Mastering Digital Electronics

An Ultimate Guide to Logic Circuits and Advanced Circuitry

—

Hubert Henry Ward

Apress®

Maker Innovations Series

Jump start your path to discovery with the Apress Maker Innovations series! From the basics of electricity and components through to the most advanced options in robotics and Machine Learning, you'll forge a path to building ingenious hardware and controlling it with cutting-edge software. All while gaining new skills and experience with common toolsets you can take to new projects or even into a whole new career.

The Apress Maker Innovations series offers projects-based learning, while keeping theory and best processes front and center. So you get hands-on experience while also learning the terms of the trade and how entrepreneurs, inventors, and engineers think through creating and executing hardware projects. You can learn to design circuits, program AI, create IoT systems for your home or even city, and so much more!

Whether you're a beginning hobbyist or a seasoned entrepreneur working out of your basement or garage, you'll scale up your skillset to become a hardware design and engineering pro. And often using low-cost and open-source software such as the Raspberry Pi, Arduino, PIC microcontroller, and Robot Operating System (ROS). Programmers and software engineers have great opportunities to learn, too, as many projects and control environments are based in popular languages and operating systems, such as Python and Linux.

If you want to build a robot, set up a smart home, tackle assembling a weather-ready meteorology system, or create a brand-new circuit using breadboards and circuit design software, this series has all that and more! Written by creative and seasoned Makers, every book in the series tackles both tested and leading-edge approaches and technologies for bringing your visions and projects to life.

More information about this series at `https://link.springer.com/bookseries/17311`.

Mastering Digital Electronics

An Ultimate Guide to Logic Circuits and Advanced Circuitry

Hubert Henry Ward

Apress®

Mastering Digital Electronics: An Ultimate Guide to Logic Circuits and Advanced Circuitry

Hubert Henry Ward
Leigh, UK

ISBN-13 (pbk): 978-1-4842-9880-0 ISBN-13 (electronic): 978-1-4842-9878-7
https://doi.org/10.1007/978-1-4842-9878-7

Copyright © 2024 by Hubert Henry Ward

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Miriam Haidara
Development Editor: James Markham
Editorial Assistant: Jessica Vakili

Cover designed by eStudioCalamar

Cover image designed by eStudio Calamar

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales..

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit https://www.apress.com/gp/services/source-code.

Paper in this product is recyclable

Dedicated to my wife, Ann.

You have been so supportive and patient while I typed away, forever, on my laptop.

Love always…

Pinch! Pinch! Pinch!

Table of Contents

About the Author ... xvii

About the Technical Reviewer ... xix

Introduction ... xxi

Chapter 1: Introduction to Logic Gates ... 1

Integrated Circuits, or ICs ... 1

Gate Technology ... 2

The Main Differences Between TTL and CMOS ... 2

Logic Families ... 3

Fan-Out and Fan-In ... 3

Unused Inputs ... 4

Handling Logic ICs ... 4

The Historical Circuits of the Logic Gates ... 4

The Laws of Logic ... 5

Diode-Resistor (DR) Logic ... 5

Analysis of Diode-Resistor Circuit 1 ... 9

Analysis of Diode-Resistor Circuit 2 ... 12

Analysis of Circuit 3 ... 14

The “4000” Series ... 19

The PLA, or Programmable Logic Array ... 20

The Binary Number System ... 21

Binary Numbers ... 21

Converting from Decimal to Binary 22
Converting from Binary to Decimal 23
Exercise 1 25
Adding and Subtracting Binary Numbers 25
Worked Example 25
Exercise 2 26
Subtracting Binary Numbers 26
Worked Example 27
Exercise 3 27
The Logic Gates 28
The AND Gate 28
The NAND Gate 30
The OR Gate 31
The NOR Gate 32
The EXOR Gate 34
The NOT Gate or Inverter 35
Summary 36
Chapter 2: Boolean Algebra 37
What Is Boolean Algebra 37
The Basic Concept 38
The NOT Gate 38
The AND and NAND Gates 39
The OR and NOR Gates 42
The Exclusive OR Gate, That Is, the EXOR Gate 44
Deriving Boolean Expressions from Logic Circuits 46
Boolean Derivation Circuit 1 47
Boolean Derivation Circuit 2 50

Boolean Derivation Circuit 3 52
Building Logic Circuits from Boolean Expressions 53
Build Logic Circuit Example 1 54
Build Logic Circuit Example 2 56
Build Logic Circuit Example 3 57
Exercise 1 59
Exercise 2 60
The Laws of Boolean Algebra 60
Commutative Law 60
Commutative Example 1 61
Commutative Example 2 61
Associative Law 61
Associative Law Example 1 61
Associative Law Example 2 63
Distributive Law 64
Distributive Law Example 1 64
Distributive Law Example 2 65
Distribution Law Example 3 66
Absorption Law 67
De Morgan's Theory 69
De Morgan's Example 1 70
De Morgan's Example 2 71
De Morgan's Examples 3 72
The OR Function with NAND Gates 77
Summary 80

Chapter 3: Simplifying Boolean Expressions 81
Some Fundamental Identities 81
The Inverse Law 81
The Identity Law 83
The Null Law 84
The Idempotent Law 85
The OR Version of the Idempotent Law 86
The OR Version of the Identity Law 86
The OR Version of the Null Law 87
The OR Version of the Inverse Law 88
Using Boolean Algebra to Minimize Expressions 90
Simplification Example 1 90
Simplification Example 2 91
Simplification Example 3 93
Simplification Example 4 95
Simplification Example 5 96
Simplification Example 6 98
Karnaugh Maps 100
Karnaugh Map Example 1 100
Using the Karnaugh Map 102
Karnaugh Map Example 2 106
Simplification Examples 110
Simplification Example 7 111
The 1st and 2nd Canonical Formats and the Minterms and Maxterms 112
The 2nd Canonical Format 113
Simplification Example 8 121

Simplification Example 9 124
Simplification Example 10 130
Simplification Example 11 132
Summary 138

Chapter 4: Moving On from the NAND Gate 139

The SR Latch 139
The De-bounce Circuit 144
The Basic SR Latch with NOR Gates 145
The Indeterminate State 148
The Clocked $\bar{S}\,\bar{R}$ Latch 149
The Master-Slave Clocked SR 153
The JK Flip Flop 157
Using the JK Flip Flop 162
The D-Type Latch 170
The T Latch 172
The Main Configurations for the JK Flip Flop 173
The JK Flip Flop 173
Summary 176

Chapter 5: Design Methods for Digital Circuits 177

Combinational and Sequential Logic 177
Combinational Logic 177
Sequential Logic 178
Representing a Digital System 178
Asynchronous and Synchronous Logic Systems 179
The Ripple Counter 180
Design Example 1: The Modulo 10 Counter 181

Design Example 2: A Non-sequential Output 184
Design Example 3: A Synchronized Sequential Circuit....................................187
Exercise 5.1 199
Design Example 4: A Synchronized Up Counter 200
Exercise 5.2...211
Design Example 5: A Modulo 6 Binary Counter 213
Determining the Inputs for the Three D-Type Latches 215
The D_0 Inputs 215
The D_1 Inputs 218
The D_2 Inputs 220
Synopsis..222

Chapter 6: State Example 3 A Bit Stream Monitor...........................223

State Diagrams 223
The State Diagram of the JK Flip Flop 224
Creating the JK Flip Flop State Table 225
Methodology for Designing Sequential Digital Logic Circuits 231
State Diagram Example 1: The Synchronized Binary Counter...........................232
Determining the Inputs for the Four D-Type Latches 235
The D_0 Input 236
Exercise 1 236
The D_1 Input 237
The D_2 Input 238
Exercise 2 239
State Diagram Example 2: The Design of a Modulo 10 Binary Counter Using State Diagrams 242
The State Table 245
Determining the Inputs for the Four D-Type Latches 247

The D_0 Input 247
The D_1 Input 249
Exercise 3 251
State Diagram Example 3: A Bit Stream Monitor 252
The D_0 Inputs 258
The D_1 Expression 259
State Diagram Example 4 262
The D_0 Expression 264
Exercise 4 267
Moore's and Mealy Diagrams 269
Summary 271

Chapter 7: Combinational Logic 273

The Tri-state Buffer 273
The Half Adder Circuit 275
The Design of the Full Adder Circuit 278
Exercise 1 286
A 3-Bit Full Adder 287
The Binary Subtractor Circuit 288
An Alternative Subtractor Circuit 294
Subtracting by Adding Decimal Numbers 294
A 4-Bit Multiplexer 298
A Demultiplexer 302
Digital Encoders 303
Application of Digital Encoders 306
The Digital Decoder 308
A Seven-Segment Decoder Chip 312
The Seven-Segment Display 314

Common Anode Seven-Segment Display..315
Common Cathode Seven-Segment Display 316
Exercise 2 323
Summary..324

Chapter 8: Shift Registers and More 325
The D-Type Latch 325
The 4-Bit Shift Register or SISO (Serial In Serial Out)....................................329
The PISO (Parallel In Serial Out) Register...330
The PIPO (Parallel In Parallel Out) Register 332
The SIPO (Serial In Parallel Out)..332
The Ring Counter 334
The Johnson Ring Counter...335
A Frequency Divider 336
The Divide by 4 Johnson Ring Counter 338
The Phase Shift Across the Latches...339
Summary..346

Chapter 9: Designing Some Useful Logic Circuits...........................347
Example 1: A Design Process for a Single Set of Traffic Lights........................347
Analysis of the Output Logic 357
Example 2: An Alternative Single Set of Traffic Lights.....................................360
Example 3: Adding a Pelican Crossing...366
An Egg Timer Circuit 378
The SN74168...379
The Practical IC We Have Looked At...391
The 7400 Quad-Two-Input NAND Gate...392
Counters..395

The 7493 Binary Counter 399
The SN74194 Multifunction Shift Register 403
Summary 407

Chapter 10: Introduction to the 555 Timer 409
The 555 Timer 409
The Pins of the 555 Timer 410
The Timer Used as a Monostable 412
The Basic Astable 418
Creating a 50/50 Duty Cycle Square Wave 424
Creating a 1Hz Square Wave 428
A PWM Application 430
Summary 436

Chapter 11: Using TINA 12 437
What Is ECAD and TINA 12 437
Running the Software 438
Creating Our First Test Circuit 440
Using a Binary Counter 447
Using Jumper Terminals 449
Creating a Macro for the 7400 IC, a Quad-Two-I/P NAND Gate 451
Using the Quad NAND 7400 Macro 457
Summary 458

Appendix: Appendix 1 459

Appendix 2: Solutions for Exercises in the Chapters 463

Appendix: Exercises 473

Index 483

About the Author

Hubert Henry Ward has nearly 25 years of experience as a college lecturer delivering the BTEC, and now Pearson's, Higher National Certificate and Higher Diploma in Electrical and Electronic Engineering. Hubert has a 2.1 Honors Bachelor's Degree in Electrical and Electronic Engineering. Hubert has also worked as a consultant in embedded programming. His work has established his expertise in the assembler and C programming languages, within the MPLAB X IDE from Microchip, as well as designing electronic circuits and PCBs using Electronic Computer-Aided Design (ECAD) software. Hubert was also the UK technical expert in mechatronics for three years, training the UK team and taking them to enter the Skills Olympics 2001 in Seoul, resulting in one of the best outcomes to date for the UK in mechatronics.

About the Technical Reviewer

Sai Yamanoor is an embedded systems engineer based in Oakland, CA. He has over ten years of experience as an embedded systems expert, working on hardware and software design and implementations. He is a co-author of three books on using Raspberry Pi to execute DIY projects, and he has also presented a Personal Health Dashboard at Maker Faires across the country. Sai is also working on projects to improve quality of life (QoL) for people with chronic health conditions. His profile can be viewed at www.linkedin.com/in/saiyamanoor/.

Introduction

The Aims and Objectives of the Book

My main aim in writing this book is to introduce you to the exciting and challenging field of digital electronics. I want to develop your desire and ability to understand how digital circuits work.

After reading this book, you should be able to do some or all of the following:

- You will understand what TTL and CMOS mean and appreciate their main differences.
- You should know what the five main logic gates are and their respective symbols and Boolean expressions.
- You should know the basics of Boolean algebra and use it to simplify logic expressions and circuits.
- You should know what Karnaugh maps are and how to use them to simplify logic circuits and expressions.
- You should know how to implement the 1st and 2nd canonical formats for Karnaugh maps.
- You will know how the JK flip flop works and how it was born out of the SR latch.
- You should be able to use the JK flip flop and the D-type latch to create a series of counters and different shift registers such as SIPO, SISO, PIPO, and PISO.
- You should understand the difference between sequential and combinational logic.

- You should be able to use a range of design techniques, that is, state diagrams, transition tables, etc.
- You should be able to create a range of combinational logic circuits such as half and full adders, binary subtractors, multiplexers, etc.
- You should understand how the 555-timer IC works and how to configure it in a range of different applications such as the monostable, the astable, and PWM.
- You should be able to design a range of logic circuits.
- You should be able to use the ECAD software TINA 12.

Prerequisites

There are no real prerequisites for you except a desire to learn about this exciting and challenging field of digital electronics.

Summary

In this introduction I have tried to give you a feel for what you will learn when you read this book. I hope I have achieved a good balance between explaining how things work and giving you enough examples that you understand the explanations and find the book useful. I know I cannot cover everything there is about this or any subject, as it is a growing area of electronics and there is always something new and exciting happening. However, I hope there is enough in the book for most of your needs and that you find it a useful resource to help in your career as a digital electronics engineer. Happy reading.

CHAPTER 1

Introduction to Logic Gates

In this chapter we will look at what an integrated circuit (IC) is and the main classification of integrated circuits. We will look at the main logic gates and how their circuitry has developed from the Diode-Resistor (DR) Logic circuit to the main Transistor-Transistor Logic (TTL) and CMOS technology.

We will look at using an Electronic Computer-Aided Design (ECAD) package that will allow us to simulate the circuits and create truth tables for the logic gates. We will study how we can use these truth tables to analyze how these circuits work.

Integrated Circuits, or ICs

An IC is a piece of silicon onto which a number of circuits have been manufactured. ICs are put into classifications according to the number of gates they contain that can vary from just a few, that is, a small IC with up to ten gates, to 10,000 gates on a very large-scale IC. We should appreciate that the technology is still advancing with more and more gates being out on these ICs.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_1

Gate Technology

The gates themselves are actual circuits made up to perform logic functions such as NOT, AND, NAND, OR, NOR, and EXOR.

The physical circuits that make up these logic functions have developed over the years, but they can be classified as follows:

RTL: Resistor-Transistor Logic

DTL: Diode-Transistor Logic

TTL: Transistor-Transistor Logic

CMOS: Same as TTL except field-effect transistors (FETs) are used instead of transistors

As the technology has moved on, the RTL and DTL have gone out of use. This leaves TTL and CMOS the most common types of this technology. Both types have their advantages over each other, but both types are still available.

TTL is normally quicker than CMOS, but the power used is higher than CMOS. Also, TTL normally uses a 5V supply voltage, whereas CMOS can be supplied from 3 to 15 volts.

The Main Differences Between TTL and CMOS

The three main areas of comparison for both types of logic families are propagation delay and power and voltage levels.

The Propagation Delay and Power and Voltage Levels for TTL

The normal power dissipation per gate is 10mW.

The normal propagation delay is 10ns.

The normal supply voltage, termed VCC, is 4.75–5.25 volts.

Logic “1” is normally from 2V to VCC and logic “0” is 0–0.8V.

The Propagation Delay and Power and Voltage Levels for CMOS

The normal power dissipation per gate is 10nW.

The normal propagation delay can vary from 25ns to 50ns.

The normal supply voltage, termed VDD, is 3–15 volts.

Logic "1" is normally from 1/3rd VDD to VDD and logic "0" is 0V to 1/3rd VDD.

Logic Families

Both TTL and CMOS ICs are grouped together in what are called families. The main family for the TTL ICs is the 74 series, and the main family for the CMOS ICs is the 4000 series. There are a series of letters that can be added to the device numbers, and the following explains what some of the letters mean:

- **L** is for low power.
- **LS** is for low power using schottky diodes.
- **H** is for high speed.
- **B** is for buffered.
- **UB** is for unbuffered.

These letters are mostly for the 74 series but some CMOS gates do have the U and UB letters added to them as well, with the same meanings.

Fan-Out and Fan-In

Fan-out quantifies how many gates can be connected to one of the outputs of the IC. A typical value for the fan-out of TTL gates is ten gates that can be connected to one of the outputs. For a CMOS device, the fan-out is usually accepted as being 50.

Fan-in quantifies how many devices can be fed into an input of the logic gate. With respect to TTL, the fan-in is restricted to just 1. This is because it is bad practice to connect the output of two or more gates together. This means no TTL gate would be able to feed more than one of its outputs to the input of a TTL gate. This same idea is applied to CMOS logic gates.

Unused Inputs

It is very bad practice to leave any unused inputs unconnected; this is termed “floating.” If we leave an input floating, hoping that it would go to a logic “1,” it would more than likely float to the wrong logic level. You should always tie unused inputs to a logic level you know won’t cause any problems with the normal operation of the IC.

Handling Logic ICs

Due to the high input impedance of the CMOS logic gates and the capacitance at their input, these logic gates are susceptible to static charge building up at the input. This can lead to severe damage to the CMOS logic gate. It is good practice to wear an earthing strap in order to discharge the buildup of charge.

TTL gates do not suffer from this type of risk. Also, the more modern CMOS devices are less susceptible, but it is still advisable to take care of handling CMOS logic devices.

The Historical Circuits of the Logic Gates

If the engineer has some understanding of the way the logic gates came about and what actual circuitry goes into making the logic gates we use, then they should be able to appreciate some of the terminology and

characteristics of the gates. When engineers began to look into creating these circuits, they were trying to develop circuits that would follow the laws of logic.

The Laws of Logic

Logic is a science that has only one of two possible results. The results are “True” and “False.” This means that the only type of question that can have a logical result is a closed question that has only one of two possible results: “True” or “False.” The actual answers could also be replaced by “Yes” or “No” or possibly “Closed” or “Open,” but the question must have only two possible answers. For example, “Has the kettle boiled?” The only logical answers are “Yes” and “No.” However, “How long will it be before the kettle boils?” is not a logical question.

Engineers soon realized that if every task could be reduced to a series of closed-type questions, then some type of circuit could be used to perform many everyday tasks. Switches or sensors could be used to determine if a condition had been met or not met, and then other circuits could be turned on or turned off accordingly. These are logical events with their logical response. Sensors and relays were used to create many logically operated circuits, hence the term “Relay Logic.” Relays had their problems as they required a voltage that could drive enough current to energize the magnet in the relay coil. Also, they had a switching life, which meant that they would eventually need replacing. Engineers wanted to create smaller devices that didn’t fail due to the limited switching life of relays and that used low voltages.

Diode-Resistor (DR) Logic

A diode is a device that acts like an electronic switch. It has two terminals: the anode “A” and cathode “K.” The symbol for a diode is shown in Figure 1-1.

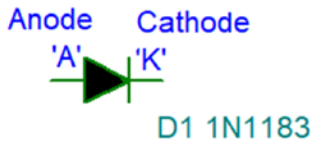


Figure 1-1. *The Basic Symbol for a Diode*

You might think that engineers can't spell, and you might be right, but we use the letter "K" for cathode as the letter "C" is used with respect to capacitance and the Coulomb. That's why we use the letter "I" for current, not just to confuse you.

The arrow shape is placed there to show the direction of conventional current flow through the diode, that is, flowing from the anode to the cathode. Just to make it more confusing, because we can, we sometimes talk about "electron flow," which flows in the opposite direction to "conventional current flow." However, in all our analyses of electrical and electronic circuits, we always use "conventional current flow."

Now back to the diode. For the switch to close and so allow current to flow through it, the anode has to become around 0.7V more positive than the cathode. It should also be noted that when the switch is closed, the maximum volt drop across the diode is fairly constant at around 0.7V. We will use the voltage of 0.7V in our analysis. Therefore, if the cathode of the diode was connected to ground, then, assuming the diode will be conducting, the maximum voltage at the anode would be around 0.7V. With that in mind, we can use an ECAD (Electronic Computer-Aided Design) software to investigate what the function of the following circuits would be. The ECAD software I use in this book is TINA version 12. There are other ECAD software packages, and each has its pros and cons. If you want to go through the simulations, but are new to TINA, then you may want to read Chapter 13 first or alongside this chapter.

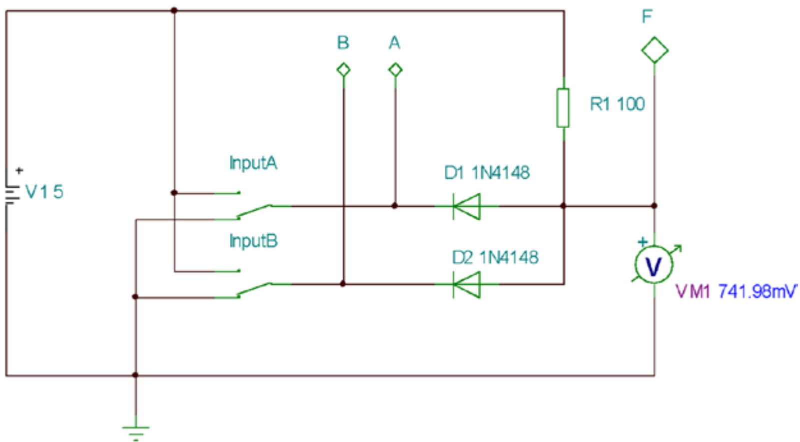


Figure 1-2. *Diode-Resistor Circuit 1*

Figure 1-2 is a simple DRL (Diode-Resistor Logic) circuit that we will look at. The switches "InputA" and "InputB" are there to represent the two inputs that can be switched to +5V, which, in TTL, is used to represent a logic "1," or switched to 0V, which is used to represent a logic "0." As there are two inputs, there are only four possible combinations of the input. Note the number of possible combinations can be determined using the following relationship:

$$\text{number of combinations} = 2^n$$

where n is the number of inputs.

$$Therefore, if\ n = 2, number\ of\ combinations = 2^2 = 4.$$

That being the case, a truth table can be developed to determine what function the circuit actually performs. The truth table is shown in Table 1-1. You should simulate the circuit with both switches connected to ground as shown in Figure 1-2. This complies with the first line in the truth table. Inputs A and B are at logic "0."

When the logic probe, at the bottom of R1, is lit, then the output, given the symbol "F" in the truth table, is said to be a logic "1." I use the letter "F" but some use "Y"; it is really down to personal preference. I use F as I don't really bother with logic circuits that have more than five inputs, A–E. When the output "F" is at a logic "1," then the voltage at the junction of the resistor would be around 5V. When the logic probe is not lit, the output is said to be at a logic "0," and the voltage would also be around 0V.

The procedure for the simulation is to change the switches as follows and record the logic at the output, the bottom of R1, as indicated by the logic probe. You should also record the voltage as measured by the DC voltmeter:

Firstly, leave switch "A" and switch "B" both connected to ground, a logic "0." Record the output "F" and the voltage at the output.

Now change switch "A" to the plus 5V, a logic "1," leaving switch "B" connected to ground, a logic "0." Record the output "F" and the voltage at the output.

Now change switch "A" back to ground and change switch "B" up to the 5V, a logic "1." Record the output "F" and the voltage at the output.

Now simply change switch "A" back up to the 5V, a logic "1," leaving switch "B" also connected to 5V, a logic "1." Record the output "F" and the voltage at the output.

This means that you should have gone through all the four possible combinations of inputs to the circuit and you should have completed the truth table as shown in Table 1-1.

Table 1-1. *Diode-Resistor Circuit 1 Truth Table*

Row	Input A	Input B	Output Voltage	Output F
1	0	0	741.98mV	0
2	1	0	772.18mV	0
3	0	1	772.18mV	0
4	1	1	4.99V	1

Truth tables are used to show the relationship between the inputs and outputs of the device. The truth table should define when the output "F" will be a logic "1." You should be able to see that the output "F" is a logic "1" only when both A and B are at a logic "1." This then suggests that this circuit can perform the logical AND function in that F will be a logic "1" only when both "A" AND "B" are a logic "1." Also, you should see that when any one of the inputs is switched to 0V, its respective diode will be switched on, and the volt drop across both diodes is 772.18mV, or 0.7712V. This agrees with our interpretation of how a diode works. When both switches are connected to the +5V supply, then the diodes will turn off, and so no current will flow through them. This then means that no current will flow through the resistor, R1, and so there can be no volt dropped across it. This is why the output voltage rises to 4.99V. The voltage is not the full 5V as there will be some leakage current through the diodes. However, the 4.99V is within the TTL range to represent a logic "1."

Analysis of Diode-Resistor Circuit 1

If we start with both input switches connected to 0V, as shown in Figure 1-2, then this means that both diodes will be turned on, as their anodes are more positive than their cathodes. This means that both diodes will allow current to flow through them. This means that current will be flowing through the resistor R1. This, in turn, means that there will be a volt drop across R1. The volt drop will be approximately 4.3V as the two diodes require a voltage of around 0.7V to keep them turned on. You should appreciate that 0.7V is too low to illuminate the voltage probe connected to the bottom of R1. It is also within the TTL range to represent a logic "0." This action is shown as being true with the first row in the truth table; see Table 1-1.

Now if switch A is connected to the +5V supply, the cathode of D1 is made more positive than its anode, which is held at 0.7V, due to the fact that D2 is still conducting, and so D1 is turned off. However, there is still a path for current to flow through R1, via D2, and the voltage at the voltage probe is still around 0.7V. The probe is still unlit. This is reflected in row 2 of the truth table.

Now, if switch A is connected to ground and switch B is connected to the +5V supply, D2 stops conducting but D1 is now conducting. This means current can still flow through R1, and so the voltage probe stays turned off. This is reflected in row 3 of the truth table.

Now if switch A is connected to +5V, while switch B is also connected to +5V, the cathodes of both diodes are at +5V. This means that both diodes will be turned off, and there will be no path for current to flow through R1. This means that there will be no voltage dropped across R1, and the voltage at the probe rises to around +5V. The probe now turns on. This is reflected in row 4 of the truth table.

Now set up the circuit shown in Figure 1-3, the Diode-Resistor Circuit 2.

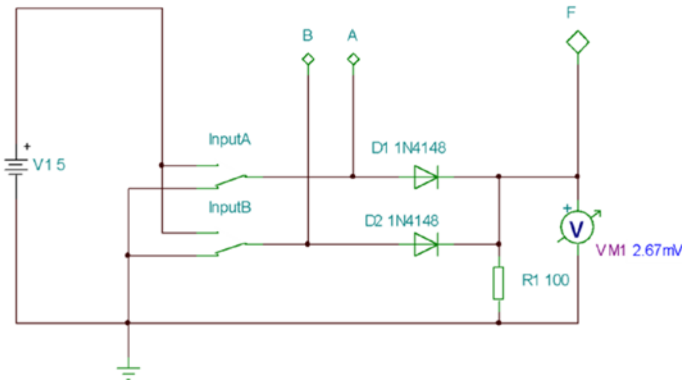


Figure 1-3. *Diode-Resistor Circuit 2*

Use the same switch sequence as with the previous simulated experiment with respect to the two-input AND gate and so fill in the truth table, shown in Table 1-2.

Table 1-2. *Diode-Resistor Circuit 2 Truth Table*

Row	Input A	Input B	Output Voltage	Output F
1	0	0	2.67mV	0
2	1	0	4.23V	1
3	0	1	4.23V	1
4	1	1	4.26V	1

There should be three sequences when the circuit produces a logic “1” at the output “F.” In this type of response from the truth table, it can be said that the output will be a logic “1” when the sequence in row 2 is true OR when the sequence in row 3 is true OR when the sequence in row 4 is true. Therefore, from the truth table, it should be possible to state that “F” will be a logic “1” when

Input A is a logic “1”

or

input “B” is a logic “1”

or

inputs A and B are both logic “1.”

This means that the circuit can be said to perform the logic OR function, that is, F is a logic “1” when the A OR B input is a logic “1.” However, the truth table also includes the AND function, as F will be a logic “1” when “A” AND “B” are both a logic “1.” Because of this, the circuit is said to perform the “inclusive OR function” as it performs the “OR” function but also includes the logical “AND” function. However, as we will see, it is normally referred to as the “OR” function.

Analysis of Diode-Resistor Circuit 2

Starting with both switches connected to ground, as shown in Figure 1-3, the anodes of both diodes are at 0V. This means both diodes are turned off, and so there is no path for current to flow through R1. This means that the voltage at both ends of the resistor R1 must be the same. As one end is connected to 0V, then the top of R1 will be at 0V, and the probe will not light. This is reflected in row 1 of Table 1-2.

Now, when switch A is connected to +5V, its anode will be more positive than its cathode, and so it will turn on. Therefore, current can flow through R1. Also, as the maximum volt drop we can have across a conducting diode is around 0.7V, the voltage at the top of R1 will rise to 4.3V, that is, the +5V supply minus the 0.7V dropped across the diode, and so the probe turns on. This is reflected in row 2 of Table 1-2.

Now, if switch A is connected to ground and switch B is connected to +5V, nothing changes as D2 is now conducting instead of D1. This is reflected in row 3 of Table 1-2.

Finally, if both switches are connected to +5V, both diodes conduct and so the probe is still lit. This is reflected in row 4 of Table 1-2.

The previous two circuits have demonstrated that the simple diode-resistor circuit can perform the logic “AND” and the logic “OR” function. This is OK but it is not possible to create the logic “NOT” function, whereby the output is the NOT, that is, the opposite, of the input, using Diode-Resistor Logic. Now simulate the following circuit shown in Figure 1-4.

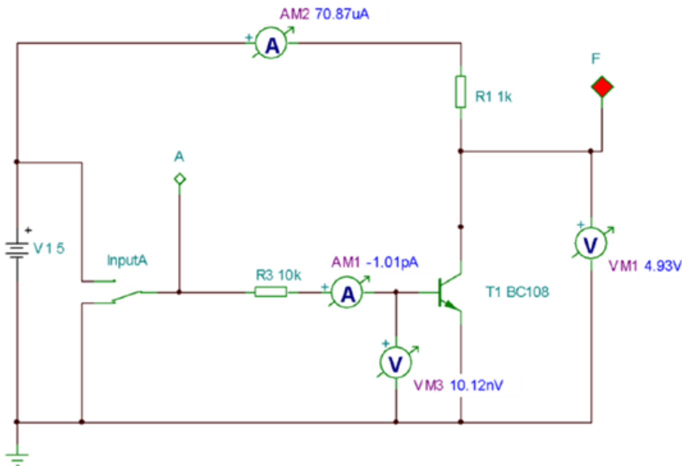


***Figure 1-4.** The NOT Gate Circuit Using RTL, Resistor-Transistor Logic*

With this circuit there is only one input, and so there are only two possible combinations. It should be clear after simulating the circuit that this circuit simply performs the NOT function as the output "F" is the opposite or "NOT" of the input. The result of the simulation is shown in Table 1-3.

***Table 1-3.** The Truth Table for the NOT Gate Circuit in Figure 1-4*

A	F	Voltage Out
0	1	4.93V
1	0	96.81mV

Analysis of Circuit 3

With switch A connected to ground, as shown in Figure 1-4, there can be no current flowing into the base of the transistor T1. Indeed, the simulation shows a very small current flowing out of the base of T1. To explain this small current, we need to appreciate that the NPN transistor can be viewed as having two diodes back-to-back as shown in Figure 1-5.

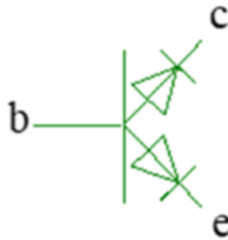


Figure 1-5. *The Two Back-to-Back Diodes of an NPN Transistor*

The diode with its anode connected to the base of the transistor and its cathode connected to the emitter is a real diode, and so it acts in the same way as a normal diode. The other diode also has its anode connected to the base of the transistor but its cathode connected to the collector. This second diode is a very weak diode, and even though, in Figure 1-4, it is reverse biased, its anode currently at 0V and the cathode at +5V, it does break down and a very small current may leak through it as shown in the simulation. Under normal conduction, when the transistor is turned on, this second diode soon breaks down and allows current to flow through it in the other way than the direction the arrow shows it should flow.

Transistors are current-controlled devices, and they need current to flow into the base to turn them on. With switch A, in Figure 1-4, connected to 0V, this is not happening as the anode of the base-emitter diode is turned off, and so T1 is turned off. This means that no current can flow through R1, and so the voltage at the collector of T1 is +5V as no voltage is dropped across R1. The probe is lit. This is reflected in row 1 of Table 1-3.

Now if switch A is connected to +5V, then the voltage at the base of T1 is connected to +5V via R3. This then means that the anode of the diode between the base of T1 and its emitter becomes forward biased. This is a real diode and it will turn on, which makes current flow into the base of T1. This forces T1 to turn on fully, and so current can flow through R1. The fact that T1 is turned fully on virtually connects the collector of T1 to ground. This then turns off the voltage probe. This is reflected in row 2 of Table 1-3.

Moving things on a bit, you should simulate the circuit shown in Figure 1-6.

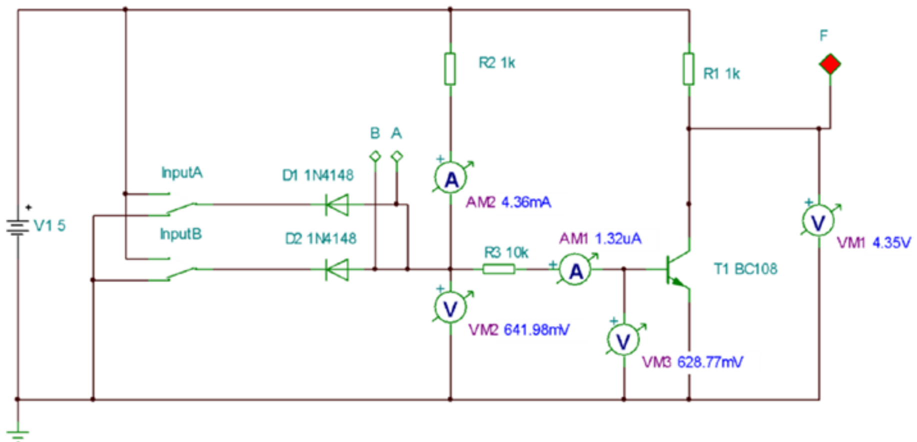


Figure 1-6. *The Diode-Transistor Logic Circuit*

Using the switches in the same way as with the previous circuits, you should simulate the circuit and complete the truth table as shown in Table 1-4.

Table 1-4. The Truth Table for the Circuit Shown in Figure 1-6

Input B	Input A	Output F	Output Voltage	Voltage at Base of T1	Voltage at Junction R2-R3	Current into Base of T1
0	0	1	4.35V	628.77mV	641.98mV	1.32μA
0	1	1	3.79V	646.76mV	672.13mV	2.54μA
1	0	1	3.79V	646.76mV	672.13mV	2.54μA
1	1	0	42.99mV	700.39mV	4.5V	379.55μA

Using this truth table, it should become clear that the function is the exact opposite of the AND function. This is termed the NOT AND, which is shortened to the NAND function.

This circuit still uses diodes, and so it should be included in the DRL series of circuits. However, as we have explained previously, within the traditional NPN transistor, there is an actual diode in the base-emitter region of the transistor. Using this fact, the two diodes in the circuit of Figure 1-6 could be replaced by two transistors as shown in the transistor-transistor circuit in Figure 1-7.

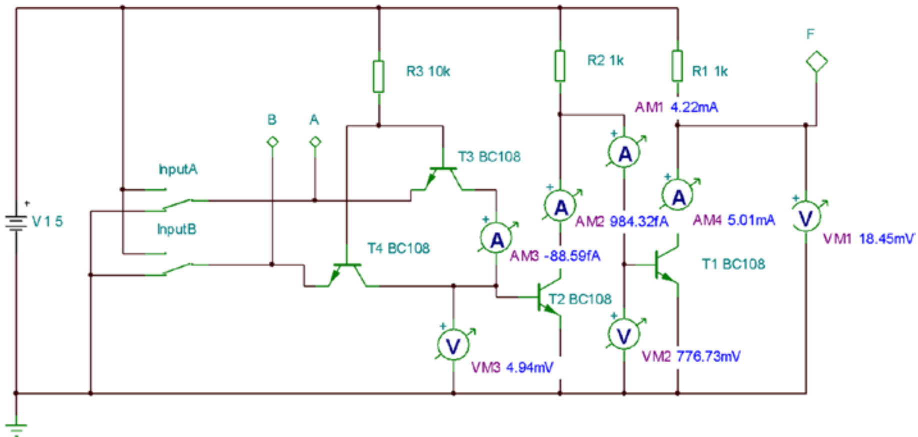


Figure 1-7. The Transistor-Transistor Circuit

The actual transistors used in TTL devices are not constructed the same way as with the simulations. They only have one base and one collector terminal, but they do have multiple emitters. However, to simulate these transistors within TINA, multiple transistors may be used with all the bases and all the collectors connected together; this gives us a single transistor with multiple emitters. If we simulate the circuit, then the truth tables shown in Tables 1-5 and 1-6 can be constructed.

Table 1-5. *The Truth Table for the Circuit in Figure 1-7 Showing the Voltage Results*

B	A	F	VM1	VM2	VM3
0	0	0	18.5mV	776.73mV	4.94mV
0	1	0	18.5mV	776.73mV	25.32mV
1	0	0	18.5mV	776.73mV	25.32mV
1	1	1	4.93V	20.82mV	746.06mV

Table 1-6. *The Truth Table for the Circuit in Figure 1-7 Showing the Current Measurements*

B	A	F	AM1	AM2	AM3	AM4
0	0	0	4.22mA	984.32fA	–88.59fA	5.01mA
0	1	0	4.22mA	997.98fA	1.37mA	5.01mA
1	0	0	4.22mA	997.98fA	–1.37mA	5.01mA
1	1	1	–990.92fA	4.98mA	1.19mA	6.43pA

When either of the inputs goes to 0V, that is, a logic “0,” then either T3 or T4 will turn on. This will drag the voltage at the base of T2, down to around 0V, that is, the saturation voltage across the transistor emitter-collector junction of T3 and T4. This means that T2 will turn off and the current that was flowing through T2, from the resistor R2, must now flow into the base of T1. This then turns T1 on clamping the voltage at its base to 0.77V, the approximate diode drop across the base-emitter junction diode. This will drag the collector of T1 down to around 0V; see VM3 in Table 1-5. In this way, when either one of the two inputs goes to logic “0,” then the output also goes to a logic “0.” This is also the case if both inputs are switched to 0V, that is, a logic “0.”

However, if both the inputs are switched to +5V, that is, to a logic “1,” both the diodes, inside the base-emitter junction of the transistors T3 and T4, turn off. This means that the base voltage at T2 will rise until T2 turns on. This will then clamp the base voltage of T2 at around 0.7V. This then means that the base voltage of T1 is dragged down to the saturation voltage of T2, that is, around 21mV; see VM2 in Table 1-5. This means T1 turns off and the voltage at its collector, which is the output of the circuit, goes to 4.93V or VCC; see VM1 in Table 1-5.

Table 1-6 shows how the current around the circuit changes as the inputs are switched through their various options.

I hope that by examining the two truth tables, Tables 1-5 and 1-6, you can see that this circuit performs the function of an AND gate as the output “F” is a logic “1” only when both input A and input B are at a logic “1.”

The circuits in Figures 1-4 and 1-7 are just some examples to show the basic concept of how the logic gate ICs are actually constructed. The actual circuits are more complicated; note each device data sheet usually shows the actual circuit for the IC. The diagram shown in Figure 1-8 is the circuit from the data sheet for the 7402 NOR gates.

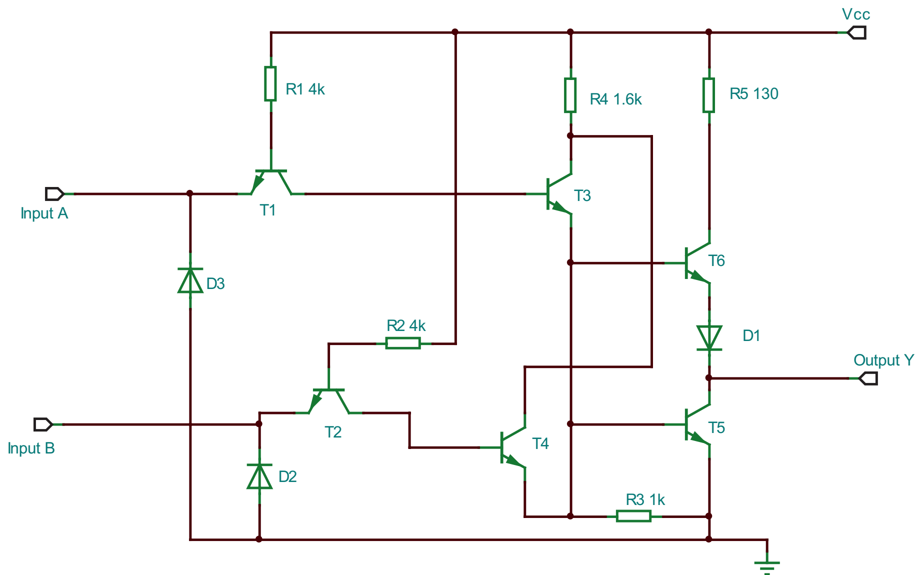


Figure 1-8. *The Internal Circuit of a 7402 NOR Gate*

The "4000" Series

With respect to CMOS ICs, the most common family is the "4000" series. CMOS logic is a technology using the Complementary Metal-Oxide-Semiconductor field-effect transistors, that is, FETs, to create the actual circuits to carry out the logic functions. The FETs draw virtually no current, as they are controlled by the voltage at the input. This makes these gates very useful in battery-powered applications. Also, they will work with a wide range of supply voltages such as 3–15 volts.

Just as the logic functions can be created from BJTs, with the TTL families, then the same logic functions can be created from FET circuits. There is no real need to go into this sort of detail, and that is left to you should you want to. However, Figure 1-9 shows a basic circuit, using FETs, that will perform the NOT logic function.

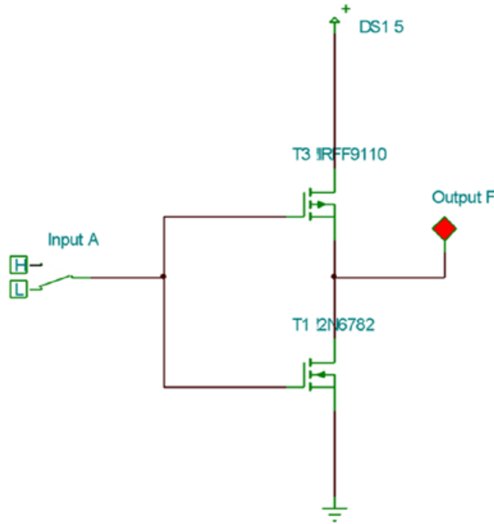


Figure 1-9. *The NOT Function Using FETs*

If you create this circuit in TINA, or your ECAD software, then simulate it; you should see that it does perform the NOT function, that is, the output "F" is the opposite of the input "A" at all times.

The PLA, or Programmable Logic Array

This is a chip whereby the interconnections of the gates have not been fixed; they are left open. The design engineer uses a computer program to design how the connections of the gates can be made to make a circuit of their choice. This circuit can then be "blown" down onto the chip to make the circuit.

The Binary Number System

As logic has only two possible states, then a number system that uses only two possible numbers would be very useful. The binary number system has, as its base number, the number 2, as it has only two unique values. These are the numbers "1" and "0." For this reason, the binary number system is an ideal system for representing logic. This is because

The logic result "True" can be represented by the binary "1."

The logic result "False" can be represented by the binary "0."

Also

The logic term "Closed" can be represented by the binary "1."

The logic term "Open" can be represented by the binary "0."

It should be noted that in electrical terms the binary number "1" would be ideally represented by a voltage of 5V and the binary number "0" would be 0V, using TTL. However, as it is very hard to reach 0V, as transistors usually saturate around 0.2V, then the two binary or logic numbers would be represented by the following:

Logic "0" is 0–0.8V.

Logic "1" is 2–5V.

It would be useful to have a better understanding of the binary number system and how it relates to our normal number system, the denary system. The following gives you an insight into the binary number system.

Binary Numbers

These are a series of "0s" and "1s" that represent numbers. We, as humans, cannot easily interpret binary numbers as we use the denary number system. The denary number system uses the base number 10, which means all the columns we put our digits in to form numbers are based on powers of 10. For example, the thousands column is based on 10^3, and the

hundreds column is based on 10^2. The tens column is 10^1 and the units column is 10^0. Try putting 10^0 in on your calculator, using the x^y button, and you will find it equals 1; in fact, any number raised to the power 0 will equal 1. The binary number system is based on the powers of 2 and $2^0 = 1$. Therefore, each of the columns represents 2^n, where n is the order of the columns, that is, 0, 1, 2, 3, 4, etc.

Converting from Decimal to Binary

Probably the first step to understanding binary numbers is in creating them, that is, converting decimal to binary. There are numerous ways of doing this, but I feel the most straightforward method is to repeatedly divide the decimal number by 2, the base number of the binary system. This is shown in the following:

Example 1

Convert 66 to binary.

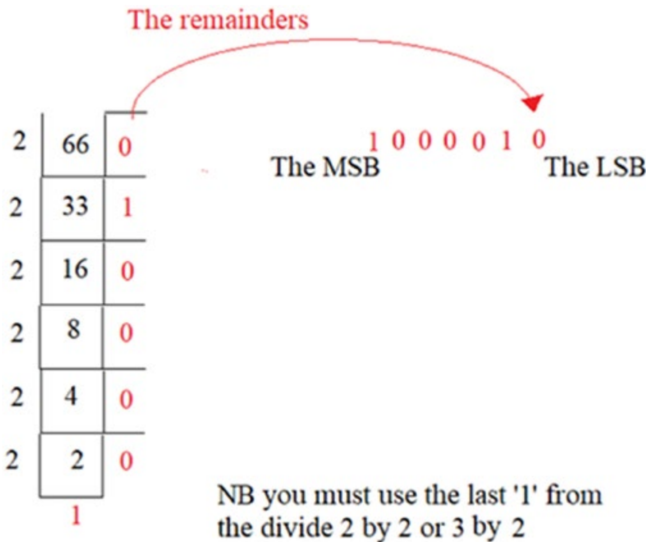


Figure 1-10. *Repeatedly Divide 66 by 2*

Simply keep on dividing the number by 2, putting the answer underneath as shown in Figure 1-10, with the remainder to the side. You should note that all the remainders are either **0** or **1**. These digits actually make up the binary number. Note also the last division always results in an answer "**1**"; we stop there, no more dividing.

To create the binary number, we take the top of the remainders, as shown, and put it into the least significant bit, or column, for the binary number. The other remainder digits follow on, thus making up the complete seven-digit number.

Converting from Binary to Decimal

It would be useful to determine if the binary number shown does actually relate to 66 in decimal. This is done by converting back into decimal the binary number 1 0 0 0 0 1 0. To do this we must realize that numbers are displayed in columns. The columns are based on the base number of the system used. With binary numbers the base number is 2. Therefore, the columns are based on powers of 2. This is shown in Table 1-7.

Table 1-7. *The Columns of the Binary Number System*

Base No.	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Equivalent	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Binary Number		**1**	**0**	**0**	**0**	**0**	**1**	**0**

To complete the conversion, we simply sum all the decimal equivalents where there is a 1 in the binary column.

In this case the sum is 64 + 2 = 66 as there are only 1s in the 64 column and the 2 column of the decimal equivalent row.

Example 2

Convert 127 to binary and check the result.

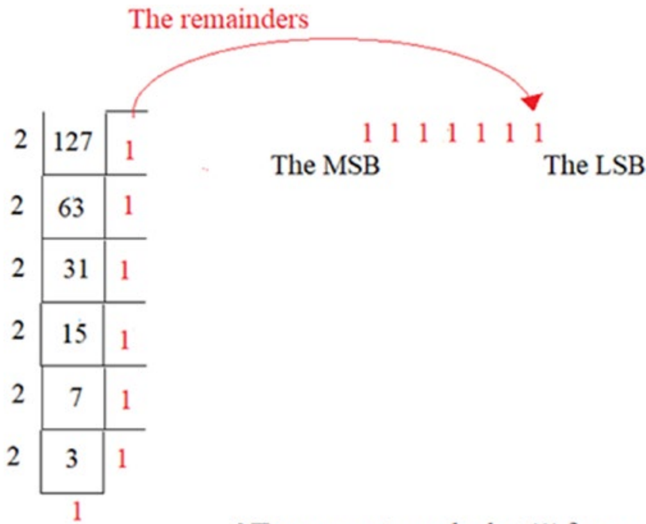


Table 1-8. *The Conversion Back to Decimal*

Base No.	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Equivalent	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Binary Number	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

To complete the conversion, we simply sum all the decimal equivalents where there is a 1 in the binary column, as shown in Table 1-8.

In this case the sum is 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127, which is what we expect.

Exercise 1

Convert the following numbers to binary and check your results by converting back to decimal. **Show all workings out**:

99

255

137

Adding and Subtracting Binary Numbers

Adding and subtracting numbers is perhaps the most basic operation we can carry out on numbers. Binary numbers follow the same rules as decimal, but there are only two allowable digits. Also, computers don't actually subtract numbers as the following will show.

Worked Example

Remember binary numbers have only two digits: "0" and "1."

Add 23 to 21 in 8-bit binary.

Method:

Convert 23 and 21 into 8-bit binary and add. Remember the following four rules:

0+0 = 0

0+1 = 1

1+0 = 1

1+1 = 0, with 1 to carry

23 in 8-bit binary is

0 0 0 1 0 1 1 1

Note we must state all 8 bits as it is 8-bit binary.

By the same process, 21 in binary is

0 0 0 1 0 1 0 1

Therefore, the sum is

```
  0 0 0 1 0 1 1 1
+ 0 0 0 1 0 1 0 1
-----------------
  0 0 1 0 1 1 0 0
```

To check your answer, put the result into the lookup table and then add the decimal equivalents as shown in Table 1-9.

Table 1-9. *Converting the Result Back to Decimal*

Power	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal Equi.	128	64	32	16	8	4	2	1
Binary Number	0	0	1	0	1	1	0	0

The sum is 32 + 8 +4 = 44.

Exercise 2

Add the following decimal numbers in 8-bit binary notation. Note: Check your answers.

20 + 21, 35 + 123, 125 + 75

Subtracting Binary Numbers

Microprocessor-based systems actually subtract numbers using a method that is addition. This involves using the 2's complement of a number, and it is best explained by the following example.

Worked Example

128 – 28

Convert the two numbers to binary using the method shown previously.

128 in 8-bit binary is 10000000. **Note we must use all 8 bits.**

28 in 8-bit binary is 00011100.

Take the 2's complement of 00011100 as this is the number that we are subtracting from 128.

Only create the 2's complement of the number we are subtracting with.

Note we must use a full 8-bit number, putting extra 0 in where needed.

To take the 2's complement, firstly take the complement and then add binary 1 to the complement. The complement of the binary number is found by simply flipping all the bits, that is, a "0" becomes a "1" and a "1" becomes a "0."

The complement of 00011100 is 1 1 1 0 0 0 1 1.

Add binary 1: 1 1 1 0 0 0 1 1 + 0 0 0 0 0 0 0 1 = 1 1 1 0 0 1 0 0.

Now add the 2's complement to the first binary number as shown:

```
  1 0 0 0 0 0 0 0
+ 1 1 1 0 0 1 0 0
-----------------
```

The result is 0 1 1 0 0 1 0 0

Note the last carry into the ninth digit is discarded as there can only be the specified number of digits, 8 in this case. Don't forget we added 1 so we should give it back.

The binary result converts to 100 in decimal. This is the correct result.

Exercise 3

Subtract the following decimal numbers using 8-bit binary 2's complement:

128 – 28, 79 – 78, 55 – 5, 251 – 151

Check your answers in the usual way.

In Chapter 7 of the book, we will create a logic adder circuit and subtractor circuit. This will look at a different method of subtracting binary numbers.

The Logic Gates

The preceding text has shown how the actual gates are constructed out of transistors with resistors for TTL and MOS FETs with resistors for the CMOS families. This was to give you an idea of the principle behind the actual circuitry for the logic gates.

There are six main logic gates, and they are shown, in the following figures, with their IEC and ANSI symbols and truth tables.

The AND Gate

With the TTL families, a typical two-input AND gate IC would be the 7408. Figure 1-11 shows this gate using the IEC standard labels. I prefer the IEC symbols as the symbology inside the rectangle does give some indication of the function of the gate. The "&" symbol suggests that this gate performs an AND function. This is shown in Figure 1-11.

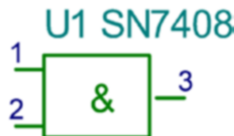


Figure 1-11. *The TTL AND Gate 7408 Using the IEC Symbol*

With the CMOS families, the AND gate would be 4081, which is an IC with four two-input AND gates. The ANSI symbol for the AND gate is shown in Figure 1-12.

Figure 1-12. *The CMOS AND Gate Using the ANSI Symbol*

I don't use these ANSI symbols as there is no indication of their function; you basically have to commit to memory what the symbol means. Figure 1-13 shows the typical pin-out for the 7408 IC.

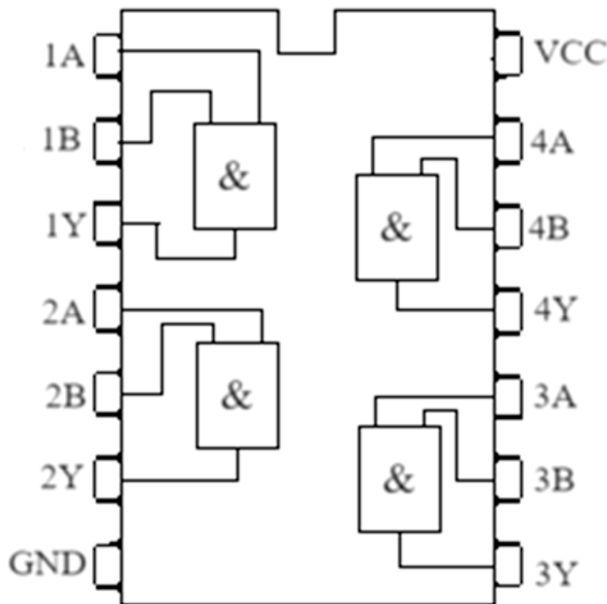


Figure 1-13. *The Actual Gate Arrangement for the Four-Two-I/P-AND-Gate IC 7408*

The NAND Gate

With respect to the TTL family, the NAND gate is the 7400. The IEC symbol is shown in Figure 1-14.

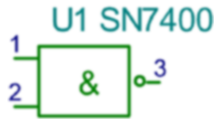


Figure 1-14. *The IEC Symbol for the 7400 TTL NAND Gate*

The phrase "NAND" means the "Not AND." This means that the output is the opposite of the output of the AND gate. The small circle, which is sometimes a small triangle, on the output is there to show that this is the NOT of the AND, which means it is a NAND gate. Figure 1-15 shows the NAND gate using the ANSI symbol.

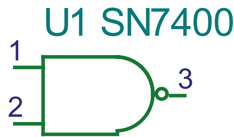


Figure 1-15. *The ANSI Symbol of the CMOS 4011 NAND Gate*

Note the small circle on the output means the NOT AND, which means it is the NAND gate.

The basic pin-out diagram for a two-input NAND gate is shown in Figure 1-16.

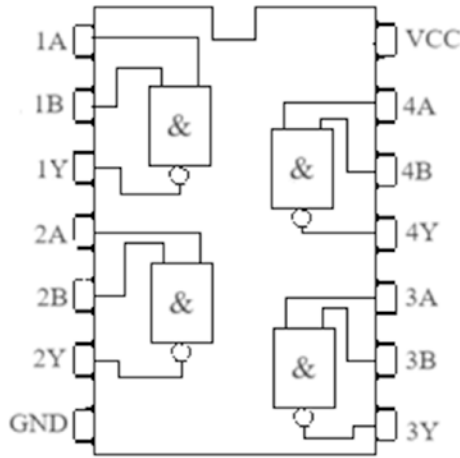


Figure 1-16. *The Basic Pin Arrangement for the 7400*

The OR Gate

The IEC symbol for the OR gate is shown in Figure 1-17.

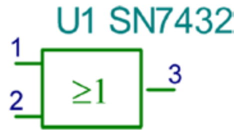


Figure 1-17. *The IEC Symbol for the OR Gate*

The equal to or greater than 1 symbol tries to indicate that the output will be a logic “1” when the inputs are equal to or greater than 1. This will be the case when input A is a logic “1” or input B is a logic “1” and also when both “A” and “B” are logic “1.” The ANSI symbol for the OR gate does not give any indication of what it is used for. This is shown in Figure 1-18.

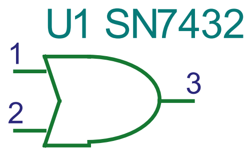


Figure 1-18. *The ANSI Symbol for the OR Gate*

This symbol gives no indication of what the logic function is for the gate. That is one reason I prefer to use the IEC symbols. Also, the IEC symbols are easier to create in a drawing package.

The basic pin-out of the 7432 is shown Figure 1-19.

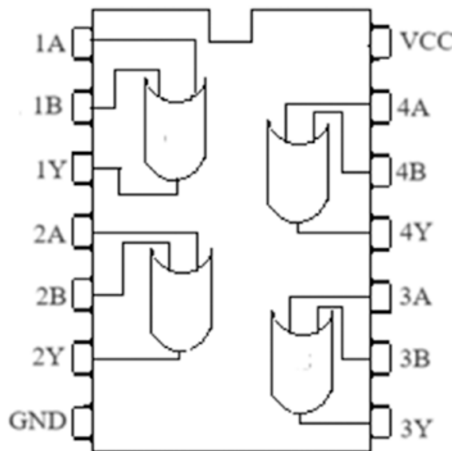


Figure 1-19. *The Basic Pin-Out Diagram for the 7432*

The NOR Gate

The IEC symbol for the NOR gate is shown in Figure 1-20.

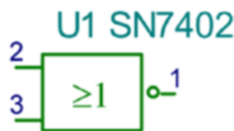


Figure 1-20. *The IEC Symbol for the Logic NOR Gate*

Figure 1-21 shows the ANSI symbol for the NOR gate. Note the simple addition of the circle on the output.

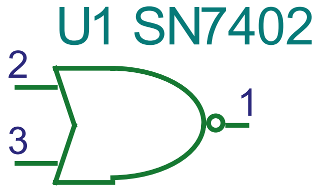


***Figure 1-21.** The ANSI Symbol for the Logic NOR Gate*

The basic pin-out diagram for the 7402 is shown in Figure 1-22.

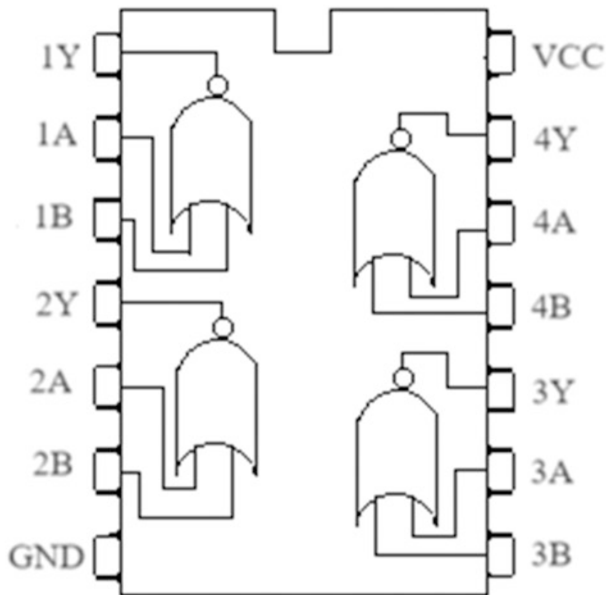


***Figure 1-22.** The Basic Pin-Out Diagram for the 7402*

The EXOR Gate

The IEC symbol for the EXOR gate is shown in Figure 1-23.

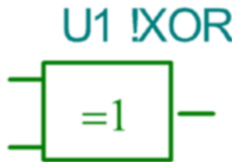


Figure 1-23. *The IEC Symbol for the EXOR Gate*

One thing that is worth mentioning about this symbol is that the software TINA uses the symbol "!", which is the symbol of "NOT" in C programming. The symbology of "=1" is there to try and indicate that the output "F" will be a logic "1" when the inputs total 1. This will be true only when input A OR input B is a logic "1" and not when both are a logic "1." Figure 1-24 shows the ANSI symbol for the EXOR gate.

Figure 1-24. *The ANSI Symbol for the EXOR Gate*

The basic pin-out of the 74HX386 is shown Figure 1-25.

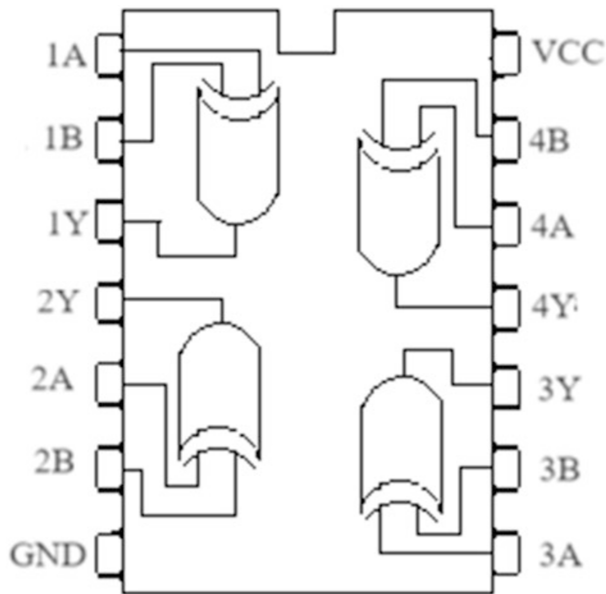


Figure 1-25. *The Basic Pin-Out Diagram for the 74HX386 EXOR Gate*

The NOT Gate or Inverter

The IEC symbol for the NOT gate or inverter gate is shown in Figure 1-26. The ANSI symbol is shown in Figure 1-27.

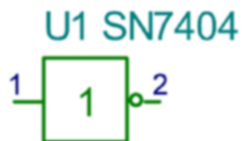


Figure 1-26. *The IEC Symbol for the Inverter or NOT Gate*

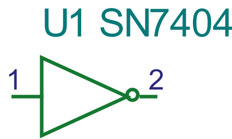


Figure 1-27. *The ANSI Symbol for the NOT Gate*

The basic pin-out diagram for the 7404 is shown in Figure 1-28. The inputs are on the "A" pins, while the outputs are on the "Y" pins.

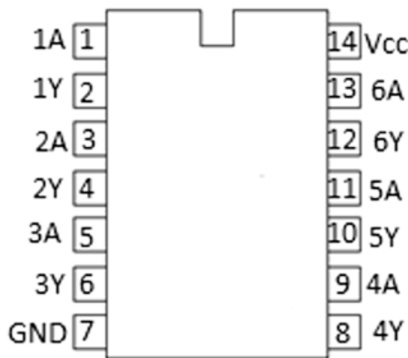


Figure 1-28. *The Basic Pin-Out Diagram for the 7404 Hex Inverter*

Summary

In this chapter we have looked at the basic construction of DRL, DTL, TTL, and CMOS logic gates. We have also looked at the two different systems used to symbolize the logic gates that we will use. We have studied the binary number system and why it is used to represent logic.

In the next chapter, we will investigate the use of Boolean algebra before going on to use it to describe the function of logic circuits. We will study the 1st and 2nd canonical formats of describing truth tables.

CHAPTER 2

Boolean Algebra

In Chapter 1 we looked at the six basic logic gates and how they can be constructed using different devices. In this chapter we will study Boolean algebra and how it can be used to describe the six basic logic gates. We will also study the basic laws of Boolean algebra.

We will learn how we can use truth tables to create Boolean expressions from them. Finally, we will go on to study De Morgan's theory.

What Is Boolean Algebra

In nearly every aspect of engineering, we like to have a method of describing systems using equations or expressions. This concept is true of digital electronics, and it is to this end that engineers use Boolean algebra. Boolean algebra is a type of mathematics that deals with operators used in logic. Its origins are found in a book first written by George Boole back in 1854, hence where it gets its name. It uses letters to describe variables used in the functions, and these variables can normally only take up the binary values of "1" and "0." Hence, Boolean algebra is used to describe the function of logic gates and logic circuits. Therefore, it is an area of electronics that the would-be engineer should try to master. I hope the following text will teach you all you need to know about this topic and show you it is not too difficult to master.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_2

The Basic Concept

All digital gates or circuits use inputs to produce an output. The number of inputs used ranges from one up to any number. It is normal to represent the individual inputs using letters, A, B, C, D, etc. All the inputs can only have one of two possible states:

Either logic "0" or logic "1"

The output is also allocated a letter, usually "F" or "Y." We will use the letter F. The output can, as with the inputs, only take up one of two possible states:

Either logic "0" or logic "1"

With Boolean algebra we write the expression that will produce a logic "1" at the output, that is, when F will equal to 1. We will start by looking at the six basic logic gates and write down the Boolean expression for them. In Chapter 1 we looked at the symbols for these gates; in this chapter we will look at some simple circuits that will allow us to simulate these gates and so create the truth tables for the gates.

The NOT Gate

This is the simplest of gates as it has only one input and one output. The function of the gate is to make the output the inversion of the input, that is, the NOT of the input.

Figure 2-1 shows the IEC symbol for the NOT gate.

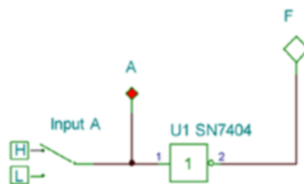


Figure 2-1. *The Test Circuit for the NOT Gate*

The Boolean expression is

$$F = \overline{A}$$

If we simulate the circuit as shown in Figure 2-1, we should see that the output "F" is the inverse, or NOT, of the input.

The AND and NAND Gates

If we simulate the circuit as shown in Figure 2-2, we should be able to construct the truth table for these two logic functions. Note, as the circuits in this chapter will be using logic gates, you choose the DIG, for digital simulations, when simulating the circuits in TINA, instead of the DC we used in Chapter 1.

Truth tables are a method by which we can record the response of the output, "F," to all the possible combinations of the inputs to the logic gate or logic circuit. As we are simulating logic devices and circuits, the inputs can only take up one of two possible states. We can use this understanding to determine the number of possible combinations of the inputs using the expression

$$Number\ of\ combinations = 2^n$$

The term "n" relates to the number of inputs. Therefore, as these simulations will only use logic gates with two inputs, we can determine the number of combinations as

$$Number\ of\ combinations = 2^2 = 4$$

Now simulate the circuit and complete the truth table for the AND and NAND functions.

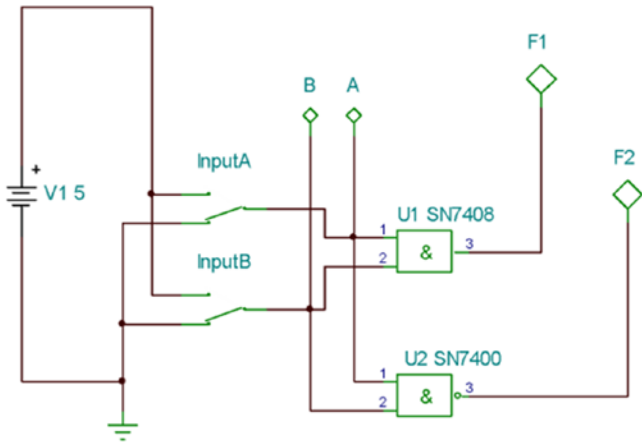


Figure 2-2. *The Simulation Circuit for the Logical AND and NAND Functions*

In this first circuit, we are using the single pole double throw switch, with the 5V battery, to change the inputs from 0V, logic "0," to 5V, logic "1." In the future simulations, we will use the High Low switch to change the states of the inputs. Both these switches are available from the "Switches" tab on the component category menu in the TINA software.

To fill in the truth table, as shown in Table 2-1, we should change the inputs A and B, starting with both at logic '0', and record the state of the two outputs: F1 and F2. Note "F1" is the output of the AND gate and "F2" is the output of the NAND gate. We have included the NAND output with the AND output, because it does show clearly that the NAND is the opposite of the AND.

Table 2-1. *The Truth Table for the AND and NAND Gates*

Input A	Input B	Output F1 AND Function	Output F2 NAND Function
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

The convention to constructing truth tables is to write the inputs in an alphabetical manner from left to right. Then, when writing the values for the inputs, we increase them as we normally write numbers from right to left, that is, the least significant bit, the LSB, on the right going to the most significant bit, the MSB, on the left. This convention of labeling the inputs makes input A the MSB and input B the LSB.

However, we could have labeled the inputs in the other way around, but then that would have made Input B the MSB and input A the LSB. This will become more important when we move on to Karnaugh maps.

Completing the simulation should allow you to complete a truth table that is the same as that shown in Table 2-1. We can see that with the AND gate, the output "F1" is only a logic "1" when both inputs, A AND B, are a logic "1." You can also see the output "F2" is the exact opposite of the AND function. That is what NAND or Not AND means. The response is the NOT, or inverse, of the AND function. The Boolean expressions are as follows:

The AND function is

$$F = A.B$$

The NAND function is

$$F = \overline{A.B}$$

The bar, above the A.B, for the NAND function is used to indicate the NOT or inverse of the term.

The OR and NOR Gates

If we simulate the circuit as shown in Figure 2-3, we should be able to construct the truth table for these two logic functions.

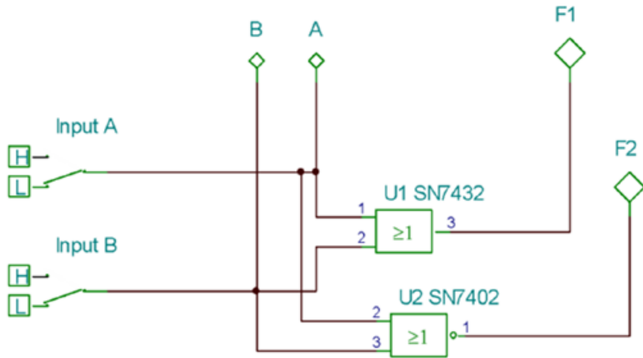


Figure 2-3. *The Simulation Circuit for the Logical OR and NOR Functions*

We are now using the High Low switches that are available in TINA specifically for use with digital inputs. With respect to the inputs "A" and "B," TINA allows us to allocate a key press from the keyboard to the switches. I have allocated "A" to input A and "B" to input B. The process to do this is to simply double-click the mouse on the switch, and then the properties window should appear. If you now click the mouse on the small black arrow next to the (-), a drop-down menu should appear. You can now simply select the key press you want to use. This process is shown in Figure 2-4.

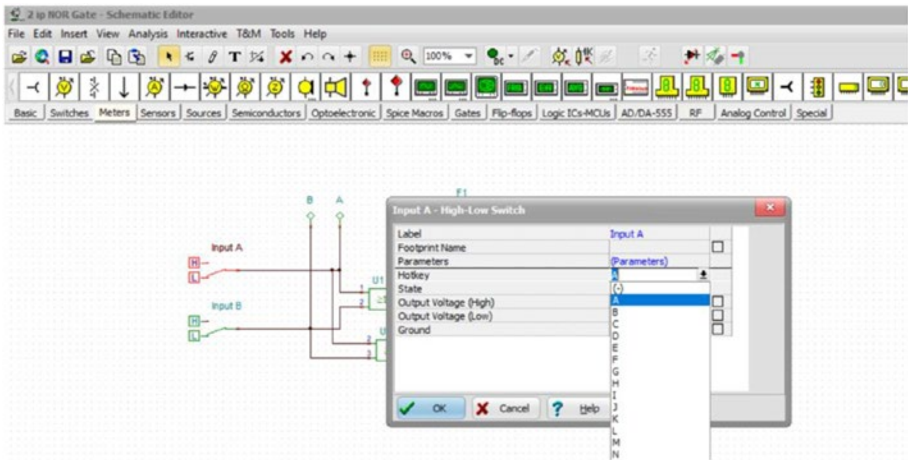


Figure 2-4. Allocating a Key Press in TINA

Now, if you go through the normal sequence of inputs, you should be able to complete the truth table as shown in Table 2-2.

Table 2-2. The Truth Table for the OR and NOR Gates

Input A	Input B	Output F1 OR Function	Output F2 NOR Function
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

The output F1 shows that the output is a logic “1” when input A or input B is a logic “1.” Moreover, the output F1 is a logic “1” when input A AND input B are a logic “1.” That is why this OR gate is sometimes, and should be, called the “inclusive OR gate” as it includes the logical AND function. When engineers built these gates in the early days, they could not

produce a true OR function. Later, as we will find out, engineers produced a gate that did not include the AND function and was the true OR function. However, the term *OR gate* has now been established as belonging to this gate. Therefore, engineers decided to call this new gate the EXOR, or exclusive OR, gate as it excludes the AND function.

You should be able to see that the NOR output "F2," shown in Table 2-2, is the opposite of the OR function. The Boolean expressions are as follows:

The OR is

$$F = A + B$$

The NOR is

$$F = \overline{A + B}$$

Again, we see the use of the bar to indicate the NOT or inversion.

The Exclusive OR Gate, That Is, the EXOR Gate

If we simulate the circuit as shown in Figure 2-5, we should be able to construct the truth table for the logic function for the EXOR. The NOT EXOR is not a common gate, and it is not found in TINA. Therefore, I will not use it here.

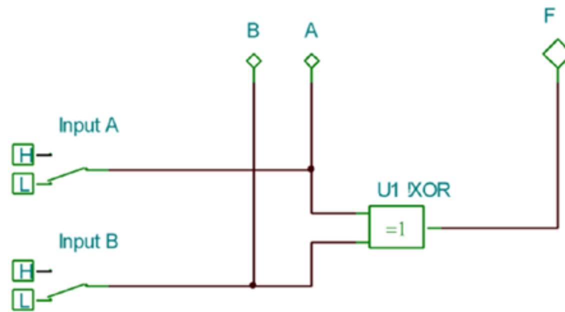


Figure 2-5. *The Simulation Circuit for the Logical EXOR Function*

Going through the simulation, by changing the input switches as before and recording the response of the output "F," should enable us to complete the truth table as shown in Table 2-3.

Table 2-3. *The Truth Table for the EXOR Gate*

Input A	Input B	Output F EXOR
0	0	0
0	1	1
1	0	1
1	1	0

We can see from the truth table that the response is the true OR response, as the output F is a logic "1" only when input A OR input B is a logic "1." It does exclude the AND function.

The Boolean expression for the EXOR function is

$$F = A \oplus B$$

We could use the truth table shown in Table 2-3 to derive a different Boolean expression for the EXOR gate. Indeed, we could do the same for the other gates and any truth table we might want to use. The process is to write down the combination of the inputs for every occurrence when the output F is a logic "1." The Boolean expression will then be the OR of each combination. In Table 2-3 we see that there are two such occurrences. The first occurrence is when A = 0 and B = 1. This would be written as

$$\overline{A}.B$$

The bar above the "A" variable means NOT A, that is, A is NOT a logic "1" as it is a logic "0."

The second occurrence is when A = 1 and B = 0. This would be written as

$$A.\overline{B}$$

This means that using the truth table, as shown in Table 2-3, the expression for the EXOR gate's output "F" is

$$F = \overline{A}.B + A.\overline{B}$$

However, we use the Boolean expression for the EXOR as stated earlier.

Deriving Boolean Expressions from Logic Circuits

Now that we have got the expressions for the basic logic gates, it would be useful to look at how we can apply Boolean algebra to logic circuits. We need to be able to derive the Boolean expression from a logic circuit and create the logic circuit that complies with a Boolean expression. We will study an approach for both tasks that I have found works for me. There may be other methods, but I know these methods work well.

Boolean Derivation Circuit 1

With respect to the logic circuit shown in Figure 2-6, we will derive the expression for it. There are two approaches we can use, one that involves changing the inputs through all their possible combinations and creating a truth table and the other by inspection of the circuit. We will go through both approaches now.

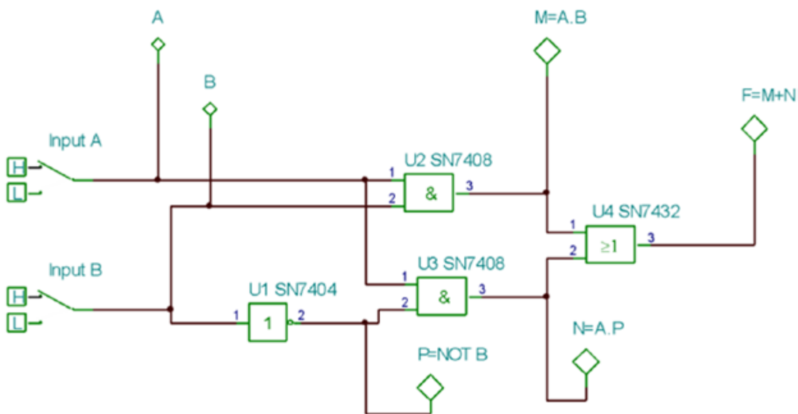


Figure 2-6. *The Figure for Derivation Circuit 1*

We should start by working backward from the output. Doing that, the first gate we come across is an OR gate. We can describe the two inputs to that OR gate as “M” and “N.”

Therefore, we can say

$$F = M + N$$

If we now look at the input “M” to that OR gate, we can see that it is the output of an AND gate. The two inputs to that AND gate come directly from the inputs A and B.

Therefore, we can say that

$$M = A.B$$

If we now look at the other input, that is, the "N" input, to the final OR gate, we see that it, too, is the output of an AND gate. One of the two inputs to that AND comes directly from the input "A," and we can say the other is "P."

Therefore, we can say that

$$N = A.P$$

Looking further back, we can see that the input "P" is the output of a NOT gate that gets its input directly from "B."

Therefore, we can say that

$$P = \overline{B}$$

There are no more gates between where we are and the input switches; therefore, we can start to substitute back in the expressions, until we get to the final gate.

Therefore, substituting for "P" in the expression for "N," we can say that

$$N = A.\overline{B}$$

Then we can substitute for "M" and "N" in the expression for "F."

The final expression becomes

$$F = \left(A.B\right) + \left(A.\overline{B}\right)$$

If we simulate the circuit, we should be able to change the inputs through all their possible combinations and so complete the truth table as shown in Table 2-4.

Table 2-4. *The Truth Table for the Simulation of Derivation Circuit 1*

Input A	Input B	Output F
0	0	0
0	1	0
1	0	1
1	1	1

We can now use this truth table to derive a Boolean expression for the circuit. The process is to write down the combination of the inputs for every occurrence when the output F is a logic "1." There are two such occurrences, and the Boolean expression will be the OR of each combination. This means that

$$F = combination1 + combination2$$

If we now insert the state of the inputs for all the combinations, we get

$$combination1 = A.\overline{B}$$

$$combination2 = A.B$$

This means that the complete expression for the circuit is

$$F = A.\overline{B} + A.B$$

I have not put brackets around the two sets of variables; that is because the dot ".", or AND term, groups the variables together and the cross "+", or the OR, separates the terms. It is really down to personal preference if you want to include the brackets or not. I do sometimes to make it clear what variables or terms are grouped together.

The expression for "F" shows us that the expression we have derived from the circuit is the same as the expression we have derived from the truth table. This should give us some confidence that we are correct in our work. However, it is not always the case that the two approaches give the exact same expression.

Boolean Derivation Circuit 2

We will now derive the expression for the circuit shown in Figure 2-7 using the same two approaches.

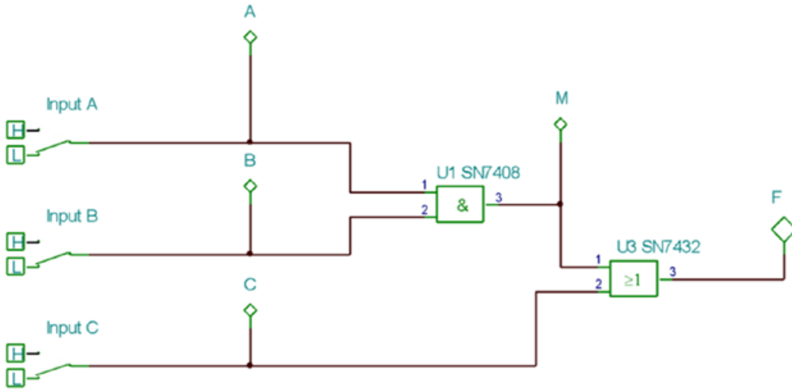


Figure 2-7. *The Figure for Derivation Circuit 2*

Starting from the output, the first gate we come across is an OR gate. We can describe the two inputs as "M" and the input C.

Therefore, we can say

$$F = M + C$$

The input M is the output of an AND gate whose inputs come directly from the inputs A and B.

Therefore, we can say

$$M = A.B$$

All we need to do now is substitute for "M" in the expression for "F." Therefore, we can say

$$F = \left(A.B\right) + C$$

Simulating the circuit allows us to complete the truth table shown in Table 2-5. As there are three inputs, there will be 2^3 combinations, that is, 8 combinations of input states.

Table 2-5. *The Truth Table for the Simulation of Derivation Circuit 2*

Input A	Input B	Input C	Output F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

There are five occurrences when the output "F" is a logic "1," and the Boolean expression will be the OR of each combination. This means that

$$F = combination1 + combination2 + combination3 \\ + combination4 + combination5$$

Looking at each combination, we get

$$combination1 = \overline{A}.B.C,\ combination2 = A.\overline{B}.\overline{C},\ combination3 = A.\overline{B}.C,\ combination4 = A.B.\overline{C},\ combination5 = A.B.C$$

If we now insert the state of the inputs for all the combinations, we get

$$F = \overline{A}.B.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.\overline{C} + + A.B.C$$

The two expressions are not the same, but that should not worry us. We might find that we can minimize this expression after reading Chapter 3.

Boolean Derivation Circuit 3

We will now derive the expression for the circuit shown in Figure 2-8.

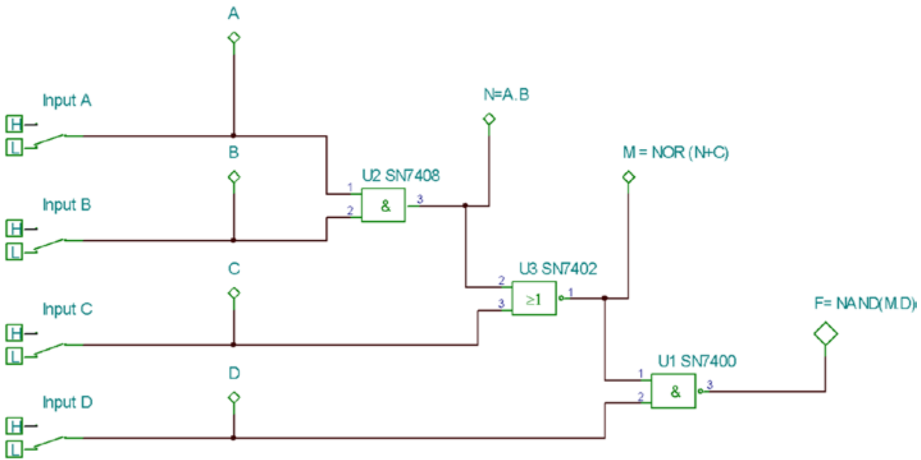


Figure 2-8. *The Simulation for Derivation Circuit 3*

Starting from the output, the first gate we come across is a NAND gate. We can describe the two inputs as “M” and the input D.

Therefore, we can say

$$F = \overline{M.D}$$

The input D is direct from the switch for input D. The input "M" is the output of a NOR gate that has two inputs, N and C.

Therefore, we can say

$$M = \overline{N + C}$$

The input "C" is direct from the switch for input "C." However, the input "N" is the output of an AND gate. The inputs to that AND gate are direct from the circuit inputs A and B.

Therefore, we can say

$$N = A.B$$

All we have to do now is substitute back to the output "F." Therefore, substituting for "N" in the expression for "M," we get

$$M = \overline{(A.B) + C}$$

Now substituting for "M" in the expression for "F", we get

$$F = \overline{\overline{(A.B) + C}.D}$$

Building Logic Circuits from Boolean Expressions

It is equally important to be able to build logic circuits from the Boolean expressions as it is to be able to derive the expressions from the circuits. The process is fairly similar, and it is best explained by going through some examples.

Build Logic Circuit Example 1

With respect to the following Boolean expression, build the circuit that fits the expression:

$$F = (A.B) + A$$

It does not really matter if you start at the inputs and work forward or start at the output and work backward. I prefer to start at the output. There are two terms that make up the expression for “F” because we can group the terms in the bracket, that is, $(A.\ B)$, into one variable, such as “M.” Therefore, we can rewrite the expression as

$$F = M + A$$

This means that the gate at the output is a two-input OR gate. One input comes direct from the switch for input A. However, the other input, “M,” is the output of a two-input AND gate. The two inputs for this AND gate come direct from the switches for input A and input B. Therefore, the circuit can be constructed as shown in Figure 2-9.

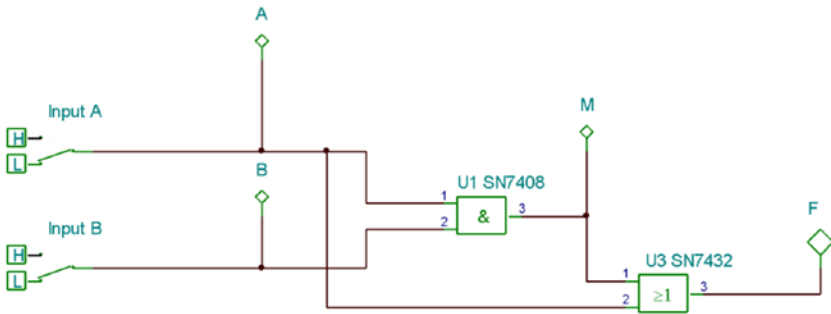


Figure 2-9. *The Circuit for Build Example 1*

We can operate the two input switches as normal to complete the truth table shown in Table 2-6.

Table 2-6. *The Truth Table for the Simulation of Build 1*

Input A	Input B	Output F
0	0	0
0	1	0
1	0	1
1	1	1

We can derive a Boolean expression from the truth table. The process is to write down the combination of the inputs for every occurrence when the output F is a logic "1." There are two such occurrences, and the Boolean expression will be the OR of each combination. This means that

$$F = combination1 + combination2$$

If we now insert the state of the inputs for all the combinations, we get

$$F = A.\overline{B} + A.B$$

However, if you look carefully while you simulate the circuit, you should see that the input B has no effect on the output "F." The circuit acts as though this input was not there. Indeed, if you look at the truth table, you will see that the output "F" is a logic "1" regardless of the logic of the input B. This suggests that the expression for the output could be

$$F = A$$

We will see in Chapter 3 that the initial expression can be simplified using Boolean algebra to F = A.

Build Logic Circuit Example 2

The Boolean expression for Example 2 is

$$F = \left(A + B\right) . \left(\overline{B.C}\right) . \left(A + C\right)$$

This can be rewritten using three terms that are ANDed at the output. Therefore, we can say

$$F = M.N.O$$

where the expressions for the three terms are

1. $M = A + B$
2. $N = \overline{B.C}$
3. $O = A + C$

This means that there are four gates used to make up the circuit, which are

- One three-input AND gate
- One two-input NAND gate
- Two two-input OR gates

The circuit that complies with the Boolean expression is shown in Figure 2-10.

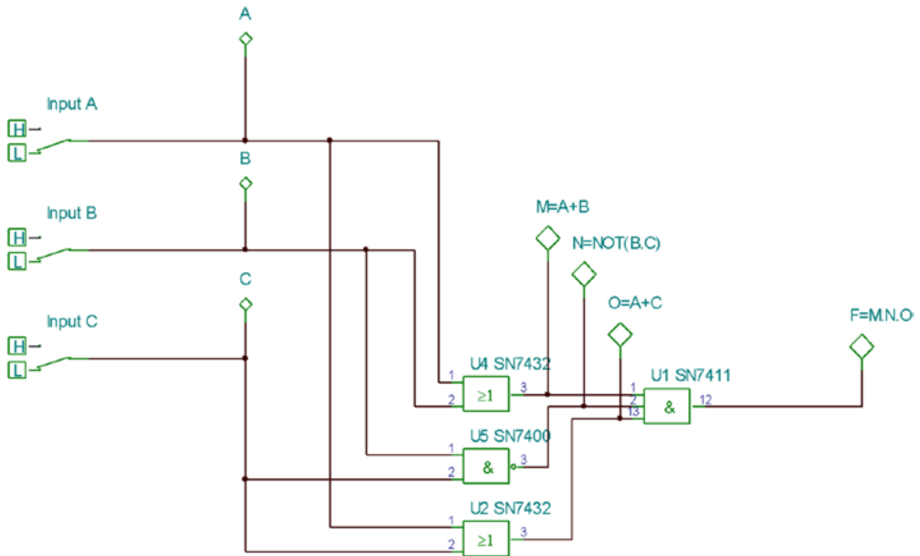


Figure 2-10. *The Circuit for Build Example 2*

Build Logic Circuit Example 3

The Boolean expression for Example 3 is

$$F = \overline{(A+B).C} + D$$

There are two terms that are ORed together at the output "F." You might see that there are two terms that are ANDed, but the bar above the $\overline{(A+B).C}$ groups this together as one term, which is ORed with "D."

Therefore, the expression for "F" is

$$F = M + D$$

One input comes directly from the switch for input D. The other input, "M," is the output of a two-input NAND gate. One of the inputs to that NAND gate comes directly from the switch for input C, whereas the other input, "N," is the output of a two-input OR gate.

Therefore, we can say

$$M = \overline{N.C}$$

The two inputs to the OR gate for "N" come directly from the switches for inputs A and B.

Therefore, we can say

$$N = A + B$$

The circuit will use three gates, which are

Two two-input OR gates

One two-input NAND gate

The circuit that complies with the Boolean expression is shown in Figure 2-11.

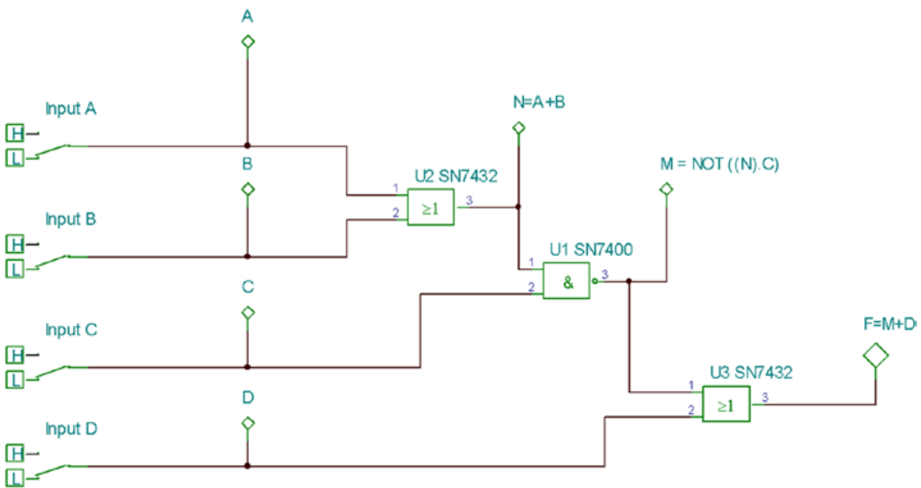


***Figure 2-11.** The Circuit for Build Example 3*

Now it is time for you to try some exercises yourselves.

Exercise 1

Derive the Boolean expression from the following circuits shown in Figures 2-12 and 2-13.

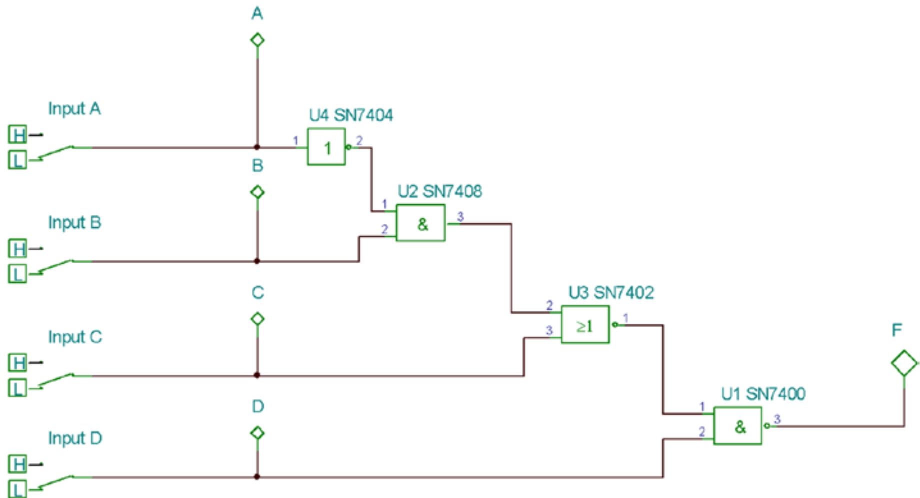


Figure 2-12. *Circuit for Derivation Exercise 1*

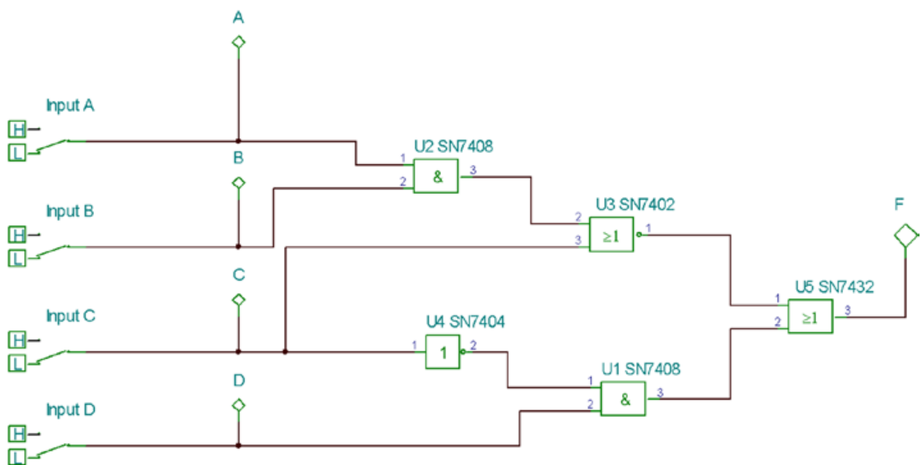


Figure 2-13. *Circuit for Derivation Exercise 2*

Exercise 2

Build the logic circuits that comply with the following Boolean expressions and simulate them using TINA:

1. $F = (A + C).\,B$
2. $F = (A.\,B) + (C.\,D)$
3. $F = \left(\overline{A} + B\right).C$
4. $F = \overline{\left(A + B\right).C} + D$

The solutions are in the appendix.

The Laws of Boolean Algebra

Like all areas of mathematics, Boolean algebra adheres to a set of rules or laws. We need to understand these laws so that we can use Boolean algebra to simplify the expressions and so the logic circuits we design. The following is an attempt to explain these laws. I say attempt as you will have to practice using the laws before you will fully understand them.

Commutative Law

This states that when using the AND and OR operators, it does not matter which way around we place the variables. It is similar to when we multiply or add two numbers, that is:

$$2 \times 3 = 6$$

$$3 \times 2 = 6$$

or

$$3 + 5 = 8$$

$$5 + 3 = 8$$

This can be applied to Boolean algebra.

Commutative Example 1

$F = A.\ B$ is the same as $F = B.\ A$.

Commutative Example 2

$F = A + B$ is the same as $F = B + A$.

Associative Law

This is similar to the commutative law, but it extends to groups of terms within an expression. If we look at the following mathematical operation, it might help:

$$(2 \times 3) \times 3 = 18$$

This can be rewritten as

$$(3 \times 3) \times 2 = 18$$

or written as

$$(3 \times 2) \times 3 = 18$$

As long as we use the same variables, we will get the same result.

Let's apply this to Boolean algebra.

Associative Law Example 1

Simulating the circuit shown in Figure 2-14 can be used to show that

$$F1 = (A.B).C$$

is the same as

$$F2 = (A.C).B$$

or the same as

$$F3 = (B.A).C$$

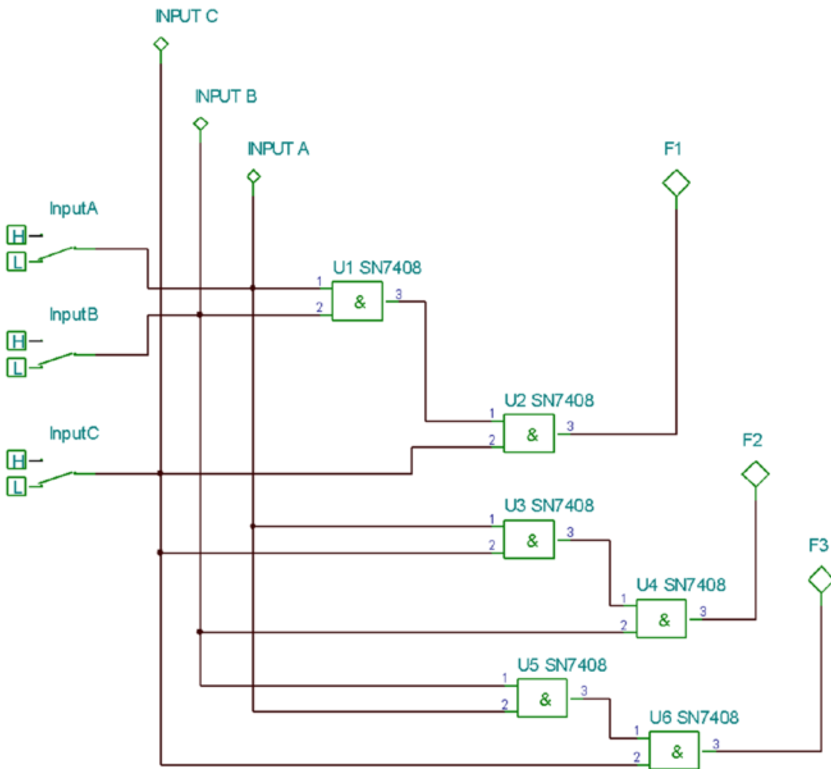


Figure 2-14. *The Simulation Circuit to Test Associative Example 1*

If you simulate the circuit, you will see that all three outputs work in the same way.

Associative Law Example 2

Simulating the circuit shown in Figure 2-15 can be used to show that

$$F1 = (A + B) + C$$

is the same as

$$F2 = (C + B) + A$$

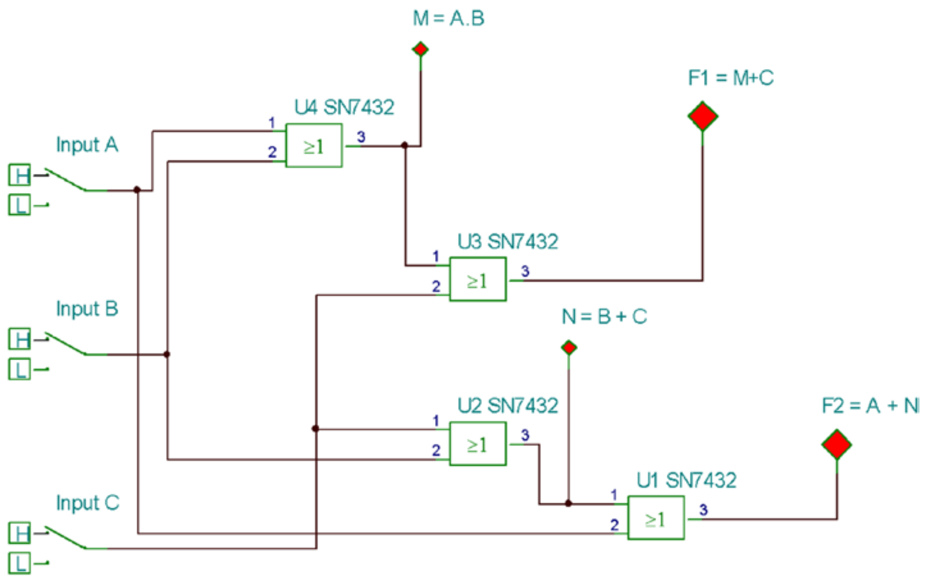


Figure 2-15. *The Simulation Circuit to Test Associative Example 2*

Again, if we simulate the circuit, we see that the two outputs work in the same way.

Distributive Law

As we use Boolean expressions, we come across situations that copy the mathematical rules. This is where, in mathematics, you can expand brackets to expand the expression. The distributive law works in a similar way in that terms can be distributed among other terms inside brackets.

Distributive Law Example 1

Consider the expression:

$$F1 = A \cdot (B+C)$$

If we treat the brackets as a mathematical function, we could expand the brackets by simply multiplying everything inside the brackets by what is outside the brackets. This would produce the following:

$$F2 = A \cdot B + A \cdot C$$

This is an example of the distributive law as the term “A” can be distributed to all the terms inside the brackets.

If you simulate the test circuit shown in Figure 2-16, you should be able to confirm the distributive law works with this example.

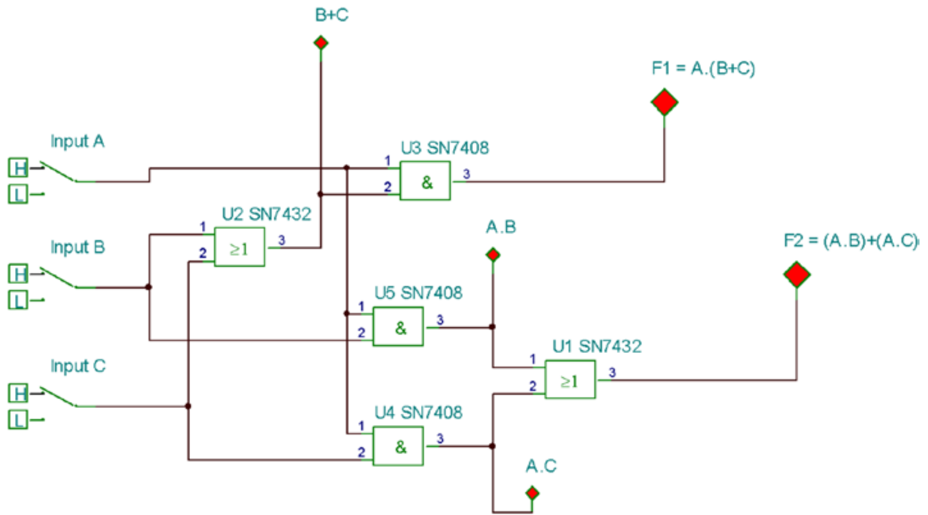


Figure 2-16. *Test Circuit for Distributive Law Example 1*

Distributive Law Example 2

Consider the following Boolean expressions:

$$F1 = A + (B.C)$$

$$F2 = (A + B).(A + C)$$

If you simulate the test circuit shown in Figure 2-17, you should be able to confirm the distributive law works with these examples.

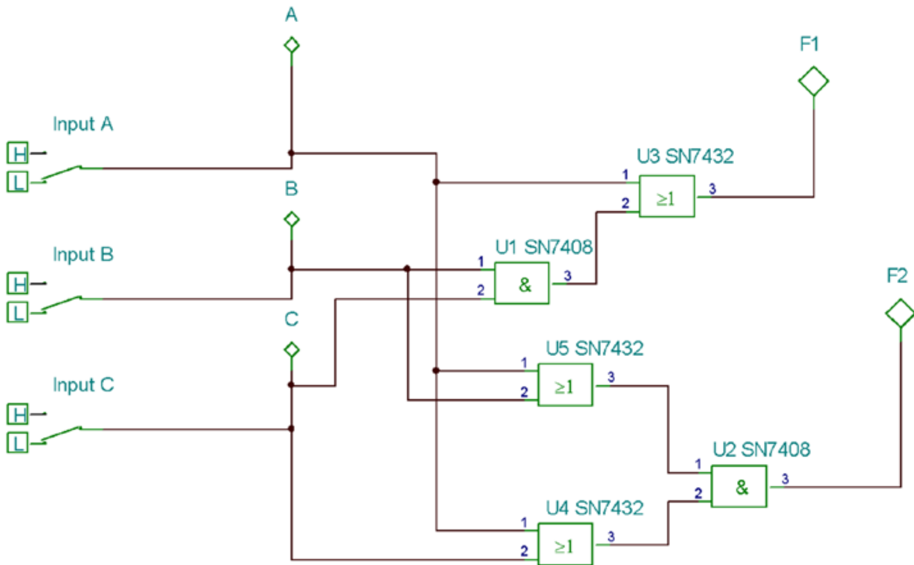


Figure 2-17. *Test Circuit for Distributive Law Example 2*

Distribution Law Example 3

Consider the following Boolean expressions:

$$F1 = (A + B).(C + D)$$

$$F2 = (A.C + A.D) + (B.C + B.D)$$

If you simulate the test circuit shown in Figure 2-18, you should be able to confirm the distributive law works with these examples.

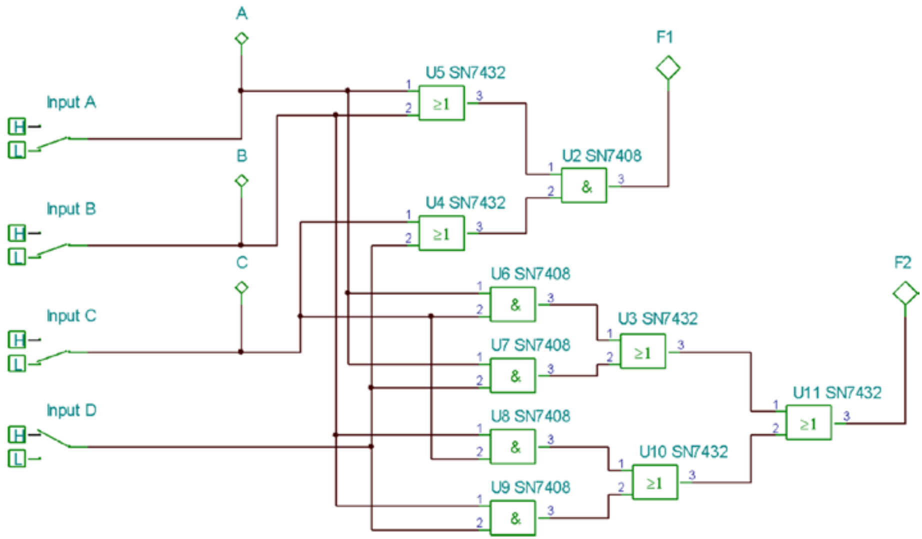


Figure 2-18. *Test Circuit for Distributive Law Example 3*

Absorption Law

This allows us to absorb variables into different terms within the expression.

Consider the circuit shown in Figure 2-19.

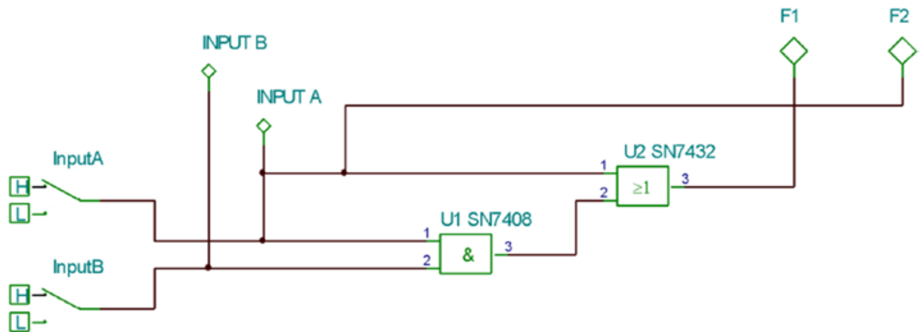


Figure 2-19. *The Simulation Circuit for Absorb 1*

The circuit has two outputs F1 and F2. I hope that by now you will be able to derive the expressions for both outputs. They are as follows:

$$F1 = A + (A.B)$$

If we use the inverse of the distributive law, then we can take the variable "A" out as a common factor. This gives

$$F1 = A.(1 + B)$$

If we use the distributive law, you should see that we get back to the original expression.

In Chapter 3 we will see that

$$1 + B = 1$$

This means we have

$$F = A.1 = A$$

Therefore, the expression for F1 has been absorbed into the expression for F2:

$$F2 = A$$

Note you do not need to put the brackets around the second term, A.B; it is just down to personal preference.

Simulate the preceding circuit so that you can draw up the truth table. From the truth table, determine if B has any effect on the output. Then determine if the circuit performs the Boolean expression

$$F = A$$

The truth table is shown in Table 2-7.

Table 2-7. *Truth Table*

Input A	Input B	Output F1	Output F2
0	0	0	0
0	1	0	0
1	0	1	1
1	1	1	1

From Table 2-7 we can see that both outputs are the same and they work in exactly the same way as the input A.

This is an application of the absorption law in that if F = A+A.B, then F is absorbed by combining the two terms to become just A as F = A.

The only way to get experienced at applying these laws is to use them. There will be many times when you need to minimize the Boolean expressions you come across. It is then that you will be able to use these laws. In Chapter 3 we will go through some examples that show how we can use these laws to minimize Boolean expressions.

De Morgan's Theory

This theory is widely used in transforming logical expressions and circuits so that we might be able to change the circuit from one that has gates we do not have to one that uses gates we do have. This was very useful in the early days of logic designs as the most common logic gate we had at our disposal was the NAND gate. The other gates were still very rare. It requires plenty of practice to become experienced with it, but it can be put to good use.

The theory states

Any Boolean expression can be transformed by carrying out the following steps:

- Change the AND to OR and change the OR to AND.
- Invert each term in the expression.
- Finally, invert the complete expression.

This is best explained with an example.

De Morgan's Example 1

If $F1 = \overline{A} + \overline{B}$, see if we can transform this into $F2 = \overline{A.B}$. This is done as follows:

Firstly, change the "+" to the ".", that is, change the OR to AND.

Therefore, we have

$$F \approx \overline{A}.\overline{B}$$

Secondly, invert each term in the expression:

Therefore, we have

$$F \approx \overline{\overline{A}}.\overline{\overline{B}}$$

We should appreciate the double negatives, that is, the two bars above each term, will simply cancel each other out.

Therefore, we have

$$F \approx A.B$$

Finally, invert the whole expression.

Therefore, we have

$$F2 = \overline{A.B}$$

This would suggest that we can make this transformation. I hope you can see that if it does work, we will have gone from a circuit that used three gates, that is, two NOT gates and one OR gate, to a circuit that uses only one NAND gate. Remember NAND gates were the most common gate.

Now simulate the circuit shown in Figure 2-20 and confirm that the transformation works.

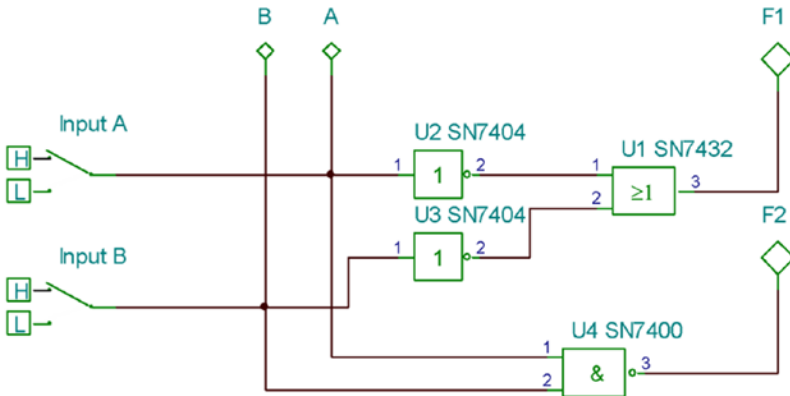


Figure 2-20. *Test Circuit for De Morgan's Example 1*

Simulating this circuit confirms that F1 is the same as F2.

De Morgan's Example 2

Use De Morgan's to transform the expression

$$F1 = A.C + (B + C)$$

Change the "." to the "+" and the "+" to the ".".

Therefore, we have

$$F \approx (A + C).(B.C)$$

Then invert each term in the expression.

Therefore, we have

$$F \approx \left(\overline{A}+\overline{C}\right).\left(\overline{B}.\overline{C}\right)$$

Now invert the whole expression.

Therefore, we have

$$F2 = \overline{\left(\overline{A}+\overline{C}\right).\left(\overline{B}.\overline{C}\right)}$$

This transformation can be tested by simulating the circuit shown in Figure 2-21.

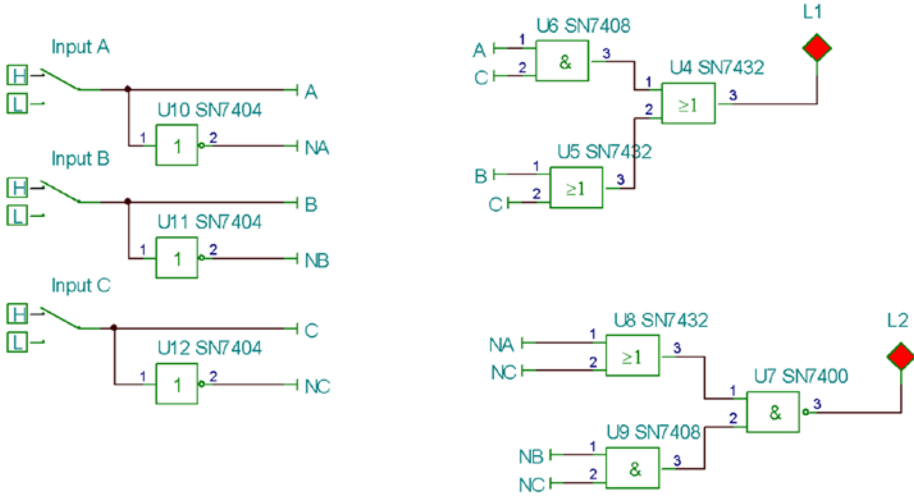


Figure 2-21. *Test Circuit for De Morgan's Example 2*

Simulating this circuit confirms that F1 is the same as F2.

De Morgan's Examples 3

Use De Morgan's to prove that

F = (A+B).(C+D) is the same as

$$F = \overline{\left(\overline{A}.\overline{B}\right) + \left(\overline{C}.\overline{D}\right)}$$

Change the “.” to the “+” and the “+” to the “.”.

Therefore, we have

$$F \approx \left(A.B\right) + \left(C.D\right)$$

Then invert each term in the expression.

Therefore, we have

$$F \approx \left(\overline{A}.\overline{B}\right) + \left(\overline{C}.\overline{D}\right)$$

Now invert the whole expression.

Therefore, we have

$$F = \overline{\left(\overline{A}.\overline{B}\right) + \left(\overline{C}.\overline{D}\right)}$$

The proof of this can be obtained by simulating the circuit shown in Figure 2-22.

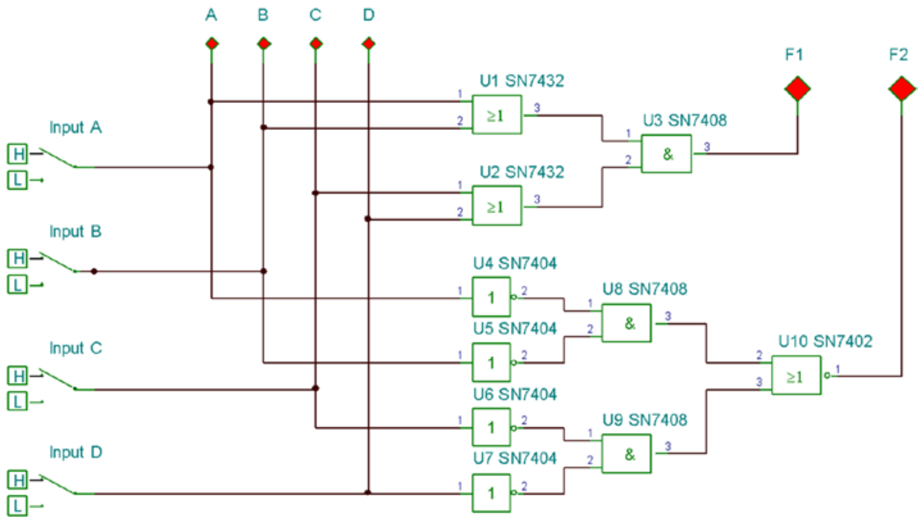


***Figure 2-22.** Test Circuit for De Morgan’s Example 3*

Simulating this circuit confirms that F1 is the same as F2.

De Morgan's theory does not have to be applied to every part of the expression. It can be applied to small parts of the expression. For example, we have shown that $F = (A + B).(C + D)$ can be expressed as $F = M.N$, but we can also say that

$$F = M.(C + D)$$

This means that $M = A + B$ and we can apply De Morgan's to just this part of the expression for F in the usual way.

Firstly, change the + to a **.**.

Therefore, we have

$$M \approx A.B$$

Secondly, invert the terms.

Therefore, we have

$$M \approx \overline{A}.\overline{B}$$

Now invert the complete expression.

Therefore, we have

$$M = \overline{\overline{A}.\overline{B}}$$

Now we can replace this expression for M in the original expression for F.

Therefore, we have

$$F2 = \overline{\overline{A}.\overline{B}}.(C + D)$$

The proof of this can be obtained by simulating the circuit shown in Figure 2-23.

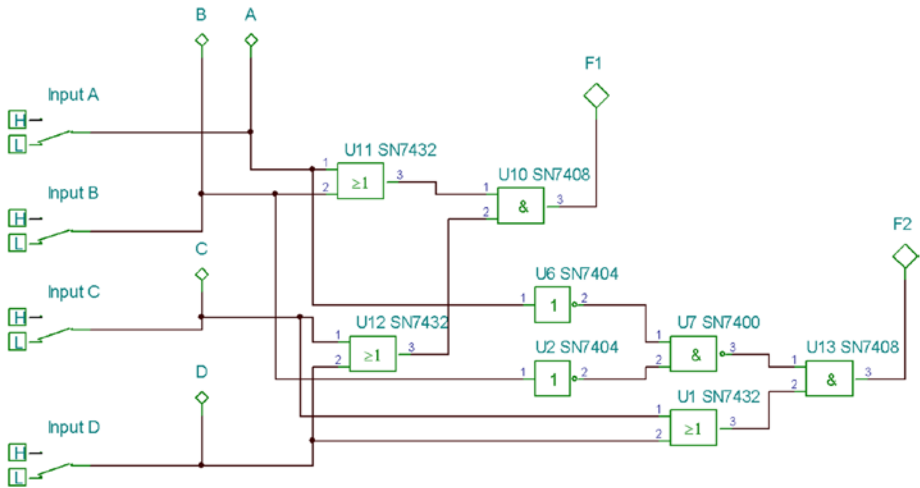


Figure 2-23. *Test Circuit for De Morgan's Example 4*

Simulating this circuit confirms that F1 is the same as F2.

This type of transformation of expressions may seem tedious and unnecessary, but there may be times when with a bit of manipulation, the engineer can use up some spare gates on a chip and so save buying another IC for the circuit.

A more important application of De Morgan's theory is to change the type of gates used in a circuit. Usually this means ensuring that only inverting gates are used in a logic circuit. As an example, we will see if the AND function can be performed using the NAND gate. We know the NAND is the NOT of the AND, so it might follow that the AND is the NOT of the NAND. If this idea is to work, we need to see if the NOT gate can be constructed from a two-input NAND gate. The test circuit is shown in Figure 2-24.

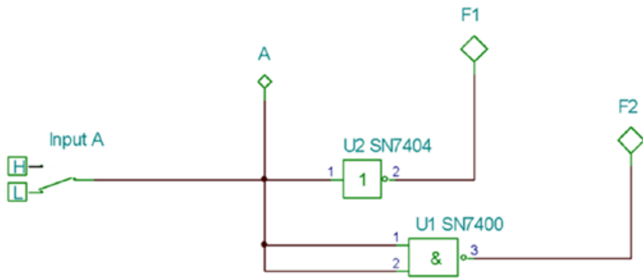


***Figure 2-24.** Test Circuit to Show a NOT Gate Made from a NAND Gate*

Simulating this circuit confirms that F1 is the same as F2. This means the NOT can be made with a two-input NAND gate. We can now proceed to see if the AND function can be constructed from two NAND gates. The test circuit is shown in Figure 2-25.

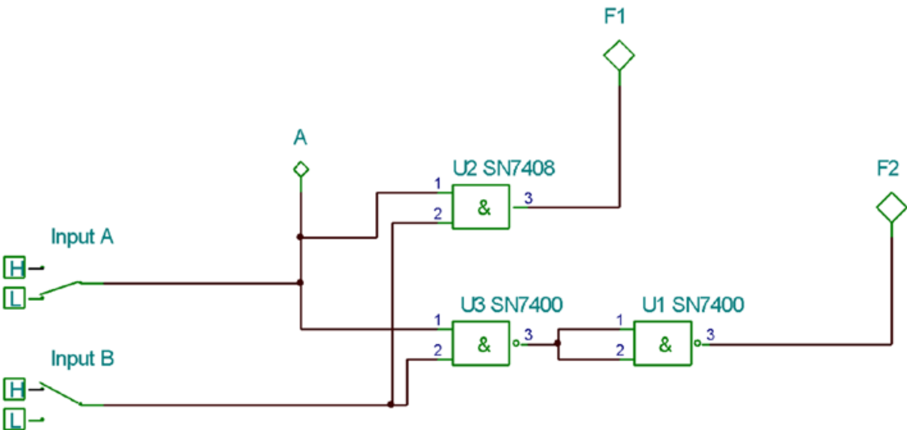


***Figure 2-25.** Creating an AND Gate from NAND Gates*

Simulating this circuit confirms that F1 is the same as F2, which means we can construct the AND function using NAND gates.

The OR Function with NAND Gates

We have seen how we can create the NOT and the AND function using NAND gates. We will now look at how we can create the OR function using NAND gates.

We will start by using De Morgan's to transform the expression

$$F = A + B$$

Change the "+" to ".".

Therefore, we have

$$F \approx A.B$$

Now invert the terms.

Therefore, we have

$$F \approx \overline{A}.\overline{B}$$

Finally, invert the whole expression.

This gives

$$F = \overline{\overline{A}.\overline{B}}$$

We have already shown how we can create a NOT gate with a NAND gate. Therefore, the complete circuit for the OR function using NAND gates is as shown in Figure 2-26.

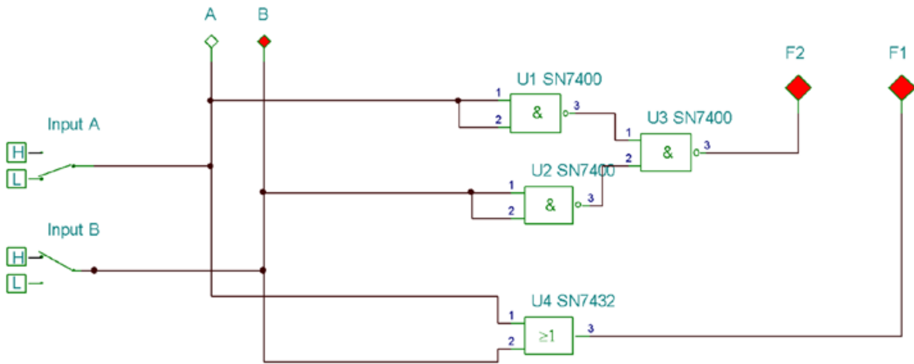


Figure 2-26. *Creating the OR Function from NAND Gates*

If you simulate the circuit and go through the normal sequence of inputs, you should be able to complete the truth table as shown in Table 2-8.

Table 2-8. *The Truth Table for the Circuit Shown in Figure 2-26*

Input A	Input B	Output F1	Output F2
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

From the table we can see that the circuit operates as expected and creates the OR function.

The logic circuit shown in Figure 2-27 shows the Boolean expression

$$F = (A + B).(C + D)$$

implemented with only NAND gates.

If we consider the A+B, we have shown that the OR function can be implemented using three NAND gates in the circuit shown in Figure 2-26. We have also shown that the AND function can be implemented with two NAND gates (see Figure 2-25). Therefore, we should be able to implement the expression as expected using only NAND gates. This is shown in Figure 2-27.

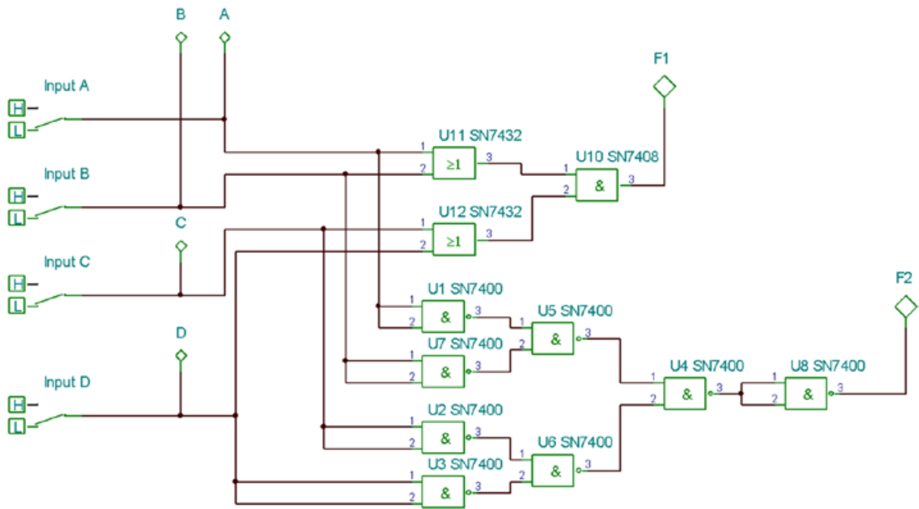


Figure 2-27. *Test Circuit Using Just NAND Gates*

Simulating this circuit confirms that F1 is the same as F2.

It might, at first, seem a waste as the original circuit uses only three gates, whereas the circuit using just NAND gates uses eight gates. However, as these logic gates normally come within an IC that includes four gates, then both circuits use two ICs, which will take up the same room on a PCB and most likely cost the same. This is something you may need to take account of when designing logic circuits. Also, if your company has a surplus of NAND gates but a shortage of NOR gates, then you might be better using the NAND gate circuit.

Summary

In this chapter we have studied some of the fundamental aspects of Boolean algebra. We have created the basic Boolean expressions for the main five logic gates. We have studied how to create truth tables using a recognized convention. We have then studied the main laws of Boolean algebra and also De Morgan's theory. We have learned how we can use the ECAD software TINA to simulate the logic circuits and confirm our work as well as create the truth tables of the circuits we have studied.

In the next chapter, we will look at the two main methods of minimizing Boolean expressions and hence logic circuits.

CHAPTER 3

Simplifying Boolean Expressions

In this chapter we will study how to use Boolean algebra to minimize the logic circuits we design. We will also look at an alternative approach to minimizing logic circuits and Boolean expressions, that is, using Karnaugh maps.

Some Fundamental Identities

Before we look at using Boolean algebra to simplify expressions, we should look at some other aspects of Boolean algebra that are useful to know.

The Inverse Law

$F = A.\overline{A}$ will always equal to 0.

This is the AND version of the "Inverse Law."

To test this concept, we will simulate the circuit shown in Figure 3-1.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_3

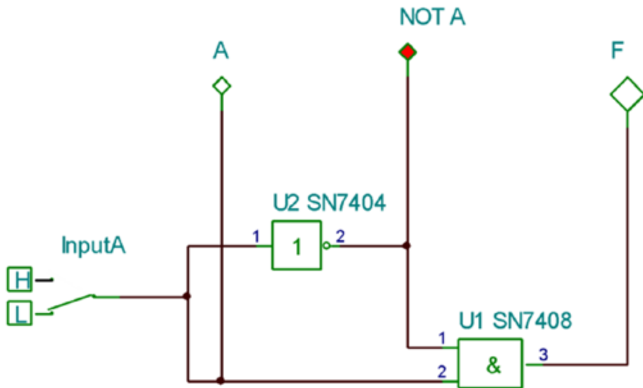


Figure 3-1. *The Simulation of the Logic AND of the Inverse Law*

Operating the switch as normal, you should be able to create the truth table as shown in Table 3-1.

Table 3-1. *The Truth Table of the Simulation of the AND of the Inverse Law*

Input A	Output F
0	0
1	0

There is only one input, which produces only two possible input states. We can see that the output F is always at a logic "0."

This concept can also be confirmed using the truth table shown in Table 3-2.

Table 3-2. *Confirmation of* $A.\overline{A} = 0$

A	$\overline{A}$	$A.\overline{A}$
0	1	0
1	0	0

This result is what should happen, as with the logical AND function, the output "F" will only be a "1" when all inputs are at a logic "1."

The Identity Law

The next expression we will look at is

$$F = A.1 = A.$$

This is the AND version of the "Identity Law."

This means that the value at "F" depends upon the value of "A." To confirm this, we will simulate the circuit shown in Figure 3-2.

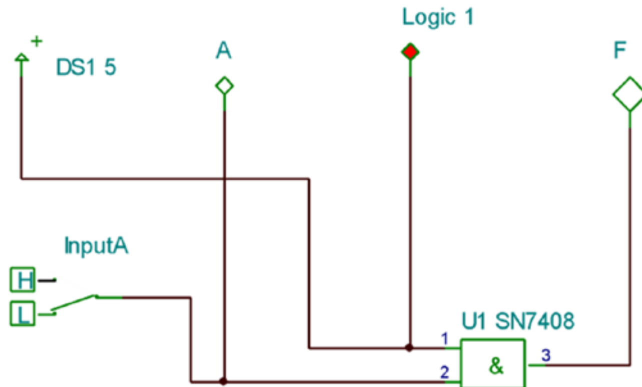


Figure 3-2. *The Simulation of the AND Version of the Identity Law*

Operating the input A as normal produces the following truth table as shown in Table 3-3.

Table 3-3. *The Truth Table of the Simulation of A AND 1*

Logic 1	Input A	Output F
1	0	0
1	1	1

The truth table does confirm the statement F = A.1 always responds with the logic of the input A.

The Null Law

The next statement we will look at is

$$F = A.0 = 0.$$

This is the AND version of the "Null Law."

The circuit to test this is shown in Figure 3-3.

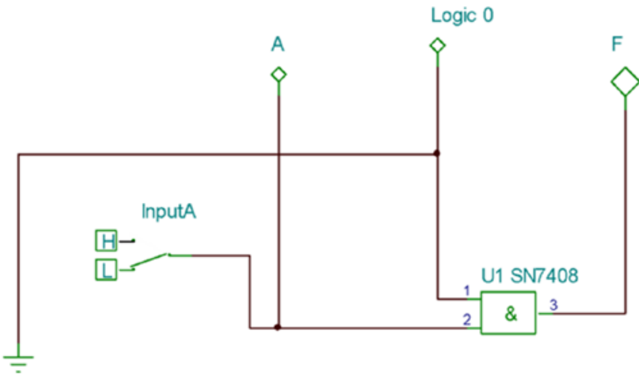


Figure 3-3. *The Simulation of the AND Version of the Null Law*

Operating the input A as normal produces the following truth table as shown in Table 3-4.

Table 3-4. *The Truth Table of the Simulation of the AND Version of the Null Law*

Logic 0	Input A	Output F
0	0	0
0	1	0

The truth table does confirm the statement F = A.0 always responds with the logic "0."

The Idempotent Law

It might be useful to try and explain what this strange word means. An idempotent variable is a variable where if you multiply it by itself or perform any mathematical operation on it with itself, the result will always be the same as the variable before it was operated on. This is true of the binary variables "1" and "0," that is, $1 \times 1 = 1$ and $0 \times 0 = 0$.

The relationship for the AND function is

$$F = A.A = A$$

This is the AND version of the "Idempotent Law."

It can easily be confirmed with the simulation of the circuit as shown in Figure 3-4.

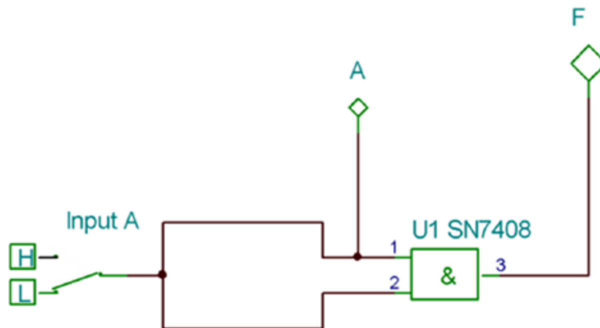


Figure 3-4. *The Simulation of A AND A*

The OR Version of the Idempotent Law

Now consider the expression

$$F = A + A = A.$$

This is the OR version of the "Idempotent Law."

This can easily be confirmed by simulation of the circuit as shown in Figure 3-5.

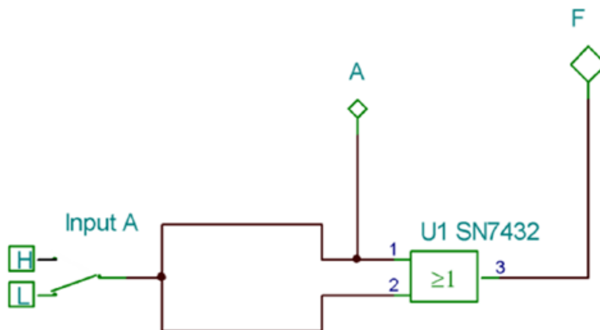


Figure 3-5. *The Simulation of the OR Version of the "Idempotent Law"*

The OR Version of the Identity Law

We will now look at the OR version of the "Identity Law" using the expression $F = A + 0 = A$.

This can be tested by simulating the circuit as shown in Figure 3-6.

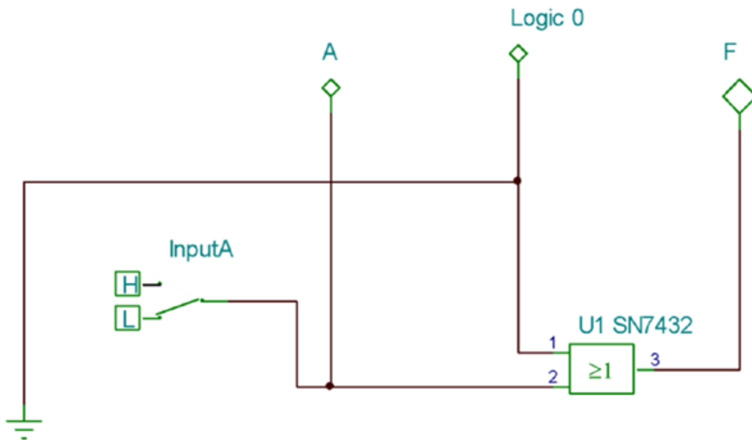


Figure 3-6. *The Simulation of the OR version of the "Identity Law"*

Operating the input A as normal produces the following truth table as shown in Table 3-5.

Table 3-5. *The Truth Table of the Simulation of the OR Version of the "Identity Law"*

Logic 0	Input A	Output F
0	0	0
0	1	1

This confirms the expression works as we expected as the output simply reflects the state of the input A.

The OR Version of the Null Law

Now consider the expression

$$F = A + 1 = 1$$

This is the OR version of the "Null Law."

The simulation circuit to test this law is shown in Figure 3-7.

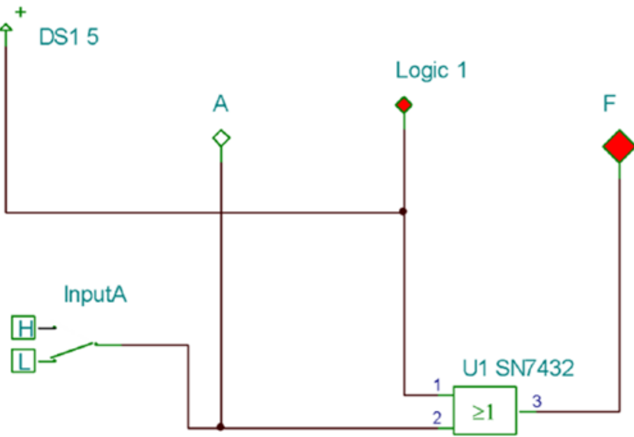


Figure 3-7. *The Simulation of the OR Version of the "Null Law"*

Operating the input A as normal produces the following truth table as shown in Table 3-6.

Table 3-6. *The Truth Table of the Simulation of the OR Version of the "Null Law"*

Logic 1	Input A	Output F
1	0	1
1	1	1

This confirms the expression works as we expected as the output is always a logic "1."

The OR Version of the Inverse Law

We will now consider the following expression:

$$F = A + \overline{A} = 1$$

This is the OR version of the "Inverse Law."

The simulation circuit to test this law is shown in Figure 3-8.

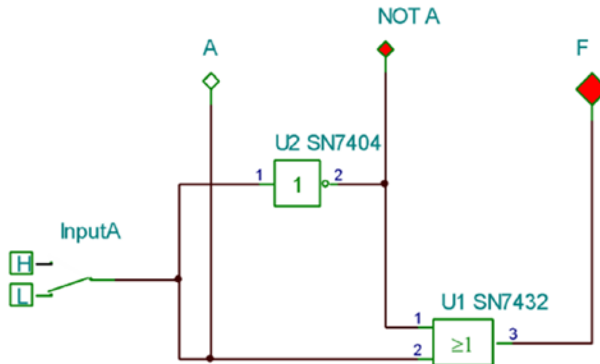


Figure 3-8. *The Simulation of the OR Version of the "Inverse Law"*

Operating the input A as normal produces the following truth table as shown in Table 3-7.

Table 3-7. *The Truth Table of the Simulation of the OR Version of the "Inverse Law"*

Input NOTA	Input A	Output F
1	0	1
0	1	1

This confirms the expression works as we expected as the output "F" is always a logic "1." We will use the preceding laws in the next section where we use Boolean algebra to minimize some logic circuits.

Using Boolean Algebra to Minimize Expressions

It takes a very experienced engineer to design the most efficient circuit for a given specification at the very first attempt. Usually, the engineer will design a circuit that will perform the required function, but then, after careful examination, they will be able to redesign it with a circuit that is more efficient in that it uses fewer components than the first one. This usually means simplifying the circuit. In logical circuit design, there are a number of ways of simplifying the circuits. One method involves writing down the expression for the circuit and then using the related laws for Boolean algebra to simplify the expression to its lowest form. Hopefully, the following sections in this chapter will give you some insight into this important aspect of logic circuit design.

Simplification Example 1

We will start off by considering the expression

$$F = A + A.B$$

We could use the inverse of the distributive law, covered in Chapter 2, by taking the "A" term out of the brackets. This would give

$$F = A.(1 + B)$$

If we use the distributive law to expand the brackets, we would come back to

$$F = A + AB$$

Now using the Null Law on the term in the brackets, we have

$$(1 + B) = 1$$

Therefore, the expression now becomes

$$F = A.(1)$$

which by distribution or the Identity Law becomes

$$F = A$$

Therefore, we have shown by using Boolean algebra that

$$F = A + AB = A$$

We can confirm this by simulating the circuit shown in Figure 3-9.

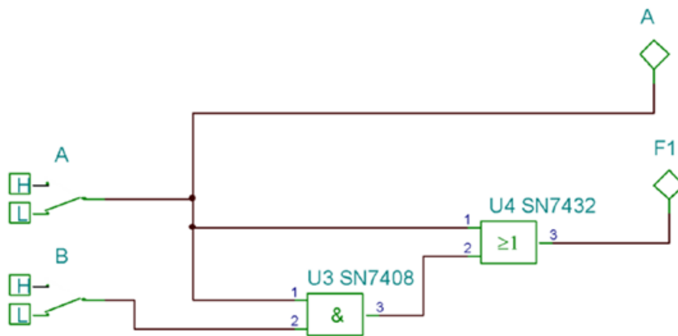


Figure 3-9. *The Simulation of F = A + A. B is the same as F = A*

This confirms the simplification, as both logic probes perform in exactly the same way. Also, we can see that operating the input switch "B" has no effect on the output "F."

Simplification Example 2

Consider the following expression:

$$F = (A + B).(A.B + A.C)$$

If we expand the brackets to comply with the distributive rule, we have

$$F = A.A.B + A.A.C + B.A.B + B.A.C$$

This expansion is carried out by taking each term in the first pair of brackets, one at a time, and ANDing it into each term in the second pair of brackets. We are ANDing them because there is an AND, ".", between the two pairs of brackets.

Using the Idempotent Law, we have shown that A.A = A; therefore, the expression becomes

$$F = A.B + A.C + B.A.B + B.A.C$$

However, using the commutative law, we can rewrite as

$$F = A.B + A.C + A.B.B + A.B.C$$

Knowing B.B. = B we have

$$F = A.B + A.C + A.B + A.B.C$$

We can simplify parts of the expression on their own, and so use the inverse of the distributive law as follows:

$$F = A.B\left(1+1+C\right)+A.C$$

Using the Null Law, we have

$$1+C=1.Therefore, A.B\left(1+1+C\right)=A.B\left(1\right)=A.B$$

Therefore we have

$$F = A.B + A.C$$

We can now take the variable "A" out as a common factor, which gives

$$F = A.\left(B+C\right)$$

Therefore, the expression now becomes

$$F = A.(B + C)$$

We can check this simplification by simulating the following two circuits as shown in Figure 3-10, L1 is the original expression, L2 is the simplified expression.

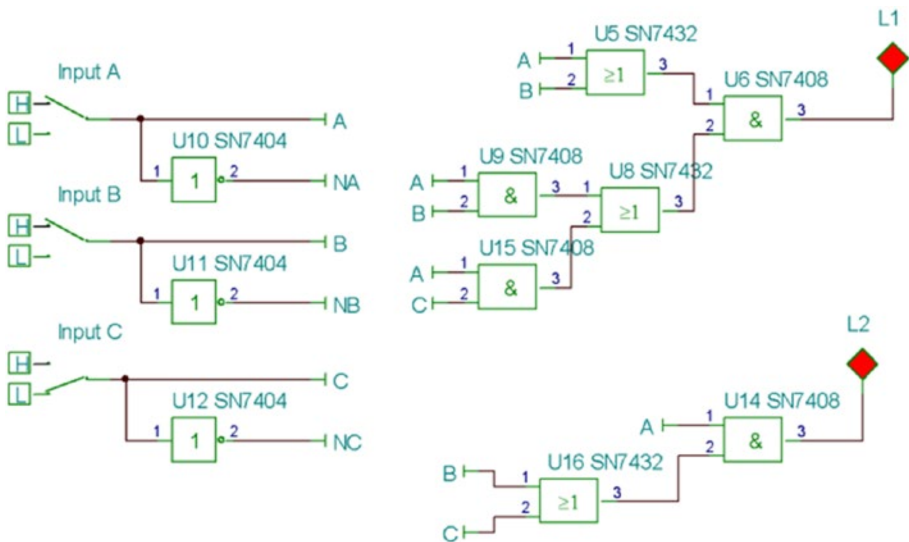


***Figure 3-10.** The Test Circuits for Simplification Example 2*

The two circuits work in exactly the same way, which confirms that the simplified expression is correct.

Simplification Example 3

Consider the expression

$$F1 = \overline{A}.B + \overline{A}.\overline{B}$$

Using the inverse of the distributive law, we can take the $\overline{A}$ out as a common factor. Therefore, we have

$$F = \overline{A}.\left(B + \overline{B}\right)$$

Using the Inverse Law, we can show that

$$B + \overline{B} = 1$$

Therefore, we have

$$F2 = \overline{A}.(1) = \overline{A}$$

This can be tested using the circuit shown in Figure 3-11.

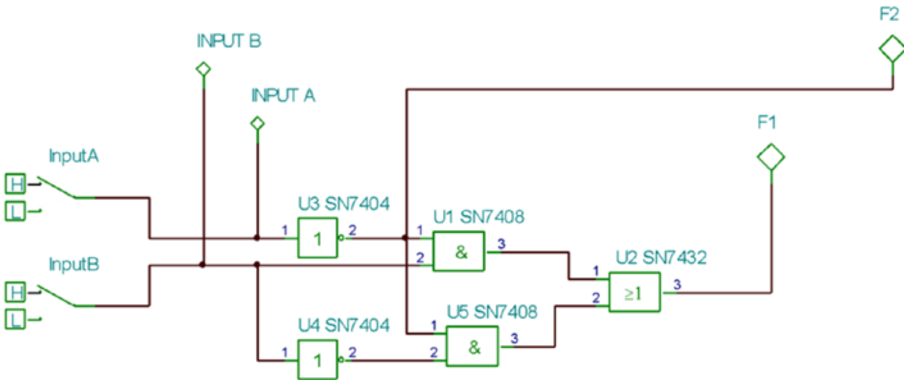


Figure 3-11. *The Test Circuit for Simplification Example 3*

Simulating this circuit shows that the two expressions, F1 and F2, and their outputs work in exactly the same way and that input B has no effect on the output.

Simplification Example 4

Simplify the following expression:

$$F1 = A.B + \overline{A}.B + \overline{A}.\overline{B} + A.C$$

Using the inverse of the distributive law, we can take a $\overline{A}$ from two terms.

Therefore, we get

$$F = A.B + \overline{A}.\left(B + \overline{B}\right) + A.C$$

Using the Inverse Law, we know $B + \overline{B} = 1$.

Therefore, we get

$$F = A.B + \overline{A}.1 + A.C = A.B + \overline{A} + A.C$$

Therefore, we have

$$F = A.B + A.C + \overline{A}$$

We can take the “A out as a common factor, which gives

$$F = A.\left(B + C\right) + \overline{A}$$

Therefore

$$F2 = A.\left(B + C\right) + \overline{A}$$

If we simulate the circuit shown in Figure 3-12, we will see that both expressions for L1 and L2 work in exactly the same way.

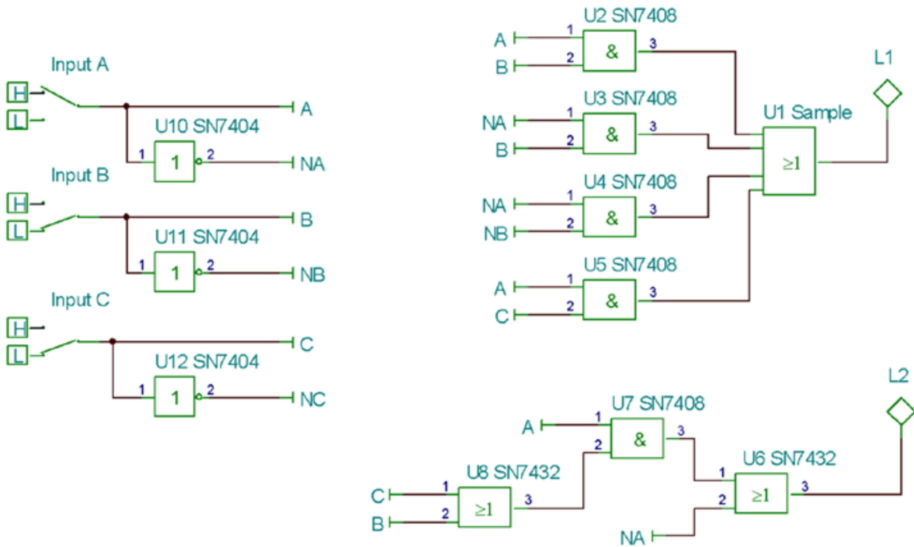


Figure 3-12. *The Test Circuit for Simplification Example 4*

Simplification Example 5

Consider the expression

$$F1 = (A + C).(\overline{A} + B).(B.\overline{C} + BC) = L1$$

Using the distributive law, we have

$$F = (A + C).(\overline{A} + B).B(\overline{C} + C)$$

Knowing $\overline{C} + C = 1$, we have

$$F = (A + C).(\overline{A} + B).B$$

Expanding the last pair of brackets gives

$$F = (A + C).(\overline{A}.B + B.B)$$

Now expanding both pairs of brackets gives

$$F = A.\overline{A}.B + A.B.B + C.\overline{A}.B + C.B.B$$

The Inverse Law shows that

$$A.\overline{A} = 0$$

Therefore, the $A.\overline{A}.B = 0$
This gives:

$$F = A.B.B + C.\overline{A}.B + C.B.B$$

Knowing $B.\ B = B$ gives

$$F = A.B + C.\overline{A}.B + C.B = L2$$

Taking B out as a common factor gives

$$F = B.\left(A + C.\overline{A} + C\right)$$

Using the commutative Law

$$F = B.\ (A + C + \overline{A}.C)$$

$$F3 = B.(A + C + \overline{A}.C) = L3$$

We can confirm this simplification by simulating the circuit shown in Figure 3-13 L1 is for expression F1, L2 is for F2 and L3 is for F3.

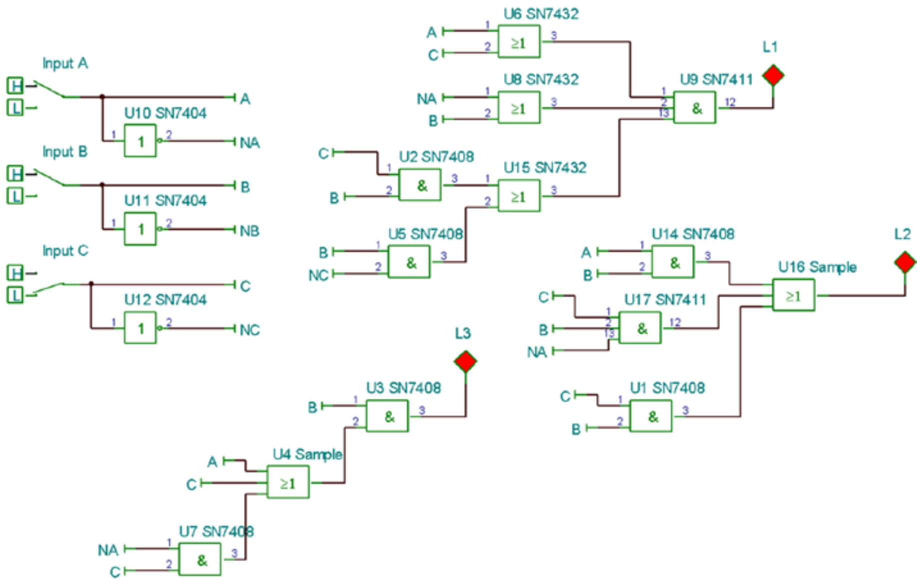


Figure 3-13. *The Test Circuit for Simplification Example 5*

The simulation confirms that the simplification works.

Simplification Example 6

Sometimes we may need to use truth tables to simplify the Boolean expression. We could try looking at truth tables to simplify the following expression further:

$$F3 = B + \overline{A}.\overline{B}$$

The process of using truth tables is shown in Table 3-8.

***Table 3-8.** Truth Tables to Aid Simplification*

1	2	3	4	5	6	7
B	A	$\overline{B}$	$\overline{A}$	$\overline{A}.\overline{B}$	$B + \overline{A}.\overline{B}$	$B + \overline{A}$
0	0	1	1	1	**1**	1
0	1	1	0	0	**0**	0
1	0	0	1	0	**1**	1
1	1	0	0	0	**1**	1

Column 6 is the expression we are trying to minimize. If we look at the first column, the response of the logic "B," we should be able to see that the last two logic values, that is, the two logic "1"s, are repeated in the last two values in column 6.

If we now look at column 4, we can see that the first two values, that is, logic "1" and then logic "0," are repeated in the first two values in column 6. This means we can get the same logic values as stated in column 6 by taking the values of column 1 and ORing them with the values in column 4. This logical OR operation is shown in column 7. We can confirm that this is true with the test circuit shown in Figure 3-14; there L1 is the expression for column 6 and L2 is the expression for column 7.

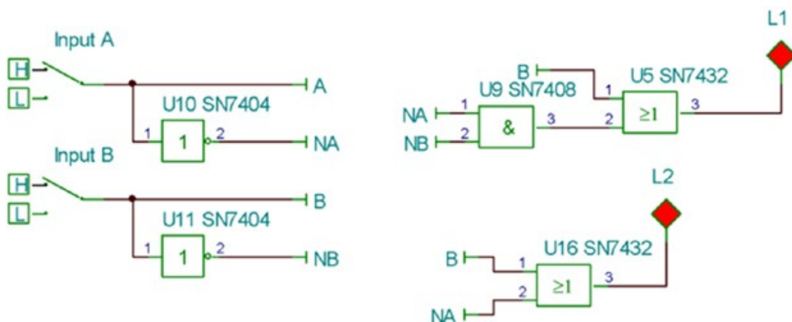


***Figure 3-14.** The Test Circuit for Example 6*

Examples 4, 5, and 6 shows us that we will have to think out of the box to simplify the Boolean expressions and make use of truth tables. Only with practice will we become good at applying Boolean algebra. Perhaps the next method may be easier.

Karnaugh Maps

Karnaugh maps are used by engineers as an aid or alternative to minimizing the number of gates used in a logic circuit. They are mainly used with circuits that have two to six inputs; any more than that and the maps become very cumbersome, and therefore Boolean algebra may be a better approach.

However, before we can use Karnaugh maps, we need to learn the rules for using them. The best way to determine how to use Karnaugh maps is probably to go through an example of using them.

Karnaugh Map Example 1

Consider the Boolean expression

$$F = \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C$$

Hopefully, this expression can be minimized. The procedure, using Karnaugh maps, is to draw the map and then indicate the cells that are logic "1"s with respect to the expression. The size of the map depends upon the number of variables, or inputs, in the expression or circuit.

A one-variable expression will need a Karnaugh map with just two cells as shown in Figure 3-15a. A two-variable expression needs a map with four cells as shown in Figure 3-15b.

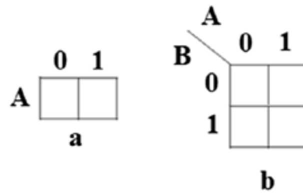


***Figure 3-15.** The Two- and Four-Cell Karnaugh Maps*

The expression in Example 1 has three variables, and the empty map is shown in Figure 3-16.

A B	0 0	0 1	1 1	1 0
C				
0				
1				

***Figure 3-16.** The Karnaugh Map for a Three-Input Expression*

The first two inputs or variables “A and B” are shown as being horizontally placed above the map. These define how many columns there are in the map. As there are two variables, then there will be four combinations of their values. Remember the number of combinations can be calculated using 2^n, where “*n*” is the number of variables, and so there will be four columns. It is also important to realize that as we move along the horizontal, the value of only one variable can be changed at a time. That is why the next column after 0,1 is 1,1, as going to 1,0 would mean two variables would have to change at once. This process of only changing one variable at a time is termed using the “Grey scale.”

The third variable or input, in this case “C,” is on the vertical and so defines how many rows there are in the map. There are only two values that the variable “C” can take up, so there will be only two rows.

Using the Karnaugh Map

To put the logic "1"s in their place in the map, we examine the expression and decide when F will equal a logic "1." In this case we see that as

$$F = \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C$$

The output will be a logic "1" with any one the following three terms:

- $\overline{A}.\overline{B}.C$
- $\overline{A}.B.C$
- $A.\overline{B}.C$

Putting the binary logic into the terms means that the output will be a logic "1" when

1. A = 0 & B = 0 & C = 1
2. A = 0 & B = 1 & C = 1
3. A = 1 & B = 0 & C = 1

We need to remember with respect to an input variable, for example, input "A," when the input is a logic "1" we use "A" and when it is a logic "0" we use "A ."

We have the first term $\overline{A}.\overline{B}.C$, which means that both A and B inputs would be a logic "0." This puts us in the first column of the Karnaugh map. Now we can see that the input "C" would be a logic "1." Therefore, this would put us in the second row of the Karnaugh map. This means that we must put a "1" in the first column but in the second row as shown in Figure 3-17.

C \ AB	00	01	11	10
0				
1	**1**			

Figure 3-17. *Inserting the First Term in the Map*

Using the same logic, the second term $\overline{A}.B.C$ would go into the Karnaugh map as shown in Figure 3-18.

C \ AB	00	01	11	10
0				
1		**1**		

***Figure 3-18.** Inserting the Second Term in the Map*

Using the same logic, the third term $A.\overline{B}.C$ would go into the Karnaugh map as shown in Figure 3-19.

C \ AB	00	01	11	10
0				
1				**1**

***Figure 3-19.** Inserting the Third Term in the Map*

Combining all the terms together and putting the logic "0"s, we have the complete map as shown in Figure 3-20.

C \ AB	00	01	11	10
0	**0**	**0**	**0**	**0**
1	**1**	**1**	**0**	**1**

***Figure 3-20.** Showing All Three Terms in the Map*

There is no real need to include the "0"s in the map as shown; it is only to show an alternative logic we can use in the map. We will look at this alternative later in the chapter.

We now draw loops around the logic "1"s that are next to each other. A loop can have one, two, four, or eight "1s" in it, and we should make the loops as big as possible. However, we must realize that the map is spherical, like a ball, in that all edges touch each other. This concept is

hard to appreciate, but I hope, as we make the loops in the maps, you will see how to interpret this concept.

This means that the logic "1" at A = 0 and B = 0 and C= 1 is next to the logic "1" at A = 1 and B = 0 and C = 1. Note also we can include the logic "1"s in more than one loop. Also, as you go through the terms in the Boolean expression, you can place more than one "1" in a cell.

The loops for this map are as shown in Figure 3-21.

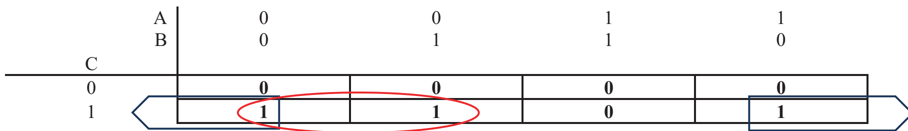


Figure 3-21. *The Completed Map with the Two Loops Added*

There are two loops in the map, and I hope you can see that loop2, in black, comes about because the left-hand edge of the map connects to the right-hand side. In the same way, the top of the map connects to the bottom of the map.

The two loops mean F will be true with loop1 or with loop2:

$$\therefore F = loop1 + loop2$$

The next step is to derive the Boolean expression for each loop as follows.

When two or more logic "1"s are looped together in a loop, the Boolean term is the logics that are common in the loop.

With loop1 A is 0 in both cells and C is 1 in both cells. Therefore, the Boolean expression for that loop is

$$Loop1 = \overline{A}.C$$

An alternative approach is to state what variables DO NOT CHANGE in the loop. You can see that in loop1 A is always a logic "0" and C is always a logic "1" but "B" changes from logic "0" to logic "1". Therefore, the expression includes the A and C but it does not have "B" in it. This really means that it does not care what logic the input "B" takes on.

With loop2 B is 0 in both cells and C is 1 in both cells. Therefore, the Boolean expression for that loop is

$$Loop2 = \overline{B}.C$$

This means that F will be "1" via both possibilities. Therefore, there will be AND and OR functions in the expression. The complete expression is as follows:

$$F = \overline{A}.C + \overline{B}.C$$

This means that

$$F1 = \overline{A}.\overline{B}.C + \overline{A}.B.C + A.\overline{B}.C$$

minimized to

$$F2 = \overline{A}.C + \overline{B}.C$$

using a Karnaugh map.

However, from our work with Boolean algebra, we can see that the input "C" is common to both terms. Therefore, using the inverse of the distributive law, we can see that

$$F3 = \left(\overline{A} + \overline{B}\right).C$$

We can simulate the circuit to confirm that all three expressions produce the same. The simulation circuit is shown in Figure 3-22.

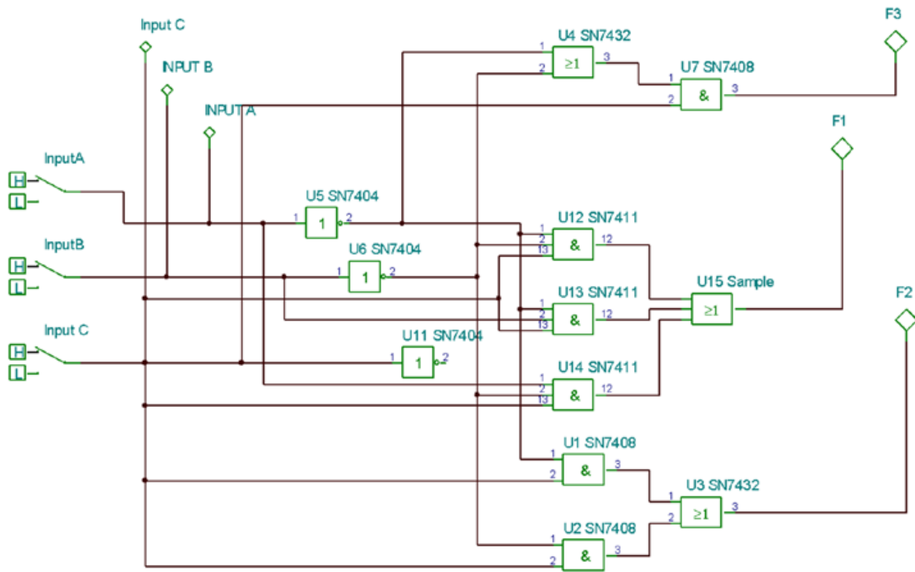


Figure 3-22. *Test Circuit for Karnaugh Map Example 1*

This confirms that the Karnaugh map does work but you may be able to minimize the expression further using Boolean algebra.

Karnaugh Map Example 2

Use a Karnaugh map to simplify the expression

$$F = \left(A.B.\overline{C}.\overline{D}\right) + \left(A.B.C.\overline{D}\right) + \left(A.B.\overline{C}.D\right) + \left(\overline{A}.\overline{B}.C.D\right) + \left(A.\overline{B}.C.D\right) + \left(\overline{A}.B.C.D\right) + \left(A.B.C.D\right)$$

This is a four-variable expression, so the map will have four columns and four rows as shown in Figure 3-23.

A B / DC	0 0	0 1	1 1	1 0
0 0				
0 1				
1 1				
1 0				

Figure 3-23. *The Empty Karnaugh Map for Example 2*

It is normal to identify the individual cells in the map by starting at the top left-hand corner. This cell would be at the first column in the first row, and we could identify it using "C1." This concept is continued throughout the map ending with, in this case, "C16," which is at the bottom right-hand corner of the map.

The first term in the expression means A = 1, B = 1, C = 0, and D = 0.

This puts a "1" in the third column of the first row, , which would be "C3." Note A = 1 and B = 1 in all the cells of the third column.

This is shown in Figure 3-24.

A B / DC	0 0	0 1	1 1	1 0
0 0			**1**	
0 1				
1 1				
1 0				

Figure 3-24. *The First Term of Example 2 Inserted into the Map*

The second term means A = 1, B = 1, C = 1, and D = 0.

This puts a "1" in the third column of the second row, that is, "C7."

This is shown in Figure 3-25.

A B	0 0	0 1	1 1	1 0
DC				
0 0			**1**	
0 1			**1**	
1 1				
1 0				

Figure 3-25. *The Second Term of Example 2 Inserted into the Map*

The third term means A = 1, B = 1, C = 0, and D = 1.

This puts a "1" in the third column of the fourth row, "C15."

This is shown in Figure 3-26.

A B	0 0	0 1	1 1	1 0
DC				
0 0			**1**	
0 1			**1**	
1 1				
1 0			**1**	

Figure 3-26. *The Third Term of Example 2 Inserted into the Map*

The fourth term means A = 0, B = 0, C = 1, and D = 1.

This puts a "1" in the first column of the third row, "C9."

This is shown in Figure 3-27.

A B	0 0	0 1	1 1	1 0
DC				
0 0			**1**	
0 1			**1**	
1 1	**1**			
1 0			**1**	

Figure 3-27. *The Fourth Term of Example 2 Inserted into the Map*

The fifth term means A = 1, B = 0, C = 1, and D = 1.

This puts a “1” in the fourth column of the third row, “C12.”

This is shown in Figure 3-28.

	A	0	0	1	1
	B	0	1	1	0
DC					
0 0				**1**	
0 1				**1**	
1 1		**1**			**1**
1 0				**1**	

Figure 3-28. *The Fifth Term of Example 2 Inserted into the Map*

The sixth term is $\left(\overline{A}.B.C.D\right)$, which means A = 0, B = 1, C = 1, and D =1. This puts a “1” in the third row of the second column, that is, “C10,” as shown in Figure 3-29.

	A	0	0	1	1
	B	0	1	1	0
DC					
0 0				**1**	
0 1				**1**	
1 1		**1**	**1**		**1**
1 0				**1**	

Figure 3-29. *The Sixth Term of Example 2 Inserted into the Map*

The seventh term is $(A \bullet B \bullet C \bullet D)$; this means A = 1, B = 1, C = 1, and D = 1.

This puts a “1” in the third column of the third row, “C11.”

This is shown in Figure 3-30.

	A	0	0	1	1
	B	0	1	1	0
DC					
0 0				**1**	
0 1				**1**	
1 1		**1**	**1**	**1**	**1**
1 0				**1**	

Figure 3-30. *The Seventh Term of Example 2 Inserted into the Map*

There are two loops in this Karnaugh map. Loop 1 takes in all the “1s” in the third column, and loop 2 takes in all the “1s” in the third row as shown in Figure 3-31.

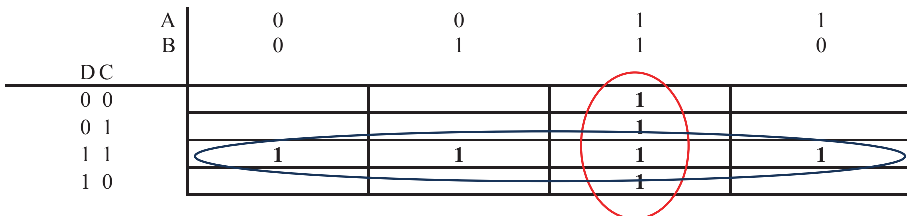


Figure 3-31. *The Two Loops Marked in the Map*

In loop 1, in red, the variables that don’t change are “A and B” as they are both a logic “1” in all four cells.

In loop 2, in black, the variables that don’t change are “C and D” as they are both a logic “1” in all four cells.

This means that

$$F = Loop1\ OR\ Loop2$$

$$F = (A.B) + (C.D)$$

This is a very simplified expression with respect to the original expression. It also shows how to use Karnaugh maps for a four-variable expression.

Simplification Examples

In this section we will go through some more examples that will simplify the Boolean expression derived from a logic circuit, using the various methods we have looked at in this chapter and Chapter 2.

Simplification Example 7

Consider the circuit shown in Figure 3-32.

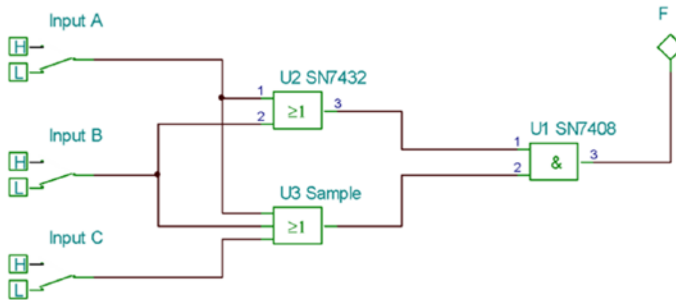


Figure 3-32. *The Circuit for Simplification Example 7*

If we simulate the circuit, we should be able to complete the truth table as shown in Table 3-9.

Table 3-9. *The Truth Table for Example 7*

Input A	Input B	Input C	Output F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

The 1st and 2nd Canonical Formats and the Minterms and Maxterms

I will use this example to explain what is meant by the 1st and 2nd canonical formats and the minterms and maxterms of a truth table.

We can create what is called the 1st canonical format by writing the expression for the terms when the output "F" is a logic "1." The terms that create the logic "1" are called the "minterms." In this example there are six combinations of the inputs that create the minterms, and they are shown in Expression 3-1.

$$F = \overline{A}.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.\overline{C} + A.B.C$$

Expression 3-1 The Minterm Expression for Simplification Example 7

Knowing what logic is indicated by the variables, with and without the bar above them, that is, A is a logic "1" and $\overline{A}$ is a logic "0," we can write a simplified representation of the expression for the minterms using the binary values. Therefore, in binary the expression for the output "F" is

$$F = \sum(010,011,100,101,110,111)$$

The "$\sum$" notation indicates that the terms, that is, the groups of the "1"s and "0"s, are ORed with each other.

Now that we have expressed the expression using binary numbers, we have to be more careful in how we create, and fill, the Karnaugh map. This is because when writing binary numbers, we always place the LSB, least significant bit, at the beginning reading from right to left. This means that the MSB, most significant bit, is the last bit on the left of the binary number. What this now means is that with the variables A, B, and C, we have defined which is the MSB, as being the one at the farthest left, that

is, the variable A in this case. It also means that the variable C is the LSB variable as it is the first reading from right to left.

If we look at the truth table, it should become apparent that we have already assigned the MSB to variable "A" and the LSB to variable "C." Note C simply toggles between 0 and 1 in each row, a clear sign that the variable "C" is the LSB.

As an alternative to using the binary values, we can use the decimal values to write an expression for the output "F." Using the decimal values, the expression for "F" is

$$F = \sum(2,3,4,5,6,7)$$

This last alternative, of writing the decimal numbers, is not as useful as writing the binary numbers. When we come to completing the Karnaugh maps, we will use the binary number format.

The 2nd Canonical Format

We have now looked at the three alternatives for writing down the expression for the 1st canonical format of the Boolean expression. When we looked at the first Karnaugh map, as shown in Figure 3-20, I indicated that we could use logic "0"s as an alternative. This alternative is termed the 2nd canonical format.

We can create the 2nd canonical format for the Karnaugh map by writing the expression for the terms when the output "F" is a logic "0"; these are known as the "maxterms." However, as this is when "F" is a logic "0," then really, we are creating an expression for $\overline{F}$. There are only two combinations in the truth table shown in Table 3-9 when the output is a logic "0." This means that in this case we can say

$$\overline{F} = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C$$

We can use Boolean algebra to create the expression for “F” from this. First, we must invert both sides of the expression, which gives

$$\overline{\overline{F}} = \overline{\overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C}$$

The double bars above the “F” cancel themselves out, and it simply becomes “F.” If we now apply De Morgan’s theory, as described in Chapter 2, we must change all the “.” to “+” and all the “+” to “.”. This would give

$$F = \overline{\left(\overline{A} + \overline{B} + \overline{C}\right).\left(\overline{A} + \overline{B} + C\right)}$$

Now we invert the individual variables, which, because we again have double bars above the variables, except the last “C” variable, gives

$$F = \overline{\left(A + B + C\right).\left(A + B + \overline{C}\right)}$$

Finally, we invert the whole expression, which gives

$$F = \overline{\overline{\left(A + B + C\right).\left(A + B + \overline{C}\right)}}$$

The double bars cancel out, and so the expression for “F” becomes

$$F = \left(A + B + C\right).\left(A + B + \overline{C}\right)$$

We don’t have to go through this complex process every time we use the 2nd canonical format. The shortcut is to invert the variables and change the “.” to “+” and the “+” to “.”. This means the variables in the terms are now ORed instead of ANDed. Also, the terms themselves are ANDed instead of ORed.

Carrying out this procedure on the 2nd canonical expression

$$\overline{F} = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C$$

gives

$$F = (A + B + C).(A + B + \overline{C})$$

Expression 3-2 The Boolean Expression for "F" Using the 2nd Canonical Format

This is what we expect the expression for "F" to be using the 2nd canonical format.

Using the binary values, we get

$$F = \pi[111,110]$$

Using the decimal values, we get

$$F = \pi[7,6]$$

or

$$F = \pi[6,7]$$

Note, with these expressions, we are using the symbol "π," which indicates the expression is from a 2nd canonical format and the variables should be ORed and the terms ANDed.

We can draw the Karnaugh maps for the expressions. As we know there are three variables, so the map would be as shown in Figure 3-33. However, we must remember which variable is the MSB and which is the LSB. In this example we have assigned the MSB and LSB slightly differently from the previous example to try and show you it is not in alphabetical order but in the order of LSB to MSB going from right to left, that is, from C to A in this case. We could have labeled the variables "M," "X," and "Y" if we wanted to.

C B	0 0	0 1	1 1	1 0
A				
0				
1				

Figure 3-33. *The Empty Karnaugh Map for a Three-Variable Expression*

The variables are placed so that they read from right to left starting with "C" as the furthest right, being the LSB, going to "A," which is the MSB. For example, using the minterm expression, Expression 3-1, which we got using the 1st canonical format, the first term is

$$\overline{A}.B.\overline{C}$$

or, using the binary values, 010.

This would put a "1" in cell C2 as A=0, B= 1, and C =0.

The next term is

$\overline{A}.B.C$, or 011

This would put a "1" in cell C3 as A = 0, B = 1, and C =1.

The next term is

$A.\overline{B}.\overline{C}$ or 100

This would put a "1" in cell C5 as A = 1, B = 0, and C =0

The next term is

$A.\overline{B}.C$ or 101

This would put a "1" in cell C8 as A = 1, B = 0, and C = 1.

The next term is

` $A.B.\overline{C}$ or 110

This would put a "1" in cell C6 as A = 1, B = 1, and C = 0

The last term is

$A. B. C$ or 111.

This would put a "1" in cell C7 as A = 1, B = 1, and C = 1.

This means the completed map obtained using the 1st canonical format would be as shown in Figure 3-34.

C B A	0 0	0 1	1 1	1 0
0		**1**	**1**	
1	**1**	**1**	**1**	**1**

Figure 3-34. *The Completed Karnaugh Map for Example 7*

In loop 1, the black loop in Figure 3-35, the only term that does not change is "A." In loop 2, the red loop in Figure 3-35, the only term that does not change is "B." Using the map, we can say that the output "F" can be expressed as

$$F = Loop1 + Loop2$$

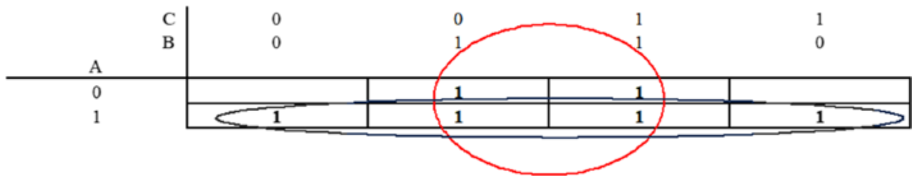


Figure 3-35. *The Karnaugh Map with the Loops Included*

Therefore, we can say

$$F = A + B$$

Now we can create the map from the 2nd canonical format using the maxterm expression. There are only two maxterms as stated in Expression 3-2. Using them we can construct the maxterm map by placing a "0" in the appropriate cells.

The binary format for the 2nd canonical format is

$$F = \pi[111,110]$$

The completed map is shown in Figure 3-36.

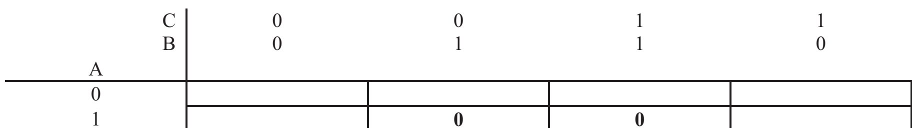


Figure 3-36. *The Karnaugh Map for the 2nd Canonical Format Using the Maxterms*

As we are using the 2nd canonical format, the process would be to identify the terms that do not change in the loop and OR them together. Then, if there was more than one loop, we would AND the loops to create the expression for the output "F." In this map there is only one loop, and the terms that do not change are "A" and "B." Therefore, the simplified expression using the 2nd canonical format is

$$F = A + B$$

This is the same as the expression obtained using the 1st canonical format. That is fine but it will not always be the case.

We should now try to simplify the expression for the circuit using Boolean algebra. To do this we will use Expression 3-1 as most engineers would use the 1st canonical format. Therefore, we start with

$$F = \overline{A}.B.\overline{C} + \overline{A}.B.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.\overline{C} + A.B.C$$

Using the inverse of the distributive law, we can rewrite this as

$$F = \overline{A}.B\left(\overline{C} + C\right) + A.\overline{B}\left(\overline{C} + C\right) + A.B\left(\overline{C} + C\right)$$

Using the OR Inverse Law $\left(\overline{C} + C\right) = 1$, we have

$$F = \overline{A}.B + A.\overline{B} + A.B$$

Now using the inverse of the distributive law again, we can say

$$F = \overline{A}.B + A\left(\overline{B} + B\right)$$

As $\overline{B} + B = 1$ we get

$$F = \overline{A}.B + A$$

We could rewrite this as

$$F = A + \overline{A}.B$$

To simplify this further, we need to examine the truth table shown in Table 3-10.

Table 3-10. *Using the Truth Table to Show* $A+\overline{A}.B = A+B$

A	B	$\overline{A}$	$\overline{A}.B$	A+B	$A+\overline{A}.B$
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

This shows that if you consider the expression $A + B$, you can see that the logic result is the same as the expression $A+\overline{A}.B$. Therefore, the expression for "F" can be simplified to

$$F = A + B$$

This agrees with both results from the Karnaugh maps.

This example shows again that using Karnaugh maps is a fairly straightforward method for minimizing Boolean expressions. However, as already stated, if the expression involves more than six terms, Karnaugh maps become very cumbersome, and Boolean algebra may be a better approach. This example does show that you will need a lot of experience to become familiar with Karnaugh maps and Boolean algebra. Even then you may have to use truth tables to help you.

Just to help confirm to you that $A+B = A+\overline{A}.B$, try simulating the test circuit shown in Figure 3-37.

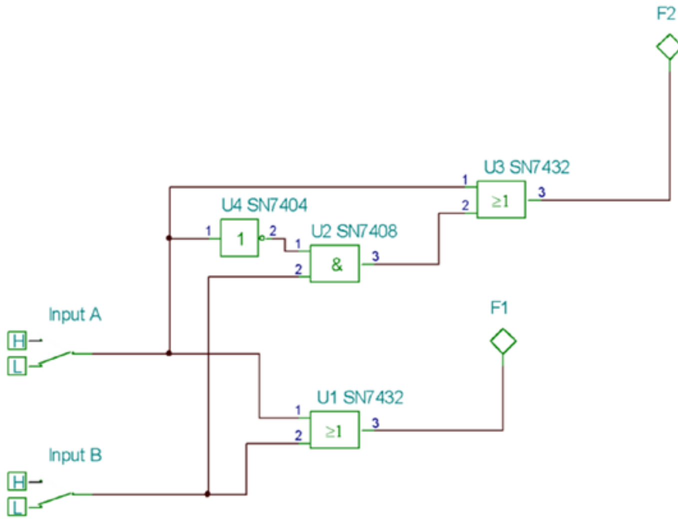


Figure 3-37. *The Test Circuit*

We can see that F2 is the output for $F = A + \overline{A}.B$, while F1 is the output for $F = A + B$. If we simulate the circuit, we should see that the two outputs work in the same way.

We should now be quite happy to appreciate that there are two formats we can use to obtain the Boolean expression for a logic circuit from the truth table, the 1st and 2nd canonical formats. In most cases engineers would use the 1st canonical format, but really if the truth table produces a lot of combinations of the inputs that produce a logic "1," then it may be easier if we use the 2nd canonical format. One such situation is with the circuit in Figure 3-39. The truth table produced 13 combinations that produced a logic "1." However, if we had used the 2nd canonical format, then we would have just three combinations that produced a logic "0" at the output. We will use this situation in Example 8 next to see if we could have used the 2nd canonical format in such a situation. You could simulate the circuit to create the truth table yourself and so confirm the above statement.

Simplification Example 8

That being the case, we will use the 2nd canonical format to produce an expression for the output "F" and then use Karnaugh maps to minimize it.

There were three combinations that produced a logic "0" in the truth Table and they would be;

$$Combination\,1\ \overline{A}.\overline{B}.\overline{C}.D$$

$$Combination\,2\ \overline{A}.B.\overline{C}.D$$

$$Combination\,3\ A.\overline{B}.\overline{C}.D$$

Using the process of inverting the individual terms, then changing the AND to an OR, and then finally ANDing the three combinations, we get an expression for the output "F":

$$F=\left(A+B+C+\overline{D}\right).\left(A+\overline{B}+C+\overline{D}\right).\left(\overline{A}+B+C+\overline{D}\right)$$

Using the binary notation, we have

$$F=\pi\left[1110,1010,0110\right]$$

There are four variables in the expression for "F," so the completed map is as shown in Figure 3-38.

AB \ DC	00	01	11	10
00				
01		0		
11		0		
10		0		

Figure 3-38. *The Completed Karnaugh Map*

The map uses variable "A" as the MSB and variable "D" as the LSB. This is because that is the way we defined the variables in that truth table.

This has two loops and so the simplified expression for the output is

$$F = Loop1.Loop2$$

You must remember we are using the 2nd canonical format.

Loop 1 is in black and the terms that don't change are

$$Loop1 = A + C + \overline{D}$$

Loop 2 is in red and the terms that don't change are

$$Loop2 = B + C + \overline{D}$$

This means that the simplified expression for the output is

$$F = \left(A + C + \overline{D}\right).\left(B + C + \overline{D}\right).$$

We can simulate the test circuit shown in Figure 3-39 to confirm that the simplification is valid.

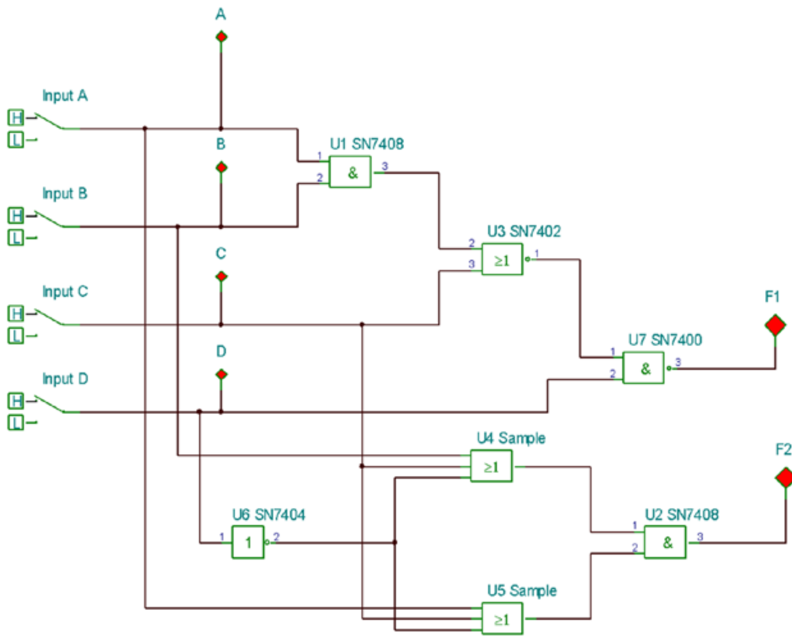


Figure 3-39. *The Test Circuit for the 2nd Canonical Format*

The output “F1” is the output for the original circuit, while F2 is the output of the simplified circuit using the 2nd canonical format. This is not really a simpler circuit, but then this example is really to show you that you can use the 2nd canonical format as an alternative approach to obtaining the Boolean expression from a truth table. However, we can apply Boolean algebra to the expression for F, which is

$$F = \left(B + C + \overline{D}\right).\left(A + C + \overline{D}\right)$$

This can be simplified to

$$F = \left(C + \overline{D}\right) + \left(B.A\right)$$

The test circuit to test these three expressions for "F" is shown in Figure 3-40.

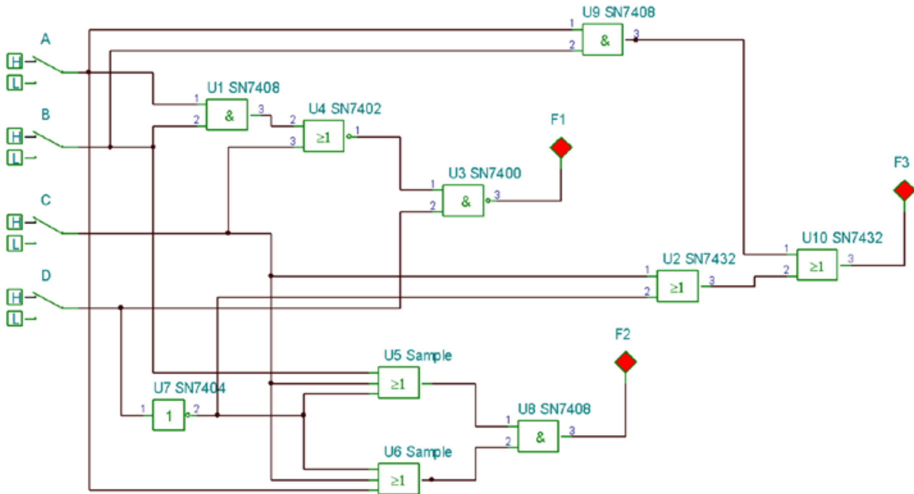


Figure 3-40. *The Test Circuit for All Three Expressions*

When we simulate the circuit, we can see that all three circuits work the same. Note the F3 output is from the now simplified expression.

Simplification Example 9

In the previous examples, we have looked at expressions that clearly show you how many variables or inputs the expression uses. In this next example, we will look at the process to deal with an expression that does not do that.

Consider the expression

$$F = \sum(0000, 0011, 0001, 1011, 1001, 1000)$$

Expression 3-3. The Binary Notation for Simplification Example 9

This uses the binary values to describe the expression for the output "F." Also, as it uses the summing term "$\sum$", we know it uses the 1st canonical format. The problem is that we can't draw up the Karnaugh map as we don't know how many variables, or inputs, are in the expression or circuit. However, if we appreciate that the highest value in the series must fit the following relationship

$$2^{n-1} < N < 2^{n}$$

where "N" is the highest value in the series and "n" is the power we raise the number 2, for binary, by, using a bit of trial and error, we can determine the value for "n," which will be the number of variables in the expression.

From the series we can see that the highest value is 11, in decimal, which is from the binary term 1011. Therefore, N = 11.

We should be able to appreciate that

$$2^{3} = 8, 2^{4} = 16$$

Letting "n" = 4 we get

$$2^{4-1} < 11 < 2^{4}$$

As this is true, it means that "n" must equal 4 and that there are four variables, which are A, B, C, and D. Then using the same approach as before, I hope you can see that the completed Karnaugh map will be as shown in Figure 3-41.

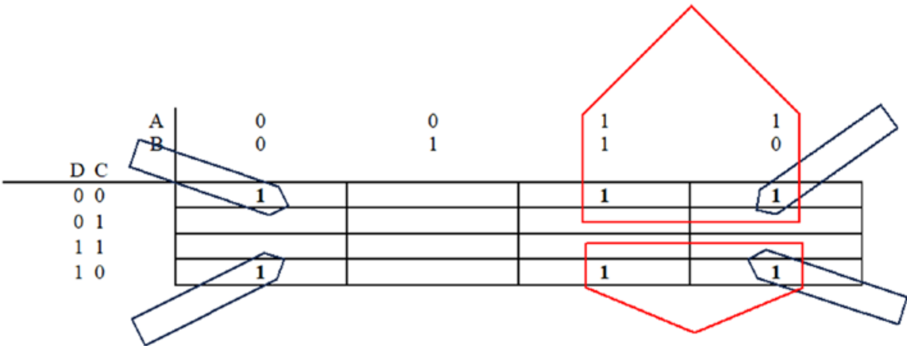


Figure 3-41. *The Completed Karnaugh Map for Example 9*

There are two loops in the map. You need to appreciate that the same minterm in the map can appear in more than one loop and that the map is spherical so both sides and tops all touch each other. Loop 1 in black envelopes all four corners, which do touch each other, because the map is spherical. We should also appreciate that the top and bottom in loop 2 in red do touch each other. As we are using the 1st canonical format, then this means that the expression is of the form

$$F = Loop1 + Loop2$$

We need to examine the loops carefully to see which variable does not change. The variables for loop 1 in black and loop 2 in red are

$$Loop1 = \overline{B}.\overline{C}$$

$$Loop2 = \overline{B}.D$$

This means the expression for the output "F" is

$$F = \overline{B}.D + \overline{B}.\overline{C}$$

Using Boolean algebra this can be simplified further to

$$F3 = \overline{B}.\left(D + \overline{C}\right)$$

This simplification can be tested; however, you have to make the circuit from the binary expression shown in Expression 3-3. This would involve the use of a six-input OR gate, which is fed from six four-input AND gates. However, as TINA only goes up to a four-input OR gate, we will need to use a four-input and a three-input OR gate to construct the six-input OR function. The final simplified expression, F3, will use one two-input AND gate, one two-input OR gate, and two NOT gates, which is quite a reduction. The completed circuit is shown in Figure 3-42.

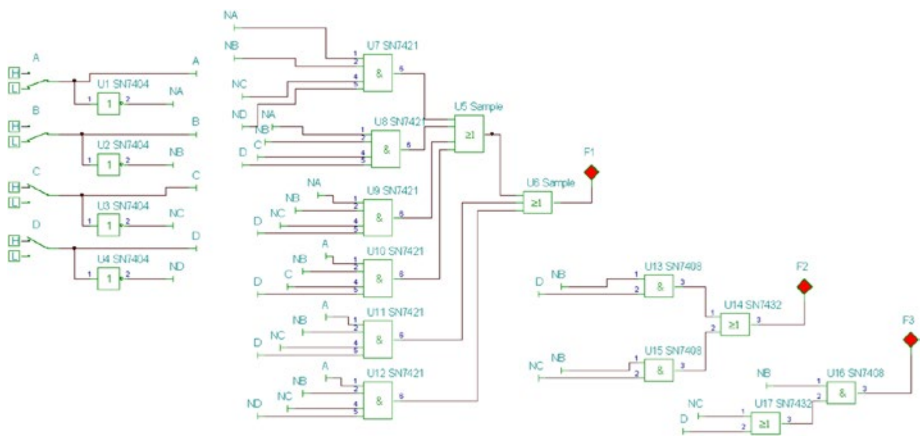


Figure 3-42. *The Test Circuit for Simplification Example 9*

This would have been rather a complex circuit to follow if we had not used the jumper terminals from within the specials option in TINA. This allows us to connect multiple points to the same connection in the circuit without having to flood the schematic with lots of wires.

I hope you can see that the construction of the circuit is correct. Then if we simulate the circuit, going through all 16 combinations of the four inputs, we will see that the three outputs, F1 being the original circuit and F2 and F3 being the simplified circuits, work in exactly the same way.

We should be able to simplify the following original expression using Boolean algebra.

The original expression is

$$F = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.C.D + \overline{A}.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.\overline{B}.\overline{C}.D + A.\overline{B}.\overline{C}.\overline{D}$$

Using the commutative law, we can move the terms around, which gives

$$F = \overline{A}.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.C.D + \overline{A}.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.\overline{B}.\overline{C}.D$$

Now using the inverse of the distributive law, we have

$$F = \overline{B}.\overline{C}.\overline{D}\left(\overline{A} + A\right) + \overline{A}.\overline{B}.C.D + \overline{A}.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.\overline{B}.\overline{C}.D$$

Knowing $\left(\overline{A} + A\right) = 1$ using the OR Inverse Law, this then gives

$$F = \overline{B}.\overline{C}.\overline{D}\left(1\right) + \overline{A}.\overline{B}.C.D + \overline{A}.\overline{B}.\overline{C}.D + A.\overline{B}.C.D + A.\overline{B}.\overline{C}.D$$

Note anything times 1 equals itself. Therefore, $\overline{B}.\overline{C}.\overline{D}\left(1\right) = \overline{B}.\overline{C}.\overline{D}$ The expression for “F” now becomes

$$F = \overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.C.D + A.\overline{B}.C.D + \overline{A}.\overline{B}.\overline{C}.D + A.\overline{B}.\overline{C}.D$$

Again, using the inverse distributive law, we have

$$F = \overline{B}.\overline{C}.\overline{D} + \overline{B}.C.D\left(\overline{A} + A\right) + \overline{B}.\overline{C}.D\left(\overline{A} + A\right)$$

Using the OR Inverse Law

$$A + \overline{A} = 1$$

the expression now becomes

$$F = \overline{B}.\overline{C}.\overline{D} + \overline{B}.C.D + \overline{B}.\overline{C}.D$$

$$F = \overline{B}.\overline{C}.\overline{D} + \overline{B}.D\left(\overline{C} + C\right)$$

$$F = \overline{B}.\overline{C}.\overline{D} + \overline{B}.D$$

Using truth tables, we can show that

$$F = \overline{B}.\overline{C}. + \overline{B}.D$$

See Table 3-11.

Table 3-11. *The Truth Table to Show that* $F = \overline{B}.\overline{C}.\overline{D} + \overline{B}.D = \overline{B}.\overline{C}. + \overline{B}.D$

B	C	D	$\overline{B}$	$\overline{C}$	$\overline{D}$	$\overline{B}.\overline{C}$	$\overline{B}.D$	$\overline{B}.\overline{C}.D$	$\overline{B}.\overline{C}. + \overline{B}.D$	$\overline{B}.\overline{C}.\overline{D} + \overline{B}.D$
0	0	0	1	1	1	1	0	1	1	1
0	0	1	1	1	0	1	1	0	1	1
0	1	0	1	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0

Inspecting the truth table, shown in Table 3-11, shows us that the last two columns are identical. This means that we can simplify the expression further to produce the same result as that from the Karnaugh map.

Finally, the expression

$$F = \overline{B}.\overline{C}. + \overline{B}.D$$

can be simplified further using, Boolean algebra, to

$$F = \overline{B}.\left(D + \overline{C}\right)$$

This is the same result. This example uses a lot of different approaches to simplify the expression. Therefore, it shows us that we need to practice all the different methods that are available to us.

Simplification Example 10

In this example we will consider a Boolean expression in which some of the minterms have a variable missing. How can this be plotted on a Karnaugh map? The expression is:

$$F = \overline{A}.\overline{B} + A.B + A.\overline{B}.C + A.\overline{C}$$

This is a three-variable expression, and so the empty map will be as shown in Figure 3-43.

C \ A B	0 0	0 1	1 1	1 0
0				
1				

Figure 3-43. *The Empty Three-Variable Karnaugh Map*

We are not using the binary number format for the expression for "F," so it is not that important which variable we make the MSB and which is the LSB. However, we have made the variable "A" the LSB and "C" the MSB.

We can see that the first term has only variables "$\overline{A}$" and "$\overline{B}$." What about the variable "C"? What it means is that this minterm $\overline{A}.\overline{B}$ does not care about the logic of "C." It will produce a logic "1" at the output F when both "A" and "B" are a logic "0" but with "C" being a logic "1" or a logic "0." It really means that this term should be

$$\overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C$$

This means we must put a "1" in the map in cells 1 and 5 as shown in Figure 3-44.

C \ AB	00	01	11	10
0	**1**			
1	**1**			

Figure 3-44. *The First Term Placed in the Karnaugh Map for Example 10*

The same can be said about the second term $A.B$ in that it does not care about the variable C. This means we will place a “1” in the map at cells 3 and 7. This is shown in Figure 3-45.

C \ AB	00	01	11	10
0	**1**		**1**	
1	**1**		**1**	

Figure 3-45. *The First Two Terms Placed in the Map*

If we add the remaining terms, we get the completed map as shown in Figure 3-46 with the loops included.

C \ AB	00	01	11	10
0	**1**		**11**	**1**
1	**1**		**1**	**1**

Figure 3-46. *The Completed Karnaugh Map for Example 10*

There are two loops, which shows the expression simplifies to

$$F = A + \overline{B}$$

This simplification can be tested by simulating the circuit shown in Figure 3-47.

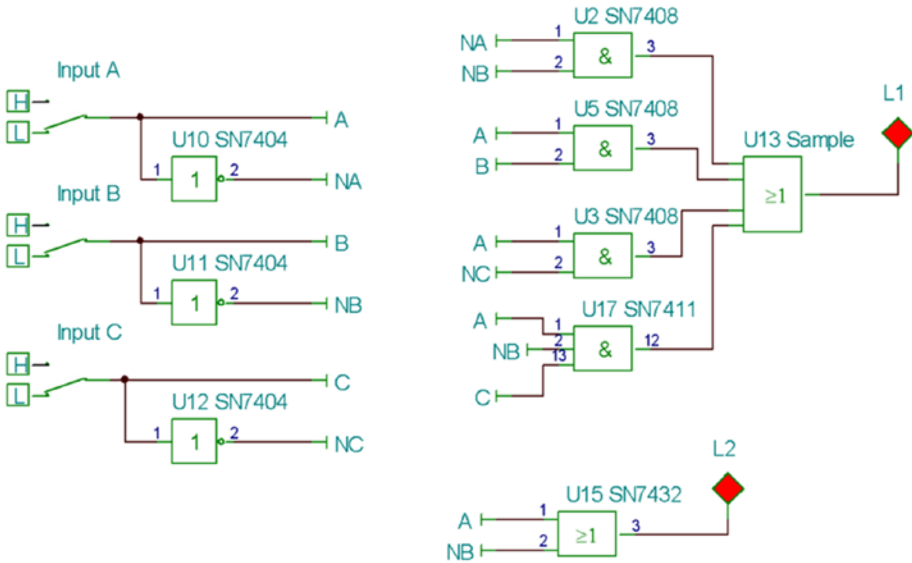


Figure 3-47. *The Test Circuit for Example 10*

Simplification Example 11

With this example I want to show you one method of completing a Karnaugh map with five variables. The process is to make two four-variable maps but label one with the fifth variable equaling a logic "0" and the other with the variable equaling a logic "1." This is explained using the following equation:

$$F = \overline{A}.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{D} + A.\overline{B}.\overline{D}.\overline{E} + \overline{A}.\overline{B}.D.E$$

The completed Karnaugh maps for the expression are shown in Figure 3-48.

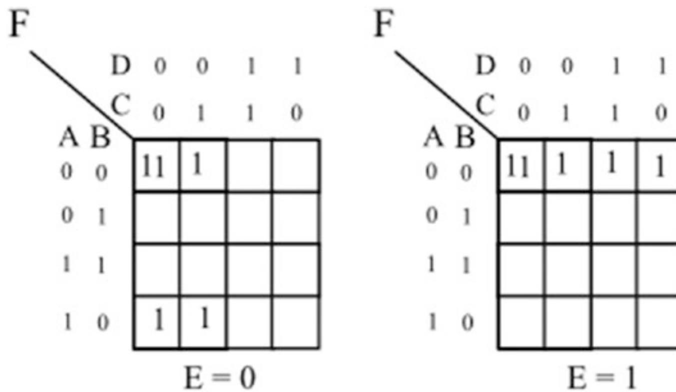


Figure 3-48. *The Karnaugh Maps for Example 11*

We are again assigning the MSB to "A" and now the LSB to "E." This expression needs careful examination to ensure we place the "1"s in the maps correctly. This is because there are some variables missing in the terms and there are five variables to accommodate. We will look at the terms one at a time and decide which cells of the maps they relate to.

The first term is

$$\overline{A}.\overline{B}.\overline{C}.\overline{D}$$

The variable "E" is missing from the term. That means that with respect to this term, the expression does not care about the logic of the variable "E"; it will work the same if E is a logic "1" or if E is a logic "0." The term could be rewritten as

$$\overline{A}.\overline{B}.\overline{C}.\overline{D}.\overline{E} + \overline{A}.\overline{B}.\overline{C}.\overline{D}.E$$

This means that the "1" will appear in the same cell in both the E = 0 map and the E = 1 map. The actual cell is cell 1, as all variables are at a logic "0" in that cell, hence the first "1" in that cell in both maps; see Figure 3-48.

The second term is

$$\overline{A}.\overline{B}.\overline{D}$$

In this term the variables "C" and "E" are missing, which means we must add a logic "1" and a logic "0" for both variables. This means the term should read as

$$\overline{A}.\overline{B}.\overline{C}.\overline{D}.\overline{E} + \overline{A}.\overline{B}.C.\overline{D}.E$$

This will put a "1" in cells 1 and 2 in both maps. That is why there are two "1"s in cell 1; see Figure 3-48.

The third term is

$$A.\overline{B}.\overline{D}.\overline{E}$$

In this term the variable "C" is missing, which means we should read this term as

$$A.\overline{B}.C.\overline{D}.\overline{E} + A.\overline{B}.\overline{C}.\overline{D}.\overline{E}$$

These will put a "1" in cells 13 and 14 in the E = 0 map but not in the E = 1 map; see Figure 3-48.

The final term is

$$\overline{A}.\overline{B}.D.E$$

This is missing the "C" variable and so it should read as

$$\overline{A}.\overline{B}.C.D.E + \overline{A}.\overline{B}.\overline{C}.D.E$$

These will put the "1" in cells 3 and 4 but in the E = 1 map only; see Figure 3-48.

If we now loop the "1"s in the maps, we get the maps as shown in Figure 3-49.

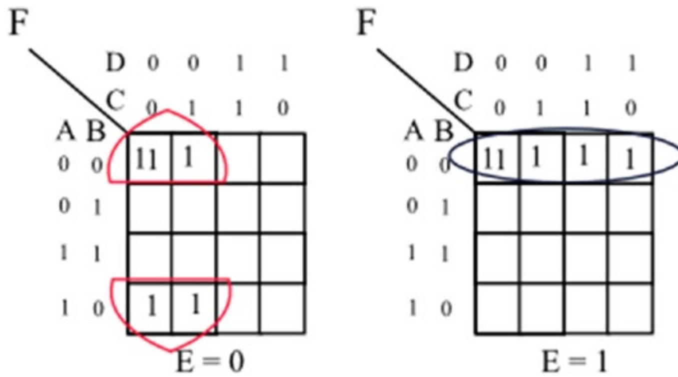


Figure 3-49. The Completed Karnaugh Maps for Example 11

This gives us two loops, one on the map E = 0 and the other on the map E = 1. Therefore, the expression for "F" becomes

$$F = \overline{B}.\overline{D}.\overline{E} + \overline{A}.\overline{B}.E$$

As always, we can test this simplification using the ECAD software. The circuit to test it is shown in Figure 3-50.

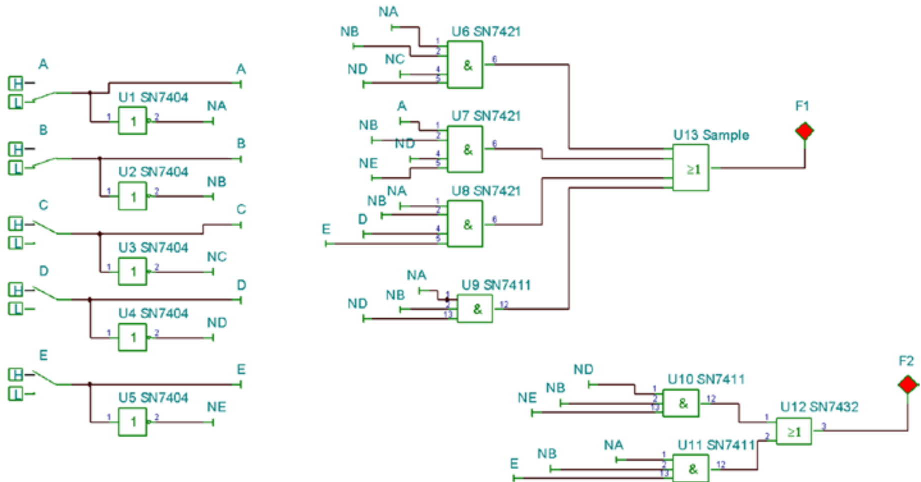


Figure 3-50. Test Circuit for Example 11

The output F1 is the output of the original circuit created using the original expression. The output F2 is the output of the simplified circuit created using the simplified expression. If we simulate the circuit, by switching the inputs through all the 32 possible combinations, we should see that the two outputs work in exactly the same way.

I hope these examples of the different methods to simplify Boolean expressions do give you enough experience to appreciate how they all work and how you can use the ECAD software to confirm your work.

The following is a set of exercises for you to try. The answers to the exercises are in the appendix.

Exercise 3.1

Use Boolean algebra to simplify the following expressions:

1. $F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.C + \overline{A}.\overline{B}.C$
2. $F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C$
3. $F = \overline{A}.B.C.D + \overline{A}.B.\overline{C}.D + A.B.D$

Exercise 3.2

Use Karnaugh maps to minimize the expressions stated in Exercise 3.1.

Exercise 3.3

Express the expressions shown in Exercise 3.1 in their binary and decimal notations.

Exercise 3.4

The truth table obtained from the simulation of the circuit shown in Figure 3-51 is shown in Table 3-12. Create the 1st and 2nd canonical expressions from the truth table and choose a suitable method to minimize the expression.

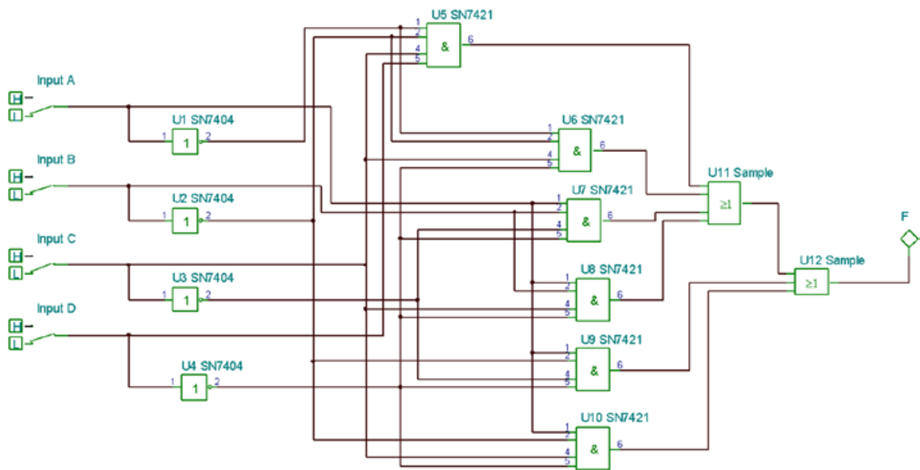


Figure 3-51. *The Logic Circuit for Exercise 3.4*

Table 3-12. *The Truth Table for Exercise 3.4*

Input A	Input B	Input C	Input D	Output F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1

(*continued*)

Table 3-12. (*continued*)

Input A	Input B	Input C	Input D	Output F
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Summary

In this chapter we have studied the fundamentals of Boolean algebra and how to use Boolean algebra to minimize Boolean expressions. We have also learned how to use Karnaugh maps to minimize Boolean expressions.

We have learned how to use the ECAD software to confirm our work by simulating the logic circuits.

We have studied the use of 1st and 2nd canonical formats and what minterms and maxterms are.

I hope you have found this chapter informative and that you have used it to help develop your abilities in handling Boolean expressions.

In the next chapter, we will study how engineers tried to make use of these logic gates they had created. We will learn about the SR latch and the JK flip flop, which are some of the building blocks of the logic circuits we use today and even the computers we use every day.

CHAPTER 4

Moving On from the NAND Gate

This chapter will discuss how engineers moved on from discovering the first NAND gate. It will introduce you to the SR latch and show how this led to the creation of the memory in the modern-day computers.

We will discuss the JK flip flop, the D-type latch, and the T latch. I hope the text and the test circuits will give you a good foundation of digital electronics and create a desire to use the circuits themselves.

The SR Latch

In Chapter 2 we stated that the NAND gate was the first and most common gate manufactured. This was basically because of the inverting action of the transistor technology in TTL. This meant that engineers started to investigate what they could do with these NAND gates. One of the first things they did was to connect two together in what I term the *cross-coupled configuration* as shown in Figure 4-1.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_4

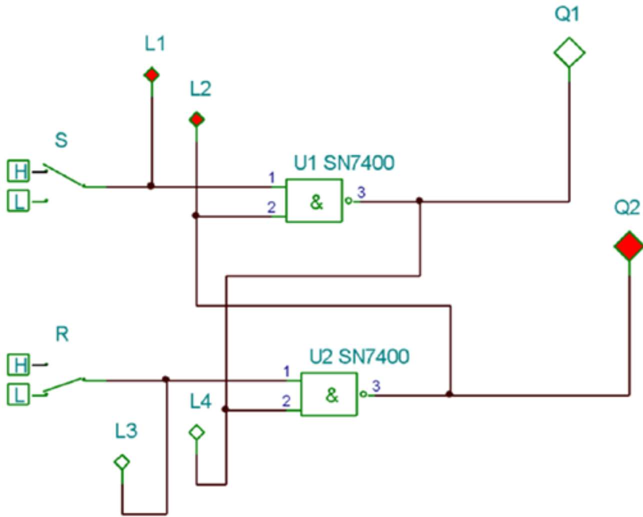


Figure 4-1. *The Cross-Coupled NAND Gate Circuit*

Table 4-1. *The Truth Table for the Two-Input AND and NAND Gate*

Input A	Input B	AND Output	NAND Output
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

To appreciate how this circuit works, you need to understand the truth table for the two-input NAND gate. This is shown in Table 4-1. Note I normally construct the truth table for the two-input AND gate and then add an extra column to include the NAND. This is because the NAND is simply the opposite response of the AND gate. Therefore, I have included it as the fourth column in red, and it is that which is of interest to us. You can see that the NAND is the exact opposite of the AND gate's response to the inputs.

This shows that the output of the NAND gate will only be a logic "0" when both inputs are at a logic "1"; at all other times, the output will be a logic "1."

This is why the "S" input to the circuit is shown high and the "R" input is shown low; see Figure 4-1. If we assumed that the output "Q1" was already low then, as this is fed to the NAND gate connected to the "R" input, this would mean that the U2 NAND gate would have two logic "0"s at its input; hence, both L3 and L4 are shown turned off. This would then ensure that the output "Q2" was high as shown in Figure 4-1. The "Q2" output is cross-fed back to the other input of the U1 NAND gate. This would then mean that, as the "S" input is high, there would be two logic "1"s fed to the U1 NAND gate; hence, both L1 and L2 are shown red, that is, turned on. This would then ensure the "Q1" output was at a logic low, that is, a logic "0"; see Table 4-1.

Now to simulate the action of the circuit, we must carefully change the inputs, S and R, as stated in Table 4-2 and note how the outputs Q1 and Q2 respond. It is important that you follow the sequence as shown in Table 4-2. Note the first row is just to agree with the initial state of the circuit as shown in Figure 4-1.

Table 4-2. *The Truth Table of the SR Latch*

Row Number	S	R	Q1 Q	Q2 $\bar{Q}$
1	1	0	0	1
2	1	1	0	1
3	0	1	1	0
4	1	1	1	0
5	1	0	0	1
6	1	1	0	1
7	0	1	1	0
8	0	0	1	1

There are two outputs from the circuit and, as in many things that have two or more consequences, we engineers give one of the outputs more importance than the other. You can also see that in all but the last row of the table, the Q2 is the opposite of the Q1. It is for that reason engineers call the Q1 the Q output and the Q2 the NOT Q Output. To help identify the outputs, we use the symbology of "Q" and "$\overline{Q}$, " that is, Q and NOT Q.

This gives Q1, the Q output, the more dominance over the NOT Q; however, as we will see later, this dominance becomes slightly more blurred. We also use the convention that when we force an output to go high, that is, to a logic "1," we are "setting" it. When we force the output to go low, that is, to a logic "0," we are "resetting" it, hence the "S" and the "R" at the inputs.

Therefore, you can see that the two occurrences shown in Table 4-2 when the Q1 or Q changed to a logic "1" were when the "S" input changed to a logic "0." These are in rows 3 and 7.

We can see that in row 4 the Q output was also a logic "1," but this was not a change in output, as Q was already a logic "1" from row 3.

We can see that in row 5 the Q changed back to a logic "0," that is, it reset, when the "R" input changed to a logic "0," while the "S" input was a logic "1."

This would indicate that you can set the Q by sending the "S" input low while keeping the "R" input high. You can then reset the Q by sending the "R" input low while keeping the "S" input high. In this way we classify the action of the circuit as being "active low," which means to activate the circuit, we must send the controlling input low.

If we now consider rows 2, 4, and 6 when both inputs had gone high after one of them had gone low, we can see that the circuit did not change from the previous condition. This is what we call the output being latched to what it was previously. Indeed, this makes this simple circuit rather useful, something we will look at shortly.

Before we do that, we must look at row 8 of Table 4-2. This is when we have driven both the "S" and "R" inputs low. We have just called this circuit an active low circuit, so this means that by driving both inputs low, we are trying to activate both the Set and the Reset response at the same time. This is something you should not do, and the outputs, both Q and NOT Q, are showing as both being high. This is when the circuit does not act as we expect. We expect the NOT Q to be the opposite of the Q, and it is not. This is what is called the "indeterminate state." It is only TINA that has shown both outputs high. In practice they may take on any logic state, and it is truly indeterminate, that is, we can't predict what will happen, something we engineers don't like. Well, of course you can say that no one will try to set the latch at the same time as they try to reset it. That would be daft. Yes, that is true, but if you say to someone don't do that, you know that at some point someone will do that. That is why engineers cannot leave a system in a state whereby they can't control what would happen. They must do something about it, and that is what they tried to do. We will see what they did later in this chapter.

The circuit, shown in Figure 4-1, is called the SR latch because operating the S input to a logic "0" while the R input is high sets the Q output to a logic 1 as shown in Table 4-1. The latching operation comes about because if the S input now changes, the Q output does not change; it stays latched at logic "1." It does not matter how many times the S input is operated; the Q output stays latched on. Try it by setting the latch while keeping the "R" input high and then repeatedly toggle the "S" input from low to high, again while keeping the R input high. You will see that the Q remains set and the NOT Q remains reset.

Indeed, the only way to return the Q output back to a logic "0" is to operate the R input, that is, send it low to a logic "0," while the "S" input is high. Operating the R input, that is, sending it low, returns the Q output back to a logic "0." This is termed resetting the Q output. Then, no matter

how many times the R input is operated, the Q output stays latched in the reset mode, that is, the Q output is at a logic “0.” That is why this circuit is called the

$$\overline{S}\,\overline{R}\ \ latch$$

Note the bars above the “S” and “R” terms are there to indicate this is an active low latch.

The De-bounce Circuit

Due to the fact that the outputs stay latched once the input goes low no matter how often the logic at the input changes from then on, this circuit can be used as a hardware de-bounce circuit.

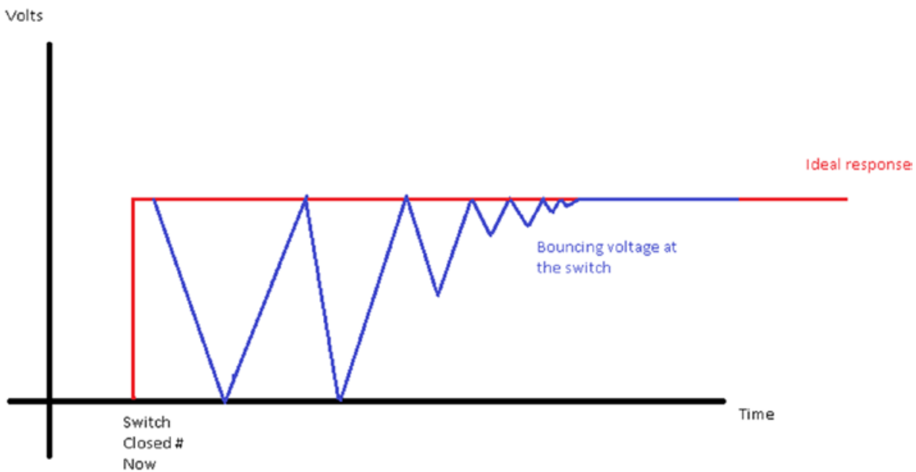


Figure 4-2. *The Bouncing Voltage Output of a Physical Switch*

Figure 4-2 shows an exaggerated representation of what happens when an electrical switch operates. Newton’s third law says that for every action there is an equal and opposite reaction. This means that when we close a

switch, the switch sets up an equal force that tries to open the switch again. This means that the voltage at the output, shown in blue, does not respond in the ideal way, shown in red. It does bounce between 0V and 5V for some time. This can cause problems in some logic circuits, especially counters. Therefore, we need a way to overcome this bouncing action. Mechanically some switches employ a spring. With microprocessors we can employ a short delay function, and with digital electronics we can use the $\overline{S}\ \overline{R}$ latch.

The Basic SR Latch with NOR Gates

The same circuit can be constructed using NOR gates instead of NAND gates as shown in Figure 4-3.

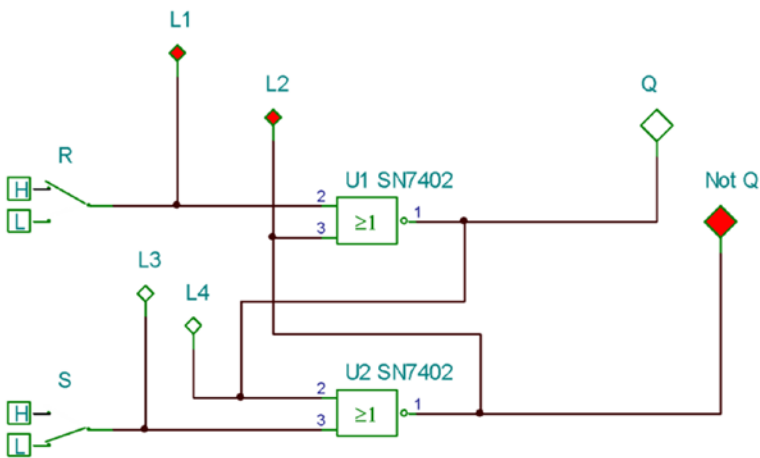


Figure 4-3. *The SR Latch with NOR Gates*

To understand how this circuit works, we need to examine the truth table of the two-input NOR gate. This is shown in Table 4-3.

Table 4-3. The Truth Table for the Two-Input NOR Gate

Input A	Input B	Output OR	Output NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

With the NOR gate, the output, shown in red, will be a logic "1" only when both inputs are at a logic "0." With any other combination of inputs, the output is a logic "0." If we look at the circuit shown in Figure 4-3, we see that the two inputs to the upper NOR gate, that is, the U1 gate, are both at a logic "1." This means that the output of the U1 NOR gate will be a logic "0." That is cross-coupled to be an input to the U2 NOR gate. This is confirmed with the L4 logic probe. Therefore, as the "S" input to the U2 NOR gate is also a logic "0," then the output of the U2 NOR gate, that is, the NOT Q output, will be a logic "1." This is then cross-coupled to become the second input to the U1 NOR gate. The U1 NOR gate now has both its inputs at a logic "1," which ensures the Q output will be a logic "0." We should also appreciate that if the "R" input changes to a logic "0," the Q output would still remain at a logic "0" as the NOR gate only requires one of its inputs to be a logic "1" to maintain its output at a logic "0"; see the truth table in Table 4-3.

Now to study the response of the NOR SR latch, you need to change the "R" and "S" inputs as stated in Table 4-4. Note the first row is set to show the condition of the SR latch as shown in Figure 4-3.

Table 4-4. *The Truth Table for the NOR SR Latch*

Row Number	S	R	Q1 Q	NOT Q $\overline{Q}$
1	0	1	0	1
2	0	0	0	1
3	1	0	1	0
4	0	0	1	0
5	0	1	0	1
6	0	0	0	1
7	1	0	1	0
8	1	1	0	0

As you go through the sequence carefully, you should see that in row 1 we have just reset the Q output by switching the "R" input high. Then, in row 2, when both the "S" and the "R" are at logic "0," nothing changes from the previous state. This is repeated in rows 4 and 6.

In row 3 we set the "Q" output when the "S" input goes high. Finally in row 8 when both inputs are at a logic high, both the "Q" and the "$\overline{Q}$" outputs are at a logic "0." This is because this is the indeterminate state that, really, we can't allow.

There are two main differences between the two SR latches, and they are that, first, the inputs for the NOR SR latch are now active high, which means when the logic at the inputs goes to a logic "1," the outputs respond accordingly. Figure 4-3 shows the logic at the R input is a logic "1" and the Q output has just reset to a logic "0."

The second difference is that the two inputs have now switched their respective positions, that is, the R input is the upper NOR gate and the S input is the lower NOR gate. It is normally a matter of personal preference which circuit you use. I normally use the NAND gate circuit, but there will be a time, in the book, when I will use the NOR gate circuit.

The Indeterminate State

The only problem with the simple $\overline{S}\,\overline{R}$ latch and the SR latch is the indeterminate condition. Note I am using the symbol for the two inputs that have a bar across the top first, because I prefer using the NAND gate circuit. This circuit is shown in Figure 4-1 and is active low. This means that the inputs must go low to set and reset the latch.

We as engineers know what will happen when we activate the Set input, that is, the Q output will go to a logic "1." Also, when the R input is activated, we know the Q will go back to a logic "0." We also know that nothing happens when the S and R inputs change after setting or resetting the Q output. The problem comes about when the S and R inputs are activated at the same time.

All these possibilities are recapped in the truth table for the active low $\overline{SR}$ latch, shown in Figure 4-1, as shown in Table 4-5.

Table 4-5. *The Truth Table for the Active Low SR Latch*

Circuit Response	S	R	Q1	Q2
Reset	1	0	0	1
No change from previous state	1	1	0	1
Set	0	1	1	0
No change from previous state	1	1	1	0
Reset	1	0	0	1
No change from previous state	1	1	0	1
Set	0	1	1	0
The indeterminate state	0	0	1	1

Engineers cannot leave the design of a circuit with the chance of an indeterminate state. Think about saying to someone, "You must not try setting and resetting the latch at the same time." We know that someone will do that. An engineer once told me that there should never be a system that could cause an accident. The engineer should think of all occurrences and make sure the reaction of the system is safe and reliable. That is something that all engineers should strive for.

The Clocked $\bar{S}\,\bar{R}$ Latch

The first thought about improving the $\bar{S}\bar{R}$ latch was to add a clock signal to it. This would also enable us to synchronize the operation. This is shown in Figure 4-4.

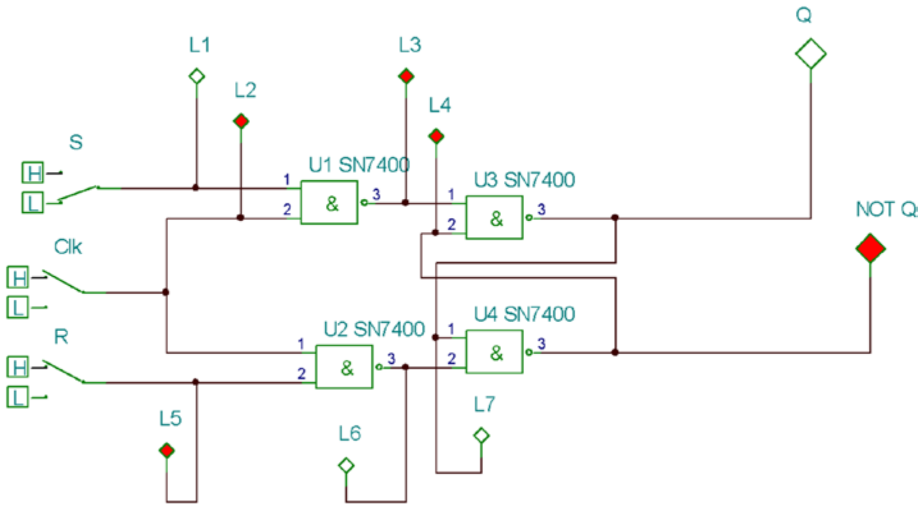


Figure 4-4. *The Clocked SR Latch with NAND Gates*

We should appreciate that with the two-input NAND gate, the only time the output would be a logic “0” is when both inputs are at a logic “1.” In Figure 4-4 the clock input, “Clk,” has just gone to a logic “1” just after the “R” input was switched to a logic “1.” This means that the U2 NAND gate would produce a logic “0” at its output; see L6 in Figure 4-4. This then means that the output of the NAND gate U4 must be a logic “1” regardless of what the logic on the other input was. Therefore, the NOT Q output of the latch must be a logic “1” as indicated by the logic probe on the NOT Q. This logic “1” is cross-coupled to become an input to the U3 NAND gate; see L4 in Figure 4-4.

The other “S” input has been kept at a logic “0” as shown in Figure 4-4; see L1. This is one of the inputs to the U1 NAND gate. The other input is from the clock, which is at a logic “1.” The fact that one of the inputs to the U1 NAND gate is a logic “0” means that the output must be a logic “1”; see L3 in Figure 4-4. This is the second input to the U3 NAND gate, and as the

other is also a logic "1" (see L4), which comes from the NOT Q output, then the Q output of the U3 NAND gate must be a logic "0"; see the logic probe for the Q output in Figure 4-4.

This is the case where the latch has just been reset with the Q at a logic "0" and the NOT Q at a logic "1." This is the state of the $\overline{S}\overline{R}$ latch as shown in Figure 4-4.

The U3 and U4 NAND gates make up the original $\overline{S}$ $\overline{R}$ Latch as shown in Figure 4-1. The two inputs are shown as L3 and L4 in Figure 4-4.

The only real inputs to this clocked SR latch are the "S" and "R" inputs. These two inputs are the only logic levels that the user controls. Therefore, the user must use the "S" input to set the latch. If the user sends this "S" input to a logic "1," then, as long as the "R" input is at a logic "0," when the clock input next goes high, the two inputs, L1 and L2, to the U1 NAND gate will be at a logic "1." This will force the output L3 to a logic "0"; see Table 4-1 to confirm the operation of the NAND gate. This will be sufficient to force the Q output to go to a logic "1"; again see Table 4-1. The fact that the "Q" output has gone to a logic "1" means that the L7 input to the U4 NAND gate goes high, which will force the NOT Q output to go to a logic "0." This will become the L4 input to the U3 NAND gate, which will keep the Q output at a logic "1," that is, set, even if the user returns the "S" input to a logic "0."

This circuit synchronized the action of the S and R inputs with the clock signal. The Set input does not activate the latch, that is, set the Q output, if the clock input is at a logic "0"; it must wait for the clock input to go to a logic "1" also. However, this clocked SR did not resolve the problem with the indeterminate state.

The truth table of the clocked SR latch will help us confirm the operation of the circuit. This is shown in Table 4-6.

Table 4-6. *The Truth Table for the Clocked SR Latch*

Row Number	S	R	Clock Clk	Q1 Q	Q2 $\bar{Q}$	State of Latch
1	0	1	1	0	1	Reset
2	0	1	0	0	1	Can't change reset
3	0	0	0	0	1	Can't change reset
4	1	0	0	0	1	Can't change reset
5	1	0	1	1	0	Set
6	1	0	0	1	0	Can't change set
7	0	0	0	1	0	Can't change set
8	0	0	1	1	0	No change set
9	0	1	1	0	1	Reset
10	0	0	0	0	1	Can't change reset
11	0	0	1	0	1	No change reset
12	1	1	1	1	1	The indeterminate state

I have gone through the changes very carefully to try and show you how the circuit responds. I hope you can see that the latch will only change over to the Set state in row 5. This is when both the "S" and "Clk" inputs

go high. Also, you should see that the latch will only change to the Reset state when both the "R" and "Clk" inputs go high; see row 9. In this way the latch is now an active high SR latch, and it will be synchronized to when the clock goes high. Note in row 4 the "S" is a logic "1" but the latch does not set because the "Clk" input is still low. Row 8 shows the same response when the "R" input goes high but the "Clk" is still low, that is, the latch does not change to the Reset state.

So we have now got an active high SR latch that is synchronized to a clock input. However, we still have the indeterminant state; see row 12. We still need to overcome this problem.

The Master-Slave Clocked SR

The next attempt was to create the master-slave clocked SR. This is shown in Figure 4-5.

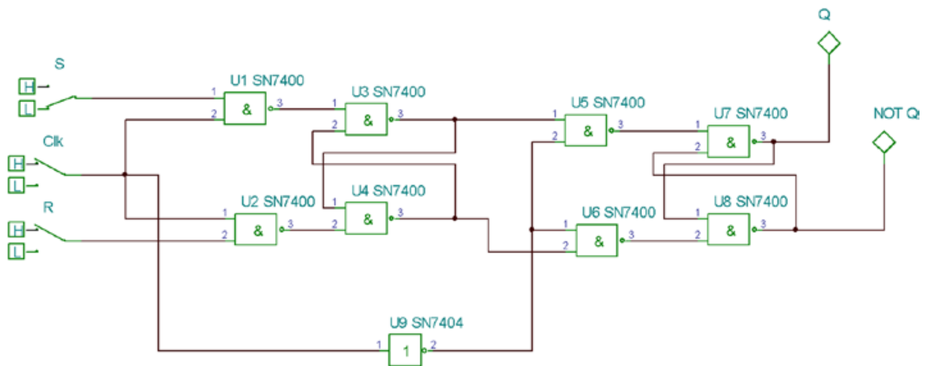


Figure 4-5. *The Master-Slave Clocked SR Latch*

This circuit consists of two clocked SR latches, the first being the master and the second being the slave. The clock input to the slave is the NOT of the clock signal to the master. This means when the master is enabled, by the clock going high, the slave is disabled, as its clock is low. Then when the master is disabled, the slave is enabled.

The circuit worked as expected, but the indeterminate state was not eradicated.

The next circuit, shown in Figure 4-6, fed back the outputs from the Q and NOT Q to the inputs. This finally removed the indeterminate state, and the circuit was named the JK flip flop. We will discuss why later.

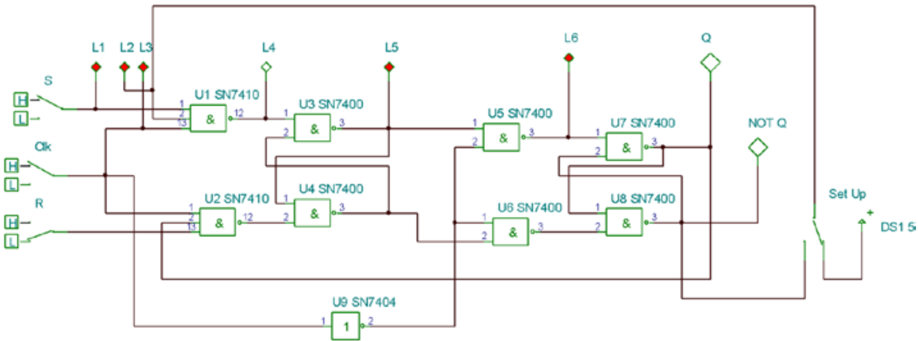


Figure 4-6. *The Master-Slave Clocked SR Latch with Feedback*

With this extension of the master-slave clocked SR latch, the indeterminate state was no longer an issue as the outputs simply toggled from logic 1 to logic 0 and back to logic 1 every time the clock signal enabled the master SR assuming we had sent both the S and R inputs high. In this way then, the logic circuit simply divided the clock signal by 2.

However, there is an extra switch in the circuit. This is labeled “Set Up Switch.” My thoughts are that this switch is just for the simulation. This circuit is rather a lot to model, and without this switch the circuit does not perform well. If you simulate the circuit, you will see that a lot of the logic on the gates is shown in black. I think this is because the software does not have enough data to determine what their logic should be.

With this “Set Up Switch” initially switched to the +5V, we ensure that the feedback logic to the U1 three-input NAND gate is a logic “1.” Then when both the clock “Clk” and the “S” input go to a logic “1” as well, all three inputs to the U1 NAND gate are at a logic “1,” and the output of the U1 NAND gate is forced to a logic “0”; see L4. This becomes one of the

inputs to the U3 two-input NAND gate, and so, regardless of the other input, the output of the U3 gate must go to a logic "1"; see L5. This logic "1" becomes the input to the U5 NAND gate. The other input to this U5 gate comes from the output of the NOT gate U9. This will have been forced to a logic "0" because the input to this NOT gate was forced high when the clock input went to a logic "1." This means that the output of the U5 gate must go to a logic "1," as indicated by the probe L6. This is fed as an input to the U7 NAND gate. This is a two-input NAND gate, and the other input has been forced to a logic "1" by the "Set Up Switch," which has been switched to +5V. This means the Q, which is the output of the U7 NAND gate, must go to a logic "0." Also, due to the cross-coupling of the U7 output to the U8 NAND gate, the NOT Q output is forced to a logic "1."

The JK flip flop has now been set up, and the "Set Up Switch" can now be changed over to replace the +5V, from the supply, with the logic "1" output of the NOT Q from the U8 NAND gate. The circuit can now work properly.

I am not sure whether this "Set Up Switch" is required just for the software, as I had to do a similar modification when I simulated the circuit using the Proteus ECAD software. It may, due to possible race hazards, be a requirement for the practical JK flip flop, although I am not sure how it could be implemented. I am quite happy to use this extra switch in the simulation as it does follow the actions of the 7476 JK flip flop we will use later.

This master-slave clocked SR with feedback circuit can be tested to show how it works if you go through the following actions:

- With the S, R, and clock inputs switched to a logic "0" and with the "Set Up Switch" switched to a logic "1," we are now ready to turn the circuit on.
- Now switch the circuit on in the digital mode by clicking the mouse on the DIG button. The circuit turns on and displays a lot of black signals at the gates.

- Now switch the S input to high. Nothing changes.
- Now switch the Clk input to high. All the black logic levels disappear, and the gates take up recognized logic levels. The Q output goes to a logic "0," that is, it is turned off, and the NOT Q goes to a logic "1," that is, it is turned on. Also, this has enabled the master clocked SR but disabled the slave, due to the action of the NOT gate U9 in Figure 4-6.
- Now change the "Set Up Switch" over. Nothing changes.
- Now change the clock to a logic "0." This disables the master but enables the slave. The Q output now goes to a logic "1," and the NOT Q goes to a logic "0." The JK has now finally been set.
- Now change the S input to a logic "0." Nothing changes.
- Now switch the R input to a logic "1." Nothing changes.
- Now change the Clk input to a logic "1." Nothing happens.
- Now switch the Clk input to a logic "0." The Q will change to a logic "0," and the NOT Q changes to a logic "1." The JK has now finally been reset.

This test sequence shows us two major aspects of the JK flip flop. First, it shows us that the Set and Reset control actions are active high. Second, it shows us that the JK flip flop activates when the clock input, Clk, goes from high to low, that is, it is active low. This is because when the clock goes low, we disable the master but enable the slave.

Now we will see what happens if we try to set the JK at the same time as we try to reset it. This means switching both the S and R inputs to a logic "1." Therefore, do the following:

- Now, with the R input still at a logic "1," switch the S to a logic "1" while keeping the Clk input at a logic "0." Nothing changes.
- Now change the Clk input to a logic "1." Nothing changes.
- Now change the Clk input back to a logic "0." The Q will change to a logic "1" and the NOT Q to a logic "0."
- Now if you simply keep on toggling the Clk input between a logic "1" and logic "0," you will see that the Q and NOT Q outputs toggle as well. However, the Q and NOT Q outputs toggle at half the rate of the clock. This means that the JK divides the clock by 2.

This means that, with this feedback configuration, the indeterminate state, which we had before, has been changed into something we can make use of, that of a counting action.

The final point to make a note of is that the S input becomes the J and the R becomes the K and the JK flip flop was born.

The JK Flip Flop

The JK flip flop is probably one of the most important logical devices ever designed. It may be viewed as an integrated circuit, as it is made up of the circuit shown in Figure 4-6 plus some more bits. It does the same for digital electronics as the opamp did for analogue electronics.

Whomever the JK flip flop can be accredited to is somewhat debatable. Some say this type of flip flops was invented by a Texas Instruments engineer Jack Kilby, hence the terms *J* and *K*. Some say Dr. Eldred Nelson coined the term while working for Hughes Aircraft. I like the Jack Kilby explanation, but I am not saying I know who first created it. After all it is basically a journey, which I have tried to explain previously, that started with the $\overline{S}\,\overline{R}\,latch$ as shown in Figure 4-1.

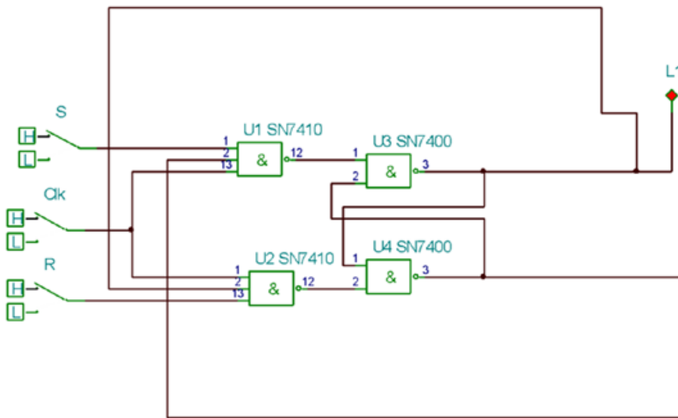


Figure 4-7. *Another Circuit That Some Say Is the JK Flip Flop*

Some say that the JK flip flop can be made up of the single clocked SR with feedback. This is shown in Figure 4-7. I have simulated this in TINA, and even though it does act correctly as the clocked SR latch shown in Figure 4-4, when both the S and the R are set to logic "1," the Q output, that is, L1 in Figure 4-7, does not toggle as the one in Figure 4-6 does. It just stays at a logic "1." I know the JK flip flop circuit is quite complicated, and an ECAD software can struggle to simulate it correctly. However, I will say, with some confidence, that the JK flip flop is made using the circuit as shown in Figure 4-6. It is not essential that we know the exact makeup of the JK flip flop. I am sure the IC will have a lot more protective circuitry than what is in Figure 4-6, but Figure 4-6 does show the basic construction

of the flip flop. As well as that, I am really just trying to show how the flip flop grew from the $\overline{S}\,\overline{R}$ *latch.* I hope I have given you a good appreciation of how it came about.

The JK flip flop is at the heart of many sequential logic circuits. It can be configured using the two inputs, J and K. We will look at the possible combinations and the JK's response in this section of the chapter.

There are a variety of symbols for the JK; however, we will use the symbol that comes with TINA. This is shown in Figure 4-8.

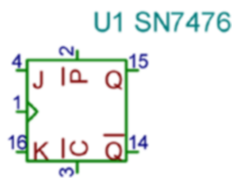


Figure 4-8. *The Symbol for the SN7476 JK Flip Flop*

There are five inputs and two outputs to the JK flip flop. Each input has the following symbols and meaning:

- The J is a normal input connected to pin 4.
- The CLK is the clock input, connected to pin 1, and it can have the following symbols associated with it:
  - The small triangle as shown in Figure 4-8, which means that the trigger that defines when the JK reacts to the clock signal is the edge of the square wave. As there is no circle or bar associated with the small triangle, then this is positive edge triggering. This means the JK will react to the rising edge of the clock signal.
  - There may be a small triangle with a small circle or bar. This indicates negative edge triggering or the falling edge of the clock signal.

- There may be just a small circle or bar at the input. This indicates low-level triggering. This means the JK will react when the clock signal is low, that is, at 0V, but not on the transition from high to low as in edge triggering.
- There may be no symbol at all, which indicates high-level triggering. This means the JK will react to the clock signal when it is high level, that is, at +5V.

However, these level triggering events are not as precise as edge triggering, and so edge triggering is normally used.

- The K is a normal input shown as being on pin 16.
- The $\overline{P}$, which may be an "S," is a Pre-set, or Set, input on pin 2. The bar above the "P" means that this input is active low. Sometimes there may be a circle in front of the input instead. This means the same thing. If there is no small circle or bar, then this input is active high, which means this input activates the JK when it goes high to a logic "1." When this input is activated, the Q output will be set regardless of the clock and other inputs.
- The $\overline{C}$ or "R" input is used to reset the "Q." As with the $\overline{P}$ or "S" input, this could be active high or active low.
- The "Q" is the main output.
- The $\bar{Q}$ is an alternative output. The bar that may be replaced with a circle indicates that this output will always be the opposite in logic of the "Q" output.

Table 4-7 lists the pins of the 7476 JK flip flop and their usage.

Table 4-7. *The Pins of the 7476 JK Flip Flop*

Pin Number	Name	Type	Usage
1	Clock	Negative edge trigger.	Used to give the JK a signal to count and others
2	$\overline{P}$ for Pre-set or S for Set	Active low. The small circle means active low. Sometimes a bar across the letter is used to indicate the same active low.	Used to force the Q to a logic 1 when this pin goes to a logic 0
3	$\overline{C}$ for Clear or R for Reset	Active low. The small circle means active low. Sometimes a bar across the letter is used to indicate the same active low.	Used to force the Q output to go back to a logic 0 when this input goes to a logic 0
4	J input	Can be a logic "1" or logic "0."	Used to configure how the JK is used
16	K input	Can be a logic "1" or logic "0."	Used to configure how the JK is used

There are two outputs, which are the Q and the $\overline{Q}$. These two outputs will be the opposite of each other depending upon the state of the inputs.

Using the JK Flip Flop

The J and K inputs are used to configure how the flip flop works. As there are two inputs, then there will be four possible combinations of those inputs we need to investigate. The first would be with both inputs at a logic "0," that is, both tied to 0V. The test circuit for this configuration is shown in Figure 4-9. To investigate how this circuit works, I am using a switch to control the clock signal.

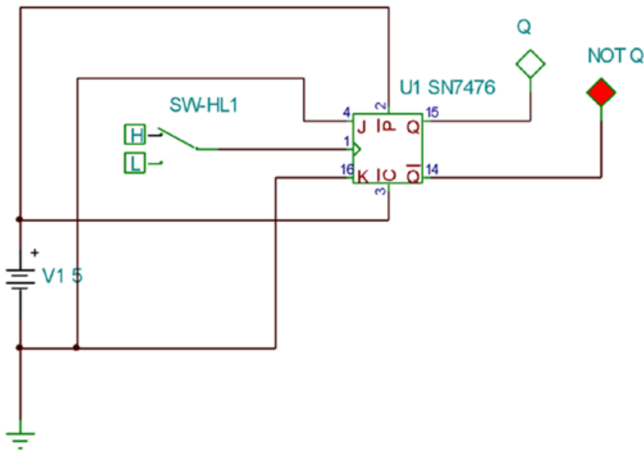


Figure 4-9. *The J and K Inputs at Logic "0"*

With this configuration the flip flop starts off with the Q at a logic "0" and the NOT Q at a logic "1." Toggling the clock, the outputs do not change. Not much use.

Now we will set the "J" to a logic "1" while the "K" is a logic "0." The test circuit for this configuration is shown in Figure 4-10.

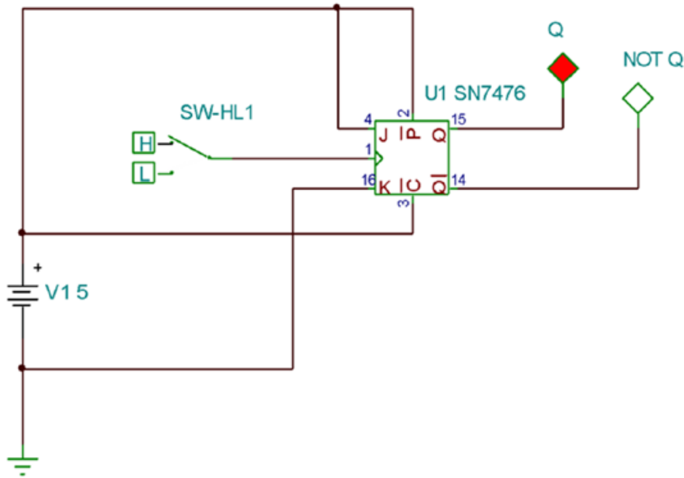


Figure 4-10. *The J at a Logic "1," the K at a Logic "0"*

Initially the Q is a logic "0" and the NOT Q is a logic "1." When the clock input first toggles, the Q goes to a logic "1" and the NOT Q goes to a logic "0." After that, toggling the clock input has no effect on the outputs. Not much use again.

Now we will set the J to a logic "0" and the K to a logic "1." The test circuit for this configuration is shown in Figure 4-11.

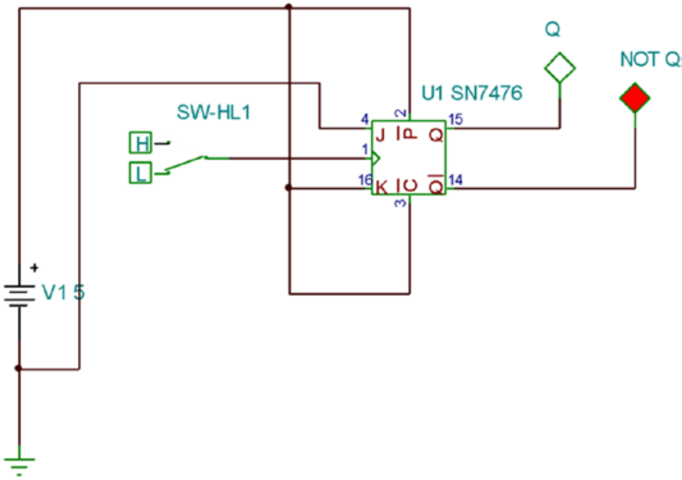


***Figure 4-11.** The J at a Logic "0" and the K at a Logic "1"*

Initially the Q is at a logic "0" and the NOT Q is a logic "1." Toggling the clock input changes nothing at the outputs.

With this next simulation, both the J and K are tied to +5V, that is, a logic "1." The test circuit for this configuration is shown in Figure 4-12.

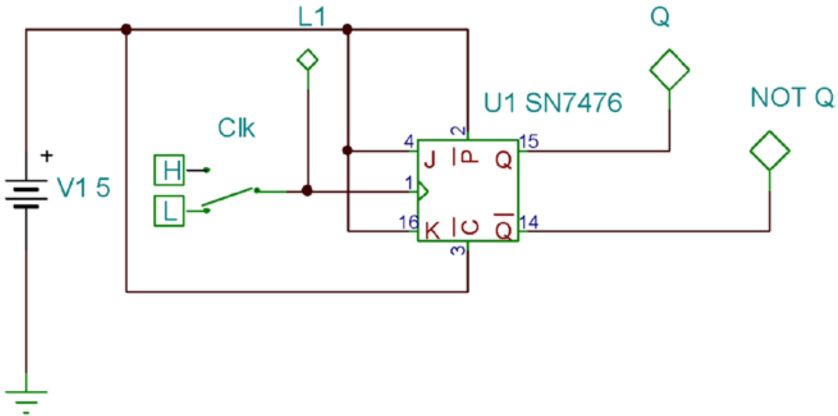


***Figure 4-12.** The Single JK Flip Flop with J and K Tied to Logic "1"*

If we go carefully through the sequence of the switch, we should be able to complete the truth table as shown in Table 4-8.

Table 4-8. *The Truth Table for the Single JK Flip Flop*

Row	Clk Clock	Q Main Output	$\bar{Q}$ Secondary Output
1	0	0	1
2	1	0	1
3	0	1	0
4	1	1	0
5	0	0	1
6	1	0	1
7	0	1	0
8	1	1	0
9	0	0	1

The first thing we can see is that when the clock first went high, the "Q" output stayed low. This may seem to be in contradiction to the symbol, which would seem to suggest the clock input is active high. However, what is happening is that the first logic "1" of the clock is the input to the master clocked SR of the JK. There is a not gate between the master and the slave, which means the clock input to the slave is now a logic "0." Now when the clock at the master goes low, the slave sees the clock go high, and so the Q output can respond. This concept confirms my idea of how the JK flip flop is constructed from the master-slave clocked SR latch as shown in Figure 4-6. Also, the data sheet for the SN7476 does describe it as having a master and a slave.

The next thing we can see from the truth table is that the "Q" stays low for rows 1 and 2 while the clock changes from logic "0" to logic "1." The Q output also stays high for rows 3 and 4. This would suggest that the Q output changes at half the rate of the clock input. This action is repeated over rows 5, 6, 7, and 8. Indeed with the J and K both tied to a logic "1," the flip flop divides the clock by 2. As you toggle the clock input slowly, you should be able to detect that for the Q to go through one cycle, the clock goes through two.

We can confirm this divide by 2 action of the JK by simulating the test circuit as shown in Figure 4-13.

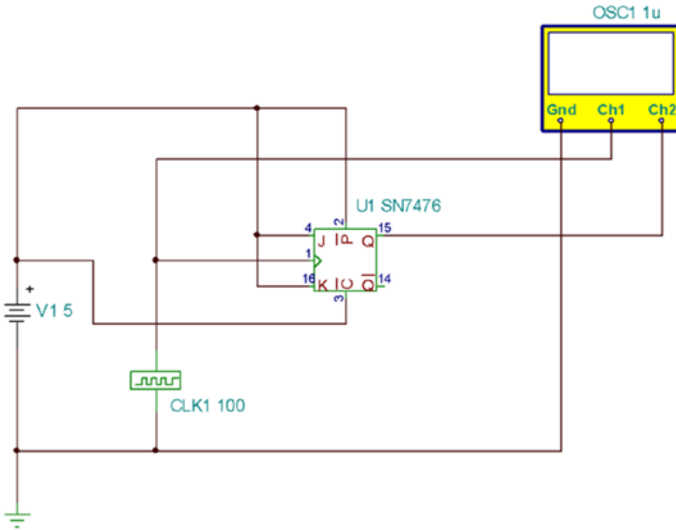


Figure 4-13. *The Test Circuit for the Divide by 2 Action of the JK Flip Flop*

With this circuit we are using an oscilloscope to compare the frequency of the input square wave, measured by channel 1 of the oscilloscope, with the output, measured by channel 2. The oscilloscope is obtained from the "Meters" tab on the main menu. We will be using the three-terminal oscilloscope and connecting it as shown in Figure 4-13. To open up the

oscilloscope and set it up, we need to click the mouse on the "T&M" option from the top menu. A drop-down menu should appear, and we need to select the oscilloscope option from that drop-down menu. The frequency of the input is 100Hz, and so we should set up the oscilloscope as shown in Figure 4-14.

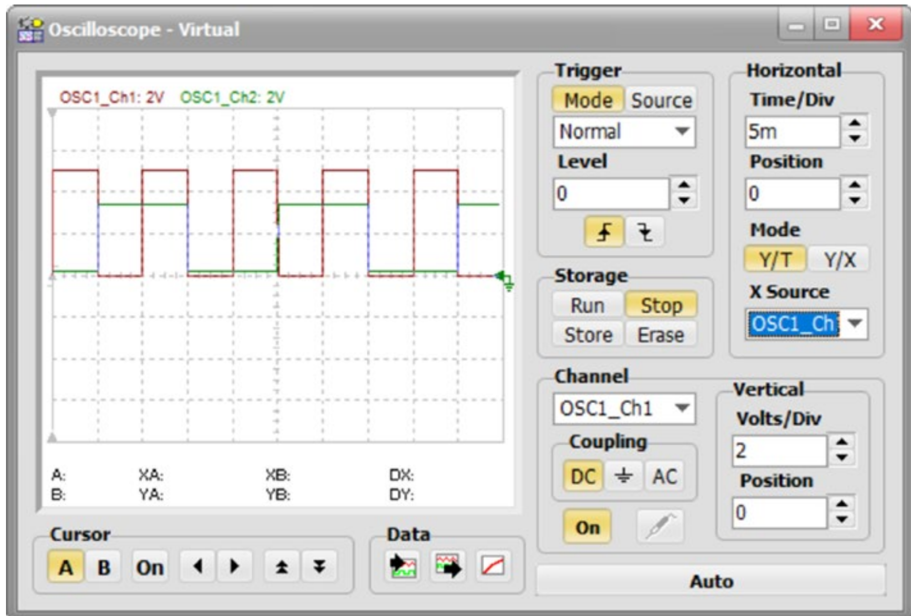


Figure 4-14. *The Oscilloscope in TINA*

We must set the "Mode" to Normal and the source for the trigger, which synchronizes the start of the display, to OSC1_Ch1. We can leave the trigger level at 0V and the edge triggering symbol set to the rising edge, which is the default setting. The Time/Div is set to 5ms so that the whole screen would measure 50ms, as there are ten divisions along the horizontal of the display. This will allow us to display five cycles of the input waveform, as at 100Hz the periodic time, that is, the time for one full cycle, is 10ms. We should set the Volts/Div for both channels to 2V per division.

To start the circuit and so display the waveforms, we simply need to click Run on the oscilloscope. Once we see some waveforms displayed on the oscilloscope, we should click the Stop button to stop the display and so freeze it on the screen. By clicking the Stop button, the Data option at the bottom of the screen should become available. By clicking the second black arrow, the "Export Curves" option, we should be able to produce the display of the two waveforms as shown in Figure 4-15.

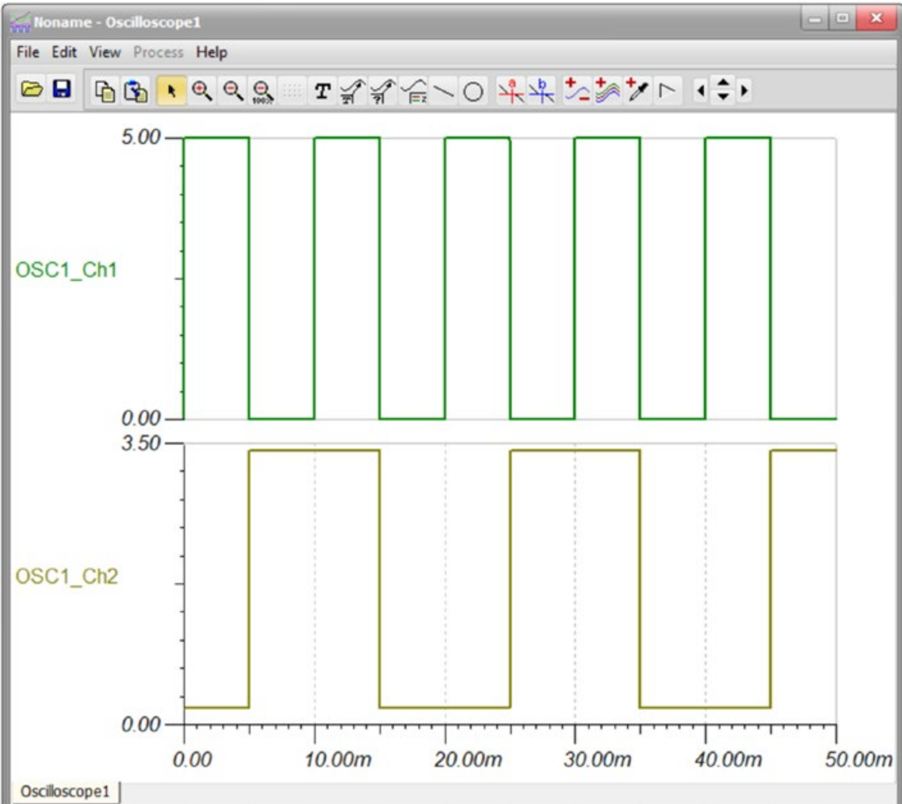


Figure 4-15. *The Two Waveforms from the Test Simulation*

The top trace, OSC1_Ch1, shows the input to the JK flip flop. The bottom trace, "OSC1_Ch2," displays the output. I hope it is clear that the top waveform is twice the frequency of the bottom trace. This does confirm that the single JK flip flop does divide by 2.

We can extend this concept of dividing by 2 by adding a second and a third JK flip flop with both the J and K inputs connected to a logic "1." The circuit for this arrangement is shown in Figure 4-16. The Q output from the first JK becomes the clock input to the second JK. Then the Q output of the second JK becomes the clock input to the third JK.

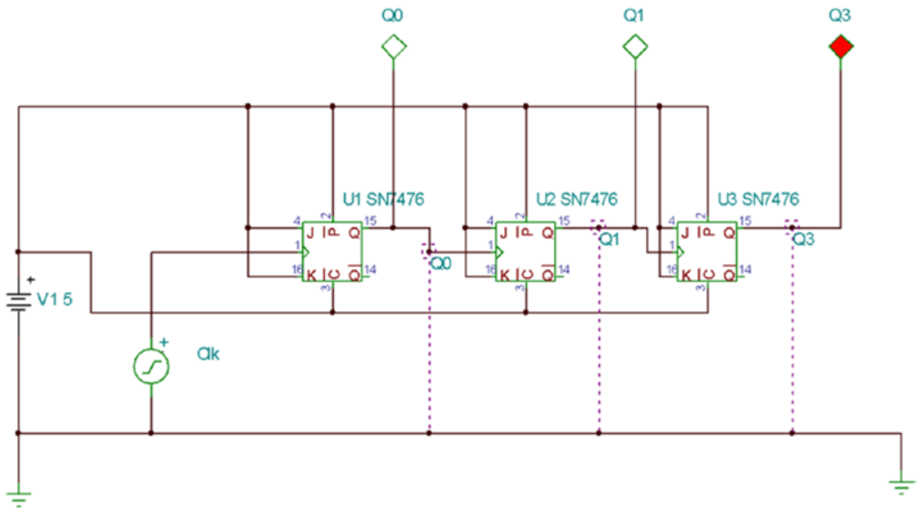


Figure 4-16. *Three JKs Cascaded Together*

Within TINA, there is an option to carry out a transient analysis. This analysis allows us to look at the response of the circuit for a short time from time zero. I have included three output probes, Q0, Q1, and Q2, which will allow us to examine the waveforms from the output Qs of the three JKs. These waveform traces are shown in Figure 4-17. If we use the waveform Q0 as the reference, then I hope you can see that the Q1 output is at half

the frequency of Q0, that is, it divides Q0 by 2, and that the Q2 output has divided the Q0 by 4. If we consider the binary number system as described in Chapter 1, I hope you can appreciate why I have added the description of the columns at the side of the waveforms. I am trying to show you that the Q0 displays the number of 1s, while Q1 displays the number of 2s and Q2 displays the number of 4s. This means that by connecting the JKs in this cascading manner, we can create a binary counter. Indeed, this series of three JKs creates a counter that counts from 0 to 7 but in binary. I have tried to show this concept by displaying the counts 0 to 7 on the x axis of the displays in Figure 4-17.

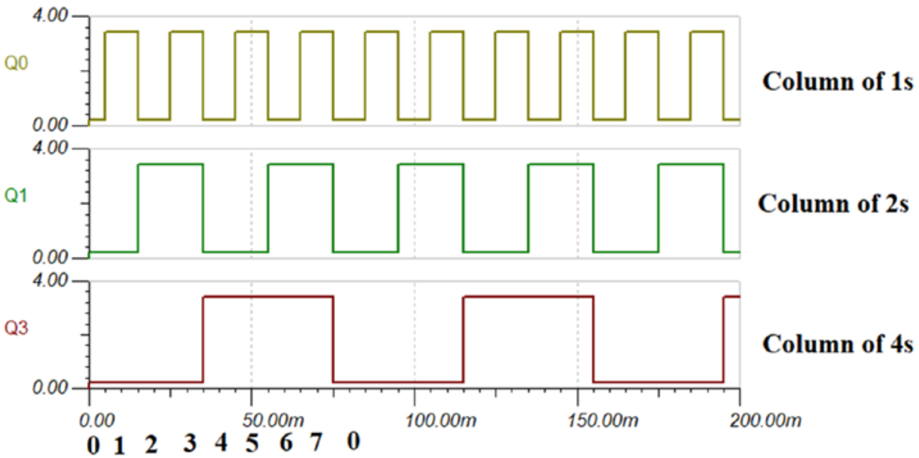


Figure 4-17. *The Timing Waveforms of the Three JKs*

The D-Type Latch

We have seen that if you tie both the J and K inputs to a logic "1," we can make a divide by 2 circuit. Later, in Chapter 5, we will see how we can make more use of the JK configured this way. The next configuration we will look at is that of making the K input the NOT of the J input. This is shown in Figure 4-18.

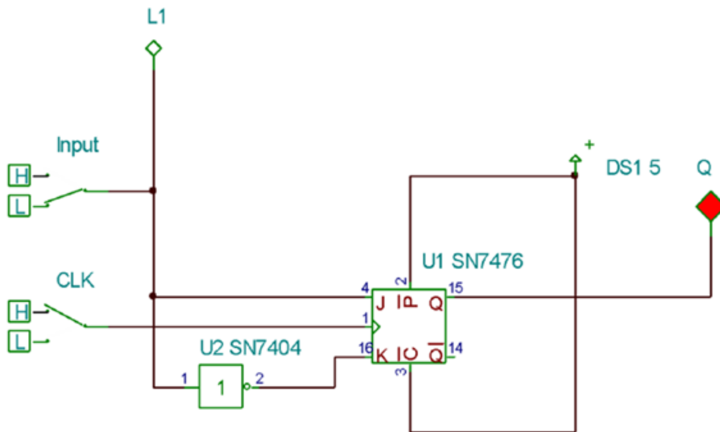


Figure 4-18. *The JK Configured as K the NOT of J*

With this configuration the output "Q" does not take on the logic of the input until the clock goes from high to low. We can see in Figure 4-18 the input is at a logic "0." The output "Q" is at a logic high, but it will eventually change when the clock is changed as follows. The CLK, which is high at present, goes low; the Q does not change. Then the CLK goes back high; the Q does not change. Now send the CLK back low; at this transition the Q does change to a logic "0." This means there is a short delay from when the input changes to when the output changes to reflect that change in the input. We can see this delay when we simulate the circuit shown in Figure 4-18.

This concept of a delay gave rise to the name for this latch, the D-type latch. This is such a useful configuration that we can buy JK flip flops with the K permanently connected as the NOT of the J. We will see in Chapter 6 how we make good use of these D-type latches.

The T Latch

This is the last configuration of the JK flip flop that we will look at. With this configuration the J and K are permanently connected together. This configuration is shown in Figure 4-19.

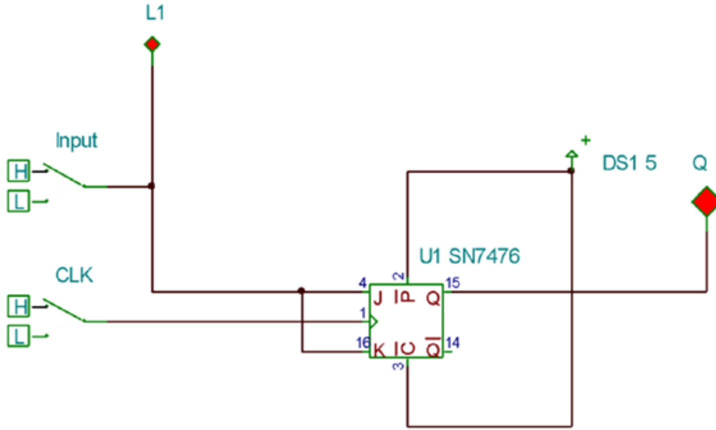


Figure 4-19. *The T Latch*

The input to the T latch can either be a logic “1” or a logic “0.” If we keep this input at a logic “1,” then the output Q will simply toggle between logic “1” and logic “0” every time the clock goes from low to high and back to low. This is really what we expected from the configuration we looked at in Figure 4-13. This toggling action is where the latch got its name from, that is, the T-type latch.

Now if the input is kept at a logic “0,” we see that the output will remain constant, with every transition from high to low of the clock, at the logic it was before the input went low. For example, if the output was at a logic “1,” then we set the input to a logic “0.” The output would remain at a logic “1” for every transition from high to low of the clock until we changed the input to a logic “1” again.

The Main Configurations for the JK Flip Flop

We can use truth tables to sum up the operation of the main configurations of the JK. They are as shown in Tables 4-9 and 4-10.

Table 4-9. *The Truth Table for the JK Flip Flop*

Row	J Input	K Input	Present Q Qn	Next Q Q+ Qn+1	Comment
1	0	0	0	0	No change from present state.
2	0	0	1	1	
3	0	1	0	0	Resets the Q to logic "0."
4	0	1	1	0	
5	1	0	0	1	Sets the Q to a logic "1."
6	1	0	1	1	
7	1	1	0	1	The Q output toggles between
8	1	1	1	0	logic "1" and logic "0."

The JK Flip Flop

With this table I am introducing you to the concept of referring to the next Q as "Q+". Sometimes the present Q can be referred to as "Qn," and therefore the next Q would be referred to as "Qn+1."

We can sum up the truth table to show what configuration of the J and K inputs we need to make the output change the way we want. This is shown in Table 4-10.

***Table 4-10.** The Transition Table for the JK*

Row	J Input	K Input	Present Q Qn	Next Q Q+ Qn+1	Comment
1	0	x	0	0	Always logic "0."
2	1	x	0	1	Change to logic "1."
3	x	0	1	1	Q goes to "1" if not already a 1 and stays at 1.
4	x	1	1	0	If J = 1 then Q goes to 0 and stays at 0.

The small "x" indicates the "don't care" state in that we don't care what logic the input takes on; the JK will react the same if the input is a logic "1" or a logic "0." We will make use of this table in Chapter 5.

We can create a similar transition table for the D-type latch. The table is shown in Table 4-11.

***Table 4-11.** The Transition Table for the D-Type Latch*

D Input	Present Q	Next Q
0	0	0
1	0	1
1	1	1
0	1	0

This table shows how the D-type latch will change from what it was when the clock goes from high to low. The output will remain at the Next Q state until the D input changes.

We can create a similar transition table for the T-type latch. The table is shown in Table 4-12.

Table 4-12. *The Transition Table for the T Latch*

T Input	Present Q	Next Q
0	0	0
1	0	1
0	1	1
1	1	0

This does show that if the T input was a logic "1," the T latch output toggles.

We can also create similar transition tables for the SR latch. There are two versions of how the SR latch can be created, one using NAND gates and the other using NOR gates. With the NAND gate, the latch is active low. With the NOR gate, the latch is active high. The transition tables for both are shown in Tables 4-13 and 4-14.

Table 4-13. *The Transition Table for the NAND SR Latch*

Row	S Input	R Input	Present Q	Next Q
1	0	1	0	1
2	x	1	1	1
3	1	0	1	0
4	1	x	0	0

If the Q is a logic "0" and the S goes to a "0" while the R is a "1," the Q will change to a logic "1." This is the first row. Now while the R is a "1," we don't care what logic there is on the S input; the Q will stay at a logic "1."

This is row 2. Now, row 3 shows us that, if R goes to a logic "0" while the S is a logic "1," the Q will return to a logic "0." In the last row, row 4, we see that as long as the S remains at a logic "1," we don't care what logic the R goes to; the Q output will remain at a logic "0."

***Table 4-14.** The Transition Table for the NOR SR Latch*

S Input	R Input	Present Q	Next Q
0	1	1	0
0	X	0	0
1	0	0	1
X	0	1	0

This works in a similar way but the logics are inverted.

These tables are very similar to each other, and they do require some careful examination when we use them. We will use most of them in Chapter 5.

Summary

In this chapter we have gone on a journey that has taken us from the simple NAND gate to the creation of the major digital device, the JK flip flop. We have then learned how the JK can be configured by connecting the J and K inputs to different logics.

In the next chapter, we are going to investigate different methods for designing logic circuits using the JK in its different configurations.

CHAPTER 5

Design Methods for Digital Circuits

In this chapter we will discuss how engineers can use different methods to design digital circuits. We will look at simply using truth tables and then move on to using state diagrams. One thing we will realize is that there is no easy method and we will have to make use of our knowledge and understanding of logic circuits as well as practice the different methods.

We will also look at the difference between combinational and sequential logic.

Combinational and Sequential Logic

Digital circuits fall into two main areas: those of combinational logic and those of sequential logic. Therefore, before we go any further, it would be useful to explain the difference between them.

Combinational Logic

All logic circuits will have inputs and outputs. With combinational logic the output will depend upon a combination of the states of the inputs to the circuit at that instant in time. Also, combinational logic circuits do not normally have any feedback in them and so no memory aspect to them.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_5

Sequential Logic

With this type of logic circuits, there is usually some sort of feedback within them. The SR latches, looked at in Chapter 4, fed back the Q to the NOT Q gate and the NOT Q to the Q gate; see Figure 4-1 in Chapter 4. This means that the output of a sequential logic circuit depends upon the input at that time and also some action that has gone on before. In this way sequential logic uses some kind of memory of what happened previously and what is happening at present to determine what will happen next.

This chapter will focus on the design of sequential logic circuits. We will look at the design of combinational logic circuits in Chapter 7.

Representing a Digital System

To aid the design of a digital system, and indeed any system, it is normal to create a diagram to represent the system. A digital system will normally have an input, or more than one input, which in general we will identify by the symbol "I." Therefore, for a system with three inputs, we will use I_0, I_1, and I_2 to identify them. The same system will usually produce one or more outputs. These outputs can be identified using the symbol "F" or "Y," but I will use "F." Therefore, if a system had three outputs, we would identify them as F_0, F_1, and F_2.

In some systems the outputs are as a direct response to the inputs. However, some systems create internal conditions that can be used in conjunction with the inputs "I," to create the conditions that are required to produce the desired outputs. These internal conditions can be represented by the symbol "Q." For example, a system with three internal conditions could represent them as Q_0, Q_1, and Q_2. When this happens, the system will combine its own internal logic conditions and the inputs to create the conditions to drive the system. Some systems will also use some logic that uses these internal conditions to create the conditions to

produce the outputs "F." These sequential systems can be represented with a block diagram as shown in Figure 5-1.

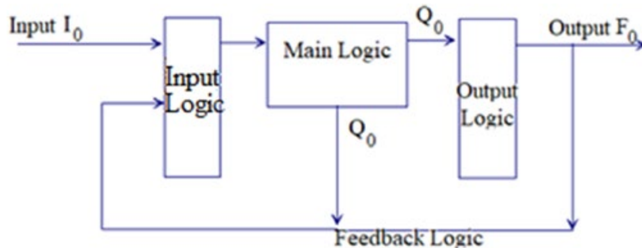


Figure 5-1. *A Simple Block Diagram of a Logic System*

The input or inputs to the main logic are obtained from the input logic. The inputs to the input logic come directly from the inputs to the system, shown as I_0 in Figure 5-1, and from the internal feedback logic shown as either Q_0 or indeed F_0. This internal feedback logic can come from the output or from the main logic circuitry. Not all sequential logic systems will require the output logic, but most systems will require the input logic. With the "ripple counter," which we will look at next, this internal input logic is only one NAND gate, a two-input NAND gate, used with the modulo 10 counter. In other circuits we will look at in this chapter, the input logic may be more complex, and there may be some output logic as well.

Asynchronous and Synchronous Logic Systems

An asynchronous system is a system whose operation is not synchronized to a clock. The devises used in the system are free-running as they don't have a clock signal synchronizing when they operate with each other.

A synchronous system is one in which the devices are all synchronized to operate with each other using a clock signal that is fed to them all.

The Ripple Counter

This is an asynchronous system as not all the JKs will be synchronized to a clock. In this example we will cascade four JKs together, with the first one connecting to the input signal that the circuit will count. Normally this input signal will come from a clock. So that we can examine what happens to each JK at each change of the clock, we will use a switch that we can toggle ourselves. This configuration will create a binary counter that will count the transitions of the input from low to high and back to low. The count will go from 0 to 15. The input at the first JK starts the action, but then the Q output becomes the input of the next JK, and a ripple action is set up that runs throughout the counter, hence the name of the counter. The circuit is shown in Figure 5-2. As the clock only goes to one JK, this is an asynchronous system.

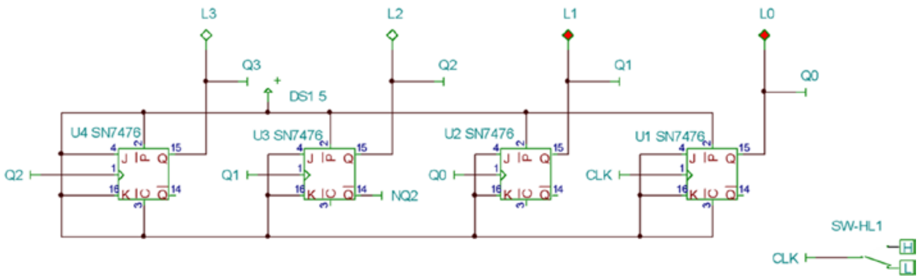


Figure 5-2. *The Ripple Counter*

The circuit shown in Figure 5-2 is that of the basic ripple counter. In this circuit both J and K inputs, of all the JKs, are tied to a logic "1." We have seen, with the feedback in the master-slave clocked SR shown in Figure 4-6 in Chapter 4, this action made the Q output toggle with the clock input. So the JK would simply count input pulses. In this circuit four JKs are cascaded together to form a 4-bit binary counter, one JK for each bit.

With this basic counter, all four $\overline{P}$ and $\overline{C}$ inputs are tied to VCC or logic "1." This prevents them from setting or resetting the Q outputs of the JKs.

The switch is used to simulate the input to the circuit. This circuit will count from 0 to 15 in binary, as $2^4 = 16$, that is, 0 to 15. After 15, the count will return to 0 and the count starts again. When we simulate the circuit, we see that the counter works as expected.

I have arranged the JKs to try and mimic how we write down numbers, that is, with the LSB on the right going to the MSB on the left. That is why the Q_0, the LSB, is on the right and the Q_3, the MSB, is on the left. Also, to save filling the schematic with wires that might clutter up the circuit diagram, I am using the "jumpers" that are available in TINA. The software will automatically connect any jumpers that have the same name. See the CLK and the Q0, Q1, and Q2 jumpers. The CLK signal from the switch SW-HL1 is connected to the first JK on the left of the circuit. The output of that JK becomes the clock input, on pin 1, of the next JK. This is repeated throughout the counter. I hope you can now follow the circuit shown in Figure 5-2.

Design Example 1: The Modulo 10 Counter

The basic ripple counter can be configured as a modulo 10 counter, with the addition of a simple NAND gate. The concept of the design is that the counter will count from 0 to 9. However, as soon as the counter tries to display 10, which is 1010 in binary going from right to left, the counter must reset back to 0, which is 0000, all 4 bits reset back to 0. Note it is Q1 and Q3, going from right to left, that will go to a logic 1 at the same time when the counter tries to display 1010, that is, 10 in binary, going from right to left. If these two Q outputs are fed into a two-input NAND gate, then only when both inputs are at a logic "1," which they will be when the counter tries to display 10 in binary, will the output of the NAND gate go to a logic "0." At all other times, the output will be a logic "1; see the truth table for the two-input NAND gate shown in Table 2-1 in Chapter 2. Therefore, if the output of this NAND gate is fed to the Clear or Reset

input of all four JKs, then as soon as the counter tries to display 1010 from right to left, the logic on all four Clear inputs will go to a logic "0," and the counter will reset to 0000. However, this will happen instantly, and you will not be able to see the logic on the Clear inputs change; all you will see is the counter reset to 0. This idea of the count of 10 being 1010 in binary is shown in Table 5-1. Note the table shows the Q0 to Q3 going from right to left, as this is the standard way of writing numbers.

Table 5-1. *The 4-Bit Binary Counter Output*

Decimal Count	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10 A	**1**	**0**	**1**	**0**
11 B	1	0	1	1
12 C	1	1	0	0
13 D	1	1	0	1
14 E	1	1	1	0
15 F	1	1	1	1

Table 5-1 shows how the count of the ripple counter changes. The value of 10 is in bold as this is when the counter should reset back to 0. The last six values, from 10 to 15, have the first six letters of the alphabet that are used in the hexadecimal numbering system to indicate those values. You should appreciate that all numbers are written in columns and each column can only show one digit to represent the value.

The modulo 10 ripple counter is shown in Figure 5-3.

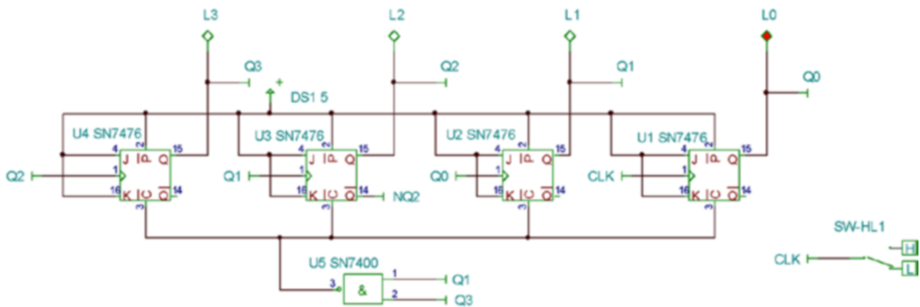


Figure 5-3. *The Modulo 10 Ripple Counter*

If you operate the clock input carefully, you should see that the counter does count up from 0 to 9. Then when it tries to display 10, the output of the small NAND gate "U5" changes from a logic "1" to a logic "0." You may need to consider the truth table of the two-input NAND gate; see Table 2-1 in Chapter 2. As the Clear inputs to all four JKs are fed from the output of this NAND gate, then, because the Clear input is active low, all four JKs clear their outputs and so they all go to logic "0." This action is almost instantaneous and so you will not see it.

Design Example 2: A Non-sequential Output

This will use a collection of JK flip flops arranged as an asynchronous circuit to create the following logic at their output: 000, 010, 001, 011, 100. The outputs would change on the falling edge of a controlling input I_0. We will use the truth table approach to design this circuit, and the truth table is shown in Table 5-2.

Table 5-2. *The Sequence of Output Changes*

Row	Q_2	Q_1	Q_0	Change
1	0	0	0	a
2	0	1	0	b
3	0	0	1	c
4	0	1	1	d
5	1	0	0	e
6	0	0	0	Back to a

The system will change the conditions from "a" to "b" and so on every time the controlling input goes high, that is, to a logic "1." This has not been shown in Table 5-2, but we need to include it in the expression we derive for the drive circuitry. We will use the table to identify when each output is triggered into changing state and what action, that is, change or event, has led to that change.

If we look at the output "Q_0," we can see that it changes state at "b" to "c," in row 2, and "d" to "e," in row 4. The action that brings about this change is "Q_1" as it is at a logic "1" for both changes. We can also see that Q_2 does not undergo any action; it is a logic "0" for both changes in Q_0. This suggests that we can trigger the "Q_0" JK from the "Q_1." We should associate this Q_0 output with the input I_0, because all changes happen when the

input I_0 goes high. Therefore, the Q_0 trigger depends upon the Q_1 and the I_0 being high, that is, at a logic "1." Therefore, the expression for the trigger to change Q_0 is $Q_0 = I_0. Q_1$.

We can see from Table 5-2 that Q_1 changes with changes a, b, c, and d but not with "e." The changes a, b, c, and d change when the input is high, but the extra condition that differentiates them from change "e" is that the output Q_2 is a logic "0." Therefore, we can say that Q_1 changes when the input I_0 is at a logic "1" and with Q_2 at a logic "0." This means we can create an expression for the trigger to change Q_1 as $Q_1 = I_0.\overline{Q_2}$.

Finally, we can see that Q_2 changes on "d" to "e," row 4, and "e" to "a," row 5. The change from "d" to "e" is identified as Q_0 and Q_1 being at a logic "1" with the input I_0 at a logic "1," while the change of "e" to "a" is Q_2 being high with the input I_0 being high. This means there are two expressions that can bring about the trigger to change Q_2 and they produce the expression $Q_2 = I_0. Q_0. Q_1 + I_0. Q_2$.

We now have the expression for the three outputs, and they are

$$Q_0 = I_0.Q_1$$

$$Q_1 = I_0.\overline{Q_2}$$

$$Q_2 = I_0.Q_0.Q_1 + I_0.Q_2$$

It is the three JKs that control when the Qs change. This is done when the inputs on pin 1 of each of the JKs are triggered. Therefore, we can use the three expressions to provide expressions for the three triggers, Tr0, Tr1, and Tr2, on the JKs. The expressions are

$$Tr_0 = I_0.Q_1$$

$$Tr_1 = I_0.\overline{Q_2}$$

$$Tr_2 = I_0.Q_0.Q_1 + I_0.Q_2$$

It is with these three expressions we can now design the drive circuit for our non-sequential logic system. The circuit will use three T-type latches, that is, latches that have their J and K inputs tied together and then both tied to +5V, as the Q output will simply toggle when triggered. The changing of the outputs of the JKs will be controlled by controlling when their respective inputs on pin 1 go low to high and back to low. It is when the logic on pin 1 of the JKs goes from high to low that, due to the NOT gate between the master and the slave in the JK, the output Q will change. This means that really the three expressions we have derived from Table 5-2 are to determine when the pin 1 of each JK will go to a logic "1," not when the Qs at the output go to a logic "1." This concept makes this design method using JKs a little more difficult to grasp. However, once you have grasped it, the method can be used.

The completed circuit is shown in Figure 5-4.

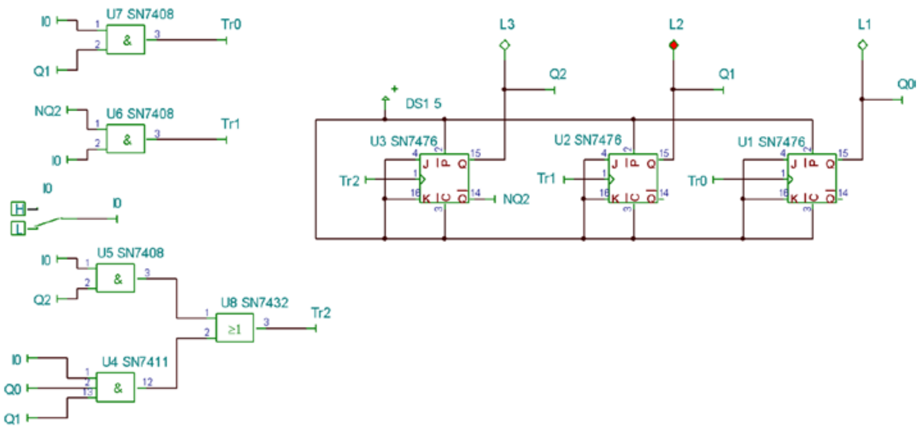


Figure 5-4. *The Non-sequential Logic System*

The logic gates U4, U5, U6, U7, and U8 make up the drive logic that implements the three expressions derived from the truth table. The input I_0 is shown as a switch, but it could easily be a sensor that goes from high to low at the required event. If we simulate the circuit, we should see that the outputs Q_0 to Q_2 change as required. This is just an example to try and

show you how you can represent a logic system with the block diagram shown in Figure 5-1 and that you can use a truth table to derive the expressions for the drive logic.

Design Example 3: A Synchronized Sequential Circuit

In the previous designs, the JKs were free-running as their triggering, or clock input, on pin 1 of the JK was fed from different sources. In this next example, we will synchronize the operation of the JKs by connecting their trigger or clock inputs to the same source. The circuit will display a non-sequential display similar to Example 2 except that the display will only change when the input source changes from high to low on all the JKs. The sequence the JKs will display is 000, 001, 011, 110, 100, and then back to 000.

The sequence will repeat until the circuit is switched off. This will be an example of an autonomous sequential circuit as there are no real inputs as the source is only used to synchronize when the display changes. This means there is no requirement for any drive logic. The signals that will feed the J and K inputs, to configure the JKs, actually come from the internal feedback obtained from the JKs themselves. There will be no output logic as the Q outputs of the JKs will provide the display.

The design process will use a table that shows the present state and the next state of the outputs as the sequence progresses. To try and arrive at the minimum solution for the circuit, we will use Karnaugh maps to determine how we can use the internal feedback to drive the J and K inputs of the JKs. Using the sequence of display, we can construct the present state and next state table. This is shown in Table 5-3.

***Table 5-3.** The Present State and Next State Table for Example 3*

Row	Present State			Next State		
	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
1	0	0	0	0	0	1
2	0	0	1	0	1	1
3	0	1	1	1	1	0
4	1	1	0	1	0	0
5	1	0	0	0	0	0

I feel I must warn you that this analysis of this method is quite long and it may need reading through carefully more than once. However, once you have grasped the concept, I am sure you will be able to use it in your own designs; be patient.

We can see that the "next state" of the current change becomes the "present state" of the outputs at the next change, that is, the next row in the table. We will see in Chapter 6, when we look at state diagrams, this is not always what happens. Also, the "next state" of the final change, at row 5, will bring the display back to the first state of the output Qs, at row 1.

To understand how this design process works, we must appreciate that it is the "present state" of the output Qs that will set up the circuit to change, as described, when the synchronizing signal goes from high to low. Also, it is the J and K inputs of the JKs that configure the JKs themselves, to ensure the outputs change as expected. In that way we need to determine how the J and K inputs are connected to the internal feedback signals. We will look at each of the J and K inputs for the JKs one at a time. We will start with the first JK and identify the inputs as J_0 and K_0. The first JK will display the output Q_0, which means we only need to consider how this output, Q_0, changes. Examination of Table 5-3 shows us how Q_0 changes, that is, when it goes from 0 to 1 or 1 to 0, or shows no change in that it goes 0 to 0 or 1 to 1. Table 5-4 will show us how to configure the settings needed for J_0 and K_0

for each of these types of changes on Q_0. This is an attempt to make use of Table 4-10, the transition table for the JK, from Chapter 4. To help with that, I have repeated that table here as Table 5-5.

Table 5-4. *The Change in Logic for the First Flip Flop JK_0*

Change	Q_0 Logic	J_0 Logic	K_0 Logic
1	0 to 1; see row 2 of Table 5-5.	1	x
2	1 to 1; see row 3 of Table 5-5.	x	0
3	1 to 0; see row 4 of Table 5-5.	x	1
4	0 to 0; see row 1 of table 5-5.	0	x
5	0 to 0; see row 1 of table 5-5.	0	x

Table 5-5. *The Transition Table for the JK from Chapter 4*

Row	J Input	K Input	Present Q Qn	Next Q Q+ Qn+1	Comment
1	0	x	0	0	Always logic "0."
2	1	x	0	1	Change to logic "1."
3	x	0	1	1	Q goes to "1" if not already a 1 and stays at 1.
4	x	1	1	0	If J = 1 then Q goes to 0 and stays at 0.

The changes in the output Q_0 are derived from Table 5-3, that is, from the present state to the next state in each of the rows of Table 5-3. The logic for the J and K that is required to bring about these changes in Q_0 is shown in Table 5-4.

The first change for Q_0 is to go from a logic "0" to a logic "1"; see row 1 of Table 5-3. The logic on the J_0 and K_0 inputs to bring this change about is stated in row 2 of Table 5-5, that is, J_0 is a logic "1" and K_0 is a logic "x," a don't care. That is why the logic of the J_0 and K_0 inputs is as stated in Table 5-4, row 1.

The next change for Q_0 is logic "1" to logic "1," that is, no change as shown in row 2 of Table 5-3. Now we must look for the 1 to 1 change, although it's not a change, in Table 5-5. This is in row 3 of Table 5-5. In row 3 of Table 5-5, we see that the J_0 needs to be a logic "x" and the K_0 needs to be a logic "0." This is reflected in row 2 of Table 5-4.

We now simply use the same approach to arrive at the remaining settings for J_0 and K_0 as shown in rows 3–5 of Table 5-4.

Now we can draw up the Karnaugh map to see if the expressions we will eventually arrive at can be simplified. As the circuit will use three flip flops, one for each Q, the map will have four columns and two rows. The empty map is shown in Figure 5-5.

Q_0 / Q_1 / Q_2	0 0	0 1	1 1	1 0
0				
1				

Figure 5-5. *The Empty Karnaugh Map*

The labeling of the rows and columns on the map reflects all the possible binary values the three output Qs, Q_2, Q_1, and Q_0, reading from right to left, can normally take up, so the labeling of the Qs on the map also reads from right to left. We have labeled the Qs in this way because we

write the required outputs in binary reading from right to left. There will be eight values in total that the three Qs can take up. So we need to know how these values would normally change. These are shown in Table 5-6.

Table 5-6. *The Normal Sequence of Three Binary Digits*

Row	Q_2	Q_1	Q_0
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

However, the circuit will only display five values. This means that three of the values in Table 5-6 will not be displayed, with this circuit, in this example. Those values are shown in rows 3, 6, and 8, that is, 010, 101, and 111. The values that will be displayed are in rows 1, 2, 4, 7, and 5, in that order. The fact that three values are not displayed means that for those values it does not matter what logic the Js and Ks of all three JKs take up. We can indicate this concept that the Js and Ks don't care by putting an "X" in the cells that correspond to those values on the map. I will use the uppercase "X" to distinguish these don't cares from the don't cares derived from the JK transition table shown in Table 5-4. With respect to those don't cares, we will use the lowercase "x."

Therefore, using the don't cares from these three values, we will start with a partially completed map, for all the JKs, as shown in Figure 5-6.

Q_2 \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0		X		
1			X	X

Figure 5-6. *The Partially Completed Karnaugh Map*

Now we will complete the map by adding the response of the Js and Ks for each of the JKs, starting with the first JK. We will start by completing the map for the J_0 input.

The first change is shown as row 1 in Table 5-4. This shows that the J must be a "1" while the K would be a don't care, that is, it could be a "1" or a "0." It does not matter; the response would be the same.

If we now look at Table 5-3, we can see that the state of all the Qs in the circuit that will create this change to the next state, when Q_0 goes from 0 to 1, is the present state shown in row 1 of Table 5-3. This is when all Qs were at a logic "0." Therefore, on the J_0 map, we must put a "1" in cell 1, which corresponds to 000 in all three Qs. This is because row 1 in Table 5-4 shows us that for the JK to change from a 0 to a 1, the J_0 should be a 1 and the K_0 is an "x," a don't care. Figure 5-7 shows the "1" entered on the map in cell 1, that is, at $Q_2 = 0$, $Q_1 = 0$, and Q0 = 0.

Q_2 \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0	**1**	X		
1			X	X

Figure 5-7. *The "1" Placed on the Map for the J_0 Configuration Pin*

Now if we look at row 2 in Table 5-4, we can see that the change for Q_0 would be 1 to 1. This would mean that the J would be a don't care "x." From Table 5-3, row 2, we can see that the present state of the Qs that would create this change would be 001. This corresponds to cell 4 on the map. Therefore, putting a lowercase "x" on the map will give the map as shown in Figure 5-8.

Q_0 / Q_1 \ Q_2	0 0	0 1	1 1	1 0
0	**1**	**X**		**x**
1			**X**	**X**

Figure 5-8. *The First Lowercase "x" in Cell 4*

The next change is a 1 to 0 as shown in row 3 in Table 5-4. The J_0 is shown as a don't care "x" in row 3, and the present state of the Qs that creates that change is 011 as shown in row 3 of Table 5-3. This would correspond to cell 3 on the map. We should place an "x" in that cell as shown in Figure 5-9.

Q_0 / Q_1 \ Q_2	0 0	0 1	1 1	1 0
0	**1**	**X**	**x**	**x**
1			**X**	**X**

Figure 5-9. *The "x" in Cell 3*

The next change is row 4 in Table 5-4, which is 0 to 0 requiring J_0 to be a logic "0." Row 4 in Table 5-3 shows that the present state of the Qs to bring about this change is 110. This corresponds to cell 6 on the map, in which we should place a "0." This is shown in Figure 5-10.

Q_0 / Q_1 \ Q_2	0 0	0 1	1 1	1 0
0	**1**	**X**	**x**	**x**
1		**0**	**X**	**X**

Figure 5-10. *The "0" in Cell 6*

The last change is shown as row 5 in Table 5-4. The change is the same as the last, that is, 0 to 0, which requires the J_0 to be a "0." The state of the Qs that brings about this change is 100 as shown as the present state in row 5 of Table 5-3. This corresponds to cell 5 on the map. Therefore, putting a "0" in that cell produces the map as shown in Figure 5-11.

Q_2 \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0	**1**	**X**	**x**	**x**
1	**0**	**0**	**X**	**X**

Figure 5-11. *The "0" in Cell 5*

We can now use this map to hopefully produce the minimal solution for the J_0 of the first flip flop. We will use the 1st canonical format to do this, which will group the minterms, that is, all the 1s on the map, to form the biggest loop. It may at first glance look as though there is only one "1" on the map. However, we should remember that as far as the JKs are concerned, all the don't cares, no matter where they have come from, can be read as either a "1" or a "0." Therefore, as we read the don't cares in cells 2, 3, and 4 as a "1," we can group cells 1, 2, 3, and 4 together to form the biggest loop as shown in Figure 5-12.

Q_2 \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0	**1**	**X**	**x**	**x**
1	**0**	**0**	**X**	**X**

Figure 5-12. *The Four Cells Looped as Loop 1*

The term that is constant in that loop is $\overline{Q_2}$. This means that the expression for J_0 is

$$J_0 = \overline{Q_2}$$

This means that we need to connect the J input of the first JK to the NOT Q output of the last JK. We will see this in the complete circuit shown in Figure 5-22.

Following a similar process for the K_0, we will create an expression for K_0. We will start with the map showing the normal binary count and so the don't cares from Table 5-5. This is shown in Figure 5-13.

Q_2 \ Q_0 Q_1	0 0	0 1	1 1	1 0
0		X		
1			X	X

Figure 5-13. *The Initial Karnaugh Map for the K_0 Input*

Now we will complete the map for the K_0 input.

The first change is shown as row 1 in Table 5-4. This shows that the J must be a "1" while the K would be a don't care, that is, it could be a "1" or a "0." It does not matter; the response would be the same.

If we now look at Table 5-3, we can see that the state of all the Qs in the circuit that will create this change to the next state when Q_0 goes from 0 to 1 is the present state shown in row 1 of Table 5-4. This is when all Qs were at a logic "0." Therefore, on the K_0 map, we must put a don't care "x" in cell 1, which corresponds to 000 in all three Qs. This is shown in Figure 5-14.

Q_2 \ Q_0 Q_1	0 0	0 1	1 1	1 0
0	x	X		
1			X	X

Figure 5-14. *The Karnaugh Map for the K_0 Configuration Pin*

Now if we look at row 2 in Table 5-4, we can see that the change for Q_0 would be 1 to 1. This would mean that the K would be a "0." From Table 5-3, row 2, we can see that the present state of the Qs that would create this change would be 001. This corresponds to cell 4 on the map. Therefore, putting a "0" on the map will give the map shown in Figure 5-15.

Q_2 \ Q_0 Q_1	0 0	0 1	1 1	1 0
0	x	X		**0**
1			X	X

Figure 5-15. *The Karnaugh Map for the K_0 Configuration Pin*

The next change is a 1 to 0 as shown in row 3 in Table 5-4. The K_0 is shown as a "1" in row 3, and the present state of the Qs that creates that change is 011 as shown in row 3 of Table 5-3. This would correspond to cell 3 on the map. We should place a "1" in that cell as shown in Figure 5-16.

Q_0 / Q_1 / Q_2	0 0	0 1	1 1	1 0
0	x	X	1	0
1			X	X

Figure 5-16. *The Karnaugh Map for the K_0 Configuration Pin*

The next change is row 4 in Table 5-4, which is 0 to 0 requiring K_0 to be a don't care "x." Row 4 in Table 5-3 shows that the present state of the Qs to bring about this change is 110. This corresponds to cell 6 on the map, in which we should place an "x." This is shown in Figure 5-17.

Q_0 / Q_1 / Q_2	0 0	0 1	1 1	1 0
0	x	X	1	0
1		x	X	X

Figure 5-17. *The Karnaugh Map for the K_0 Configuration Pin*

The last change is shown as row 5 in Table 5-4. The change is the same as the last, that is, 0 to 0, which requires the K_0 to be a don't care "x." The state of the Qs that brings about this change is 100 as shown as the present state in row 5 of Table 5-3. This corresponds to cell 5 on the map. Therefore, putting an "x" in that cell produces the map as shown in Figure 5-18.

Q_0 / Q_1 / Q_2	0 0	0 1	1 1	1 0
0	x	X	1	0
1	x	x	X	X

Figure 5-18. *The Karnaugh Map for the K_0 Configuration Pin*

We now need to place the biggest loop, or loops, on the map. Knowing that all the don't cares X and x can be read as a "1" means the map with the loop on is as shown in Figure 5-19.

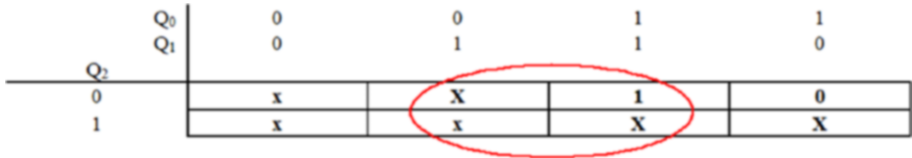


Figure 5-19. *The Karnaugh Map for the K_0 Configuration Pin*

Using this map, we can see there is only one loop, and the variable that does not change its values in the loop is Q_1, which is a logic "1" in all four cells of the loop. Therefore, the expression for K_0 is

$$K_0 = Q_1$$

This means we must connect the K input of the first JK to the Q_1 output of the second JK.

If we carry on to create the maps for the second JK using the Q_1 conditions and the last JK using the Q_2 conditions, we should get the maps as shown in Figures 5-20, 5-21, and 5-22. However, to make it easier to see what logic we need for the Js and Ks, we will create a table that shows all we need. The table for J_1 and K_1 for this circuit is shown in Table 5-7.

Table 5-7. *The Full Table to Show the Logic for J_1 and K_1*

Present State			Row	Next State			Transition	J	K
Q_2	Q_1	Q_0		Q_2	Q_1	Q_0			
0	0	0	1	0	0	1	0–0	0	x
0	0	1	2	0	1	1	0–1	1	x
0	1	1	3	1	1	0	1–1	x	0
1	1	0	4	1	0	0	1–0	x	1
1	0	0	5	0	0	0	0–0	0	x

Using Table 5-7, we can see that the Karnaugh map for J_1 is as shown in Figure 5-20.

Q_0 / Q_1 / Q_2	0 0	0 1	1 1	1 0
0	**0**	**X**	**x**	**1**
1	**0**	**x**	**X**	**X**

Figure 5-20. *The Completed Karnaugh Map for J_1*

We can see from the loop that the only output that does not change is Q_0, which is a logic "1" for all cells in the loop. Therefore, the expression for J_1 is

$$J_1 = Q_0$$

This means we must connect the J input of the second JK flip flop directly to the Q output of the first JK.

We can use Table 5-7 to complete the Karnaugh map for K_1 as shown in Figure 5-21.

Q_0 / Q_1 / Q_2	0 0	0 1	1 1	1 0
0	**x**	**X**	**0**	**x**
1	**x**	**1**	**X**	**X**

Figure 5-21. *The Completed Map for the K1 Input*

We can see from the loop that the only output that does not change is Q_0, which is a logic "0" for all cells in the loop. Therefore, the expression for K_1 is

$$K_1 = \overline{Q_0}$$

This means we should be able to connect the K_1 input to the $\overline{Q}$ output of the first JK.

Exercise 5.1

As an exercise you should create and use a table for the J_2 and K_2 inputs, similar to Table 5-7, and show that the expressions for J_2 and K_2 are

$$J_2 = Q_1$$

$$K_2 = \overline{Q_1}$$

Having completed Exercise 5.1, we should be able to create the circuit for Example 3, and this is shown in Figure 5-22.

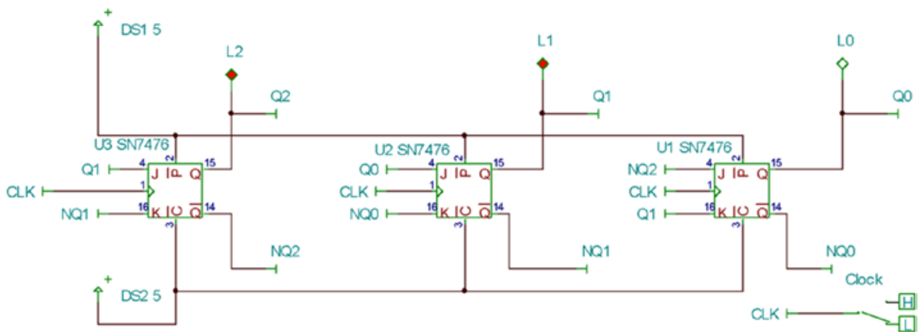


Figure 5-22. *The Working Circuit for Example 3*

We can simulate this circuit by changing the clock input from low to high and back to low. We can see that, apart from the first toggle of the clock input, the state of the Qs does change every time the clock signal goes low. The sequence does follow the sequence as specified.

Design Example 4: A Synchronized Up Counter

Now that we have been introduced to the design of a synchronized logic circuit using JK flip flops, we will move on to one of the more useful examples of this type of logic circuit. The circuit will be the 4-bit counter, similar to the ripple counter we looked at in Example 1. However, with this counter all four JKs will be synchronized to the same controlling signal. If we make this signal toggle once every second, then this counter can make the basis of a digital clock. We will look at that concept in Chapter 8.

To design the circuit, we will use the same principle of examining how the present state triggers the next state of the outputs and how we can use the internal feedback signals to control when the Js and Ks each connect to their respective logics. This will be a similar design process to the previous example, but the amount of information we will look at will make the process seem more complex. As long as we take our time and consider what we are doing, we will find that the process is not too complicated.

We will let the counter count from 0 to 15 with this example. Later, in Chapter 6, we will see what we have to do to make it a modulo 10 counter, that is, a BCD (Binary-Coded Decimal) counter. The first thing we must do is write down the “present state” and “next state” of the four outputs. To try and explain the process, we will create one table for each of the JKs. This will enable us to identify the transitions of each of the Qs separately from the “present state” to the “next state.” The table for the first JK flip flop with its output Q_0 is shown in Table 5-8.

Table 5-8. The Present State and Next State from the Q_0 Output

Present State				Row	Next State				Transition	J_0	K_0
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0			
0	0	0	0	1	0	0	0	1	0–1	1	x
0	0	0	1	2	0	0	1	0	1–0	X	1
0	0	1	0	3	0	0	1	1	0–1	1	x
0	0	1	1	4	0	1	0	0	1–0	X	1
0	1	0	0	5	0	1	0	1	0–1	1	x
0	1	0	1	6	0	1	1	0	1–0	X	1
0	1	1	0	7	0	1	1	1	0–1	1	x
0	1	1	1	8	1	0	0	0	1–0	X	1
1	0	0	0	9	1	0	0	1	0–1	1	x
1	0	0	1	10	1	0	1	0	1–0	X	1
1	0	1	0	11	1	0	1	1	0–1	1	x
1	0	1	1	12	1	1	0	0	1–0	X	1
1	1	0	0	13	1	1	0	1	0–1	1	x
1	1	0	1	14	1	1	1	0	1–0	X	1
1	1	1	0	15	1	1	1	1	0–1	1	x
1	1	1	1	16	0	0	0	0	1–0	X	1

The logic on the table for the J_0 and K_0 has been derived from the transition table shown in Table 4-10, rows 2 and 4, in Chapter 4 and repeated in Table 5-5 in this chapter. We now need to examine the logic for the J_0 and K_0 so that we can determine where we can get their logic from. We can see that the J_0 will simply toggle between a logic "1" and a don't care condition, i.e., a "X", in each row of Table 5-8. This means we can

simply keep the J_0 connected to a logic "1." We can also see that the same can be said about the K_0 even though they are in antiphase to each other. This really means that for the first JK flip flop, we can connect the J_0 and K_0 permanently to +VCC, that is, a logic "1."

Now we need to consider the second JK flip flop. The output of this would be Q_1. Therefore, using Table 5-9 we need to determine the transition states for J_1 and K_1 and their respective logics to bring about these transitions.

Table 5-9. *The Present State and Next State for Q1*

Present State				Row	Next State				Transition	J_1	K_1
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0			
0	0	0	0	1	0	0	0	1	0–0	0	X
0	0	0	1	2	0	0	1	0	0–1	1	X
0	0	1	0	3	0	0	1	1	1–1	x	0
0	0	1	1	4	0	1	0	0	1–0	x	1
0	1	0	0	5	0	1	0	1	0–0	0	X
0	1	0	1	6	0	1	1	0	0–1	1	X
0	1	1	0	7	0	1	1	1	1–1	x	0
0	1	1	1	8	1	0	0	0	1–0	x	1
1	0	0	0	9	1	0	0	1	0–0	0	X
1	0	0	1	10	1	0	1	0	0–1	1	X
1	0	1	0	11	1	0	1	1	1–1	x	0
1	0	1	1	12	1	1	0	0	1–0	x	1
1	1	0	0	13	1	1	0	1	0–0	0	X
1	1	0	1	14	1	1	1	0	0–1	1	X
1	1	1	0	15	1	1	1	1	1–1	x	0
1	1	1	1	16	0	0	0	0	1–0	x	1

We can see that there is a repetition every fourth change. However, there is no obvious choice of logic source for these transitions. Therefore, we must draw up the Karnaugh map for the J_1 and one for the K_1 and see if they provide us with the solution. The empty map for J_1 is shown in Figure 5-23.

Q_0 / Q_1 — $Q_3 Q_2$	0 0	0 1	1 1	1 0
0 0				
0 1				
1 1				
1 0				

Figure 5-23. *The Empty Karnaugh Map for the J_1 Input*

To fill the map, we will look at each row of Table 5-9. We will use the "present state" of the Q outputs to identify which cell we will be using on the map. Then we will fill that cell with the logic as indicated in that row of Table 5-9 for the J_1. As the "present state" of the first change, that is, row 1 in Table 5-9, is 0 0 0 0, then we are using cell 1 on the Karnaugh map. The logic for J_1 in row 1 is "0"; therefore, we must put a "0" in cell 1 of the map. This is shown in Figure 5-24.

Q_0 / Q_1 — $Q_3 Q_2$	0 0	0 1	1 1	1 0
0 0	0			
0 1				
1 1				
1 0				

Figure 5-24. *The "0" Inserted in Cell 1 of the Map for J_1*

The next change in row 2 shows Q_1 changing from 0 to 1. The logic of J_1 is "1," so we must put a "1" in the cell corresponding to the present state of 0001. This is cell 4 on the map, and this is shown in Figure 5-25.

$Q_3 Q_2$ \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0 0	0			**1**
0 1				
1 1				
1 0				

Figure 5-25. *The "1" Inserted in Cell 4 of the Map for J_1*

The next change in row 3 shows Q_1 changing from 1 to 1. The logic of J_1 is "x," so we must put an "x" in the cell corresponding to the present state of 0010. This is cell 2 on the map, and this is shown in Figure 5-26.

$Q_3 Q_2$ \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0 0	0	x		**1**
0 1				
1 1				
1 0				

Figure 5-26. *The "x" Inserted in Cell 2 of the Map for J_1*

The next change in row 4 shows Q_1 changing from 1 to 0. The logic of J_1 is "x," so we must put an "x" in the cell corresponding to the present state of 0011. This is cell 3 on the map, and this is shown in Figure 5-27.

$Q_3 Q_2$ \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0 0	0	x	x	**1**
0 1				
1 1				
1 0				

Figure 5-27. *The "x" Inserted in Cell 3 of the Map for J_1*

The same process can be carried out for all the remaining rows in Table 5-9, and so the correct logic for the J_1 can be entered into the map as shown in Figure 5-28.

Q_0 Q_1 $Q_3 Q_2$	0 0	0 1	1 1	1 0
0 0	0	x	x	**1**
0 1	0	x	x	1
1 1	0	x	x	1
1 0	0	x	x	1

Figure 5-28. *The Completed Map for the J_1 Logic*

To create the minimum expression for J_1, we must include all the logic "1"s and exclude all the logic "0"s. We can then decide to interpret the don't cares, the "x," as a logic "1" or logic "0." It is entirely up to us. If we include all the "x"s in cells 3, 7, 11, and 15 by interpreting them as logic "1"s, we can create the loop as shown in Figure 5-29.

Q_0 Q_1 $Q_3 Q_2$	0 0	0 1	1 1	1 0
0 0	0	x	x	**1**
0 1	0	x	x	1
1 1	0	x	x	1
1 0	0	x	x	1

Figure 5-29. *The Completed Karnaugh Map for the J_1 Input*

From inspection of the loop, we can see that the only output that does not change in the loop is Q_0. Q_0 is a logic "1" throughout the loop. That being the case, the expression for J_1 becomes

$$J_1 = Q_0$$

This means we need to connect the input J_1 directly to the output of the first JK, that is, directly to Q_0. We could have used a smaller loop, that which only included the "1"s in cells 4, 8, 12, and 16. However, if we did that, the expression for J_1 would become

$$J_1 = Q_0 \cdot \overline{Q_1}$$

This would work, but it would require an extra AND gate, which is not as efficient, so do not use this expression for J_1.

We now need to determine where the input K_1 will get its logic from. We can use Table 5-9 to determine the expression for K_1 in the same way as we did with J_1. In row 1 we can see that the present state of 0000 brings about the change of 0–0 and the K input is a don't care "x." This would be placed in cell 1 of the Karnaugh map for K_1.

The next change in row 2 is a 0–1, and the logic for K_1 is again a don't care "x." The present state that brought this change about was 0001. Therefore, we must put this "x" in cell 4 of the Karnaugh map for K_1.

The next change in row 3 is a 1–1, and the logic for K_1 is a "0." The present state that brought this change about was 0010. Therefore, we must put this "0" in cell 2 of the Karnaugh map for K_1.

The next change in row 4 is a 1–0, and the logic for K_1 is a "1." The present state that brought this change about was 0011. Therefore, we must put this "1" in cell 3 of the Karnaugh map for K_1.

These logic levels for K_1 are shown in Figure 5-30.

Q_0 / Q_1 — $Q_3 Q_2$	0 0	0 1	1 1	1 0
0 0	x	0	1	x
0 1				
1 1				
1 0				

Figure 5-30. *The First Four Cells in the Map for the K_1 Input*

If we follow the same procedure, we should be able to produce the completed Karnaugh map for K_1 as shown in Figure 5-31.

Q_0 / Q_1 — $Q_3 Q_2$	0 0	0 1	1 1	1 0
0 0	x	0	1	x
0 1	x	0	1	x
1 1	x	0	1	x
1 0	x	0	1	x

Figure 5-31. *The Completed Karnaugh Map for the K_1 Input*

We can see that, even though the "1"s and "x"s have been swapped over, the expression for K_1 is the same as J_1 as the term that is common to this loop is still Q_0. Therefore, the expression for K_1 is

$$K_1 = Q_0$$

We will see later how we can make these connections.

We can now carry on the process and determine the logic levels for J_2 and K_2 using Table 5-10.

Table 5-10. *The Table for the J_2 and K_2 Inputs*

Present State				Row	Next State				Transition	J_2	K_2
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0			
0	0	0	0	1	0	0	0	1	0–0	0	x
0	0	0	1	2	0	0	1	0	0–0	0	x
0	0	1	0	3	0	0	1	1	0–0	0	x
0	0	1	1	4	0	1	0	0	0–1	1	x
0	1	0	0	5	0	1	0	1	1–1	x	0
0	1	0	1	6	0	1	1	0	1–1	x	0
0	1	1	0	7	0	1	1	1	1–1	x	0
0	1	1	1	8	1	0	0	0	1–0	x	1
1	0	0	0	9	1	0	0	1	0–0	0	x
1	0	0	1	10	1	0	1	0	0–0	0	x
1	0	1	0	11	1	0	1	1	0–0	0	x
1	0	1	1	12	1	1	0	0	0–1	1	x
1	1	0	0	13	1	1	0	1	1–1	x	0
1	1	0	1	14	1	1	1	0	1–1	x	0
1	1	1	0	15	1	1	1	1	1–1	x	0
1	1	1	1	16	0	0	0	0	1–0	x	1

Using Table 5-10 the Karnaugh map for J_2 can be created as shown in Figure 5-32. You should try and confirm the table is correct.

Q_0 / Q_1 — $Q_3 Q_2$	0 0	0 1	1 1	1 0
0 0	0	0	1	x
0 1	0	0	1	x
1 1	x	x	x	x
1 0	x	x	x	x

Figure 5-32. *The Completed Karnaugh Map for the J_2 Input*

Using this map, the expression for J_2 is

$$J_2 = Q_0 . Q_1$$

Using Table 5-10, we can also construct the Karnaugh map for K_2 as shown in Figure 5-33.

Q_0 / Q_1 — $Q_3 Q_2$	0 0	0 1	1 1	1 0
0 0	x	x	x	x
0 1	x	x	x	x
1 1	0	0	1	0
1 0	0	0	1	0

Figure 5-33. *The Completed Karnaugh Map for the K_2 Input*

Using this map, the expression for K_2 is

$$K_2 = Q_0 . Q_1$$

The table for the final JK inputs, J_3 and K_3, is shown in Table 5-11.

Table 5-11. *The Table for the J_3 and K_3 Inputs for the 4-Bit Up Counter*

Present State				Row	Next State				Transition	J_3	K_3
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0			
0	0	0	0	1	0	0	0	1	0–0	0	x
0	0	0	1	2	0	0	1	0	0–0	0	x
0	0	1	0	3	0	0	1	1	0–0	0	x
0	0	1	1	4	0	1	0	0	0–0	0	x
0	1	0	0	5	0	1	0	1	0–0	0	x
0	1	0	1	6	0	1	1	0	0–0	0	x
0	1	1	0	7	0	1	1	1	0–0	0	x
0	1	1	1	8	1	0	0	0	0–1	1	x
1	0	0	0	9	1	0	0	1	1–1	x	0
1	0	0	1	10	1	0	1	0	1–1	x	0
1	0	1	0	11	1	0	1	1	1–1	x	0
1	0	1	1	12	1	1	0	0	1–1	x	0
1	1	0	0	13	1	1	0	1	1–1	x	0
1	1	0	1	14	1	1	1	0	1–1	x	0
1	1	1	0	15	1	1	1	1	1–1	x	0
1	1	1	1	16	0	0	0	0	1–0	x	1

Using Table 5-11 the Karnaugh map for J_2 can be created as shown in Figure 5-34.

$Q_3 Q_2$ \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	0	1	x
1 1	x	x	x	x
1 0	x	x	x	x

Figure 5-34. *The Completed Karnaugh Map for the J_3 Input*

Using this map, the expression for J_3 is

$$J_3 = Q_0.Q_1.Q_2$$

We can create the final Karnaugh map, that for K_3, from Table 5-11. The map is shown in Figure 5-35.

$Q_3 Q_2$ \ $Q_0 Q_1$	0 0	0 1	1 1	1 0
0 0	x	x	x	x
0 1	x	x	x	x
1 1	0	0	1	0
1 0	0	0	0	0

Figure 5-35. *The Completed Karnaugh Map for the K_3 Input*

Using this map, the expression for K_3 is

$$K_3 = Q_0.Q_1.Q_2$$

We can now create the circuit for the 4-bit synchronous binary counter. This is shown in Figure 5-36.

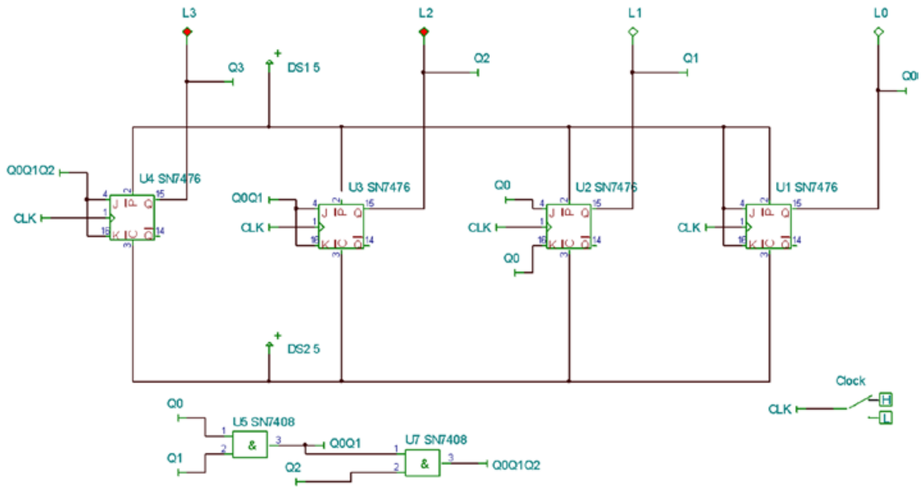


Figure 5-36. *The Completed Circuit for the 4-Bit Synchronized Counter*

We can see that we have not used a three-input AND gate for the J_3 and K_3 inputs. This is because we can use the output of the AND gate for J_2 and K_2 inputs, as this is Q_0 ANDed to Q_1. This means we only need to AND Q_2 to this output to get the three inputs we need for J_3 and K_3. This is what we do with U7. When we simulate this circuit, you will see that it does count from 0 to 15 in binary and then resets to 0.

Exercise 5.2

Using the method we have just used to create this synchronized up counter, you should design a 4-bit down counter and so confirm the circuit is as shown in Figure 5-37. To get you started, Table 5-12 shows the present and next states of the outputs from the down count.

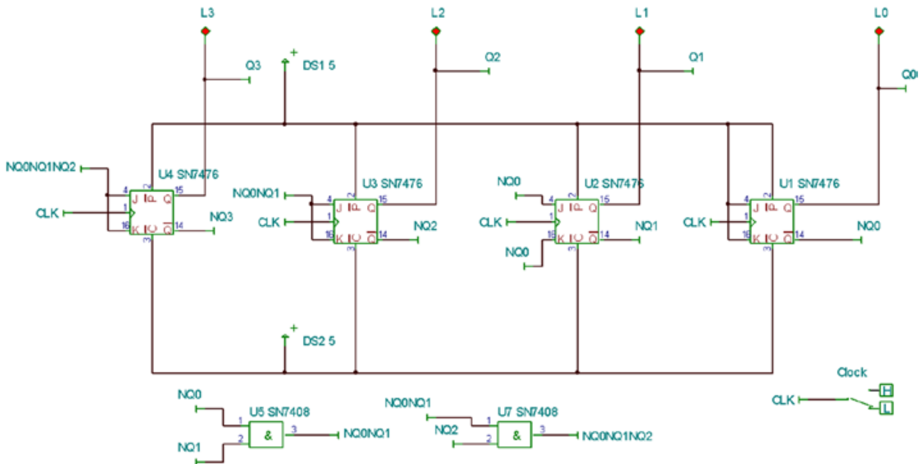


***Figure 5-37.** The Synchronized Down Counter*

***Table 5-12.** The Table for the Down Count*

Present State				Row	Next State				Transition	J	K
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0			
1	1	1	1	1	1	1	1	0			
1	1	1	0	2	1	1	0	1			
1	1	0	1	3	1	1	0	0			
1	1	0	0	4	1	0	1	1			
1	0	1	1	5	1	0	1	0			
1	0	1	0	6	1	0	0	1			
1	0	0	1	7	1	0	0	0			
1	0	0	0	8	0	1	1	1			
0	1	1	1	9	0	1	1	0			
0	1	1	0	10	0	1	0	1			
0	1	0	1	11	0	1	0	0			

(*continued*)

Table 5-12. (*continued*)

Present State				Row	Next State				Transition	J	K
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0			
0	1	0	0	12	0	0	1	1			
0	0	1	1	13	0	0	1	0			
0	0	1	0	14	0	0	0	1			
0	0	0	1	15	0	0	0	0			
0	0	0	0	16	1	1	1	1			

Design Example 5: A Modulo 6 Binary Counter

In this last example in this chapter, we will look at using a D-type latch to build a counter. In Chapter 4 we introduced the D-type latch, which is simply a JK flip flop with the K being connected as the NOT of the J input.

We will design this counter to work as a synchronized system synchronizing it to the clock signal it will be counting. The first thing to do is to determine how many latches to use. This can be determined using

$$2^{n-1} \leq N \leq 2^{n}$$

where "N" is the number of changes the system will go through. Little "n" is the number of latches needed to create the circuit. If we create the present state and next state table, we will be able to determine the number of changes and so the number of latches the circuit will require. The process will use some trial and error to determine the value for "n" that conforms to the preceding expression.

The present state and next state table is shown in Table 5-13.

Table 5-13. *The Present and Next States of the Modulo 6 Counter*

Present State PS			Row Number	Next State NS		
Q_2	Q_1	Q_0		Q_2	Q_1	Q_0
0	0	0	1	0	0	1
0	0	1	2	0	1	0
0	1	0	3	0	1	1
0	1	1	4	1	0	0
1	0	0	5	1	0	1
1	0	1	6	0	0	0

We can see that there are six changes in this counter, which means "N" = 6. To determine the value of "n," we will firstly try n = 2.

$$2^{n-1} \leq N \leq 2^n$$
$$2^{2-1} \leq 6 \leq 2^2 \therefore 2 \leq 6 \leq 4$$

This is not true as 6 is not a value between 2 and 4. Therefore, we will try letting n = 3.

This gives

$$2^{n-1} \leq N \leq 2^n$$
$$2^{3-1} \leq 6 \leq 2^3 \therefore 2^2 \leq 6 \leq 8$$
$$\therefore 4 \leq 6 \leq 8$$

This is true as 6 is a value between 4 and 8. Therefore, "n" = 3, which means the circuit will require three D-type latches. Note the clock will be sent to each of the three D-type latches. The outputs "Q" of each D-type latch will indicate the current state of the count. Each output will be numbered Q_0, Q_1, and Q_2.

Determining the Inputs for the Three D-Type Latches

The next part of the process is to determine where each D input comes from. The D inputs are numbered D_0, D_1, and D_2. The process is to use the "present state" and the "next state" of the outputs.

The D_0 Inputs

To determine where the logic for input D_0 comes from, we need to identify each logic "1" in the "next state" column for Q_0. There are three logic "1"s in this column. They are in rows 1, 3, and 5; see Table 5-13.

Having identified the logic "1"s, the next step is to write down the logic state of the three outputs from the "present state" listings for each of the three logic "1"s that have been identified. Note if the Q output is a logic "1," it is written as "Q." If the "Q" output is a logic "0," then it is written as NOT Q, that is, $\overline{Q}$. In this way a logic equation for each logic "1" in the "next state" listings for the Q_0 output is

$$D_0 = \overline{Q_2}.\overline{Q_1}.\overline{Q_0} + \overline{Q_2}.Q_1.\overline{Q_0} + Q_2.\overline{Q_1}.\overline{Q_0}$$

This expression can be implemented using a three-input OR gate fed from three three-input AND gates. We can create a test circuit as shown in Figure 5-38 to test that this drive circuit does get the first D-type latch to count 0 to 1. Note the other two latches have their respective D inputs tied to 0V.

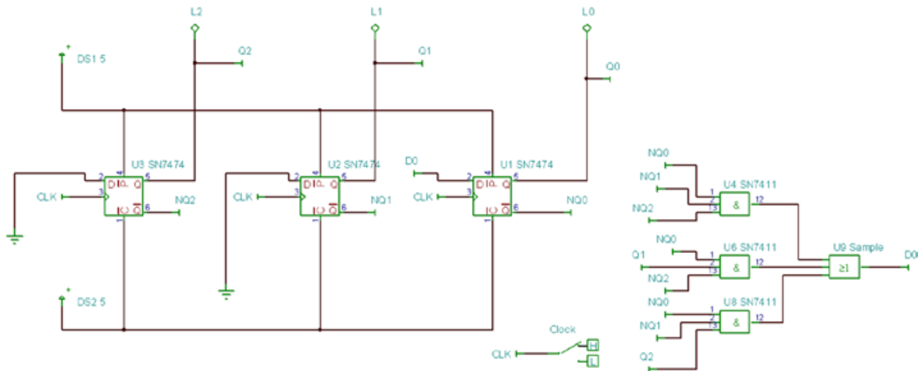


Figure 5-38. *The Test Circuit for the Input D_0*

This test circuit does show that the drive circuit does work as required. However, we could create a Karnaugh map for the input D_0 to see if it would produce a simpler logic circuit. As we use the binary values for the Qs of the circuit, it might be easier to write the expression for D_0 in its binary format. This would produce the expression

$$D_0 = \sum 000, 010, 100$$

I think that using this binary format of the variables for the expression makes completing the Karnaugh map easier than using the letters A, B, C, etc. for the variables. The three separate binary numbers, each with 3 bits, in the expression for D_0, read from right to left, the LSB being the first bit on the right of the 3 binary bits and the MSB being the last bit on the left, for each of the three binary numbers. In the Karnaugh map, we have placed the most significant output, Q_2, on the vertical. This is followed by Q_1, the lower of the two outputs on the horizontal. The least significant output, Q_0, is the upper of the two outputs on the horizontal of the map. In this way the map mimics the numbers in that the Q_2 is the farthest on the left and the Q_0 is the first output on the right. This means that reading the first of the three binary numbers, that is, the 000, we must place a "1" in cell 1 of the map. The next binary number, 010, puts the 1 in cell 2. Finally, the third

binary number, 100, goes in cell 5 of the map. The completed Karnaugh map for the D_0 input is as shown in Figure 5-39.

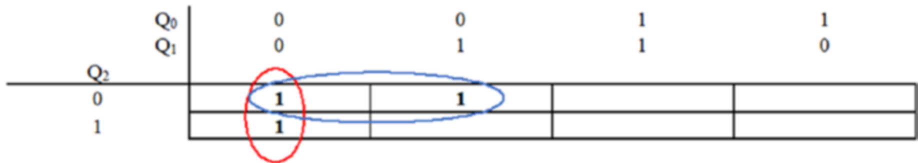


Figure 5-39. *The Karnaugh Map for the Input D_0*

This Karnaugh map does show that the expression for D_0 does minimize as the expression is

$$D_0 = Loop1 + Loop2$$

Loop 1, in red, is $\overline{Q_1}.\overline{Q_0}$ as Q_2 changes from logic "0" to logic "1" in that loop. Loop 2, in blue, is $\overline{Q_2}.\overline{Q_0}$ as Q_1 changes from logic "0" to logic "1" in that loop.

This means that the expression for D_0 becomes

$$D_0 = \overline{Q_1}.\overline{Q_0} + \overline{Q_2}.\overline{Q_0}$$

This can be implemented with two AND gates, which would feed a two-input OR gate. However, using Boolean algebra this expression could be reduced to

$$D_0 = \overline{Q_0}.\left(\overline{Q_1} + \overline{Q_2}\right)$$

This can be implemented with a two-input AND gate and a two-input OR gate, thus saving us a logic gate.

The circuit to implement this is shown in Figure 5-40.

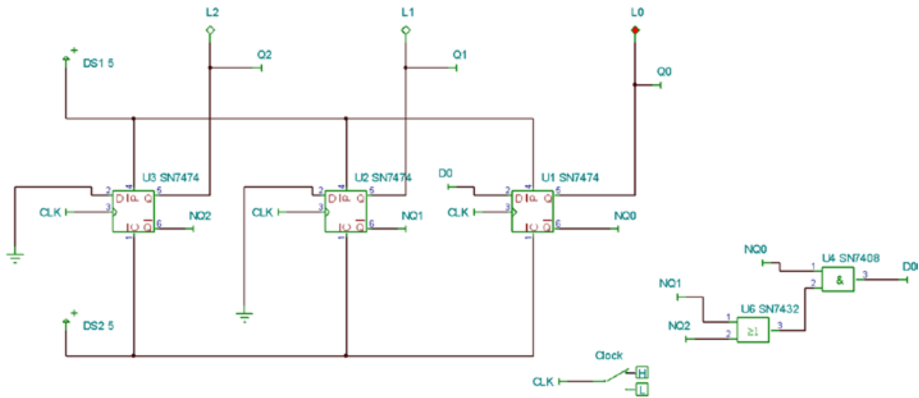


Figure 5-40. *The Minimized Circuit for the Input D_0*

We can see that the minimized circuit works as required; therefore, we should use Karnaugh maps to try and minimize the expressions for the other D inputs.

The D_1 Inputs

The next step, in the process, is to determine where the D_1 gets its inputs from. To do this the information can be obtained from Table 5-13 as before. We must look for the logic "1"s in the Q_1 column of the "next state" part of the table. There are two logic "1"s in this column. They are in rows 2 and 3. The next step is to write down the logic state of the three outputs from the "present state" listings for each of the two logic "1"s that have been identified. In this way a logic equation for the Q_1 output is

$$D_1 = \overline{Q_2}.\overline{Q_1}.Q_0 + \overline{Q_2}.Q_1.\overline{Q_0}$$

or

$$D_1 = \sum 001,010$$

The next step is to try and minimize the expression using Karnaugh maps. The Karnaugh map for this expression is shown in Figure 5-41.

Q_2 \ Q_0 Q_1	0 0	0 1	1 1	1 0
0		**1**		**1**
1				

Figure 5-41. *The Karnaugh Map for the Input D_1*

However, no loops can be made, so this expression does not minimize using the Karnaugh map, but using Boolean algebra it does minimize to

$$D_1 = \overline{Q_2}.\left(\overline{Q_1}.Q_0 + Q_1.\overline{Q_0}\right)$$

It is good practice to prove your circuit design at each stage of your design. In this way you can confirm that each part of the design works, and when you come to a stage that doesn't work as you had hoped, then you know the fault lies in the latest stage of your design. If you don't do this and only test your circuit when your design is complete, then you have no idea where the fault lies. Therefore, we should test that the drive logic for the D_1 input works correctly. The test circuit is shown in Figure 5-42.

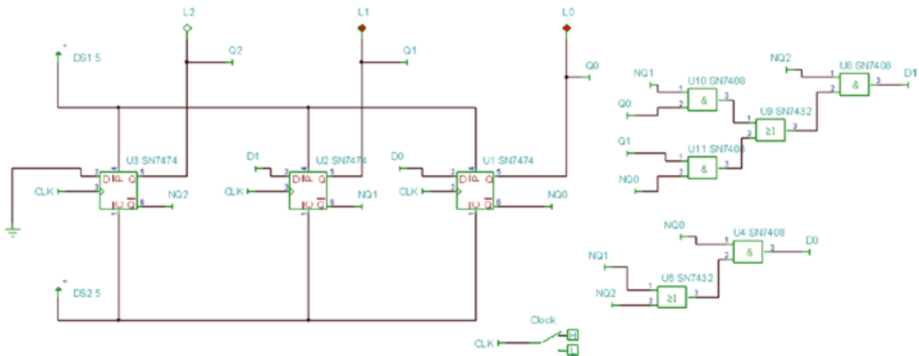


Figure 5-42. *The Test Circuit for Inputs D_0 and D_1*

The test circuit confirms that the drive logic works so far.

The D_2 Inputs

The final step in the design process is to determine where the final latch gets its inputs to the D_2 from. The process uses Table 5-13 as before. This is done by identifying each logic "1" in the "next state" column for Q_2. There are two logic "1"s in this column. They are in rows 4 and 5. Having identified the logic "1"s, the next step is to write down the logic state of the three outputs from the PS listings for each of the two logic "1"s that have been identified:

$$D_2 = \overline{Q_2}.Q_1.Q_0. + Q_2.\overline{Q_1}.\overline{Q_0}.$$

Again, use Karnaugh maps to try and minimize the expression. However, the expression does not minimize. Also, there is no obvious simplification using Boolean algebra.

This means that the expressions for each of the D inputs are

$$D_0 = \overline{Q_0}.(\overline{Q_1} + \overline{Q_2})$$

$$D_1 = \overline{Q_2}.\left(\overline{Q_1}.Q_0 + Q_1.\overline{Q_0}\right)$$

$$D_2 = \overline{Q_2}.Q_1.Q_0. + Q_2.\overline{Q_1}.\overline{Q_0}$$

The complete circuit to show the implementation of this design is shown in Figure 5-43.

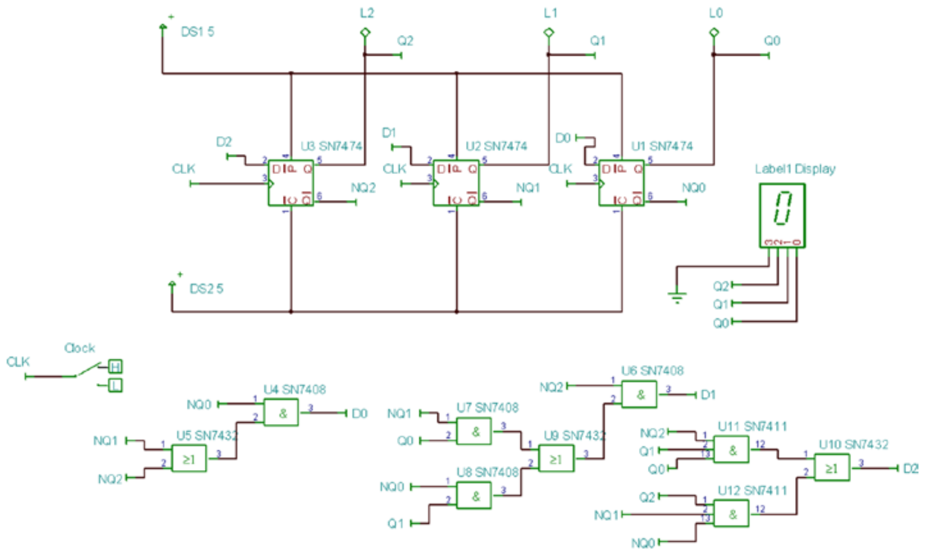


***Figure 5-43.** The Complete Modulo 6 Counter with a Seven-Segment Display*

When we simulate the circuit, we can see that the seven-segment display does count from 0 to 5 and at the next count it goes back to 0. The gates 4–12 are there to make up the drive logic for the design. We can say that the seven-segment display makes up the output logic for the design. The Q outputs from the three D-type latches make up the feedback logic of the design. Recognizing these aspects of the circuit does show that the design does conform to the basic block diagram as shown in Figure 5-1.

A final aspect of this counter, which uses the D-type latch, is that the count changes every time the clock goes from low to high. This differentiates it from the JK counter as the JK counter changes when the clock goes from high to low.

Synopsis

In this chapter we have looked at what makes up a logic circuit and how to use the "present state" and "next state" tables to design a logic circuit. We have designed some counters using the basic JK flip flop and lastly using the D-type latch.

In the next chapter, we will look at state diagrams and how they can be used to represent a logic system. We will also look at how they can be used as an aid to design logic systems. I hope you have enjoyed reading this chapter and found it useful in developing your understanding of logic circuits and their design.

CHAPTER 6

State Example 3 A Bit Stream Monitor

In this chapter we will look at how a structured design method can be applied to designing synchronous counters. The method we will use makes use of “state diagrams.” We will learn how to interpret these state diagrams and use them to design a variety of sequential logic circuits.

State Diagrams

These are a pictorial representation of how a sequential logic system will change from one state to another state. They consist of nodes in which the state of the internal logic elements may or may not be shown. They have transition lines that show how the system will move from one node to the next. The logic state of the inputs that causes the system to move along the transition lines is indicated along these lines. There is usually a legend on these transition lines to indicate what the inputs are and how they are represented; however, these outputs may be indicated inside the nodes instead of along the transition lines. The transition from one state to the next is normally synchronized to a trigger, which, although not shown in the diagram, will be applied to the system in some form.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_6

The State Diagram of the JK Flip Flop

Figure 6-1 shows the state diagram for the JK flip flop. To appreciate how it works, we will compare the diagram to the transition table of the JK. This is shown in Table 6-1.

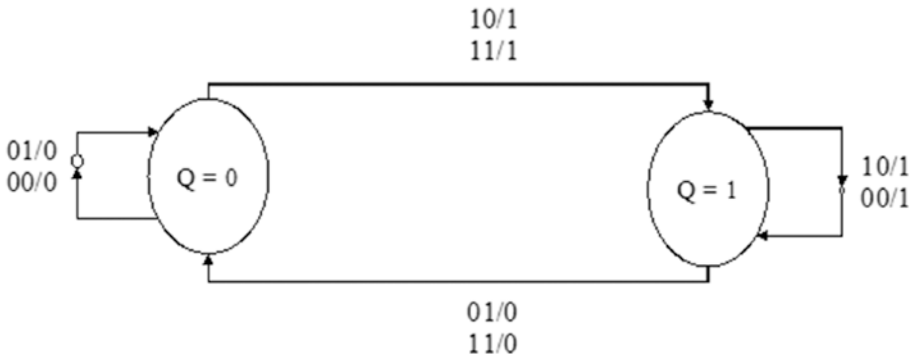


Figure 6-1. *The State Diagram for the JK Flip Flop*

Table 6-1. *The Transition Table for the JK Flip Flop*

Row	J	K	Q
1	0	X	0–0
2	1	X	0–1
3	X	0	1–1
4	X	1	1–0

The inputs, that is, the J and the K, are shown in the form of JK/Q. This also indicates the output Q. If, with the Q = 0 state, the J input was a logic “0,” then, no matter what logic level there was on the K input, the flip flop would stay with Q = 0, and the output would be at a logic “0” every time the JK was triggered. This is shown as the circulating transition arrow on the state diagram. If we look at row 1 in the transition table, Table 6-1, we see that the table shows the same result.

Now, while still in the Q = 0 state, if the J input went to a logic "1" as with the upper transition arrow, then when triggered, the Q output would go to a logic "1." This will happen no matter what logic there was on the K input. This is reflected, as being the same, in row 2 of the transition table.

Now the Q output has gone to a logic "1." We can see that if the K input was set to a logic "0," then regardless of the logic on the J input, the Q would stay at a logic "1" on the next trigger. This is indicated by the circulating arrow on the right-hand node. Row 3 of the transition table shows the same effect.

Now, with Q being a logic "1," if the K input went to a logic "1," then, regardless of the logic on the J input, the Q would change to a logic "0" on the next trigger. This is indicated with the lower transition arrow on the state diagram. Row 4 of the transition table shows the same result.

Creating the JK Flip Flop State Table

Before we progress too far with using state tables, it might be useful to show you how we can create a state diagram from some information. To help with this, we will use the information from the truth table for the JK experiment to create the state diagram shown in Figure 6-1. The truth table is shown in Table 6-2.

Table 6-2. *The Truth Table for the JK Flip Flop*

Row	J Input	K Input	Present Q	Next Q	Comment
1	0	0	0	0	No change from present state.
2	0	0	1	1	
3	0	1	0	0	Resets the Q to logic "0."
4	0	1	1	0	
5	1	0	0	1	Sets the Q to a logic "1."
6	1	0	1	1	
7	1	1	0	1	The Q output toggles between
8	1	1	1	0	logic "1" and logic "0."

To create this truth table, we can simulate the test circuit shown in Figure 6-2.

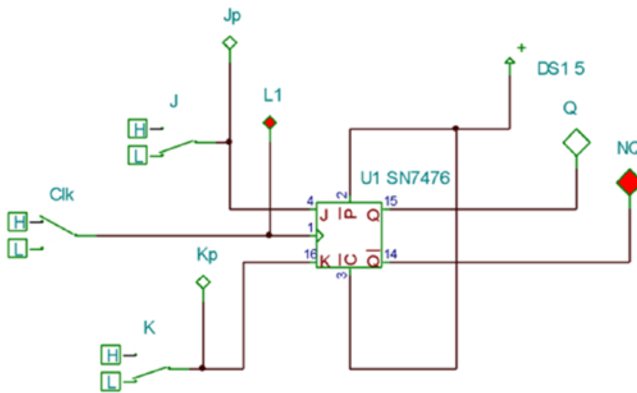


Figure 6-2. *The Test Circuit for the JK Truth Table*

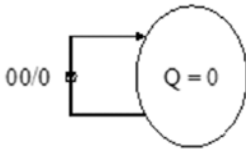
When simulating the test circuit, we need to make sure the Q output is set to that as shown in the present state for each row in Table 6-2. The table shows the same conditions for the J and K in two rows at a time. This is because we need to test how the JK will respond to the two present states,

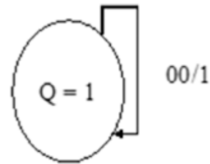
that is, the present state is a logic "0" and the present state is a logic "1," for each configuration of the J and K inputs; that is why rows 1 and 2 have the J and K both set to a logic "0." Therefore, if you carefully simulate the circuit for each setting of the truth table, you should be able to complete the table as shown in Table 6-2.

Now to create the state diagram, we will try to look at how the diagram builds up for each change shown in the truth table. The changes will be shown in Figure 6-3 starting with diagram "a" that shows the two possible states of the Q output and the consequence of row 1 in Table 6-2. Row 1 shows that with both the J and K set to a logic "0," the JK does not change. This means if the Q was a logic "0," then the diagram should show that we don't change anything. However, we should show that something has occurred as the clock will have gone from low to high and back to low. That is why we show an arrow circulating around the state Q = 0. We also indicate the logic on the J and K as well as the output. This is represented using the symbology JK/Q, which is shown as 00/0. This is because J = 0, K =0, and Q = 0. A state diagram drawn up this way, with the input and output conditions shown on the transition lines, is called a "Mealy state diagram." We will consider "Moore's and Mealy" machines at the end of the chapter.

Part "a"

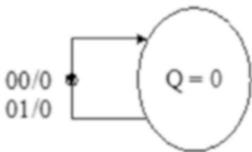
This implements row 1 when J = K = 0 and we have no change Q = 0 assuming Q was a logic "0" to begin with.

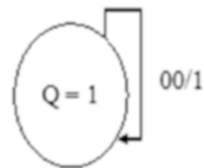




Part “b”

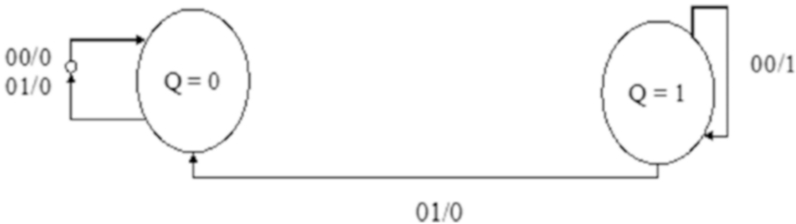
This implements row 2 when J = K = 0 and we have no change Q = 1 assuming Q was a logic “1” to begin with.





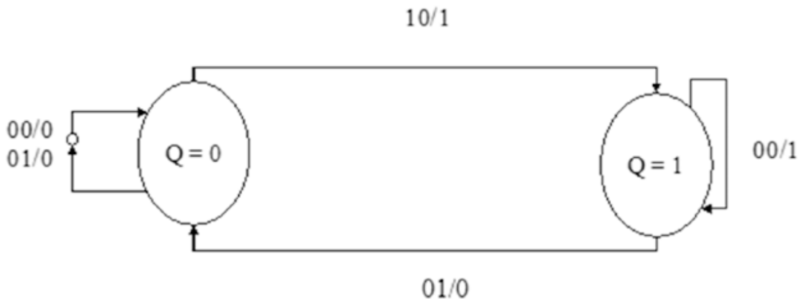
Part “c”

This implements row 3 when J = 0 and K = 1 and the Q resets to logic “0.” If we assume the Q is already at a logic “0,” we just circle around the state Q = 0.



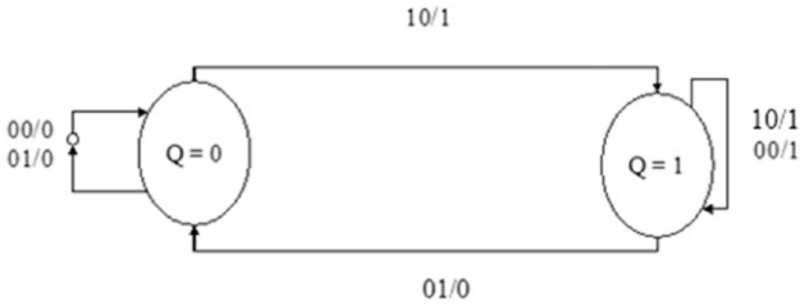
Part “d”

This implements row 4 when J = 0 and K = 1 and Q resets from 1 to 0 as if the Q was at a logic “1,” the JK flip flop would move from Q =1 to Q = 0.



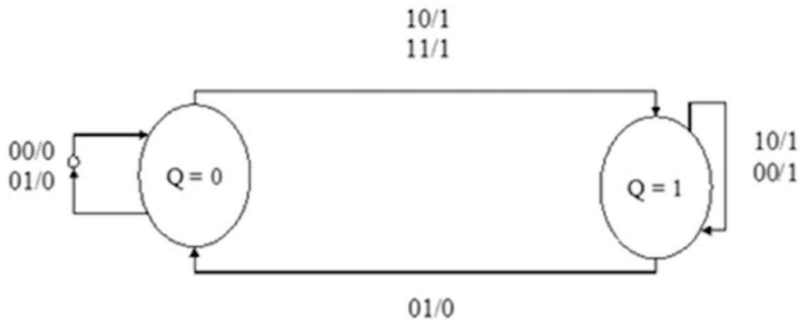
Part “e”

This implements row 5 when J = 1 and K = 0 and Q sets from 0 to 1.



Part “f”

This implements row 6 when J = 1 and K = 0 and Q sets from 1 to 1.



Part “g”

This implements row 7 when J = 1 and K = 1 and Q changes from 0 to 1.

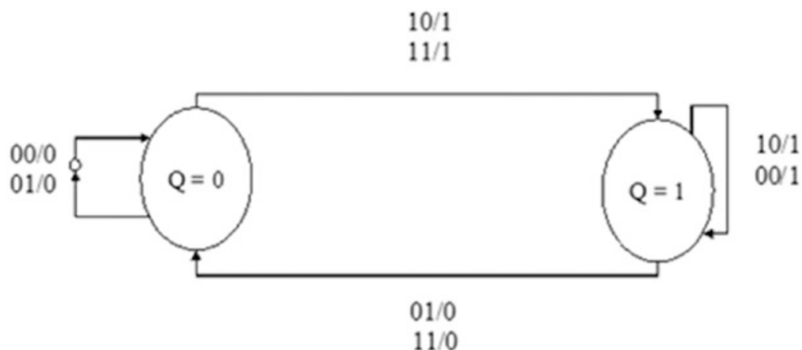


Figure 6-3. *The Stages of Construction of the JK State Diagram*

This implements row 8 when J = 1 and K = 1 and Q changes from 1 to 0.

I hope this set of figures does give you some idea of how we can create a state diagram from some given information. To be able to use state diagrams properly, it will be useful if we can understand how to interpret them correctly. Knowing how they are made is a good step forward in achieving this.

Using the truth table for the JK shown in Table 6-2, we can create a smaller table that sums up how the Q reacts to the configuration options for the J and K inputs. Table 6-3 shows how the JK reacts.

Table 6-3. *The Change of States for the JK Flip Flop*

Row	J Input	K Input	Present Q	Next Q	Comment
1	0	x	0	0	Always logic "0."
2	1	x	0	1	Change to logic "1."
3	x	0	1	1	Q goes to "1" if not already a 1 and stays at 1.
4	x	1	1	0	If J = 1 then Q goes to 0 and stays at 0.

Row 1 is a combination of rows 1 and 2 of Table 6-2. With rows 1 and 2 in Table 6-2, we can see that when the J is connected to a logic "0," the Q output will reset and go to a logic "0" no matter what logic there is on the K. Hence, row 1 of Table 6-3 shows that Q = 0 with J = 0 and K = x, that is, don't care what the logic is on the K input.

I hope we can see that row 2 of Table 6-3 is a combination of rows 3 and 4 in Table 6-2 and that row 3 of Table 6-3 is a combination of rows 5 and 6 in Table 6-2. Finally, row 4 of Table 6-3 is the final combination of rows 7 and 8 in Table 6-2. We will use this change of state table when we move on to design logic circuits using JK flip flops.

The following will be a series of examples that will hopefully show us how to use state diagrams when designing logic circuits to perform a specific function. Before we do that, it might be useful to think about a procedure that, if we follow, could help us with our designs.

Methodology for Designing Sequential Digital Logic Circuits

We all have our own way of designing systems and circuits, but it is sometimes useful to have a good starting point. I am not going to say that the method I will describe here is the best approach, but it has served me well since I was first introduced to it. There are basically eight steps you could follow to produce a working solution to your logic problem. They are the following:

1. Once you understand the specification of what your design is meant to do, you should create a state diagram or a timing diagram that represents the behavior of your system.

2. Assign a binary value for your states.

3. Determine how many flip flops your design will require and what type you will use.

4. Create a state table that describes the "present state" and "next state" of all the variables in your design.

5. Decide if some of your states can be removed.

6. Create an excitation table if required.

7. Determine the expressions for the internal inputs and the outputs of the design.

8. Create and test the circuit for your design.

You do not have to stick rigidly to the items in the list, but it will give you a good starting point. To help explain the steps, we will go through a design example now.

State Diagram Example 1: The Synchronized Binary Counter

In Chapter 5 we designed a synchronous binary counter using the JK flip flop. In this example we will design the same counter using the D-type latch.

The specification for the counter is that once started the counter will simply increment a seven-segment display from 0 to F, using the hexadecimal number system. There will be no external input except the clock signal that synchronizes when the counter will increment.

The state diagram is shown in Figure 6-4. There is nothing to trouble us with the diagram as it will have 16 states going from 0 to 15 in decimal.

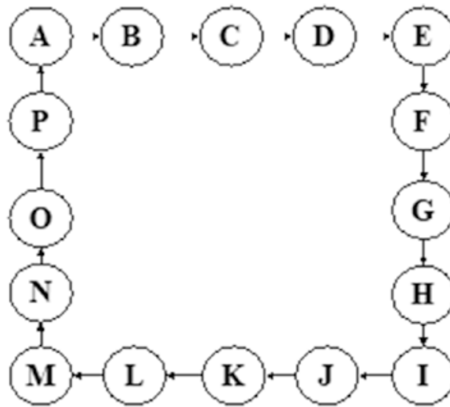


Figure 6-4. *The State Diagram for Example 1*

Step 2 says assign a binary value to each of the steps. This is done with Table 6-4.

Table 6-4. *Assigning Binary Value for Each State*

State	Binary Value
A	0000
B	0001
C	0010
D	0011
E	0100
F	0101
G	0110
H	0111
I	1000

(*continued*)

Table 6-4. (*continued*)

State	Binary Value
J	1001
K	1010
L	1011
M	1100
N	1101
O	1110
P	1111

To assign the binary values, we simply increment the value, reading from right to left, by adding a binary 1 to the previous value. This is the normal count of a 4-bit binary number, and we will use this in all our truth tables. As there are 16 states in the diagram, then we will need four D-type latches, that is, $2^3 = 8$, but $2^4 = 16$.

We can now carry out step 4, which is to create the state table. This is shown in Table 6-5.

Table 6-5. *The State Table for Example 1*

Present State				Row Number	Next State			
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0
0	0	0	0	1	0	0	0	1
0	0	0	1	2	0	0	1	0
0	0	1	0	3	0	0	1	1
0	0	1	1	4	0	1	0	0
0	1	0	0	5	0	1	0	1

(*continued*)

Table 6-5. (*continued*)

Present State				Row Number	Next State			
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0
0	1	0	1	6	0	1	1	0
0	1	1	0	7	0	1	1	1
0	1	1	1	8	1	0	0	0
1	0	0	0	9	1	0	0	1
1	0	0	1	10	1	0	1	0
1	0	1	0	11	1	0	1	1
1	0	1	1	12	1	1	0	0
1	1	0	0	13	1	1	0	1
1	1	0	1	14	1	1	1	0
1	1	1	0	15	1	1	1	1
1	1	1	1	16	0	0	0	0

There are no states in the table that are repeated, and so we can't reduce the table.

There is no need to create an excitation table as we can determine where the D inputs will get their logic from directly from the state table.

Determining the Inputs for the Four D-Type Latches

The next part of the process is to determine where each D input comes from. The D inputs are numbered D_0, D_1, D_2, and D_3. We could try to use the state diagram directly; however, the process may be more straightforward if we use the state table, as shown in Table 6-5, which indicates the "present state" and the "next state" of the outputs.

The D_0 Input

We need to determine where the D_0 input will get its logic from. To do this we need to determine the logic of the "present state" for every logic "1" in the Q_0 column of the "next state" of the Q outputs shown in the state table, Table 6-5. There are eight rows in the "next state" column for Q_0 where we have a logic "1." Using each of the eight occurrences, we can build up the expression for the D_0 input. The first term is in row 1, and the state of the outputs in the "present state" is

$$\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}$$

The next occurrence is in row 3, and the state of the outputs in the "present state" is

$$\overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0}$$

The next occurrence is in row 5, and the state of the outputs in the "present state" is

$$\overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0}$$

Once we have determined the logic for all eight occurrences, we will be able to put all eight terms to form the expression for D_0 as it will simply be the OR of all eight terms.

Exercise 1

As an exercise continue the process to determine the logic for all eight occurrences and so confirm that the expression for D_0 becomes

$$\begin{aligned} D_0 = {} & \overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0} + \overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0} + \overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0} + \overline{Q_3}.Q_2.Q_1.\overline{Q_0} \\ & + Q_3.\overline{Q_2}.\overline{Q_1}.\overline{Q_0} + Q_3.\overline{Q_2}.Q_1.\overline{Q_0} + Q_3.Q_2.\overline{Q_1}.\overline{Q_0} + Q_3.Q_2.Q_1.\overline{Q_0} \end{aligned}$$

We will try using Karnaugh maps to minimize the expression. Therefore, to possibly help us with that process, we could write the expression for the D_0 inputs using the binary format. This will give us

$$D_0 = \sum 0000,0010,0100,0110,1000,1010,1100,1110$$

We can now put these logic "1"s into the map. The Karnaugh map for this is shown in Figure 6-5.

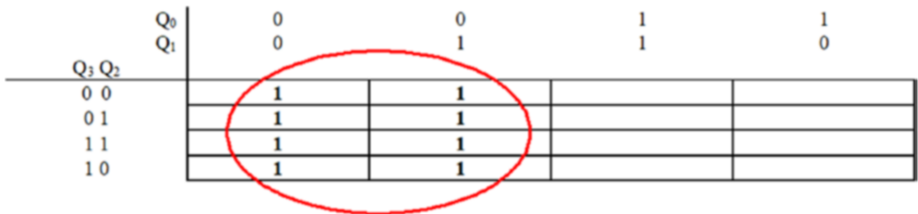


Figure 6-5. *The Karnaugh Map for the D_0 Input for State Diagram Example 1*

Using this map, the expression for the D_0 input simplifies to

$$D_0 = \overline{Q_0}$$

The D_1 Input

If we examine the "present state" and "next state" table shown in Table 6-5, we can see that in the "next state" column for Q1, the Q_1 is a logic "1" in rows 2, 3, 6, 7, 10, 11, 14, and 15. This means we need to look at the state of the Qs in the "present state" part for each of these rows. This will give us the following expression for the logic of the inputs to D_1:

$$D_1 = \sum 0001,0010,0101,0110,1001,1010,1101,1110$$

If you want to, you can go through the process to confirm you get the same result. The more you go through the process, the better you become at it.

We can put the logic "1"s in the Karnaugh map to see if it will minimize. The map is shown in Figure 6-6.

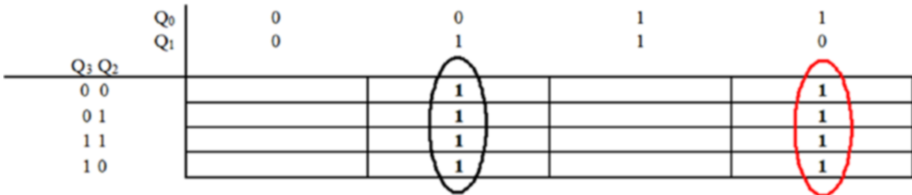


Figure 6-6. *The Karnaugh Map for the D_1 Input*

Using the map, the expression for D_1 becomes

$$D_1 = Q_1.\overline{Q_0} + \overline{Q_1}.Q_0$$

The D_2 Input

If we examine the "present state" and "next state" table shown in Table 6-5, we can see that in the "next state" column for Q_2, the Q_2 is a logic "1" in rows 4, 5, 6, 7, 12, 13, 14, and 15. This means we need to look at the state of the Qs in the "present state" part for each of these rows. This will give us the following expression for the logic of the inputs to D_2:

$$D_2 = \sum 0011, 0100, 0101, 0110, 1011, 1100, 1101, 1110$$

If we put the logic "1"s into the map, we get the Karnaugh map shown in Figure 6-7.

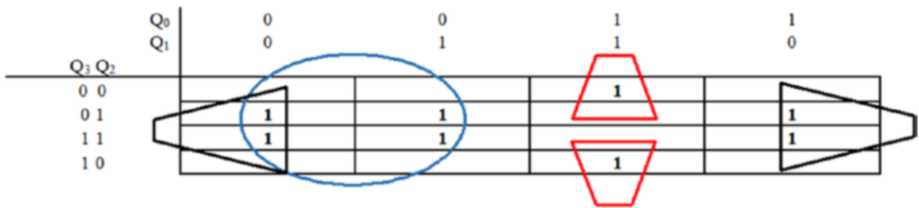


Figure 6-7. *The Karnaugh Map for the D_2 Input*

Using the map, the expression for D_2 becomes

$$D_2 = \overline{Q_2}.Q_1.Q_0 + Q_2.\overline{Q_1}. + Q_2.\overline{Q_0}$$

Exercise 2

Continue the process to show that the expression for D_3 becomes

$$D_3 = Q_3.\overline{Q_0} + Q_3.\overline{Q_1}. + Q_3.\overline{Q_2}.Q_1 + \overline{Q_3}.Q_2.Q_1.Q_0$$

Using Boolean algebra, the expression can simplify more to

$$D_3 = Q_3.\left(\overline{Q_2}.Q_1 + \overline{Q_1} + \overline{Q_0}\right) + \overline{Q_3}.Q_2.Q_1.Q_0$$

Now that we have all four expressions for the D inputs, the circuit can be created. This is shown in Figure 6-8.

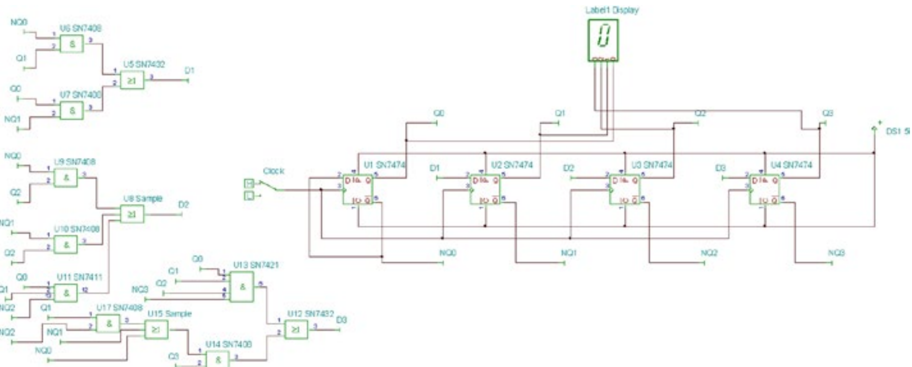


Figure 6-8. *The Complete Circuit for the Binary Counter*

The circuit uses the seven-segment display that has its own decoder in it to display the output of the counter. There is no need for any output logic with that seven-segment display as the Q outputs can drive the display directly.

To make it easier to draw up the circuit and read where the inputs come from, I am using the "jumper" connections within TINA. These are available from the specials option on the menu bar in TINA. You may need to read Chapter 11 to understand how to create this circuit.

As Figure 6-8 is rather small to show all the details, I will use Figures 6-9 and 6-10 to show the logic for the D inputs.

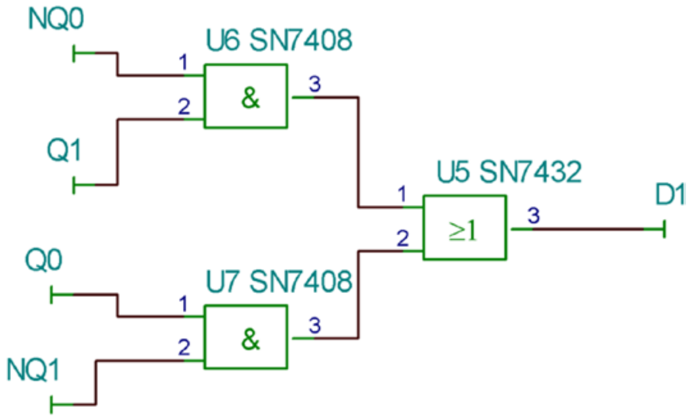


Figure 6-9. *The Logic for the D_1 Input*

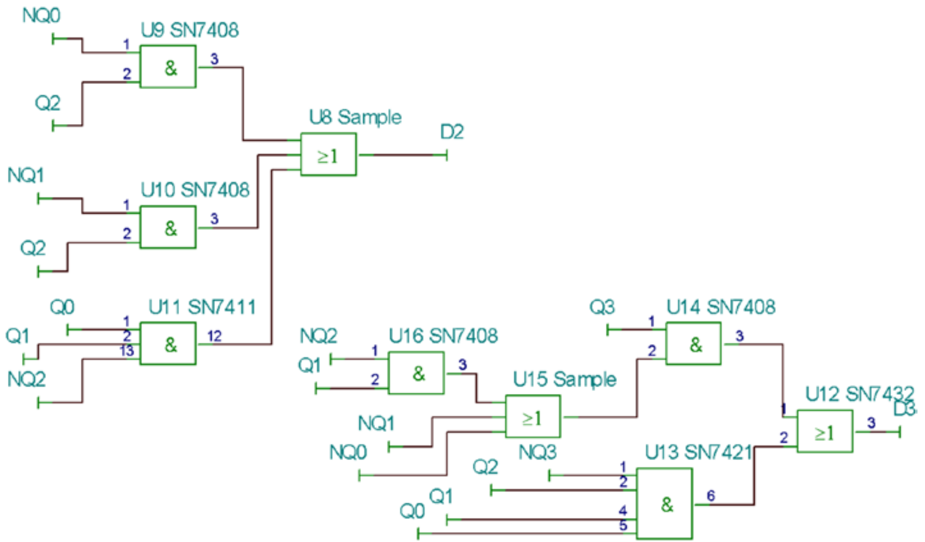


Figure 6-10. *The Logic for the D_2 and D_3 Inputs*

State Diagram Example 2: The Design of a Modulo 10 Binary Counter Using State Diagrams

This is a binary counter that will count up from 0 to 9. Then, when it tries to display the value 10, it will reset back to 0 and start again. We will create a synchronous counter with the trigger input changing every second and controlling when the system changes from one state to the next. The state diagram for this example is shown in Figure 6-11. As there are no external inputs to the system, we will simply draw the state diagram and label the states as shown in Figure 6-11.

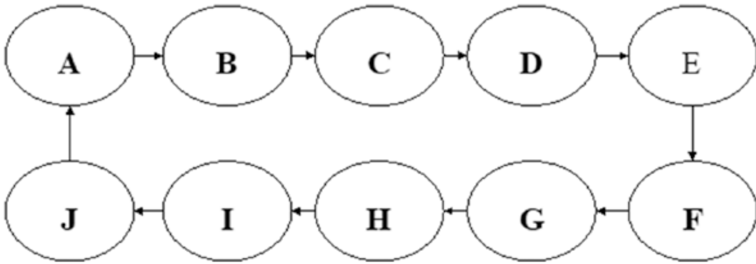


Figure 6-11. *The State Diagram for the Modulo 10 Counter*

Now that we know what the circuit should do, we should attempt to create the state diagram. Note I say attempt as we may need to revisit it, later in the design process, as it may not be good enough to describe the process. The state diagram is shown in Figure 6-11.

Starting at the top left-hand state, the system moves to the next state in the direction of the arrow with every clock tick. In this way the system can be said to have a “present state” and a “next state.” The system moves from one state to the next on each tick of the clock. We will design the counter using D-type latches.

The next thing to do is assign binary values to the states A–J. This is so that we can use Boolean algebra on any expression we create and also assign them to the outputs of our D-type latches. The assignment is done in Table 6-6.

Table 6-6. *The Assignment of the States to Binary Values*

State	Binary Value	Outputs
A	0000	Q0–Q3, which means we will require four latches. We will confirm this next.
B	0001	
C	0010	
D	0011	
E	0100	
F	0101	
G	0110	
H	0111	
I	1000	
J	1001	
	Value read from right to left	Outputs go from right to left.

It may be easier to create the state diagram with these binary values already allocated to each state. Indeed, sometimes this is how a state diagram is drawn up. The state diagram for this example drawn up this way is shown in Figure 6-12.

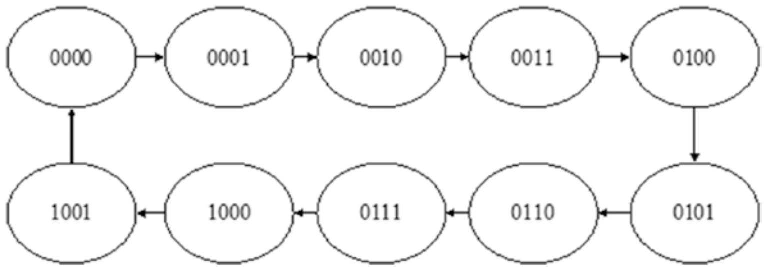


Figure 6-12. *The State Diagram with the Binary Values Allocated to Each State*

The next thing to do is to determine how many latches we need. This can be determined using

$$2^{n-1} \le N \le 2^n$$

where "N" is the number of states in the state diagram, in this case N = 10.

Little "n" is the number of latches needed to create the circuit. The process is to use trial and error to determine the value for "n" that conforms to the preceding expression.

Try n = 3.

This gives

$$2^{n-1} \le N \le 2^n$$

$$2^{3-1} \le 10 \le 2^3$$

$$\therefore 2^2 \le 6 \le 8$$

$$\therefore 4 \le 10 \le 8$$

The value 10 is not between 4 and 8, so try again but this time let n = 4.

This gives

$$2^{4-1} \le 10 \le 2^4$$

$$2^3 \le 10 \le 16$$

$$8 \le 10 \le 16$$

This is true, as the value of 10 is between 8 and 16; therefore, the circuit will require four D-type latches. Note the clock will be sent to each of the four D-type latches. The outputs "Q" of each D-type latch will indicate the current state of the count, so it is not essential for the counter to require any output logic. However, if we want to use a seven-segment display, then we may need some output logic; we will look at that after we have designed the counter. The outputs will be numbered Q_0, Q_1, Q_2, and Q_3 going from left to right. This is slightly different from reading from right to left, and our circuit may look slightly different. I am doing this to try and show you the important aspect is knowing which of the Qs is the LSB and which is the MSB, especially when you draw up the Karnaugh map. Each D-type latch will require a D input, but these will not be sourced from any direct input. We will source them from the feedback signals, which will be the Q outputs of the counter.

The State Table

Step 4, in the design process, is to create a state table. This is done by allocating the first state in the process, state "A" in this case, to the first item in the "present state" part of the table. If the system was to use an external input, that would determine which state in the diagram the system would go to next; depending upon the logic of that input, there would be a column in the state table for that input. However, with the counter we are designing, there is no external input. The system would advance to the "next state" at every cycle of the synchronizing clock signal. That being the case, and as the system is just simply counting, we know what the next state of the system would be for every possible combination of the present state part. Therefore, we should be able to appreciate that the state table, for this example, is that shown in Table 6-7.

Table 6-7. *The Present State and Next State Table for the Modulo 10 Counter*

Present State				Row Number	Next State			
Q_3	Q_2	Q_1	Q_0		Q_3	Q_2	Q_1	Q_0
0	0	0	0	1	0	0	0	1
0	0	0	1	2	0	0	1	0
0	0	1	0	3	0	0	1	1
0	0	1	1	4	0	1	0	0
0	1	0	0	5	0	1	0	1
0	1	0	1	6	0	1	1	0
0	1	1	0	7	0	1	1	1
0	1	1	1	8	1	0	0	0
1	0	0	0	9	1	0	0	1
1	0	0	1	10	0	0	0	0

The difference between this table and Table 5-7 in Chapter 5 is that we do not need to consider the JK transition states, as we are not using JK flip flops for this counter.

One point I think that is important to explain is that the “present state” part is simply made up of writing down all the possible combinations of the four Q outputs. Their logic does not rely on the logic of the “next state” part. By that I mean that in row 2 the 0001 in the “present state” part is not a consequence of the logic of the “next state” part in row 1. The 0001 is simply the next value in the incrementation of the ten possible combinations of the four Q outputs; it is just a coincidence that the next state is the same as the following present state. I hope this concept, and the principle I am trying to explain, will become clearer when we look at the other example designs.

Determining the Inputs for the Four D-Type Latches

The next part of the process is to determine where each D input gets its logic from. The D inputs are numbered D_0, D_1 , D_2, and D_3. We could try to use the state diagram directly; however, the process may be more straightforward if we use the state table, as shown in Table 6-7, which indicates the "present state" and the "next state" of the outputs. This is a repetition of the process we went through with the previous counter, but I thought it best to describe it again. I will go through the process for the D_0 and D_1 inputs. I will then ask you to do the same for the D_2 and D_3 inputs. If you feel confident enough, you can try working out the logic for all four inputs and just use my description as a check.

The D_0 Input

With this system the D inputs to the latches all come from the internal logic of the Q outputs from the D-type latches themselves. We can determine where the logic for input D_0 comes from by identifying each logic "1" in the "next state" column for Q_0. Note the D_0 input directly affects only the Q_0 output. There are five rows in Table 6-7 where a logic "1" is in this column. They are in rows 1, 3, 5, 7, and 9.

Having identified the logic "1"s in the correct column, the next step is to write down the logic state of the outputs that caused this change to come about. These will be from the "present state" listings for each of the logic "1"s that have been identified. In this way a logic equation for each logic "1" in the "next state" listings for the Q_0 output is

$$D_0 = \overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0} + \overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0} + \overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0} + \overline{Q_3}.Q_2.Q_1.\overline{Q_0} + Q_3.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}$$

This expression can be implemented using a five-input OR gate fed from five four-input AND gates. However, we might be able to simplify this expression using Karnaugh maps. To make the creation of the map easier, it might be useful to write the expression down in binary format. This would produce

$$D_0 = \sum 0000 + 0010 + 0100 + 0110 + 1000$$

Using this the Karnaugh map would be that shown in Figure 6-13.

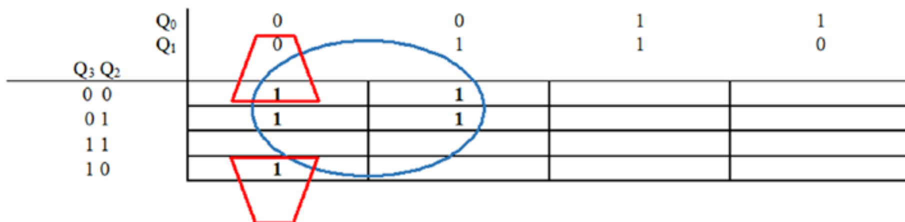


Figure 6-13. *The Karnaugh Map for the D_0 Input*

We can see from the map that the expression does minimize and it now becomes

$$D_0 = \overline{Q_3}.\overline{Q_0} + \overline{Q_2}.\overline{Q_1}.\overline{Q_0}$$

We can use Boolean algebra to show that the expression becomes

$$D_0 = \overline{Q_0}.\left(\overline{Q_3} + \overline{Q_2}.\overline{Q_1}\right)$$

This expression would require one OR gate that is fed from a single two-input AND gate, and the other input is direct from the $\overline{Q_3}$ output. The output of this OR gate feeds a two-input AND gate. The other input to the AND coming direct from the $\overline{Q_0}$. We should test this expression out to ensure that it does make the Q_0 output toggle every time the clock signal goes from low to high. The test circuit is shown in Figure 6-14.

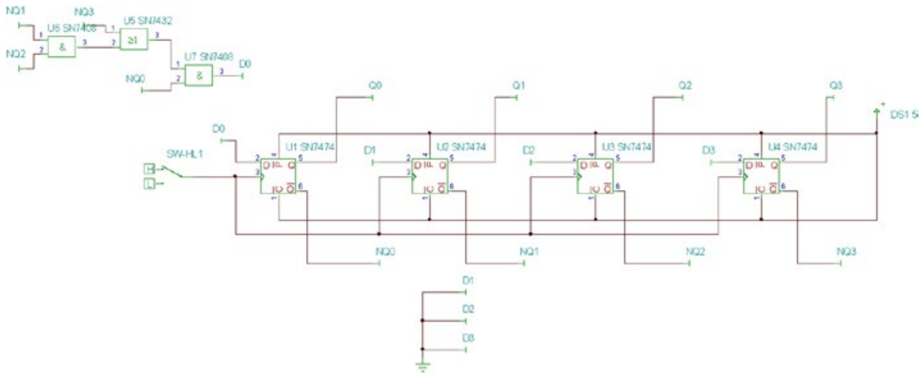


Figure 6-14. *The Test Circuit for the D_0 Input*

It is a good idea to test the parts of the circuit you design as you go along with the design. In that way you can prove that the current part of the circuit you are designing works. If you leave it until you have designed the whole circuit and there is a fault, then finding the fault on the whole circuit could be very difficult.

With this test circuit, the D inputs that are not being used are tied to 0V. If we now toggle the clock input, we will see that the Q_0 output does toggle between logic "0" and logic "1" every time the clock input goes from low to high. This is what is required, and we can now move on and determine where the D_1 input gets its logic.

The D_1 Input

If we examine the "present state" and "next state" table shown in Table 6-7, we can see that in the "next state" column for Q_1, the Q_1 is a logic "1" in rows 2, 3, 6, and 7. This means we need to look at the state of the Qs in the "present state" part for each of these rows. This will give us the following expression for the logic of the inputs to D_1:

$$D_1 = \overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0 + \overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0} + \overline{Q_3}.Q_2.\overline{Q_1}.Q_0 + \overline{Q_3}.Q_2.Q_1.\overline{Q_0}$$

This expression will require a four-input OR gate, which will be fed from four AND gates that have four inputs. We will try using Karnaugh maps to see if we can simplify the expression. That being the case, it may be easier if we write the expression for D_1 using binary format. This would be

$$D_1 = \sum 0001 + 0010 + 0101 + 0110$$

The Karnaugh map for this expression is shown in Figure 6-15.

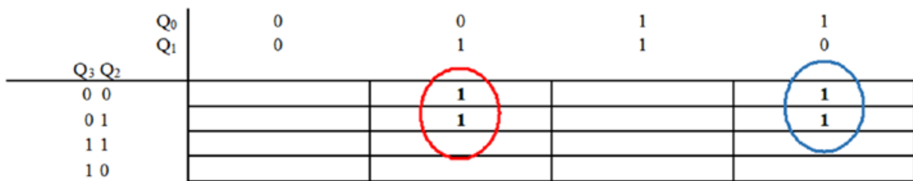


Figure 6-15. *The Karnaugh Map for the Inputs for D1*

This map has two loops, which means the expression does simplify. The simplified expression is

$$D_1 = \overline{Q_3}.Q_1.\overline{Q_0} + \overline{Q_3}.\overline{Q_1}.Q_0$$

This uses one two-input OR, which gets its inputs from two three-input AND gates.

However, again if we use Boolean algebra, this will simplify. Therefore, the expression for the D_1 input becomes

$$D_1 = \overline{Q_3}.\left(Q_1.\overline{Q_0} + \overline{Q_1}.Q_0\right)$$

This expression will use a two-input AND gate that gets one of its inputs direct from the $\overline{Q_3}$ output and the other from the output of a two-input OR gate. This OR gate gets one of its inputs from a two-input AND gate, which has Q_1 and $\overline{Q_0}$ as its inputs. The other input to the OR

gate comes from another two-input AND gate, which has $\overline{Q_1}$ and Q_0 as its inputs.

We can test this logic out by simulating the circuit shown in Figure 6-16.

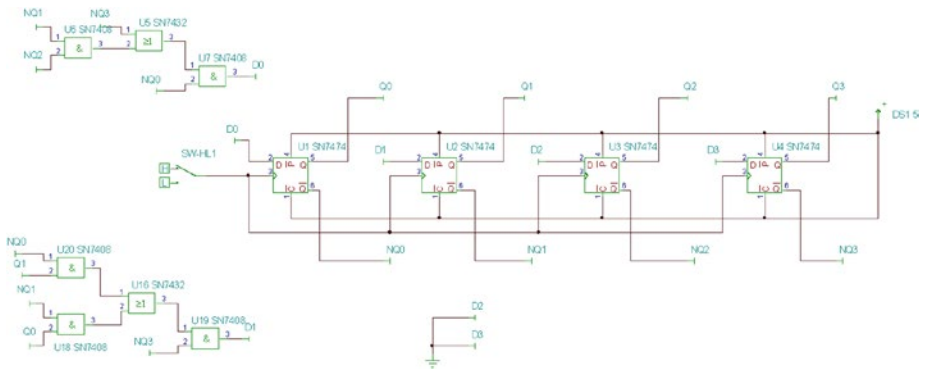


***Figure 6-16.** The Test Circuit for the D_1 Input*

When we simulate this circuit, we can see that it does work as expected.

Exercise 3

Continue with this method to show that the expressions for the inputs D_2 and D_3 are

$$D_2 = \overline{Q_3}.\left(Q_2.\overline{Q_0} + \overline{Q_1}.Q_2 + \overline{Q_2}.Q_1.Q_0\right)$$

$$D_3 = 0111,1000$$

Then continue and confirm that the circuit for the modulo 10 counter is as shown in Figure 6-17.

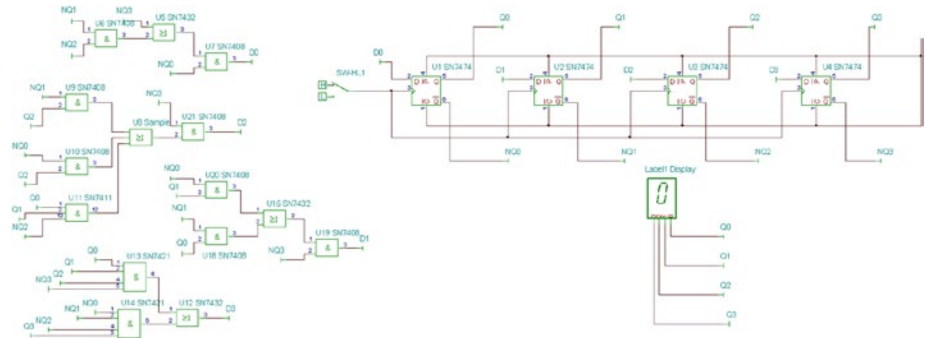


Figure 6-17. *The Completed Circuit for the Modulo 10 Counter*

State Diagram Example 3: A Bit Stream Monitor

The specification for this example is that the system will monitor a stream of bits inputted serially to the system. When the system detects three, or more, logic “1”s one after the other, the system will light a lamp to show this has happened. If the system detects a logic “0” at any time, then it will reset and the lamp will turn off.

We now need to try and create our state diagram for the design. We know there will be the following states:

State “A”: This is when the system has reset, and it will be waiting to receive three logic “1”s in a row.

State “B”: This is when the system has received one logic “1,” and it is awaiting two more logic “1”s.

State “C”: This is when the system has received two logic “1”s, and it is awaiting one more logic “1.”

State “D”: When the system has received all three logic “1”s, it is awaiting no more.

The system will remain in state "D" as long as the bits it keeps on receiving at the input are a logic "1." If it receives a logic "0," at any of the states, the system will reset and then await the next series of three or more consecutive logic "1"s.

Now that we have decided on the number of states, we can draw up the state diagram. This is shown in Figure 6-18.

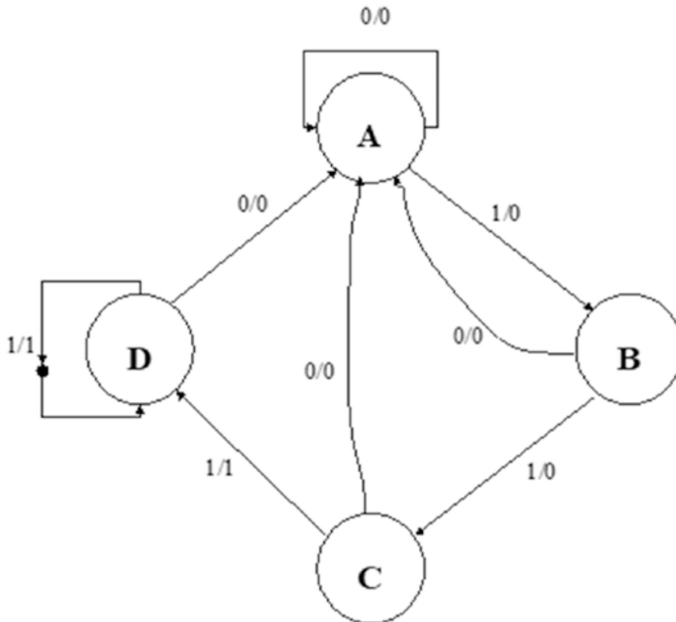


Figure 6-18. *The State Diagram for Example 3*

We should analyze the state diagram to see if it does conform to the sequence.

State A is the first state, and this is where the system should return if the system receives a logic "0" and resets. The input could be a logic "0" or a logic "1." If the input is a logic "0," then the system should remain in state "A." This is shown by the small looping arrow that just circles around state "A." The 0/0 shows what the state of the input and output is. The input logic is the first 0, and the output logic is the second "0," going from left to right.

Now if the input was a logic "1," the system should move to state "B." This is shown by the arrow going from "A" to "B." The 1/0 shows that the system had detected the first of the three logic "1"s at the input. The output would still be a logic "0" as the system is still awaiting two more logic "1"s.

Now we are at state "B." If the input now went back to a logic "0," then the system should return to state "A" where it would reset. This is shown by the arrow curving back from "B" to "A." If the input was the second logic "1," the system should move to state "C" where it would recognize the second logic "1." The output would still be a logic "0" as we are awaiting one more logic "1," hence the 1/0 on the arrow between states "B" and "C."

We are now at state "C," and the response to the input would be the same as with state "B." If the input was again another logic "1," which would be the third logic "1," the system would move to state "D." Along this arrow we would indicate the 1/1 as the input is again a logic "1" but the output should also go to a logic "1," as we have now received all three logic "1"s one after the other. However, it is only when we are at state "D" that the output should respond. Also, while at state "C" if we received a logic "0," the system would move back to state "A" and reset back to 0. This is shown by the arc from "C" to "A."

Now at state "D" if the next input was again a logic "1," the system should stay at state "D," and the output should stay at a logic "1." This is indicated by the small loop that circles around state "D." If the input went to a logic "0," then the system should move to state "A" and reset, that is, the output should turn off and the system should wait for three more logic "1"s at the input, one after the other.

This analysis shows that our state diagram is correct, so we can now move on with our design.

The next step is to assign binary values to the states in the state diagram. This is so that you can create expressions for the states and use them in the design process.

There are four states, and so we will assign the following four binary values as shown in Table 6-8.

Table 6-8. *Assigning the Binary Values to the Four States*

State	Q_1	Q_0
A	0	0
B	0	1
C	1	0
D	1	1

Having the binary values, we can now determine how many flip flops we will need to create the logic circuit to our design. We will need two Q outputs as two will enable the outputs to mimic the binary value for the four states.

We could use this expression to determine the number of flip flops:

$$2^{n-1} \leq N \leq 2^{n}$$

where "N" is the number of states in the state diagram, in this case N = 4.

Little "n" is the number of flip flops needed to create the circuit. The process is to use trial and error to determine the value for "n" that conforms to the preceding expression.

Try n = 1.

This gives

$$2^{n-1} \leq N \leq 2^{n} 2^{1-1} \leq 4 \leq 2^{1} \therefore 2^{0} \leq 4 \leq 2 \therefore 1 \leq 4 \leq 2$$

The value 4 is not between 1 and 2, so try again but this time let n = 2.

This gives

$$2^{2-1} \le 4 \le 2^2$$

$$2^1 \le 4 \le 4$$

$$2 \le 4 \le 4$$

This shows that we can use two flip flops for the design. Really, I would hope you can easily see that we need two flip flops without using this expression. However, as the number of states gets larger, then it might be useful to use this expression.

Now we can create the state table, which will indicate the "present" and "next" states of all variables. The table is shown in Table 6-9.

Table 6-9. *The State Table for Example 3*

Row	Present State			Next State		Output
	Q_1	Q_0	I	Q_1	Q_0	
1A	0	0	0	0	0	0A
2A	0	0	1	0	1	0B
3B	0	1	0	0	0	0A
4B	0	1	1	1	0	0C
5C	1	0	0	0	0	0A
6C	1	0	1	1	1	0D
7D	1	1	0	0	0	0A
8D	1	1	1	1	1	1D

Even though there are only four states in the state diagram, there are eight possible combinations of the "present state," as we need to include the state of the input. This is because for each state in the state diagram, the input could be either a logic "1" or a logic "0." This is the principle I was trying to show you when we analyzed the present state and next state table in Table 6-7.

The "present state" of the table shows us, with the values for Q_1 and Q_0, each of the states shown in the diagram, monitoring what logic the input

would go to. The "next state" of the table indicates where we would go in the state diagram in response to the input, as stated in the "I" column, and the present state of the outputs Q_1 and Q_0.

The state of the Qs and the I in the present state columns follow the standard increment of the 3-bit binary number.

The Row column indicates the number of the row but also the position in the state diagram we would be at with the present state of the Qs. The Output column indicates the logic of the output as well as the position in the state diagram we would be at with the next state of the Qs.

Row 1 is when we are in the first state of the design and the input has gone to a logic "0." The state diagram shows that we would stay in the first state "A." Therefore, the "next state" of the Qs would be as shown in row 1, that is, 0 0. The output must still be 0 as no logic "1"s have been presented to the system.

Row 2 is when we are in the first state of the design and the input has gone to a logic "1." The state diagram shows that we should move to the second state "B" in the diagram. This would mean that the Qs would go to 0 1 as shown in row 2 in the "next state." Now, as we have only received one logic "1," the output should stay at 0.

Row 3 now allows us to consider how the system would respond to the inputs if we were in the second state "B" of the system. If the input went to a logic "0," as shown in row 3, the system would reset and we would move to the first state, "A" in the system. This is reflected in the value of the Qs being 0 0 in the "next state." The output must still be a "0" as the system has reset.

Row 4 shows how the system would respond if, while we are at "B," the input went to a logic "1." This would allow the system to move to the third state, "C," in the system. This is indicated by the "next state" Qs going to 1 0. As the system has now received two logic "1"s, the output would still be "0."

Row 5 shows us that if we are in the third state C and we receive a logic "0," we would reset the system again and move back to the first state "A" of

the system. This is reflected by the 0 0 in the “next state” part and the 0 in the Output column.

Row 6 shows us that if the input went to a logic “1” while we were at “C” in the diagram, we would move to the fourth state “D” of the system. The “next state” part shows the Qs going to 1 1, which will put the system in state “D.” The output is shown as still being a logic “0” because it is not until the Qs actually both go to a logic “1” in the “present state” that the output can change to a logic “1.” The simulation may help us confirm this concept.

Row 7 shows us that the system would reset and move back to “A” in the system if the input went to a logic “0” while we were in “D.” It also shows that the output would go out, that is, back to a logic “0” as the system would reset.

Row 8 shows us that if the input went to a logic “1” while we are in “D,” the system would stay in “D,” and the output would stay on at a logic “1.”

I hope this analysis explains what the state table is trying to show us and so how you should go about creating a state table.

There are no states that are repeated, and so the system cannot be simplified.

As we are using D-type flip flops, there is no real need to create an excitation table. This is because we can determine where each D-type flip flop gets its inputs from directly from the state table shown in Table 6-9.

The D_0 Inputs

All we need to do is for every logic “1” in the “next state” of the Q_0 outputs, write down the expression for the state of the terms in the “present state” part. The first logic “1” in the Q_0 “next state” column is in row 2. There are three variables in the “present state” part, and the logic for each variable is 0 0 1. Note we must include the logic of the input “I” as this, along with the

Qs of the D-type latches, will affect the logic of the D inputs. This gives the first term for the D_0 expression:

$$\overline{Q_1}.\overline{Q_0}.I$$

The next logic "1" for Q_0 in the "next state" part is in row 6, and the state of the variables in the present state part is 1 0 1. This gives the second term for the D_0 expression:

$$Q_1.\overline{Q_0}.I$$

The final logic "1' for Q_0 in the "next state" is in row 8, and the state of the variables in the present state part is 1 1 1. This gives the third and final term for the D_0 expression:

$$Q_1.Q_0.I$$

The complete expression for D_0 is the OR of all three terms. This means that the expression for D_0 is

$$D_0 = Q_1.Q_0.I + Q_1.\overline{Q_0}.I + \overline{Q_1}.\overline{Q_0}.I$$

$$D_0 = \sum 111,101,001$$

The D_1 Expression

We do the same as for the D_0 except that we are using the Q_1 column in the "next state" part. The first logic "1" is in row 4, and the logic for the variables in the present state part is 0 1 1. This means the expression for the first term is

$$\overline{Q_1}.Q_0.I$$

The next logic "1" is in row 6, and the logic of the variables is 1 0 1. The expression for this term is

$$Q_1.\overline{Q_0}.I$$

The third and final term is in row 8, and the logic for the variables in the present state part is 1 1 1. The expression for this term is

$$Q_1.Q_0.I$$

Therefore, the expression for the D_1 input is

$$D_1 = Q_1.Q_0.I + Q_1.\overline{Q_0}.I + \overline{Q_1}.Q_0.I$$

$$D_1 = \sum 111,101,011$$

We now need to determine the expression for the output "F." There is only one row where there is a logic "1" in the output "F" in the "next state" part, and this is in row 8. This means the expression for the output "F" is

$$F = Q_1.Q_0.I$$

Now we should try and minimize the three expressions. We will try using Karnaugh maps to do that.

The Karnaugh map for the D_0 input is shown in Figure 6-19.

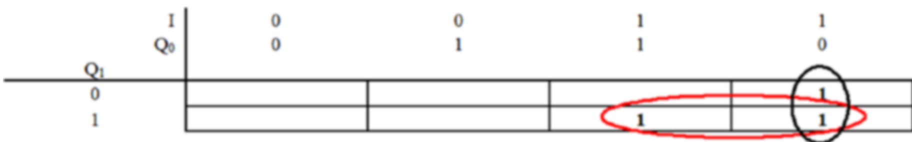


Figure 6-19. *The Karnaugh Map for the D_0 Inputs*

Using this Karnaugh map, the expression for D_0 becomes

$$D_0 = Q_1.I + \overline{Q_0}.I$$

Using Boolean algebra this can be further minimized to

$$D_0 = I\left(Q_1 + \overline{Q_0}\right)$$

The Karnaugh map for D_1 is shown in Figure 6-20.

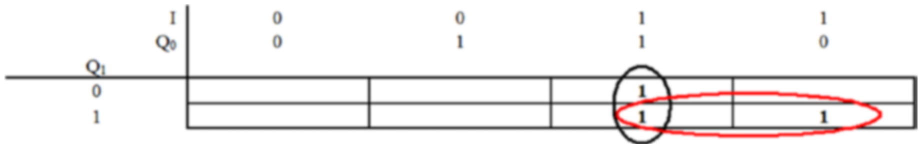


Figure 6-20. *The Karnaugh Map for the D_1 Inputs*

Using this map, the expression for D_1 can be minimized to

$$D_1 = Q_1.I + Q_0.I$$

Using Boolean algebra this can be further minimized to

$$D_1 = I\left(Q_1 + Q_0\right)$$

The expression for the output “F” cannot be minimized.

Now that we have the expressions for all the internal inputs and the output, we can create the circuit for the system. The circuit is shown in Figure 6-21.

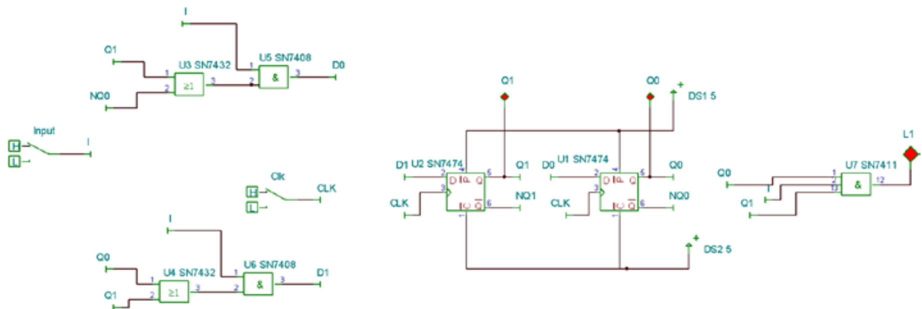


Figure 6-21. *The Complete Circuit for Example 3*

When we simulate this circuit, we will see that the output lamp will come on only after the system has had three consecutive logic "1"s at its input. This confirms our design works correctly.

State Diagram Example 4

In this example we will use the same specification as in Example 3, that is, wait for three successive logic "1"s at the input and then light an output lamp. This means we will have a similar state diagram, but we will move around it in the opposite direction. The state diagram is shown in Figure 6-22.

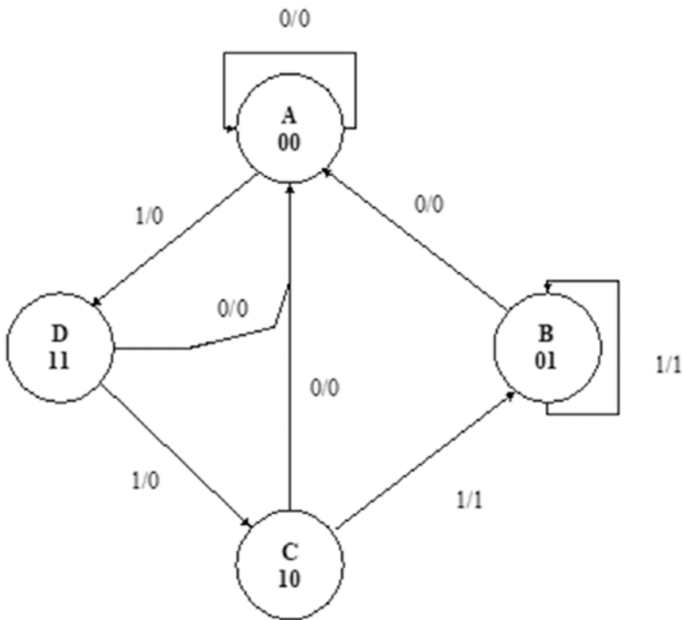


Figure 6-22. *The State Diagram for Example 4*

We will stay with the same assignment of binary values for the four states as shown again in Table 6-10.

Table 6-10. The Binary Values for the Four States

State	Q_1	Q_0
A	0	0
B	0	1
C	1	0
D	1	1

The difference is how the system will move from the "present state" to the "next state" in response to the input "I." The state table, shown in Table 6-11, will show these differences.

Table 6-11. The State Table for Example 4

Row	Present State			Next State		Output
	Q_1	Q_0	I	Q_1	Q_0	
1 A	0	0	0	0	0	0 A
2 A	0	0	1	1	1	0 D
3 B	0	1	0	0	0	0 A
4 B	0	1	1	0	1	1 B
5 C	1	0	0	0	0	0 A
6 C	1	0	1	0	1	1 B
7 D	1	1	0	0	0	0 A
8 D	1	1	1	1	0	0 C

The state of the Qs and the I in the present state columns follow the standard increment of the 3-bit binary number. You need to refer to the state diagram shown in Figure 6-22 to see how the "next state" columns show they are the correct response of the system to the "present state" logic. Along with the row number, I have indicated the state position we

would be in at the present state. With the Output column, I have indicated the state position we would be in at the next state.

To help you understand Table 6-11, we will look at row 2. The present state shows that we have $Q_1 = 0$, $Q_0 = 0$, and I = 1. This means we are at state A when we get a logic "1" at the input. This would allow the system to move to state D. This is reflected in the next state part by the $Q_1 = 1$ and $Q_0 = 1$. This means we would be at state D as indicated by the Output column, 0D. The output would still be off as this would be the first logic "1" at the input.

If we look at row 4, the present state shows us that we are at state B when we get a logic "1" at the input. The system should remain at state B with the output lit. This is reflected in the next state part and Output column of the row.

I hope the analysis of these two rows helps you understand how the table is constructed.

Now that we have completed the state table, we should be able to create the expressions for the D_0 and D_1 inputs as well as the output "F." These are created in the same way.

The D_0 Expression

As before, all we need to do is for every logic "1" in the "next state" of the Q_0 outputs, write down the expression for the state of the terms in the "present state" part. The first logic "1" in the Q_0 "next state" column is in row 2. There are three variables in the "present state" part, and the logic for each variable is 0 0 1. This gives the first term for the D_0 expression:

$$\overline{Q_1}.\overline{Q_0}.I$$

I hope I do not have to explain the other terms as they are determined using the same process. Therefore, the second term is

$$\overline{Q_1}.Q_0.I$$

Similarly, the third term is

$$Q_1.\overline{Q_0}.I$$

Therefore, the complete expression for the D_0 input is

$$D_0 = \overline{Q_1}.\overline{Q_0}.I + \overline{Q_1}.Q_0.I + Q_1.\overline{Q_0}.I$$

$$D_0 = \sum 001,011,101$$

If we carry out a similar process, we can see that the expression for the D_1 input is

$$D_1 = \overline{Q_1}.\overline{Q_0}.I + Q_1.Q_0.I$$

$$D_1 = \sum 001,111$$

Using the state table, the expression for the output "F" is

$$F = I.\overline{Q_1}.Q_0$$

However, it might be easier to realize that this is the expression for the output because after the third consecutive logic "1," the system would be at state "B" in the state diagram. This is due to the fact that we are rotating anticlockwise around the diagram as shown in Figure 6-22.

Now we can draw up the Karnaugh maps to try and minimize the expressions. The Karnaugh map for the D_0 input is shown in Figure 6-23.

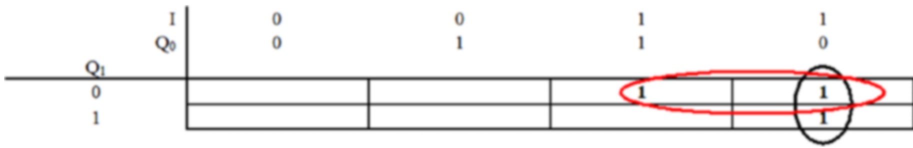


Figure 6-23. *The Karnaugh Map for the D_0 Input for Example 4*

Using this map, the expression for the D_0 input becomes

$$D_0 = \overline{Q_0}.I + \overline{Q_1}.I$$

Using Boolean algebra this simplifies to

$$D_0 = I.\left(\overline{Q_0} + \overline{Q_1}\right)$$

The map for the D_1 input is shown in Figure 6-24.

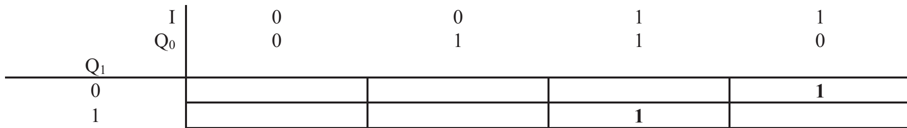


Figure 6-24. *The Karnaugh Map for the D_1 Input for Example 4*

This shows that the D_1 expression does not simplify. Therefore, using the Karnaugh map we have

$$D_1 = \overline{Q_1}.\overline{Q_0}.I + Q_1.Q_0.I$$

However, using Boolean algebra this will simplify to

$$D_1 = I.\left(\overline{Q_1}.\overline{Q_0} + Q_1.Q_0\right)$$

The expression for the output does not simplify.

Now we can create the circuit to test the design for this second sequence detector. The circuit is shown in Figure 6-25.

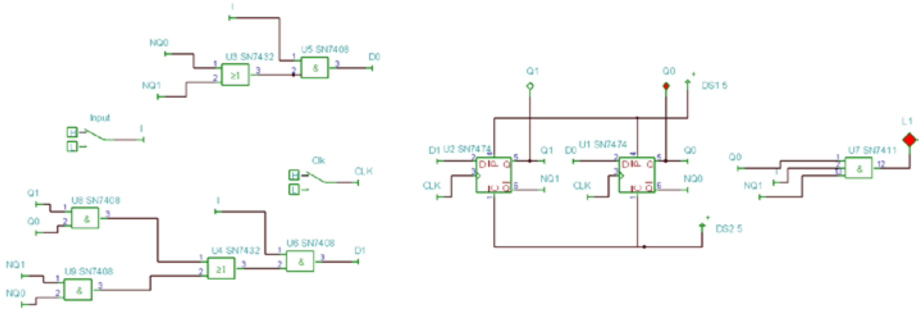


Figure 6-25. *The Circuit for Example 4*

When we simulate this circuit, we will see that the output lamp will come on only after the system has had three consecutive logic 1s at its input. This confirms our design works correctly.

Exercise 4

Using the same sequence specification as that for Example 3, show that if the state diagram was changed to that shown in Figure 6-26, the actual design circuit for the sequence detector would be that shown in Figure 6-27.

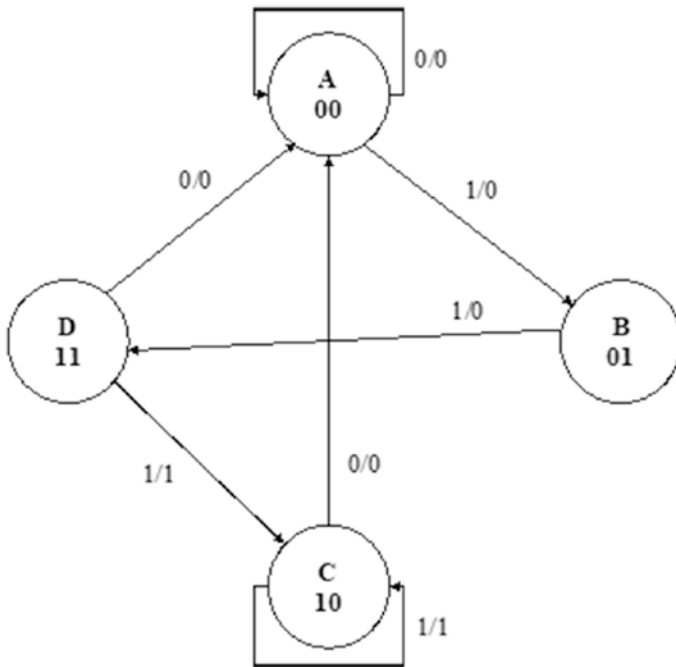


Figure 6-26. *The State Diagram for Exercise 4*

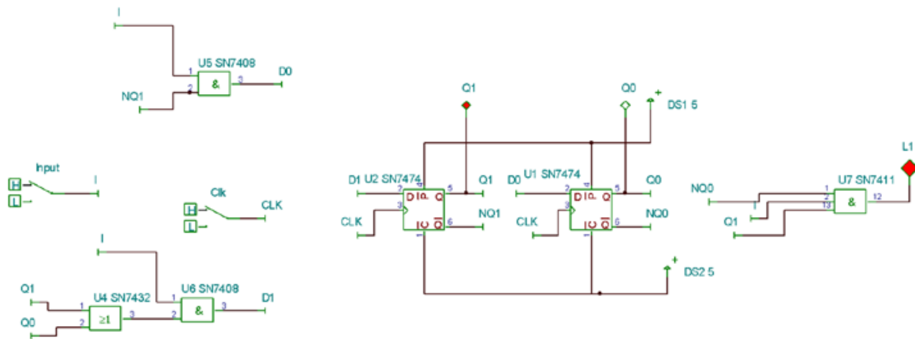


Figure 6-27. *The Circuit for Exercise 4*

To help you the expressions used in the circuit are

$$D_0 = I.\overline{Q_1}$$

$$D_1 = I.(Q_1 + Q_0)$$

$$F = I.(Q_1 + \overline{Q_0})$$

Moore's and Mealy Diagrams

There are two recognized formats for drawing up state diagrams and for representing logic systems. These are called Moore's and Mealy representations. We have already learned that the Mealy state diagram is one that adds both the input states and the output state on the transitional lines of a state diagram; see the reference to the JK flip flop state diagram construction. The state diagram of the JK flip flop is shown again in Figure 6-28 to show you what a Mealy state diagram looks like.

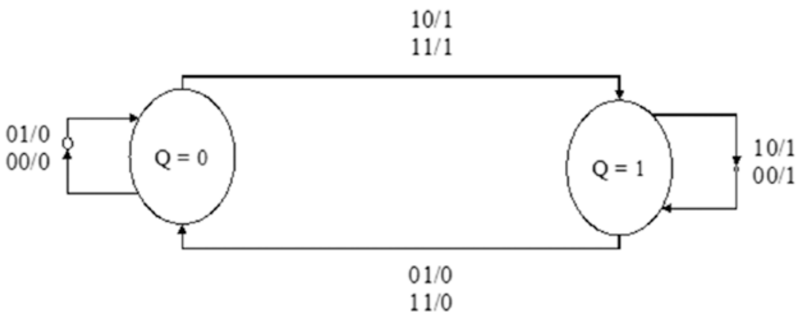


***Figure 6-28.** The Mealy State Diagram for the JK Flip Flop*

With Moore's state diagram, only the input logic is shown along the transitional lines. The output logic of each of the states is shown inside the circles along with the state identification symbol.

Figures 6-28 and 6-29 show the main difference between Moore's and Mealy state diagrams.

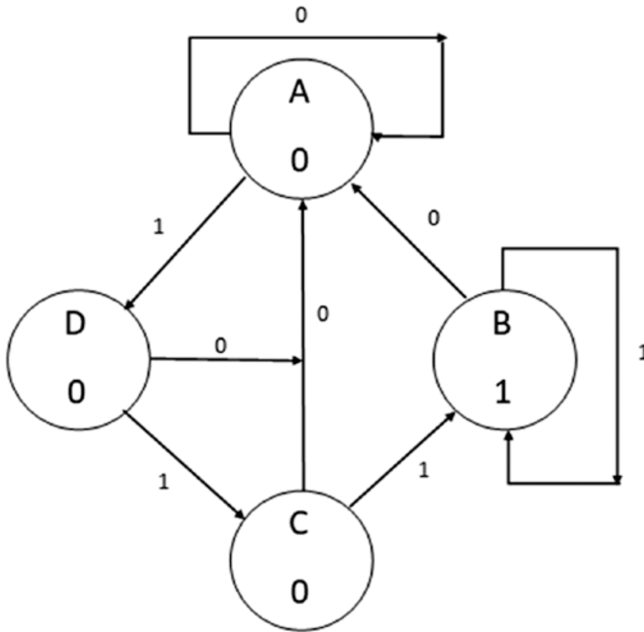


Figure 6-29. *Moore's State Diagram for Exercise 4*

The other difference between Moore's and the Mealy systems is in the way they show how the output is related to the input. With Moore's system there is no direct link between the output and the input to the system. This is reflected in the way the system is represented. Figure 6-30 shows how a Moore's system may be represented. You can see that the input does not directly affect the output.

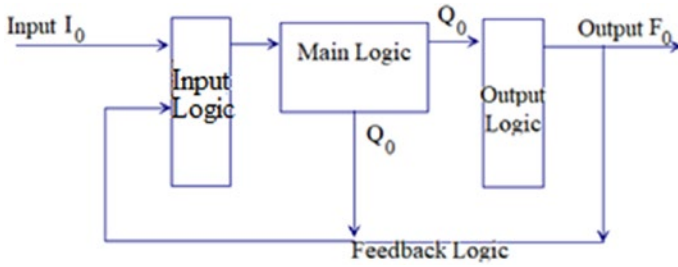


Figure 6-30. *A Representation of a Moore's System*

With a Mealy system the input does have a direct effect on the output. This would be represented on a Mealy diagram in a similar fashion to the system shown in Figure 6-31.

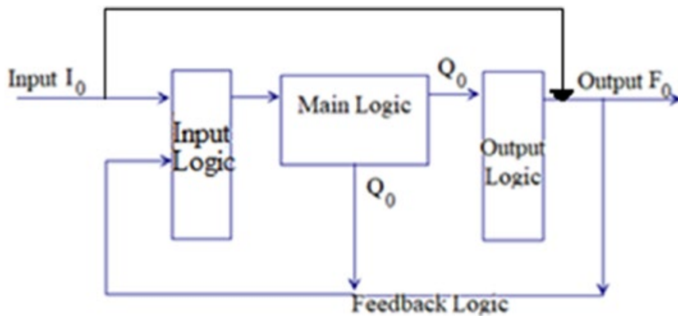


Figure 6-31. *A Mealy Representation of a System*

Summary

In this chapter we have studied some of the fundamental aspects of state diagrams. We have studied how they could be produced from a set of data, and we have studied how they can be used as an aid to designing logic circuits.

In the next chapter, we will look at combinational logic and how we can create some useful combinational logic circuits such as the half adder, the full adder, the subtractor, etc.

CHAPTER 7

Combinational Logic

In this chapter we will look at combinational logic circuits. These are circuits that respond simply to the state of their inputs at that moment in time. They don't care about what has happened before as with sequential logic circuits. There are a range of useful combinational logic circuits that make up the basis of devices today, and we will study how they work. They will include

- The half adder
- The full adder
- The multiplexer

The Tri-state Buffer

Before we go too far into the world of combinational logic, we will study a useful device that can be used in both combinational and sequential logic. This is the tri-state buffer. As its name suggests, this device has three states; well, it certainly has three pins. A single tri-state buffer is shown in Figure 7-1.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_7

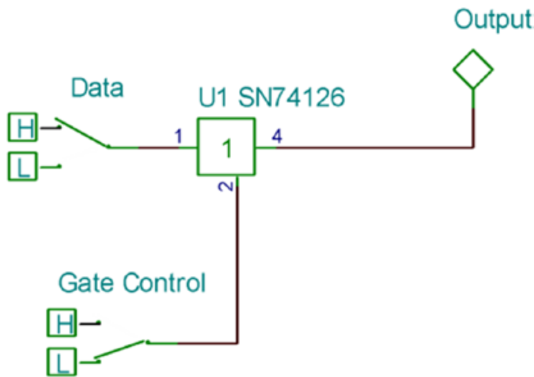


Figure 7-1. *The Tri-state Buffer*

The three terminals are the data input, the output, and the gate control. The three states are where the output is high, low, or totally disconnected. The principle by which it works is that if the gate is switched to disable the buffer, which for the 74126 is switched to a logic "0," then the output would be disconnected, and it will not respond to the logic of the input. If the buffer is enabled, by driving the gate control to a logic "1" for the 74126, then the output would take on the logic of the input. This is reflected in the truth table for the 74126 tri-state buffer. This is shown in Table 7-1.

Table 7-1. *The Truth Table for the 74126 Tri-state Buffer*

Gate Control	Input	Output
0	0	High impedance
0	1	High impedance
1	0	0
1	1	1

It is normal to buy these with four buffers in the one IC. This is the 74126 quad-tri-state buffer as shown in Figure 7-2.

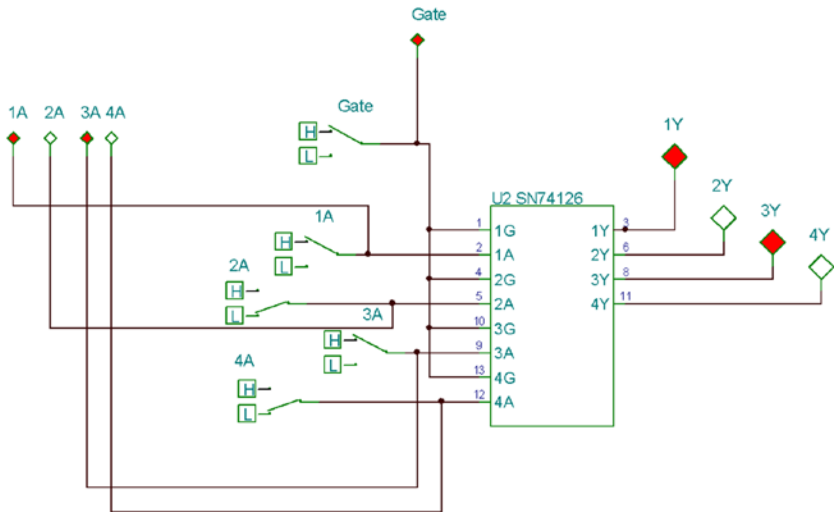


Figure 7-2. *The 74126 Quad-tri-state Buffer*

If you simulate this circuit, you will see that when the gate signal is high, at a logic "1," as shown in Figure 7-2, the output takes on the state of the input data. The input data is controlled by the four H/L switches 1A to 4A. If you switch the gate to the low state, a logic "0," you should see that the outputs do not respond to the state of the four inputs.

The 74125 is a quad-tri-state buffer, but the gate control logic is reversed; a logic "0: enables the buffer to respond to the inputs, whereas a logic "1" disables it.

The Half Adder Circuit

This is the first fundamental logic circuit we will look at. It is called the half adder as it really only does half the job in that it simply adds to binary digits and it does not care if the previous addition caused a carry to be passed on to be added as well. The output will be the sum of the two binary digits and there will be a carry produced, but the carry is not passed on if

the next adder is again the half adder. You should appreciate that if we add 4 to 7, we get a 1 that is left in the units column and a 1 that is passed on to the next column, that is, the tens column. If we were using just a half adder, then that carry would not be passed on to the tens column. An example of this in binary would be the addition of 1 + 1 in binary. The result would be a 0 with a 1 to carry on to the next column. However, if we were using half adders for all the columns, the carry digits would be lost. This means that really the half adder is not much use. However, it was the first step in creating the full adder, which we will look at next.

Before we look at the circuit for the half adder, we should try and come up with the Boolean expression for the circuit. To do this we need to think of what we want from the circuit. We will need two outputs: one that is the sum of A and B and one that is the carry that is produced when the sum is greater than the maximum value we can have in the column we are working in. To help it would be useful to reiterate the addition of two binary digits A and B. This is done in Table 7-2.

Table 7-2. *The Basic Addition of Two Binary Digits*

A	+	B	Result
0	+	0	0 and no carry
0	+	1	1 and no carry
1	+	0	1 and no carry
1	+	1	0 and 1 to carry

There is no need to include a carry from any previous column as this is the half adder. Really there is no need to record the carry, as we won't be passing it on to the next column; however, the basic addition will naturally produce a carry.

We can present a better table from Table 7-2 to show the sum and carry result from the addition of A and B, which will be the truth table for the half adder. This is shown in Table 7-3.

Table 7-3. *The Truth Table for the Half Adder*

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Using Table 7-3 we can see that there are two combinations when the sum results in a "1." The expression for the first is

$$Sum = \overline{A}.B$$

The expression for the second combination is

$$Sum = A.\overline{B}$$

Therefore, the complete expression for the sum is

$$Sum = \overline{A}.B + A.\overline{B}$$

This is the Boolean expression for EXOR gate.

There is only one combination when there's a "1" in the Carry column. The expression for this is

$$Carry = A.B$$

Therefore, knowing this we can create the circuit for the half adder as shown in Figure 7-3.

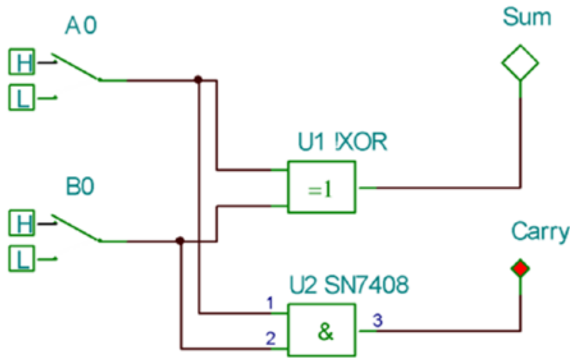


Figure 7-3. *The Circuit for the Half Adder*

If you simulate the circuit, you will see that it does work as the truth table suggests it should.

The Design of the Full Adder Circuit

We can extend this concept to try and design the circuit for the full adder. The difference with the full adder is that it must respond to a carry that may or may not be carried forward from the previous column. The possibilities that this will produce are shown in Table 7-4.

Table 7-4. *The Addition of Two Binary Digits with a Carry In*

A	B	Carry In	Result
0	0	0	0 and no carry forward
0	0	1	1 and no carry forward
0	1	0	1 and no carry forward
0	1	1	0 and 1 to carry forward
1	0	0	1 and no carry forward
1	0	1	0 and 1 to carry forward
1	1	0	0 and 1 to carry forward
1	1	1	1 and 1 to carry forward

Again, we can present a better table from Table 7-4 to show the sum and carry result from the addition of A and B with a carry in. This will be the truth table for the full adder. This is shown in Table 7-5.

Table 7-5. *The Truth Table for the Sum and Carry from the Full Adder*

A	B	Carry In (CIN)	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Using Table 7-5, we can try and build up an expression for the sum and the carry out. Using the table, we see that there are four combinations that will produce a logic "1" in the Sum column. They are

$$\overline{A}.\overline{B}.CIN$$

$$\overline{A}.B.\overline{CIN}$$

$$A.\overline{B}.\overline{CIN}$$

$$A.B.CIN$$

Therefore, the expression for the sum is

$$Sum = \overline{A}.\overline{B}.CIN + \overline{A}.B.\overline{CIN} + A.\overline{B}.\overline{CIN} + A.B.CIN$$

To help with a Karnaugh map, we could represent this expression in binary format. This would produce

$$Sum = \sum 001,010,100,111$$

Using this expression, we can fill the Karnaugh map as shown in Figure 7-4.

CIN	0	0	1	1
B	0	1	1	0
A				
0		**1**		**1**
1	**1**		**1**	

Figure 7-4. *The Karnaugh Map for the Sum Expression*

This shows that the expression cannot be simplified with Karnaugh maps. However, we can try using Boolean algebra. This will produce

$$Sum = \overline{A}.\overline{B}.CIN + \overline{A}.B.\overline{CIN} + A.\overline{B}.\overline{CIN} + A.B.CIN$$

Rearranging, this gives

$$Sim = \overline{A}.\overline{B}.CIN + A.B.CIN + \overline{A}.B.\overline{CIN} + A.\overline{B}.\overline{CIN}$$

Now taking out some common factors, we have

$$Sum = CIN.\left(\overline{A}.\overline{B} + A.B\right) + \overline{CIN}.\left(\overline{A}.B + A.\overline{B}\right)$$

If we consider the variables "A" and "B" and study the truth tables for a two-input logic – OR, NOR, EXOR, and NOT EXOR gates – as shown in Table 7-6, we might be able to consider what logic gates we could use to implement the sum function.

Table 7-6. *The Truth Table for the Inputs A and B Ignoring the And and NAND Functions*

Row	A	B	OR	NOR	EXOR	NOT EXOR
1	0	0	0	1	0	1
2	0	1	1	0	1	0
3	1	0	1	0	1	0
4	1	1	1	0	0	1

Rows 2 and 3 are the response of the EXOR gate, what I call the true OR function. Using those two rows, we can see that the Boolean expression for the EXOR is

$$F = \overline{A}.B + A.\overline{B}$$

We can see that this is the second part of the simplified expression for the sum. Therefore, we can say that

$$Sum = CIN.\left(\overline{A}.\overline{B} + A.B\right) + \overline{CIN}.\left(A \otimes B\right)$$

If we consider rows 1 and 4 of Table 7-6, we can interpret the function $\overline{A}.\overline{B} + A.B$ as being the NOT of the EXOR of A and B. This is shown in the NOT EXOR column of Table 7-6.

This means that the expression for the sum now becomes

$$Sum = CIN.\left(\overline{A \otimes B}\right) + \overline{CIN}.\left(A \otimes B\right)$$

There is obviously an EXOR gate, and we could create a NOT EXOR gate by feeding the output of an EXOR gate to a NOT gate. However, if we look at a circuit that feeds an EXOR gate with a variable and the output of a previous EXOR gate, we might find something useful from the output. The test circuit to test this principle is shown in Figure 7-5.

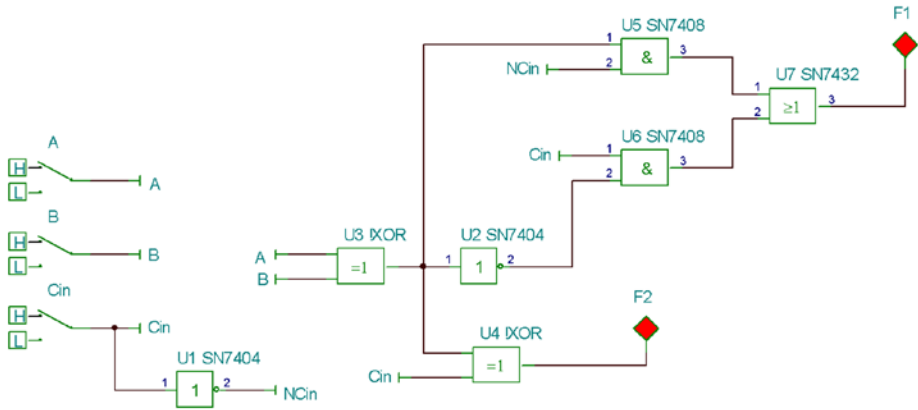


Figure 7-5. *The Test Circuit*

The gates U2, U3, U5, U6, and U7 make up the logic circuit that complies with the expression for the sum we have derived and feed the output F1. The gates U3 and U4, which are simply two EXOR gates, with the output F2, make up an alternative circuit. When we simulate the circuit, we see that the two outputs work in the same way. This means that both circuits will perform the sum function we require for the full adder. If we simulate the circuit, we should be able to fill out the truth table. This is shown in Table 7-7.

Table 7-7. *The Truth Table for the Test Circuit*

A	B	Cin	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

If we examine Table 7-7, we can see that the output "F" gives the same result as that required for the sum function in Table 7-5. This suggests that the test circuit will perform the sum function of the full adder.

If we derive the Boolean expression for the test circuit that feeds just the F2 output, shown in Figure 7-5, we may confirm that it performs the sum function we require. To obtain the Boolean expression, start from the output "F2" and work backward toward the inputs. We can see that the first gate is an EXOR gate and the two inputs are CIN and M. Therefore, this gives the expression for F as

$$F2 = CIN \otimes M$$

We can expand on the EXOR function to say that

$$F2 = CIN.\overline{M} + \overline{CIN}.M$$

The CIN input is direct from the input switch, but the M input is the output of another EXOR gate. The two inputs to that EXOR gate are A and B. Therefore, we can say that

$$M = A \otimes B$$

Therefore

$$M = \overline{A}.B + A.\overline{B}$$

Now if we substitute this into the expression for F, we get

$$F2 = CIN.\overline{\overline{A}.B + A.\overline{B}} + \overline{CIN}.\left(\overline{A}.B + A.\overline{B}\right)$$

Or we can say

$$F2 = CIN.\left(\overline{A \otimes B}\right) + \overline{CIN}.\left(A \otimes B\right)$$

This is the same expression as for the sum, and so this confirms that the test circuit will provide the sum function for the full adder.

If we now consider the carry out function, then using Table 7-5, we can see that the expression for the carry out, that is, CO, is

$$CO = \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C = \sum 011,101,110,111$$

We can try using a Karnaugh map to simplify this as shown in Figure 7-6.

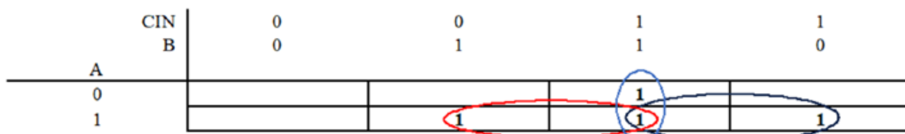


Figure 7-6. *The Karnaugh Map for the CO*

$$Carry = A.CIN + B.CIN + A.B$$

$$Carry = CIN.\left(A + B\right) + A.B$$

To implement this, we would require an OR gate that has both inputs fed from two-input AND gates: one AND gate using just the A and B inputs and the other AND gate having one input from the CIN but the other input being fed from an OR gate of A and B. However, what we really want from this second OR gate is the true OR; we don't require the AND function that the inclusive OR gate gives us. We really want the exclusive OR function that the EXOR gate gives us. I am making this point because the sum function of the full adder already has an EXOR gate that is using the inputs A and B. To clarify this point, we can simulate the second test circuit shown in Figure 7-7 to see if it performs the carry out function we require.

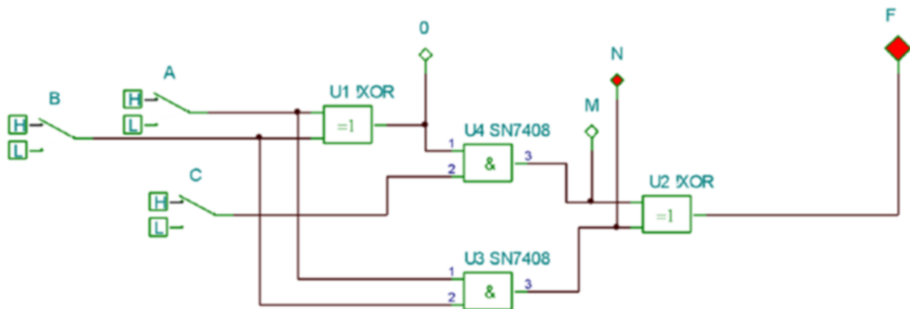


***Figure 7-7.** The Test Circuit for the Carry Out Function*

If we simulate the circuit shown in Figure 7-7, we will be able to complete the truth table for this test circuit as shown in Table 7-8. Then use that table to derive the expression for the output F, which is the carry out.

Table 7-8. *The Truth Table for the Test Circuit Shown in Figure 7-7*

A	B	C	Carry Out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

The expression for the carry out that can be obtained from this table is

$$Carry\ Out = \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$$

This confirms that the second test circuit does perform the function of the carry out operation for the full adder.

Exercise 1

As an exercise, derive the expression for the circuit shown in Figure 7-8 and confirm that it does perform the function for the carry out in the full adder.

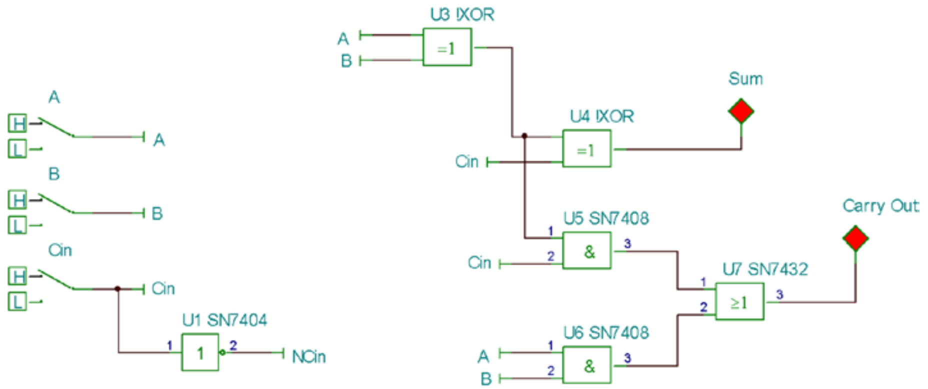


Figure 7-8. *The Test Circuit for the Full Adder*

A 3-Bit Full Adder

The circuit shown in Figure 7-9 shows the circuit for a 3-bit full adder. I have restricted it to 3 bits as to produce an 8-bit adder would simply be too big a circuit. If you simulate the circuit, you will see it does perform the full addition with carry.

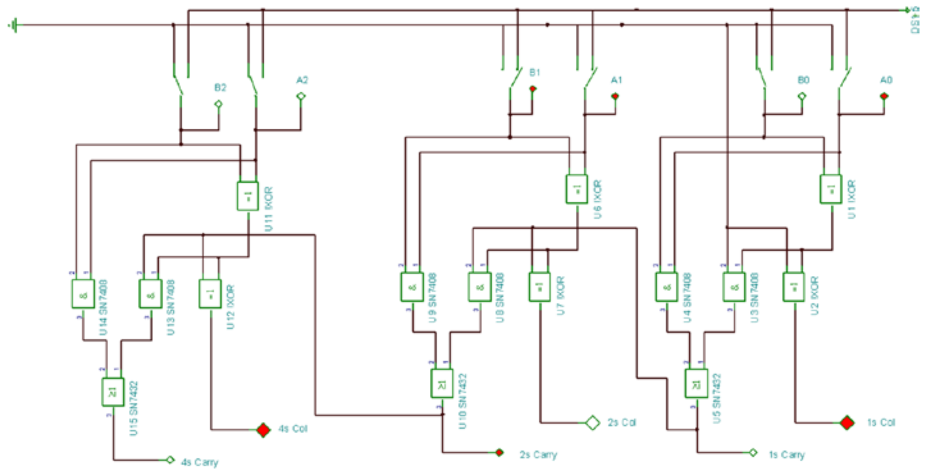


Figure 7-9. *The 3-Bit Full Adder*

I have rotated the circuit as shown, to try and show you that each individual full adder circuit represents one column in the 3-bit binary number. The columns are numbered 1s Col, 2s Col, and 4s Col, as that is what the columns 1, 2, and 3 are in binary; see Chapter 1. I hope you can see that the A input is 011, which is 3 in decimal, and the B input is 010, which is 2 in decimal. The result is 101, which is 5 in decimal, and 2 + 3 = 5. I hope this shows you that the 3-bit full adder works as expected. Also, we can increase the number of bits we can add by simply increasing the number of full adder circuits we cascade together in this fashion.

The Binary Subtractor Circuit

The natural follow-on from the full adder would be to look at creating a subtractor logic circuit. We will start designing the subtractor circuit in the same way as we designed the adder.

Before we can design a circuit, we must fully understand what we are trying to achieve. We are used to subtracting with our denary system. If the subtrahend is larger than the minuend, then we simply borrow a 1 from the column to our left. As an example, consider 12 – 7. This is shown in Figure 7-10.

1 [1]2
[1] 7
0 5

Figure 7-10. *A Subtraction Using the Denary Number System*

We must appreciate that the “1” we have borrowed is actually a value of 10, as it is borrowed from the tens column, making the 2 equal to 12 and so enabling us to subtract 7 from it. Of course, we must pay back what we borrow, so we send the borrow out to the next column; we must remember

that the “1” in this column is worth the 10 in the units column that we borrowed. The tens column now has s the 1 – 1 equaling 0. The answer is the “5” we expect.

This method works for the denary system; therefore, we should be able to employ a similar process with the binary number system, as with any number system.

An example of this is shown in Figure 7-11.

```
  11 11 11
 1  1  1  0
 1  1₁ 1₁ 1
-----------
 0  1  1  1
```

Figure 7-11. *A Subtraction Using the Binary Number System*

With the binary system, the “1” we borrow from the next column is twice the value of the current column, hence why we have two 1s as the upper borrow. This means that the first subtraction in the first column is 11 – 1, which results in a 1. Again, we have to pay back the borrow, but it will be a single “1” as shown in the lower borrow. This process works correctly as 1110 – 111 is 14 – 7 = 7, which is 111 in binary.

We can summarize the subtraction of the two binary numbers with the truth table shown in Table 7-9.

Table 7-9. *The Truth Table for Subtracting Two Binary Numbers*

Row	A	B	Borr In from Previous	Diff	Borr Out to Next
1	0	0	0	0	0
2	0	0	1	1	1
3	0	1	0	1	1
4	0	1	1	0	1
5	1	0	0	1	0
6	1	0	1	0	0
7	1	1	0	0	0
8	1	1	1	1	1

Table 7-9 shows two instances for each subtraction, that is, the 0 – 0 is shown twice, as are all the others. This is because we must consider each subtraction both when there is no borrow from the previous column and also when there is a borrow.

Row 1 of Table 7-9 is the result of the subtraction 0 – 0 but with no borrow be paid back from the previous column.

Row 2 shows that if the previous column had resulted in a borrow being paid back into this column, then the difference would be "1," and we would have had to borrow "1," that is, 11, from the next column, which we would have to pay back, hence the borrow out to the next column.

Row 3 shows the subtraction of 0 – 1 with no borrow from the previous column. This requires that we borrow one from the next column, and so the subtraction becomes 11 – 1. The result is the difference of 1, and a 1 has to be paid back to the next column.

Row 4 is again the subtraction of 0 – 1, but this time there is a 1 being paid back from the previous column. This produces a difference of "0."

This is because the borrow from the previous column would produce a 0 – 11. This would mean we would borrow "1" in from the next column, which would produce a 11 – 11 resulting in a difference of "0" and demanding we pay back the borrow from the next column.

Row 5 is a normal subtraction of 1 – 0 resulting in a difference of "1" but not requiring us to pay back the borrow from the next column as we didn't need to borrow a "1."

Row 6 is again a normal subtraction, but as we have got a borrow in from the previous column, then we have 1 – 1 = 0 with no need to borrow from the next column.

Row 7 is fairly straightforward, a 1 – 1 = 0.

With the last row, we are using a borrow in from the previous column, and so we would have 1 – 11. This would require a borrow in from the next column giving us 111 – 11, which would result in a difference of "1" and requiring us to pay back the borrow to the next column.

I know this is a rather wordy description of how the truth table in Table 7-9 works, but I hope that if you read through it a couple of times, you will see that it does start to make sense.

Having confirmed that the truth table is valid, we can use it to create an expression for the difference between A and B and for the borrow out to the next column. This is done as follows:

$$DIFF = \overline{A}.\overline{B}.BIN + \overline{A}.B.\overline{BIN} + A.\overline{B}.\overline{BIN} + A.B.BIN$$

$$DIFF = \overline{A}.\overline{B}.BIN + A.B.BIN + \overline{A}.B.\overline{BIN} + A.\overline{B}.\overline{BIN}$$

We should be able to see that the expression is the same as that for the sum expression for the full adder. Therefore, we know the Karnaugh map does not help, but we can use Boolean algebra to show that

$$DIFF = BIN.\left(\overline{A \otimes B}\right) + \overline{BIN}.\left(A \otimes B\right)$$

Examining Table 7-9 for the borrow out, we can see there are four instances where there is a logic "1" in the column. Therefore, we can say that the expression for the borrow out is

$$BOUT = \overline{A}.\overline{B}.BIN + \overline{A}.B.\overline{BIN} + \overline{A}.B.BIN + A.B.BIN$$

We will try to use the Karnaugh map to minimize this expression; therefore, it would be useful to express the expression in binary format. This gives

$$BOUT = \sum 001, 010, 011, 111$$

The Karnaugh map is shown in Figure 7-12.

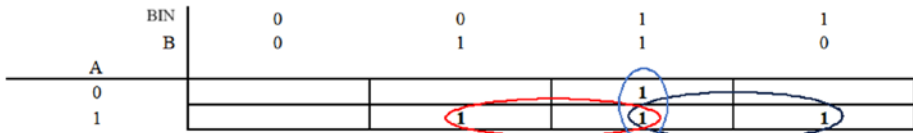


Figure 7-12. *The Karnaugh Map for BOUT*

This shows that the expression simplifies to

$$BOUT = \overline{A}.B + \overline{A}.BIN + B.BIN$$

Using Boolean algebra this simplifies further to

$$BOUT = \overline{A}.B + BIN.\left(\overline{A} + B\right)$$

Therefore, a basic subtractor circuit can be achieved with the circuit shown in Figure 7-13.

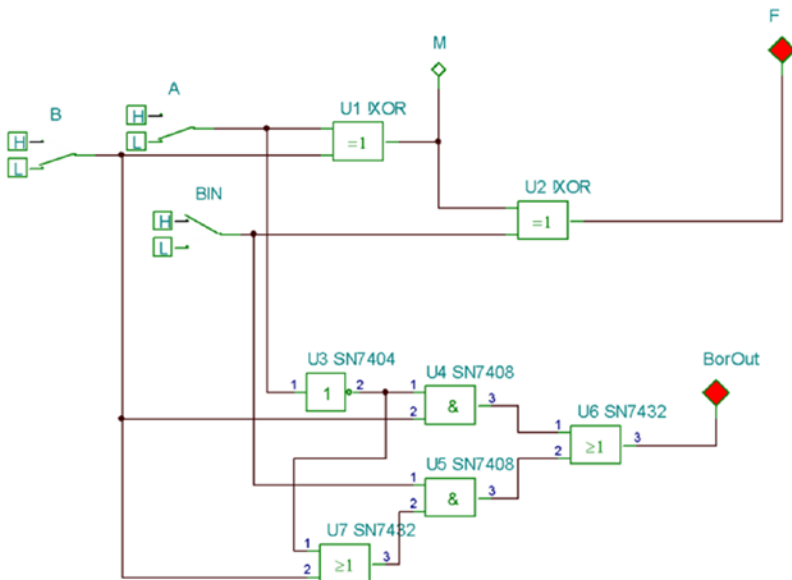


Figure 7-13. *The Basic Subtractor Circuit*

If we simulate this circuit, we should be able to create the same truth table as shown in Table 7-9. That is what did happen when we simulated the circuit.

This means that we can create a 3-bit subtractor with the circuit shown in Figure 7-14.

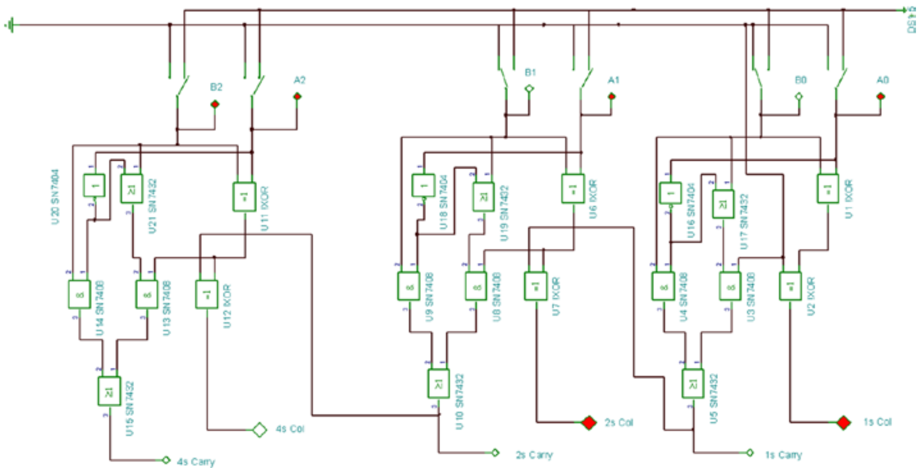


***Figure 7-14.** The 3-Bit Subtractor Circuit*

Figure 7-14 shows the subtractor circuit performing the subtraction 7 – 4, which results correctly in 3. It is shown in binary as 111 – 100 = 011, which is correct. You can perform other subtractions, and the circuit works correctly as long as the minuend is greater than the subtrahend.

An Alternative Subtractor Circuit

As a programmer I know that we can subtract by adding, and this aspect is not restricted to binary numbers; however, with binary numbers it is quite an easy process.

Subtracting by Adding Decimal Numbers

To help explain how the process works, we will apply the process to two decimal numbers. In this example we subtract 77 from 89. The result should be 12. Here is the process.

We must create the complement of the number we are subtracting with; this is termed the "subtrahend." The number we are subtracting from is termed the "minuend." To create the complement of a number, we determine what we have to add to the number to raise it to the maximum value in the columns used in that number. With the denary system, the maximum value in any column is 9. Therefore, to raise 77 to its maximum of 99, we need to add 22. This really means that 22 is the complement of 77, as 77 + 22 = 99. Another example is that the complement of 32 is 67, as 67 + 32 = 99.

Now that we have a complement of 77, we must add a 1 to this complement. This would give 22 + 1 = 23. Now we add this value to the minuend, that is, we do 23 + 89; the result is 112. The only thing we need to do is pay back the 1 we added to the complement. This is the 1 in the third column, and so we lose this "1" and we are left with the correct value of 12. This is what the result would be. We will go through another example again; 55 – 21 = 34. The process is

55		55
- 21	Complement is	78
	Add 1	1 1
	Sum all	34
	To pay back 1	
Answer is		34

This process when applied properly will work in any number system. What makes it work so well with the binary number system is that the maximum value we can have in any binary column is "1." This means that the complement of binary "0" is "1" and the complement of binary "1" is "0." This means that we can determine the complement of a binary number by simply inverting each bit of the binary number, for example:

The complement of 10011111 is 01100000.

This means then when subtracting binary numbers, we need only invert all the bits of the binary subtrahend and then add 1 to the result. Finally, we must add that number to the minuend and payback the 1 we

borrowed to add to the complement. As an example, we will do the 55 – 21 subtraction, but using binary numbers.

First, convert 55 to binary.

$$55 \div 2 = 27\ r\ 1$$

$$27 \div 2 = 13\ r\ 1$$

$$13 \div 2 = 6\ r\ 1$$

$$6 \div 2 = 3\ r\ 0$$

$$3 \div 2 = 1\ r\ 1$$

The binary number is then the last "1" followed by the remainders, which means 55 in binary is

$$55 = 110111$$

This is 1 + 2 + 4 + 16 + 32 = 55.

Similarly, 21 to binary gives

$$21 \div 2 = 10\ r\ 1$$

$$10 \div 2 = 5\ r\ 0$$

$$5 \div 2 = 2\ r\ 1$$

$$2 \div 2 = 1\ r\ 0$$

Therefore, 21 in binary is 10101. However, we must use six binary digits as the minuend uses six binary digits. This means that 21 = 010101. Now we must invert all six binary digits of the 6-bit number for 21. This means the complement of 010101 is 101010. We now carry out addition as follows:

$$\begin{array}{rr} & 110111 \\ + & 101010 \\ + & 1 \\ \hline \text{Throw away} & 100010 \end{array}$$

Note the "1" that would have gone into the seventh column is the "1" we must discard to pay back the extra 1 we have added. The 6-bit number 100010 converts back to 34 in decimal, which is what we expect. We could go through some more examples, and they would work correctly, but we should think about designing the subtractor circuit using logic gates. As this is basically an adding process, then the circuit can be based on the full adder circuit shown in Figure 7-9. If we assume the minuend of the two numbers is represented by the A digits and the subtrahend is represented by the B digits, then to change the circuit to a subtractor, we need to invert all the B digits and then add the now inverted B to the A but also include an extra 1. This extra 1 could be produced by including a carry in to the first full adder circuit. There is no carry in to the first full adder in Figure 7-9.

This means that one possible binary subtractor circuit could be that shown in Figure 7-15.

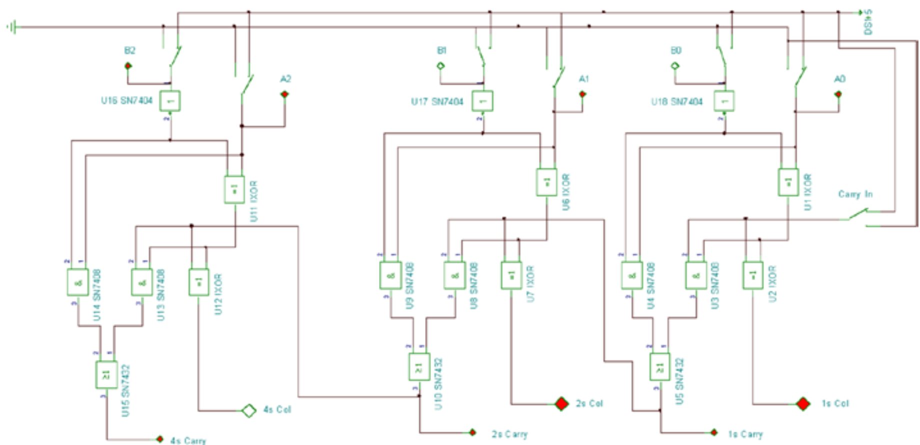


Figure 7-15. *The Logic Subtractor Circuit*

The circuit shown in Figure 7-15 is showing the A digits as 111, which is 7 in binary. The B digits are 100, which is 4 in binary. The result is 011, which is 3 in binary. This is the correct result for 7 – 4, which is 3. This suggests that the subtractor circuit works, and if you try some more three-digit examples, then the circuit works correctly.

I have gone through a detailed description of the process to design the half adder, full adder, and subtractor circuits to try and show a design process. The process can be described as follows:

1. Think carefully about what it is you are trying to make happen.
2. Write down the process.
3. Then compile a truth table for the events.
4. Then create Boolean expressions from those truth tables.
5. Then create a circuit that fulfils those expressions.

This process does require a lot of experience with the tasks you are designing your circuits for. However, that experience will only come with practice. So you should start with a task that you are familiar with.

A 4-Bit Multiplexer

This next example is a multiplexer, which is a circuit that will take a series of inputs, just four in this design, and send them out on just one output. This means the circuit must, in this case, have four inputs and one output. It must also have a method by which the user can select which input is passed on to the output. As we are using only four inputs, then we need only two binary digits to enable us to select one of the four as $2^2 = 4$. As we are using these two control inputs to select one of the four inputs to the multiplexer, we will name the two numbers S_0 and S_1. We can create a truth table to show how these numbers select the four inputs as shown in Table 7-10.

Table 7-10. *The Truth Table for S_0 and S_1*

S_0	S_1	Input
0	0	A
0	1	B
1	0	C
1	1	D

The output F will be from input A OR B OR C OR D, which requires a four-input OR gate. Using Table 7-10 we can create the expression for each input such that

$$A = \overline{S_0}.\overline{S_1}$$

$$B = \overline{S_0}.S_1$$

$$C = S_0.\overline{S_1}$$

$$D = S_0.S_1$$

Therefore, the expression for the output F is

$$F = A.\left(\overline{S_0}.\overline{S_1}\right) + B.\left(\overline{S_0}.S_1\right) + C.\left(S_0.\overline{S_1}\right) + D.\left(S_0.S_1\right)$$

This will require one four-input OR gate as the final gate at the output and also four three-input AND gates that use the S_0 and S_1 selectors to enable the AND gate to pass the required input on to the output. For example, when the S_0 and S_1 selectors are both a logic "0," then the AND gate with the A input will be enabled, while the other three AND gates are disabled. The circuit for the 4-bit multiplexer is shown in Figure 7-15.

Figure 7-16 shows the two selectors set to a logic "0." This means that only the top AND gate is enabled and its logic "1" is passed on to the output. If you simulate the circuit, you will see that the circuit works as expected, passing the selected input, according to the S_0 and S_1 bits, on to the output.

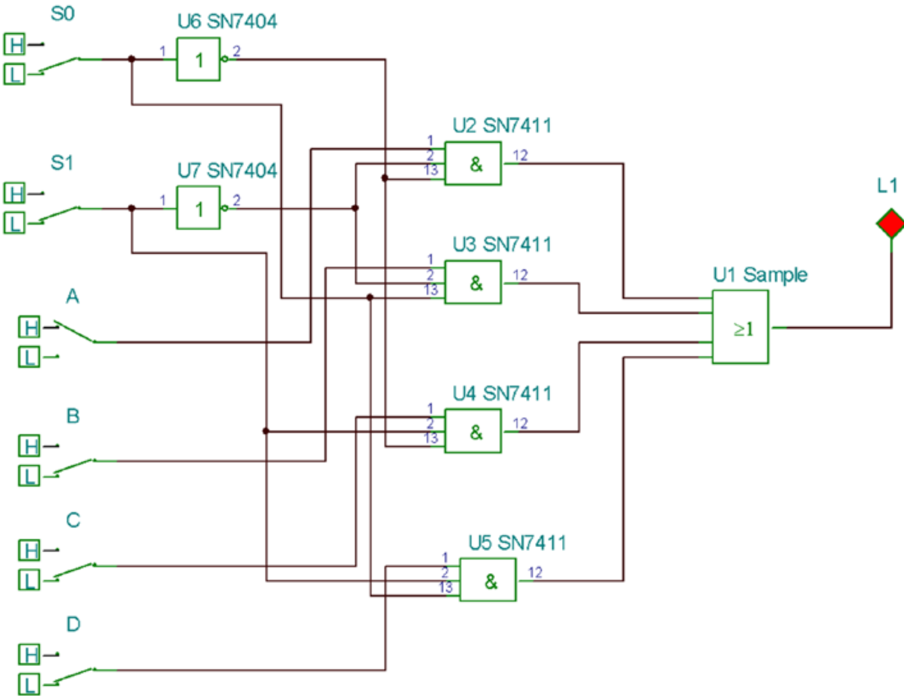


Figure 7-16. *The 4-Bit Multiplexer*

The circuit relies on the principle that an AND gate can be enabled to take on the logic of one of its inputs if the other inputs are kept at a logic "1." If the other inputs are at a logic "0," then the AND gate is disabled. We can test this concept by simulating the test circuit shown in Figure 7-17.

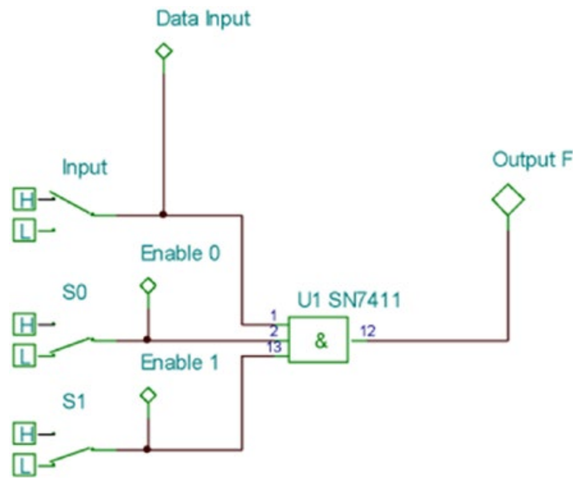


Figure 7-17. *The Test Circuit for Enabling and Disabling of the AND Gate*

If we simulate the test circuit taking the switches through all their logic combinations, we should be able to complete the following truth table.

Table 7-11. *The Truth Table for the Test Circuit Shown in Figure 7-17*

Enable 1	Enable 0	Data Input	Output F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Using Table 7-11 we can see that the output "F" only takes on the logic of the input data when both enable inputs are at a logic "1."

A Demultiplexer

This circuit works in the opposite way to the multiplexer. This means there is one input stream that can be switched onto one of a number of different outputs; we will use four outputs in this case. Just as with the multiplexer, we will require two selection bits S_0 and S_1. The four outputs are A, B, C, and D, and the data input is I. Knowing this we can create an equation for the four outputs as

$$A = \overline{S_0}.\overline{S_1}.I$$

$$B = \overline{S_0}.S_1.I$$

$$C = S_0.\overline{S_1}.I$$

$$D = S_0.S_1.I$$

This shows that the circuit uses four three-input AND gates and two NOT gates. The circuit is shown in Figure 7-18.

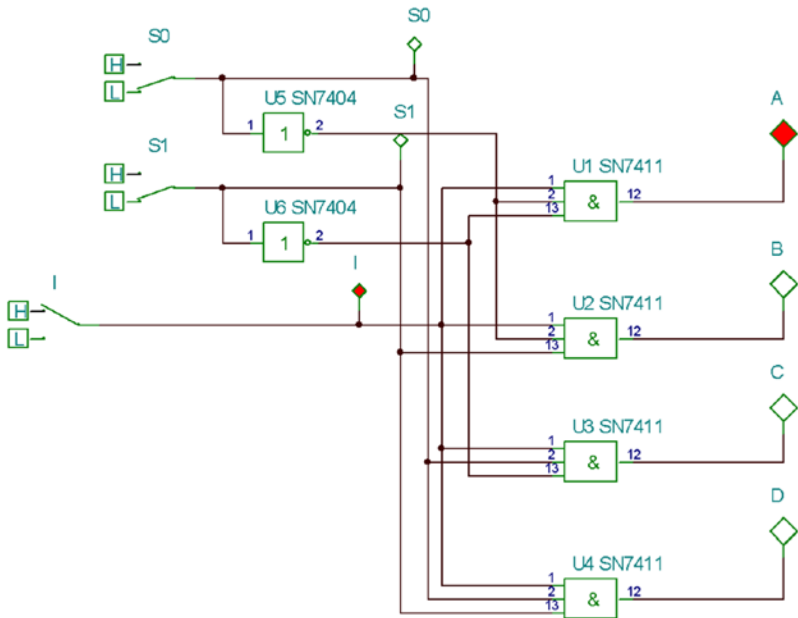


Figure 7-18. *The 4-Bit Demultiplexer*

This circuit shows the input "I" at a logic "1" and it being sent to the output "A" by setting both selection bits to a logic "0." If you simulate the circuit, you will see that the other outputs B, C, and D can be selected.

The TTL 74LS138 is a 1-to-8 demultiplexer IC, the TTL 74LS139 is a 1-to-4, and the CMOS CD4514 is a 1-to-16 output demultiplexer.

Digital Encoders

These are logic circuits that have a number of logic inputs, and the output will display which of them has gone to a logic "1" or to a logic "0" depending upon which logic they are active with. They are usually arranged as 2^n encoders, where "n" is the number of outputs and 2^n is the number of inputs. This means an encoder with just two outputs can encode four inputs, whereas an encoder with eight inputs will have three outputs.

We will look at a 4-bit encoder first. This will have four inputs, A, B, C, and D, and so two outputs Q_0 and Q_1. The circuit will work on the more common use for the encoder such that the two outputs will indicate which one of the four inputs has been activated. It is also more normal for the encoder to be active low. This means that if the third input has gone low, then the outputs will indicate it has gone low by sending both its outputs, Q_0 and $Q_{1,}$ low. If the first input had gone low, then only the Q_0 output would go low.

Now that we have understood how the circuit works, we can create a truth table to show what will happen for all possible input variations. The truth table is shown in Table 7-12.

Table 7-12. *The Truth Table for the 4-Bit Encoder*

Row	D	C	B	A	Q_1	Q_0
1	1	1	1	1	1	1
2	1	1	1	0	1	0
3	1	1	0	1	0	1
4	1	0	1	1	0	0
5	0	1	1	1	X	X

The expression for Q_0 is

$$\overline{Q_0} = \left(D.C.B.\overline{A}.\right) + \left(D.\overline{C}.B.A.\right)$$

Using Boolean algebra, we have

$$\overline{Q_0} = D.B.\left(C.\overline{A} + \overline{C}.A\right)$$

Therefore, we have

$$\overline{Q_0} = D.B.\left(A \otimes C\right)$$

The expression for NOT Q_1 is

$$\overline{Q_1} = A.\overline{B}.C.D + \left(A.B.\overline{C}.D\right)$$

Using Boolean algebra, we have

$$\overline{Q_1} = A.D.\left(\overline{B}.C + B.\overline{C}\right)$$

Therefore, we have

$$\overline{Q_1} = A.D.\left(B \otimes C\right)$$

The expression would produce a logic "1" at the respective outputs, so we need to invert the end term to ensure the outputs go to a logic low when we want them to. Therefore, the expressions for Q_0 and Q_1 are

$$\overline{Q_0} = \overline{D.B.\left(A \otimes C\right)}$$

$$\overline{Q_1} = \overline{A.D.\left(B \otimes C\right)}$$

The circuit to implement this encoder is shown in Figure 7-19.

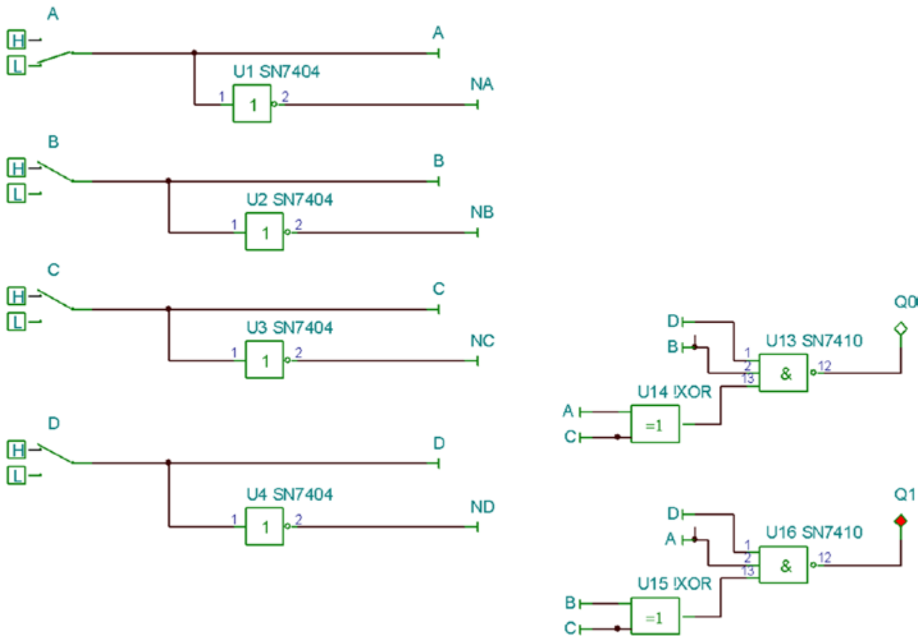


Figure 7-19.

If we simulate the circuit, we can see that the circuit works quite well. There are some issues in that when two or more inputs are sent low, both outputs go high. Also, there is no way of detecting when input D goes low.

Application of Digital Encoders

We have shown that a digital encoder is a circuit that enables the output to show which of the available inputs has become active. They are normally an active low circuit. Then for each of the different codes, a unique event can be triggered. One application of a digital encoder could be a rotary encoder. This could be where we use eight inputs to represent eight positions around the circumference of the position sensor. The output is then used to indicate the position of the rotor around the circumference. Figure 7-20 is a possible representation of the rotary encoder.

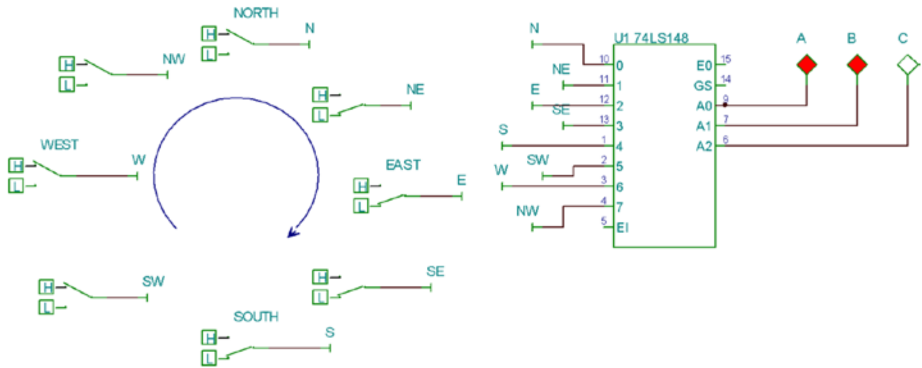


Figure 7-20. *The Rotary Encoder*

Figure 7-20 shows the output with a value of 4 in decimal. This means the switch has rotated to the fourth contact, starting from NE as contact 1 to a position of South. It would be a simple matter of describing what position the eight possible values as the output would mean as shown in Table 7-13. The table shows when the output goes to a logic "0."

Table 7-13. *The Table for the Rotary Compass*

C	B	A	Position
1	1	1	North
1	1	0	North East
1	0	1	East
1	0	0	South East
0	1	1	South
0	1	0	South West
0	0	1	West
0	0	0	North West

I hope that when you simulate the circuit shown in Figure 7-20 you do get a sense of how it can be used as a rotary compass. Also, this is a priority encoder in that if the sensor on the NE position was low, which would make “A” show a logic “0,” that is, position 1, but at the same time the South sensor was low, the output would show that only the “C” output would be low, displaying the value of 4 in decimal. This is not really a problem as the encoder would display the furthest position the rotor had moved to.

The Digital Decoder

This is a device that works in the opposite way to the encoder. Whereas the encoder used a code at its output to indicate which input had become active, the decoder uses a code at its input to determine which of its outputs becomes active. In this way we can use “n” inputs to determine which of 2^n outputs is activated. For example, a three-input decoder can determine which of eight outputs is activated. This would be referred to as a 3-to-8 decoder; the 74LS138 is one such device. We will look at a 2-to-4 to see how this can be designed. The 74LS139 is a basic 2-to-4 decoder. It has two decoders on the chip, and a basic test circuit is shown in Figure 7-21.

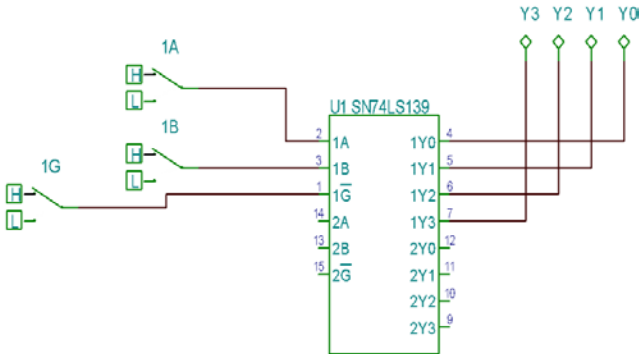


Figure 7-21. *The Basic Test Circuit for the 74LS139*

There are two enable pins, one for each decoder, and they are 1G and 2G. The bar above each input indicates they are both active low, that is, we need to switch these enable pins to a logic "0" to enable their respective decoder.

If we simulate the circuit through all four input combinations of A1 and B1, we should be able to complete the truth table shown in Table 7-14.

Table 7-14. *The Truth Table for the Test Circuit Shown in Figure 7-21*

B1	**A1**	Y_3	Y_2	Y_1	Y_0
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

We should appreciate that like most decoders, an output becomes active when its input goes to a logic "0," that is, they are active low. From the table we can see that when both the inputs are at a logic "0," that is, the data is 0 in binary, then output Y_0 is the only one active. Then when the input data is "1," then only the output Y_1 is at a logic "0" and so activated. Similarly, if the input data is 2 in binary, then the output Y_2 is active.

If we study the truth table for the circuit, we can see that the expressions for the four outputs are

$$\overline{Y_0} = \overline{B_1}.\overline{A_1}$$

$$Therefore, Y_0 = \overline{\overline{B_1}.\overline{A_1}}$$

$$\overline{Y_1} = \overline{B_1}.A_1$$

$$Therefore, Y_1 = \overline{\overline{B_1}.A_1}$$

As an exercise use Table 7-14 to determine the expressions for Y_2 and Y_3 and so confirm that the circuit shown in Figure 7-22 is correct.

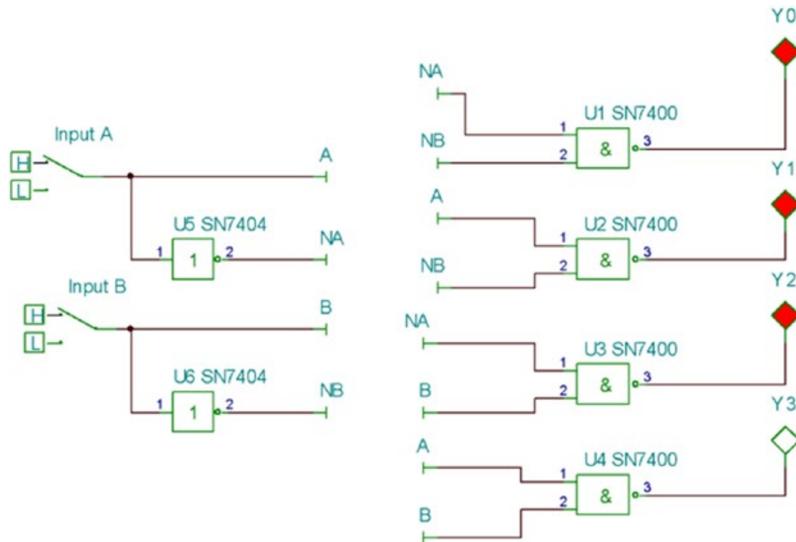


***Figure 7-22.** The Test Circuit for the 2-to-4 Decoder*

If you simulate the circuit shown in Figure 7-22, you should see that the output that goes low corresponds to the current number, in binary, that is present at the input. Figure 7-21 shows that when the data at the input equals 3 and then only Y_3 goes low.

An application of the 74LS138, which is a 3-to-8 decoder, is to control the chip select on the memory chips. Using this method, a 16-bit address bus, bits 0–15, could be used to select a memory location in a 32k memory area. The address lines 0–12 can be used to select the individual locations in a 4k memory chip, leaving address lines 13, 14, and 15, to enable one of the eight 4k memory chips. This then enables 16 address lines to select 32k of random-access memory.

When the IC is enabled, by switching G1 to a logic “1” and G2A and G2B to a logic “0,” the decoder will activate one of the eight outputs while deactivating the other seven. It will use the logic at the three inputs, A, B

and C, to decode which output will be activated. Figure 7-23 shows all 3 inputs at logic "1," that is, the binary input is 111, which is 7 in decimal. Therefore, only output 7 is activated. With the 74LS138 the output is sent to a logic "0" to activate it, that is, they are active low. If we simulate the circuit, we should be able to complete the truth table shown in Table 7-15.

Table 7-15. *The Truth Table for the 74LS138 Decoder*

G2B	G2A	G1	C	B	A	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	1	0	0	0	1	1	1	1	1	1	1	0
0	0	1	0	0	1	1	1	1	1	1	1	0	1
0	0	1	0	1	0	1	1	1	1	1	0	1	1
0	0	1	0	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	0	1	1	1	0	1	1	1	1
0	0	1	1	0	1	1	1	0	1	1	1	1	1
0	0	1	1	1	0	1	0	1	1	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1

This device is normally used in controlling when a chip select input on the number of chips, eight in this case, is activated. It uses three data lines to turn one of the eight chips on. Normally the chips have active low chip select pins. The principal operation of the 74LS138 is shown in Figure 7-23.

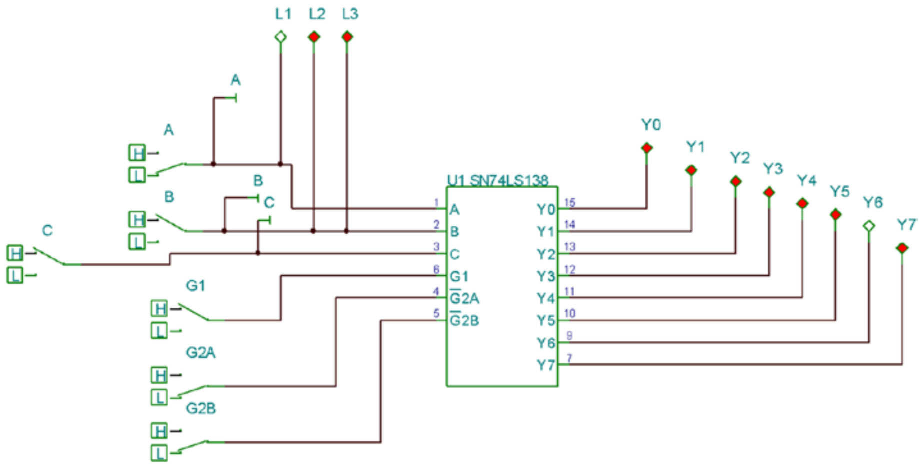


Figure 7-23. *The Operation of the 74LS138*

Simulating the circuit shows that it does use the three inputs to control which output is sent low. This agrees with the truth table shown in Table 7-15.

A Seven-Segment Decoder Chip

The 74LS49 is a decoder chip that can be used to take the four inputs from a binary counter and decode their outputs to drive the seven diodes in a seven-segment display. The truth table that shows how the four inputs drive the required diodes is shown in Table 7-16.

Table 7-16. *The Truth Table for the Seven-Segment Display Driver*

Q_3	Q_2	Q_1	Q_0	A	B	C	D	E	F	G	No.
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	0	0	1	1	9
1	0	1	0	1	1	1	0	1	1	1	A
1	0	1	1	0	0	1	1	1	1	1	B
1	1	0	0	1	0	0	1	1	1	0	C
1	1	0	1	0	1	1	1	1	0	1	D
1	1	1	0	1	0	0	1	1	1	1	E
1	1	1	1	1	0	0	0	1	1	1	F

The outputs are shown as "1" because the seven-segment display is a common cathode. If it was a common anode, then all logic would have to be inverted.

To fully appreciate what this is about, we should look at the seven-segment display.

The Seven-Segment Display

This is a device that can be used to display numbers, so it can be used to create a display for a digital clock. A typical seven-segment display is shown in Figure 7-24.

Figure 7-24. *A Typical Seven-Segment Display*

There are seven LEDs (light-emitting diodes) in a display, hence the name seven-segment display. Some displays have an extra LED for the decimal place, or dot. The LEDs can be switched on in different arrangements to display the numbers 0–9 and if required the letters A, B, C, D, E, and F as in the hexadecimal number system. We can get red, green, blue, yellow, and white LEDs as well as extra-bright LEDs.

LEDs have an anode terminal and a cathode terminal. These terminals are sometimes labeled A and K as C can stand for capacitance or the Coulomb. Electrical current can only flow one way through the diode, and conventional current flows from the anode to cathode. To make the current flow through the LED, the anode voltage must be around 2.2V higher than that of the cathode.

There are two main types of seven-segment displays, which are common anode and common cathode.

Common Anode Seven-Segment Display

With this type of seven-segment display, the anodes of all seven LEDs are usually connected to a +5V supply. To turn an LED on, the cathode must be connected, independently, to ground or 0V. A resistor is inserted between the cathode and ground to limit the current that flows through the LEDs to prevent them from destroying themselves. This arrangement is shown in Figure 7-25.

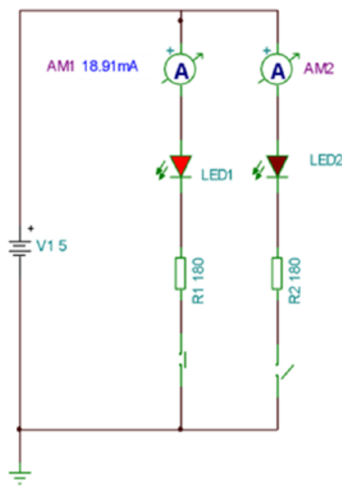


Figure 7-25. *The Basic Circuit to Turn on an LED in Common Anode*

In Figure 7-25 LED1 is shown illuminated as it is switched to ground. LED2 is off as its cathode is not switched to ground. The 180Ω resistor is there to limit the current flowing through the LEDs to around 19mA; too much current and the LEDs would burn out and be destroyed.

Common Cathode Seven-Segment Display

With this type of seven-segment display, the cathodes of all seven LEDs are usually connected to a 0V supply. Then to turn each LED on, the anode of each LED can be connected, independently, to a +5V supply. This arrangement is shown in Figure 7-26.

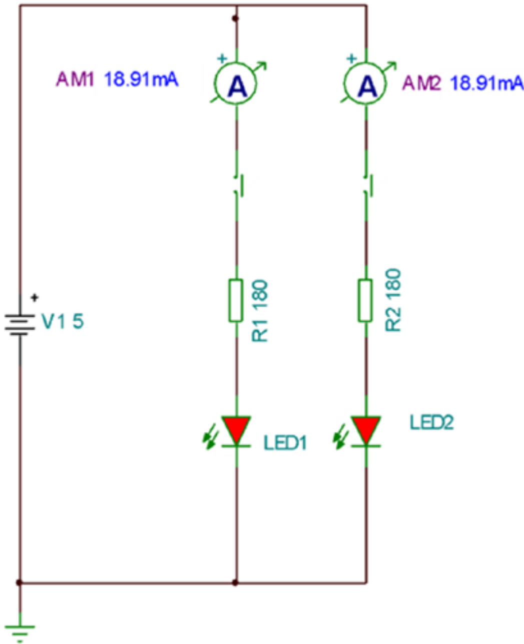


Figure 7-26. *The Common Cathode Arrangement*

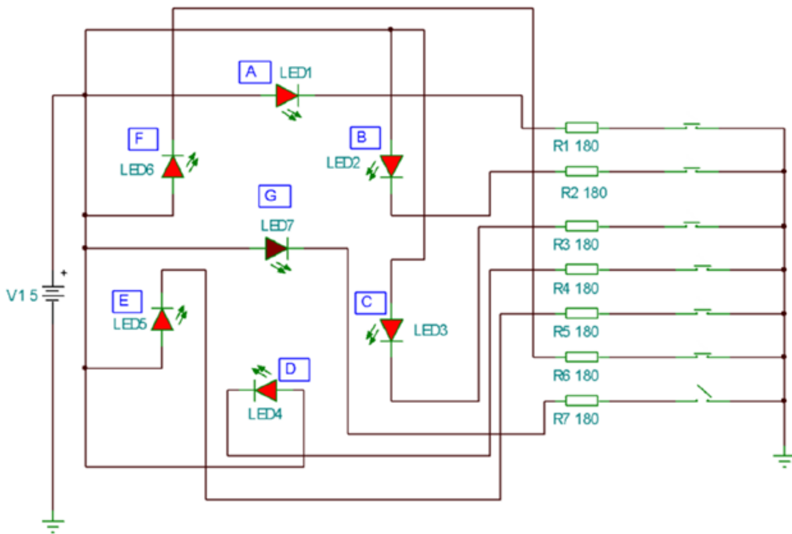


Figure 7-27. *The Circuit of a Common Anode Seven-Segment Display*

Figure 7-27 is an attempt to show you the circuitry of the common anode display. The seven LEDs are laid out to form a ring going from LED “A” to LED “F” in six LEDs. Then there is the seventh LED, LED “G,” that lies central to the display. Figure 7-27 shows the six outer LEDs turned on by closing their respective switches to connect their respective cathodes to ground or 0V via the series resistor. This means those six LEDs are switched on and current flows through them. It is hoped that you can see that this forms the number zero.

Knowing the basics of what a seven-segment display is and how we can turn on the LEDs, we will use Table 7-16 to look at some of the logic circuitry inside the 74LS49 driver IC for the seven-segment display. The IC must take the four inputs, from the output of the binary counter, and decode them to turn on and off the appropriate LEDs to display any of the 16 values the seven-segment display can show.

From inspection of Table 7-16, we should be able to derive expressions for when each segment would be lit. We will create two such expressions, one for when segment B is lit and the other for when segment C is lit. To do this we must determine the state of the input Qs for each logic “1”, if we use the 1st canonical format, or logic “0”, if we use the 2nd canonical format, in the B and C columns of Table 7-16.

With respect to the B column, it would be easier to use the 2nd canonical format as there are only six “0”s but ten “1”s. Therefore, using the 2nd canonical format, we have

$$\overline{segB} = \left(\overline{Q_3}.Q_2.\overline{Q_1}.Q_0\right) + \left(\overline{Q_3}.Q_2.Q_1.\overline{Q_0}\right) + \left(Q_3.\overline{Q_2}.Q_1.Q_0\right) + \left(Q_3.Q_2.\overline{Q_1}.\overline{Q_0}\right) + \left(Q_3.Q_2.Q_1.\overline{Q_0}\right) + \left(Q_3.Q_2.Q_1.Q_0\right)$$

In binary format this is

$$\overline{segB} = \sum 0101,0110,1011,1100,1110,1111$$

Similarly, using the 2nd canonical format, the expression for $\overline{segC}$ is

$$\overline{segC} = \left(\overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0}\right).\left(Q_3.Q_2.\overline{Q_1}.\overline{Q_0}\right).\left(Q_3.Q_2.Q_1.\overline{Q_0}\right).\left(Q_3.Q_2.Q_1.Q_0\right)$$

In binary format it is

$$\overline{segC} = \sum 0010,1100,1110,1111$$

We can use Karnaugh maps to see if these expressions will simplify. The map for $\overline{segB}$ is shown in Figure 7-28.

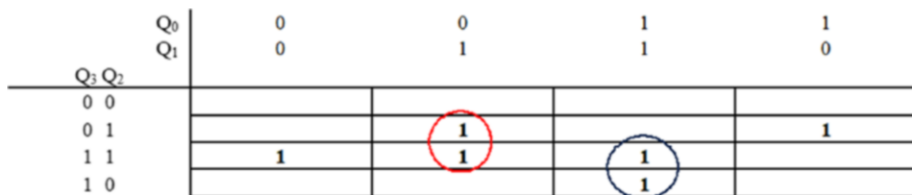


Figure 7-28. *The Karnaugh Map for* $\overline{segB}$

This does simplify slightly to

$$\overline{segB} = \left(Q_2.Q_1.\overline{Q_0}\right) + \left(Q_3.Q_1.Q_0\right) + \left(Q_3.Q_2.\overline{Q_1}.\overline{Q_0}\right) + \left(\overline{Q_3}.Q_2.\overline{Q_1}.Q_0\right)$$

The map for $\overline{segC}$ is shown in Figure 7-29.

Q3 Q2 \ Q0 Q1	0 0	0 1	1 1	1 0
0 0		1		
0 1				
1 1	1	1	1	
1 0				

Figure 7-29. *The Karnaugh Map for* $\overline{segC}$

This does simplify slightly to

$$\overline{segC} = \left(\overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0}\right) + \left(Q_3.Q_2.\overline{Q_0}\right) + \left(Q_3.Q_2.Q_1\right)$$

We can now convert each expression using the shortcut, which is to invert the variables and change the "." to "+" and the "+" to ".". This means the variables in the terms are now ORed instead of ANDed. Also, the terms themselves are ANDed instead of Ored:

$$segB = \left(\overline{Q_2} + \overline{Q_1} + Q_0\right).\left(\overline{Q_3} + \overline{Q_1} + \overline{Q_0}\right).\left(\overline{Q_3} + \overline{Q_2} + Q_1 + Q_0\right).\left(Q_3 + \overline{Q_2} + Q_1 + \overline{Q_0}\right)$$

$$segC = \left(Q_3 + Q_2 + \overline{Q_1} + Q_0\right).\left(\overline{Q_3} + \overline{Q_2} + Q_0\right).\left(\overline{Q_3} + \overline{Q_2} + \overline{Q_1}\right)$$

We can test these expressions by simulating the test circuit shown in Figure 7-30.

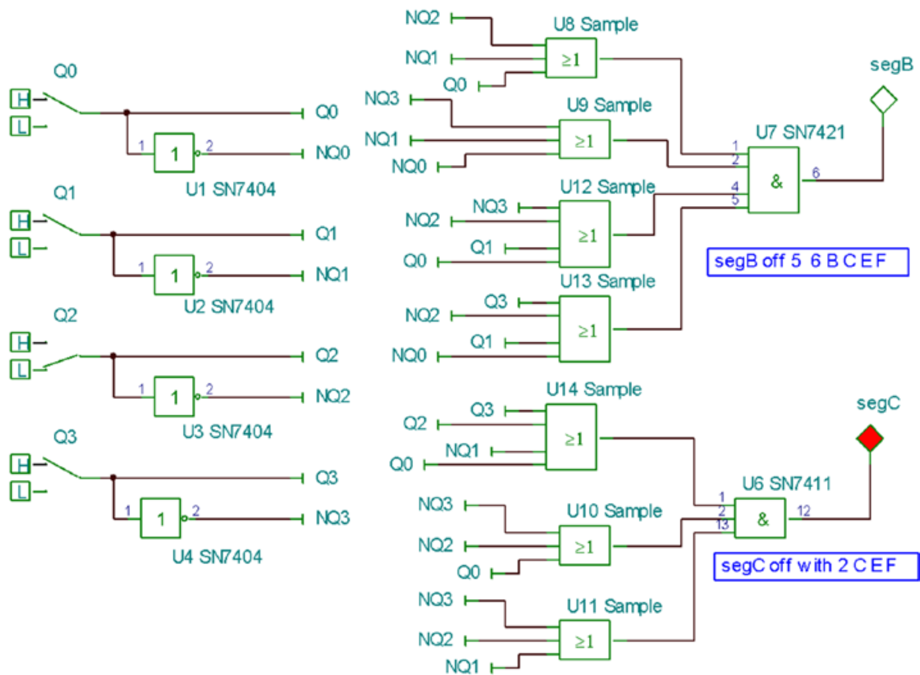


Figure 7-30. *The Test Circuit for the segB and segC Expressions*

From inspection of Table 7-16, we can see that the segments will not turn on for the following numbers:

- segB does not turn on for numbers 5, 6, 11 or B, 12 or C, 14 or E and 15 or F.
- segC does not turn on for numbers 2, 12 or C, 14 or E and 15 or F.

If we simulate the circuit, we should see that the two logic probes do turn off for the predicted numbers. This confirms that the two expressions are correct.

This is only part of the logic needed for the decoder to drive the seven-segment display, but as an exercise in logic design, it does use a lot of what we have studied in this book.

We could try using the 2nd canonical expression without transforming it. The expression really states when the seg will be turned off. So that might allow it to show when the seg is turned on using the fact the expression would not be true. This approach is best explained with an example. We will use this approach to create a circuit segF. Using Table 7-16 we can see that there are five occurrences when there is a logic “0” in the segF column. Using those occurrences, the expression for $\overline{segF}$ is

$$\overline{segF} = \sum 0001, 0010, 0011, 0111, 1101$$

It is easier to read the binary format from Table 7-16; however, it is also useful to describe the expression in terms of the variables. This then gives the expression for $\overline{segF}$ as

$$\overline{segF} = \left(\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0\right) + \left(\overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0}\right) + \left(\overline{Q_3}.\overline{Q_2}.Q_1.Q_0\right) + \left(\overline{Q_3}.Q_2.Q_1.Q_0\right) + \left(Q_3.Q_2.\overline{Q_1}.Q_0\right)$$

We can use the binary format to fill in the Karnaugh map. This is shown in Figure 7-31.

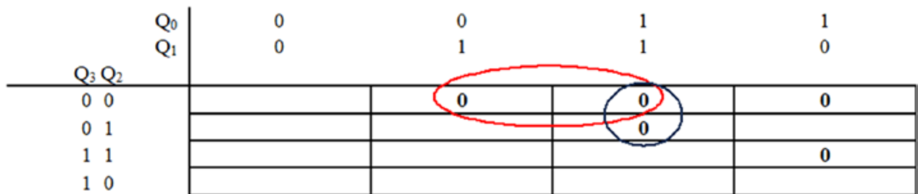


Figure 7-31. *The Karnaugh Map for* $\overline{segF}$

This does simplify to

$$\overline{segF} = \left(\overline{Q_3}.\overline{Q_2}.Q_1\right) + \left(\overline{Q_3}.Q_1.Q_0\right) + \left(Q_3.Q_2.\overline{Q_1}.Q_0\right) + \left(\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0\right)$$

This can simplify further using Boolean algebra to

$$\overline{segF} = \left(\overline{Q_3}.\overline{Q_2}\right)\left[Q_1 + \left(\overline{Q_1}.Q_0\right)\right] + Q_0\left[\left(\overline{Q_3}.Q_1\right) + \left(Q_3.Q_2.\overline{Q_1}\right)\right]$$

We can test the expression out using the circuit shown in Figure 7-32.

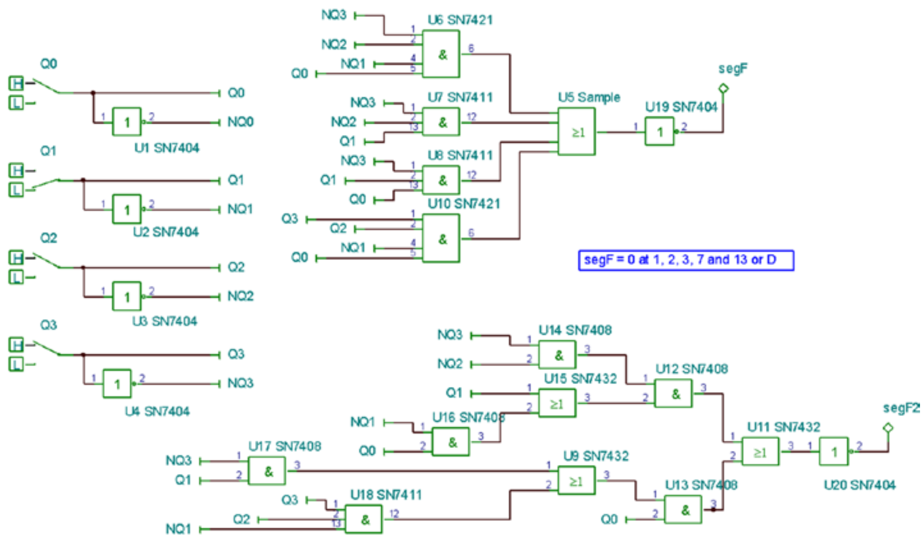


Figure 7-32. *The Test Circuit for the* $\overline{segF}$

There are a couple of points we can confirm with this test circuit. The first is that each output has a NOT gate inserted between the last OR gate and the logic probe. This is because the expression for $\overline{segF}$ will be true when the variables meet the stated values. This means the logic probe would go high, to a logic "1." However, in the real application of this expression, we want the output to go to a logic "0" when the variables agree with the expression and stay at a logic "1" for all other values. In this way the logic probe would mimic the value for the $\overline{segF}$ column in Table 7-16. Therefore, the circuit requires this extra NOT gate.

As we simulate the circuit, we see that the two logic probes respond in exactly the same way. This means that the simplification of the initial expression for $\overline{segF}$ was correct. However, the circuit for the simplified expression looks more complicated. That is true, but it may use gates that are more easily available than the original expression.

The simulation shows that the use of the 2nd canonical format can work and may provide an easier approach, but you need to be aware of what logic you want from the expression.

Exercise 2

You should try and create the expressions for segA and segG using Table 7-16. The answer will be in the appendix.

The circuit shown in Figure 7-33 shows a possible circuit that uses the 74LS49 decoder being used to drive a seven-segment display. The 74LS49 has outputs that are active high, that is, they go to a logic “1” when activated. However, the seven-segment display in TINA is a common anode. Therefore, we need to invert the output of the driver to change the logic “1” outputs to logic “0.” Also, we should insert resistors to limit the current in each of the LEDs. A value of 220 is a reasonable and should limit the current to around 15mA in each LED.

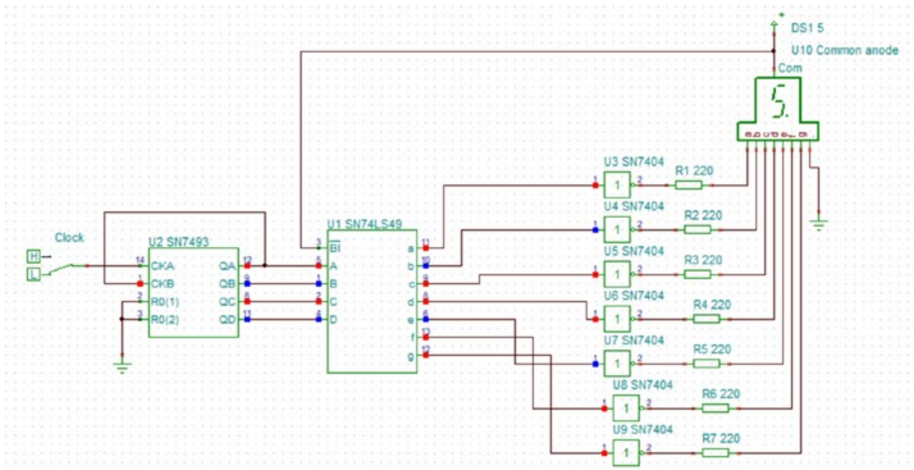


Figure 7-33. *The 74LS49 Used to Drive a Seven-Segment Display*

If we examine Figure 7-33, we can see the seven-segment display is showing the number 5. We can see that the logic on the output of the SN74LS49 driver is showing a blue square on output "b." This means it is at a logic "0," that is, it is turned off. This is what we predicted would happen with segB when the number displayed was 5. The output "e" is also turned off, but we have not looked at that segment.

Summary

In this chapter we have looked at several applications of digital circuits, mostly combinational logic circuits being used in some everyday applications. I hope you have found them interesting and the explanations have helped you understand how they work and how we could design such circuits.

In the next chapter, we will look at applying what we have learned to some applications we might come across, that is, memory registers, shift registers, and PISO and SIPO registers.

CHAPTER 8

Shift Registers and More

In this chapter we will look at some useful sequential logic circuits. We will also look at circuits that make use of the D-type latch.

The D-Type Latch

In Chapter 5 we concentrated on using the JK flip flop to create some sequential circuit such as a counter. In this chapter we will concentrate more on using the D-type latch.

I have stated that the J and K inputs are used to configure how the flip flop operates. The next useful way of configuring the JK flip flop is to make the "K" input the NOT of the "J" input. This arrangement is shown in Figure 8-1.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_8

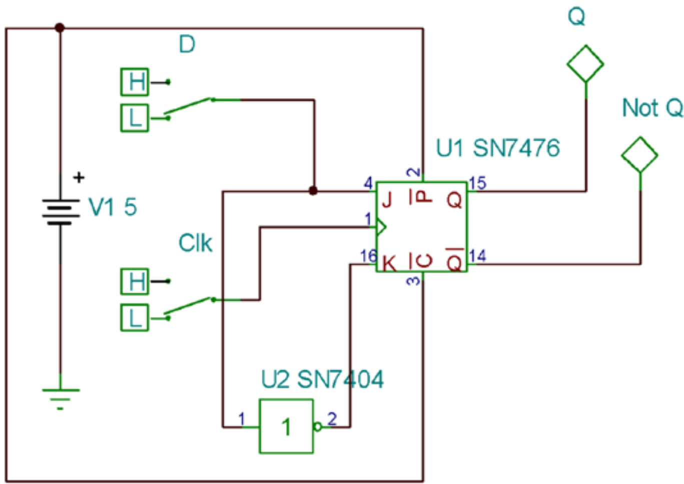


Figure 8-1. *The JK with the K Set as the NOT of J*

The D-type latch introduces a delay from changing the data at the input until it has passed on to the output. That is why we call this the D-type latch, that is, D for delay.

It is not the easiest concept to see what is happening from the simulation. However, the principle of the operation is that the "D" input determines what logic we want the Q to change to. However, the "Q" output will not respond until the clock goes high. This is really ensuring that transfer of the "D" logic onto the "Q" output can be synchronized to a clock. The input is labeled "D" as the "D" stands for data. That being the case, we can use this D-type latch to synchronize when we can pass data on to some other location. This could be in synchronizing when we pass data from a data bus to a memory cell. This is an operation that is at the heart of any solid-state memory device. We will extend this idea later in the chapter.

This principle of the D-type latch is so useful that JK flip flops are now manufactured with the "K" input being the NOT of the "J" and are called the D-type latch. Figure 8-2 shows a common flip flop configured this way.

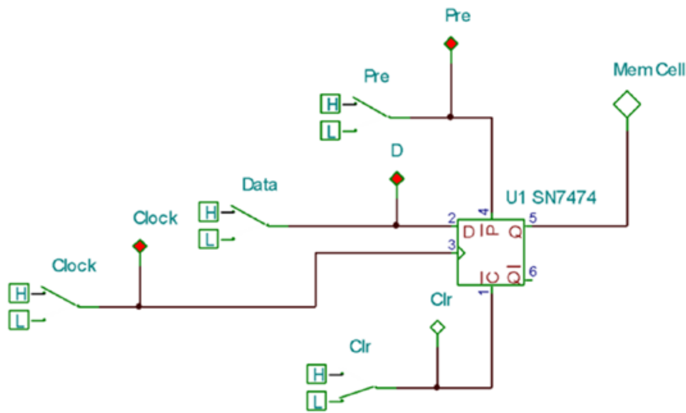


***Figure 8-2.** The Simple Test Circuit for the D-Type Latch*

The test circuit shown in Figure 8-2 allows us to study what happens when the inputs change.

The first test is to send the Pre-set switch down to a logic “0.” This immediately sets the output MemCell to a logic “1.” This confirms that the Pre-set input is active low and that it will immediately set the output to a logic “1” independent of the clock.

The next test is, with the Pre-set input put back to a logic high, the output stays at a logic “1,” but when we send the Clr input low, the output goes back to a logic “0.” This confirms that the Clr input is active low and that it will immediately reset the output to a logic “0” independent of the clock.

Now send the Clr input back to a logic “1,” the output stays at a logic “0.” Now send the Data switch to a logic high. Nothing happens. Now toggle the clock switch. You will see that when the clock toggles from low to high, the output will change to a logic “1.” Now change the Data switch back to low and toggle the clock again. You will see that only when the clock goes from low to high will the output change back to a logic “0.”

This shows that the output will copy the logic at its data input only after the clock has gone from low to high.

This then shows how the D-type latch can be used to create a memory register. This basic concept of how this could be used to transfer data, waiting on the data bus in a computer system, into a memory register is shown in Figure 8-3.

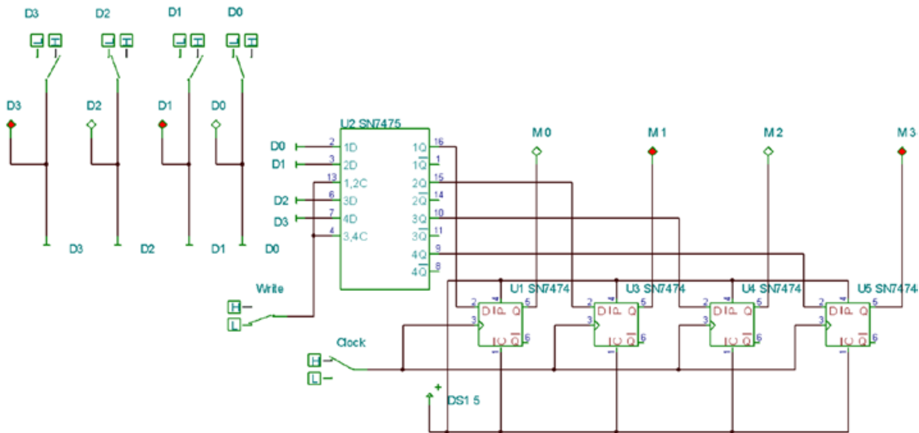


Figure 8-3. *A 4-Bit Data Bus to a 4-Bit Memory Register*

The four switches D0–D3 control the data sitting on the data bus. The other two switches, WRITE and Clock, control when the data on the data bus is allowed to be written to the four Qs in the memory register. Try changing the data and see that it only goes onto the 4 memory bits M0–M3 when both WRITE and Clock are at a logic "1." In this way then the D-type flip flop can form the basis of a memory register.

The test circuit is using the SN7475, the U2 chip on the circuit, as a latching device. The IC is split into two 2-bit latches that can be connected to a 4-bit data bus whose data is waiting at the inputs: 1D, 2D, 3D, and 4D. There are two control inputs on pins 13 and 4, which are labeled 1,2C and 3,4C. This is how the IC can be split into two halves. The data will only be latched onto the outputs of the IC, 1Q, 2Q, 3Q, and 4Q, when both these control pins go to a logic "1." If the control switch goes back to a logic "0," the data that has just been passed on to the outputs remains as it was.

The data on the data bus can change, but this change will not be reflected on the outputs until the two control inputs go high again.

As we are using a 1Hz square wave input at the clock input to all four D-type latches, just to give us time to see things changing, the latches will be constantly passing the data at their D inputs onto their respective outputs. However, as the data at their respective D inputs comes from the output of the 7475 latch, the data does not change until the write switch is sent to a logic "1."

This test circuit should give us some idea that a memory register could be made from the D-type latch, which, as we have looked at, was born from the $\overline{S}\,\overline{R}$ latch we looked at in Chapter 4.

The 4-Bit Shift Register or SISO (Serial In Serial Out)

Now that we have seen how the D-type latch is created, we will look at another important use of it. This is as a shift register, which is an important aspect of how computers communicate with the outside world. The circuit for the shift register is shown in Figure 8-4.

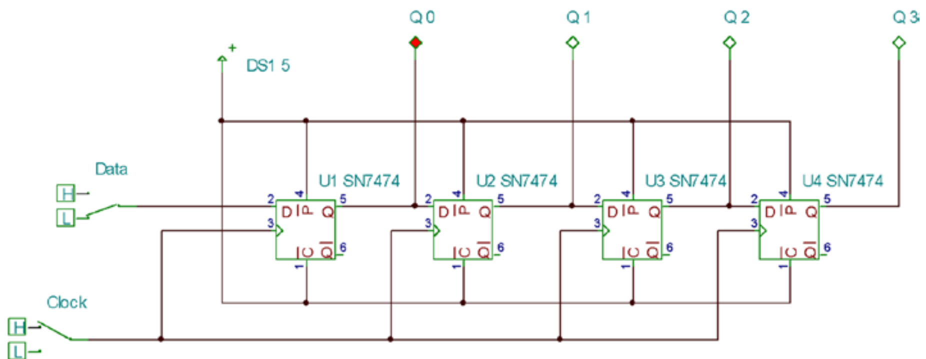


Figure 8-4. *The 4-Bit Shift Register*

I have restricted the circuit to 4 bits just for convenience. However, the principle can easily be applied to 8 bits or more. In this circuit there is only one D-type latch that takes the data, to be shifted, into the register. This is the first latch, and it gets its input from the switch labeled Data. I am using a switch so that we can set the data we want to shift in the register. The other three latches get their data input from the Q output of the previous latch. The clock is applied, in parallel, common to all the latches at the same time. This is controlled by the switch labeled Clk.

To simulate the circuit, set the Data input to a logic "1" while the clock is connected to logic "0." Now turn the circuit on, and the probe L1 should show high, a logic "1." All the Q outputs should show low, a logic "0." Now send the clock high and then low. The data should have been shifted into the first D-type latch, and Q0 should have gone high. Now switch the Data input back to a logic low. Now each time you toggle the clock input, you should see the current Q go low and the next Q in the shift register should go high. In this way you should see that data does shift through the register until it falls off the end after sending the Q3 high and then low.

This makes good use of the D-type latch, and the circuit is sometimes referred to as the "SISO" (Serial In Serial Out) register.

The PISO (Parallel In Serial Out) Register

The next important register we will look at is the PISO. This again uses the D-type latch, but it also uses the Pre-set or Set input of the latch. You should appreciate that the Pre-set input can be used to set the Q output of the D-type latch regardless of what is happening with the main Data input and the clock. This means that, when we are ready, we can make the Q outputs go to any logic we want. The main use of this circuit is to take data into the latches in a parallel fashion, as from a data bus, and then shift it out as an output from a computer. This is one of the basic actions of a UART (Universal Asynchronous Receive and Transmit IC) chip. The basic circuit for a 4-bit PISO is shown in Figure 8-5.

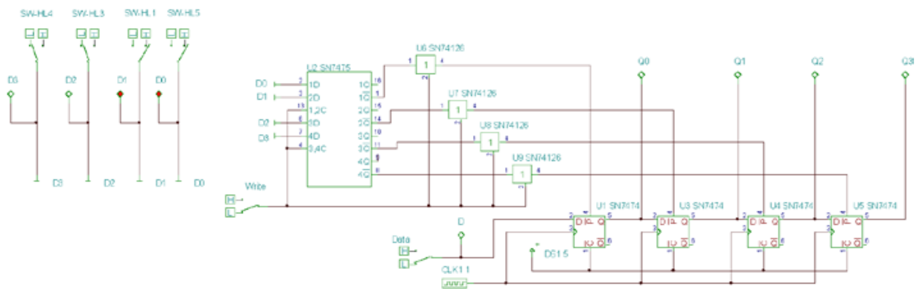


Figure 8-5. *The 4-Bit PISO Circuit*

We again use the SN7475 latching IC as a method by which we can control when we enable the data present on the data bus, the four switches D0–D3 in this case, to be loaded into the four memory cells, U1, U3, U4, and U5. It is the switch "Write" that controls when the enabling logic "1" is applied to the 1,2C and 3,4C control pins on the 7475 IC.

One difference between this test circuit and that shown in Figure 8-3 is the outputs of the 7475 that we are using are the NOT Q outputs. This is because the Pre-set inputs, the "P," on each of the D-type latches are active low. We looked at this in Figure 8-2.

The other difference is that we are using four tri-state buffers between the outputs of the 7475 and the Pre-set inputs to the D-type latches. This is because we want to isolate the D-type latches from the 7475 if we are not writing data to them.

The clock is running at 1Hz, but the data won't be loaded into the D-type latches until the write signal goes to a logic "1." Note the write signal enables the tri-state buffers at the same time as it sends the data from the data bus onto the outputs of the 7475. As we enable the tri-state buffers at the same time as we pass the data through the 7475, the data is loaded in parallel to the D-type latches. When we switch the write signal back to a logic low, this disconnects the D-type latches from the 7475 and

so the data bus. Now as the D-type latches are still being clocked and the data at the first D-type latch is a logic "0," the data that has just been loaded in is shifted through the four latches. In this way the circuit does act as a PISO (Parallel In Serial Out) register.

The data would then be shifted out of the register, and after the last bit has gone, a new 4-bit data is sent into the register in a parallel fashion.

The PIPO (Parallel In Parallel Out) Register

With this circuit, data is presented to all four D inputs at once. Then, on the next transition of the clock from low to high, the data passes through the latches and appears at all four outputs at once. Hence, we get a parallel in and parallel out register. The basic circuit of a PIPO is shown in Figure 8-6.

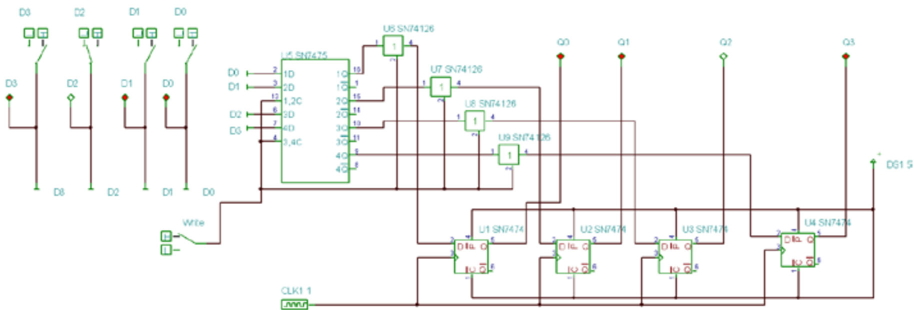


Figure 8-6. *The Basic PIPO Circuit*

The write input gives us the ability to disable the load function.

The SIPO (Serial In Parallel Out)

This is the last of the data shifting registers. The basic circuit for the SIPO is shown in Figure 8-7.

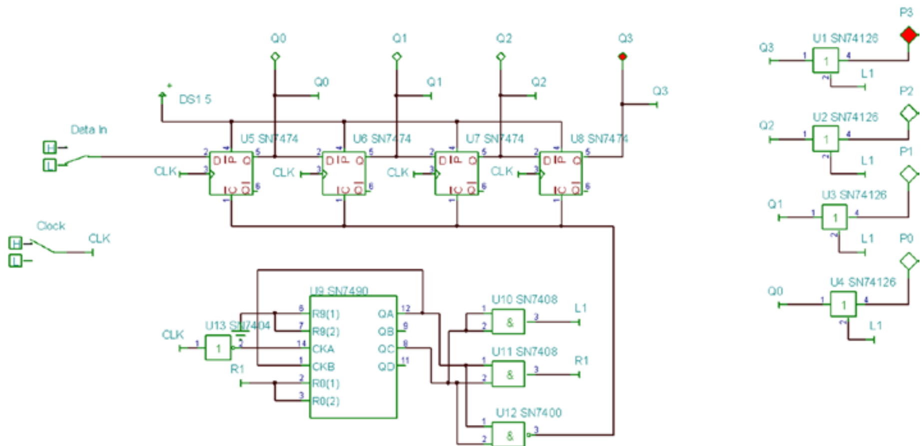


Figure 8-7. *The Basic SIPO*

The principle is that we allow 4 bits of data to be shifted into the register and when the fourth bit enters the register, we enable the data present at the four tri-state buffers to be passed onto their four outputs. These outputs would be connected to the parallel bus.

To control the timing of the latching action, we are using a simple counter IC, the SN7490. When the counter reaches a value of 4, the U10 AND gate puts a logic "1" on its output "L1." This output is connected to the trigger inputs on all four tri-state buffers. This means when the count gets to 4, the data present on the Qs is allowed to pass onto the P outputs.

The clock signal to the counter is passed through a NOT gate first because the shift register shifts on a low-to-high transition, whereas the counter counts on a high-to-low transition.

Both the shift register and the counter are cleared on the fifth count controlled by the U12 AND gate.

The Ring Counter

With the basic shift register we have considered so far, when the initial data input gets to the final latch, that is, Q3, with the 4-bit shift register, that data will be lost on the next transition of the clock. With the ring counter, that data is not allowed to fall off, as it is fed back to the input data on the next transition of the clock. The basic circuit for the ring counter is shown in Figure 8-8.

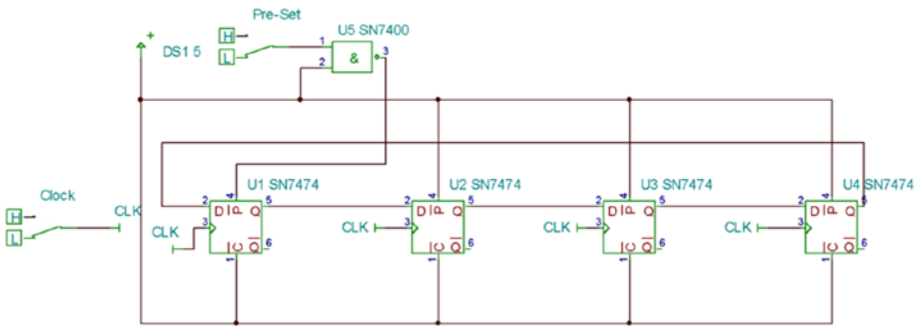


Figure 8-8. *The Basic Ring Counter*

We are using the Pre-set to momentarily go to a logic "1," at the first switch on of the circuit. When the switch is connected to the logic "1," as the other input is connected permanently to VCC, that is, a logic "1," there will be two logic "1"s at the input to the NAND gate U5. This then forces the output of the NAND gate to go to a logic "0." As this output is connected to the active low P input of the first latch, this will then force the Q output, of that latch, to a logic "1." Then at the next transition of the clock, that logic "1" will shift right to the Q of the second latch. Assuming the Pre-set switch has gone back to a logic "0," the Q output of the first latch will go to a logic "0," as this was the logic of its D input at the time of the first shift. As we go through more transitions of the clock, the logic "1" will shift through the counter with a logic "0" replacing it in the Q it has just shifted from. When the logic "1" arrives at the final D-type latch, then at the next transition,

this logic "1," instead of falling off the counter and so being lost, is fed back to the D input on the first D-type latch. Then, on further transitions of the clock, the cycle repeats.

The Johnson Ring Counter

This is similar to the basic ring counter except that it is the NOT Q of the final latch that is fed back to the input at the first latch. The circuit for the Johnson counter is shown in Figure 8-9.

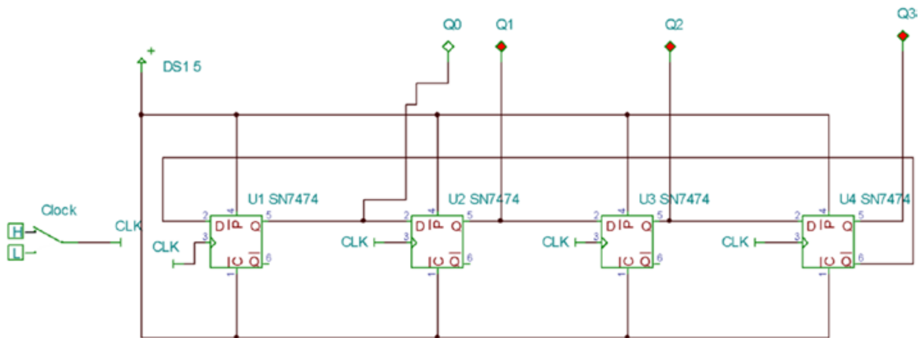


Figure 8-9. *The Johnson Ring Counter*

This counter does not need a Pre-set input. This is because, as with the final latch, the Q is at a logic "0" but the NOT Q is at a logic "1." It is this logic "1," from the NOT Q of the last D-type latch, that is fed back as the D input to the first latch. In the first shift through the register, the output counts up as in 1000 (1), 1100 (3), 1110 (7), and then 1111 (15). This is then followed by a reducing count of 0111(14), 0011(12), 0001(1), and 0000 (0), reading from left to right. Then the cycle repeats.

A Frequency Divider

The Johnson ring counter can be used as a divider circuit. If there are four stages as shown in Figure 8-9, this gives us four options for the divide rate. If the feedback was taken from the NOT Q on the first D-type latch, then the counter could divide by 2. It will be a divide by 4 if the feedback is taken from the second D-type latch, 6 if taken from the third latch, and then 8 if the feedback is taken from the last latch. Figure 8-10 shows the feedback taken from the third latch with Figure 8-11 confirming this would divide the clock frequency by 6.

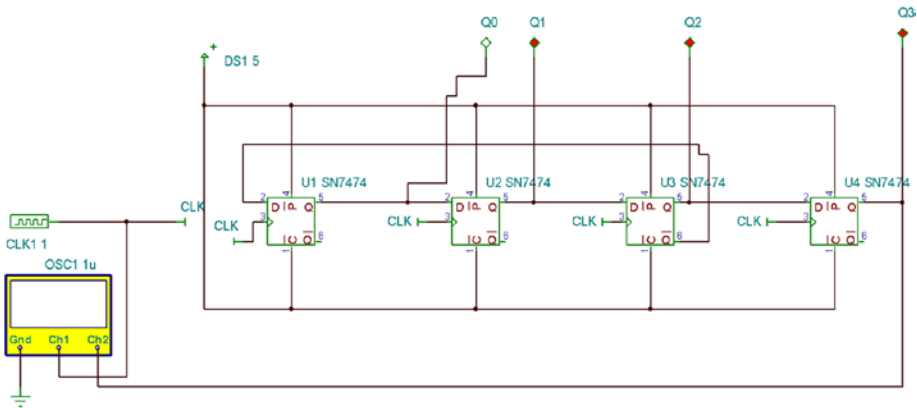


Figure 8-10. *The Divide by 6 Johnson Ring Counter*

The feedback is taken from the NOT Q of the third D-type latch. Note it does not matter from which D-type latch we monitor the output of the counter. Figure 8-10 shows us connecting channel 2 of the oscilloscope to the Q of the fourth D-type latch. Channel 1 is measuring the frequency of the input clock; the frequency is set to 1Hz.

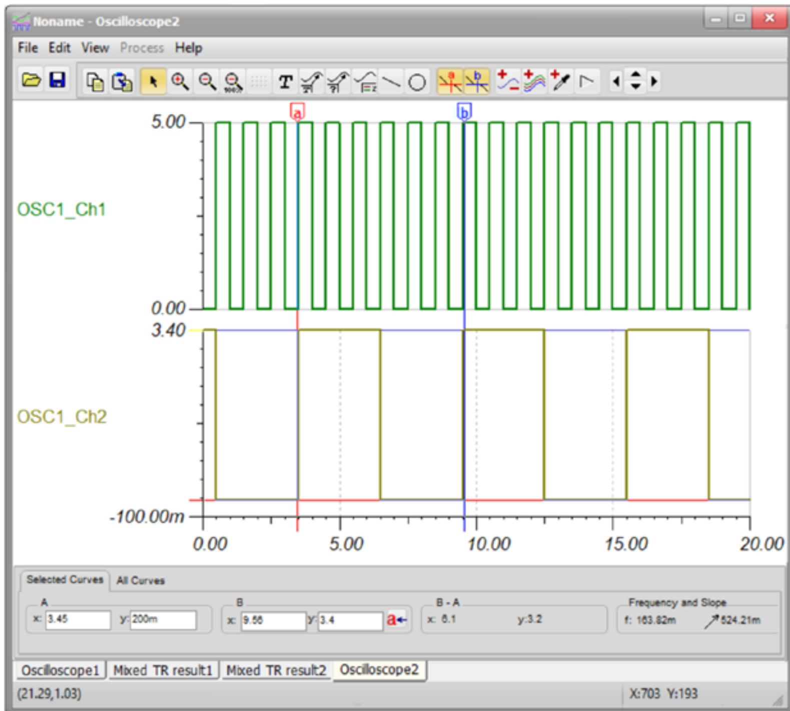


***Figure 8-11.** The Two Traces from the Oscilloscope*

If we examine the channel 2 trace, shown in Figure 8-11, we can see that it takes 6s to complete one full cycle of the counter's output. We can use this time to calculate the frequency of the output using

$$F = \frac{1}{T} = \frac{1}{6} = 0.167\ Hz$$

This confirms that the output frequency is 1/6th that of the input frequency.

The Divide by 4 Johnson Ring Counter

The circuit to test this counter is shown in Figure 8-12.

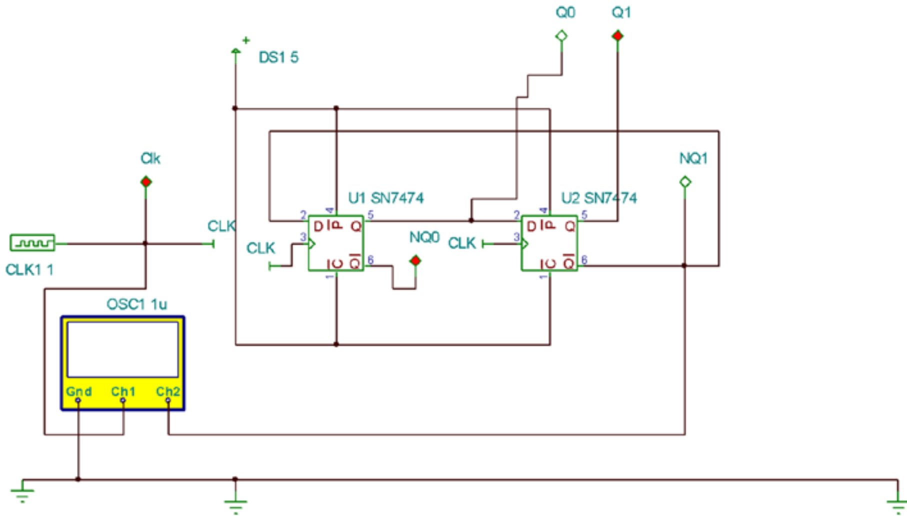


Figure 8-12. *A Divide by 4 Johnson Ring Counter*

The test circuit shown in Figure 8-12 uses only two D-type latches. The NOT Q of the second latch is fed back to the D input of the first latch. This should divide the input frequency by 4.

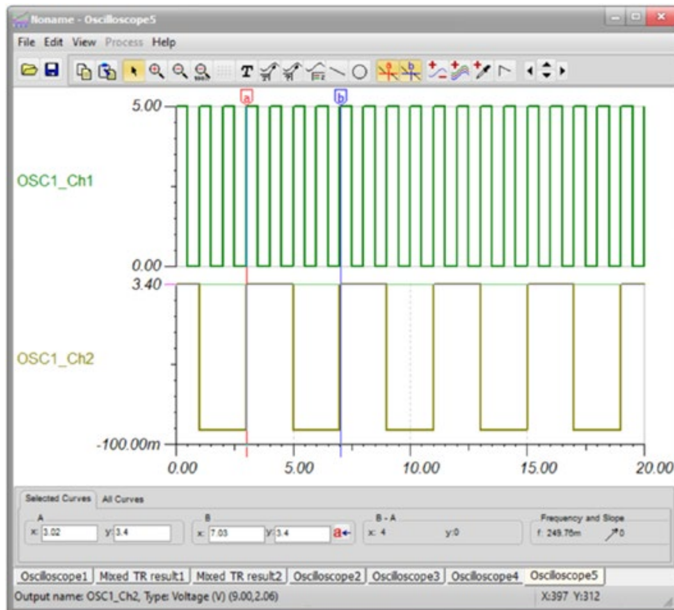


Figure 8-13. *The Two Oscilloscope Traces*

In Figure 8-13 the channel 2 trace displays the output of the counter. We can see that the periodic time for this trace, measured using the B -A option, is 4 sec. This means the frequency is 1/4Hz. This is the input frequency of 1Hz divided by 4. This confirms that this is a divide by 4 circuit.

The Phase Shift Across the Latches

In this next simulation, we are going to consider the phase shift across each of the registers in the counter. This will mean examining more than two traces. The oscilloscope will not provide us with more than two traces. However, TINA allows us to overcome this problem using the transient analysis. This enables us to look at the first few seconds, or minutes, in total from the startup of the circuit. We need to set up the circuit as shown in Figure 8-14. We need to add an output for everywhere we want to see

what is happening. We also need to insert a logic probe at these outputs. This setup is shown in Figure 8-14. A more detailed description of how to set up a transient analysis is given in Chapter 11.

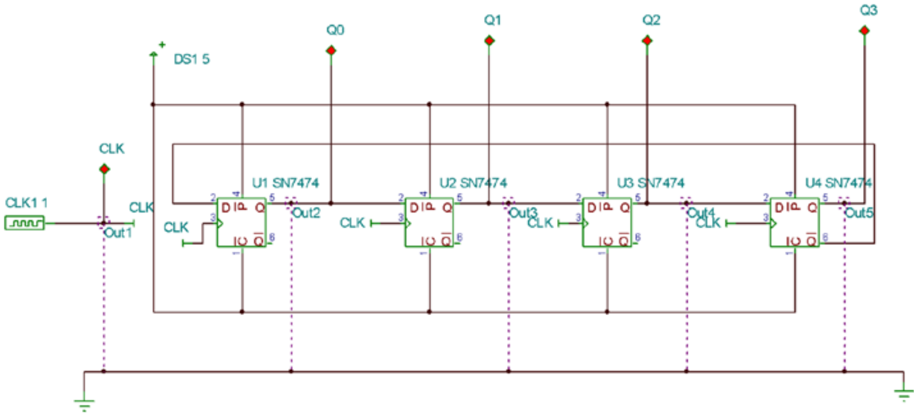


***Figure 8-14.** The Test Circuit to Consider the Phase Shift*

You should be able to see the five outputs we have inserted in the circuit. These outputs are available from the Inset option on the main menu bar, as shown in Figure 8-15.

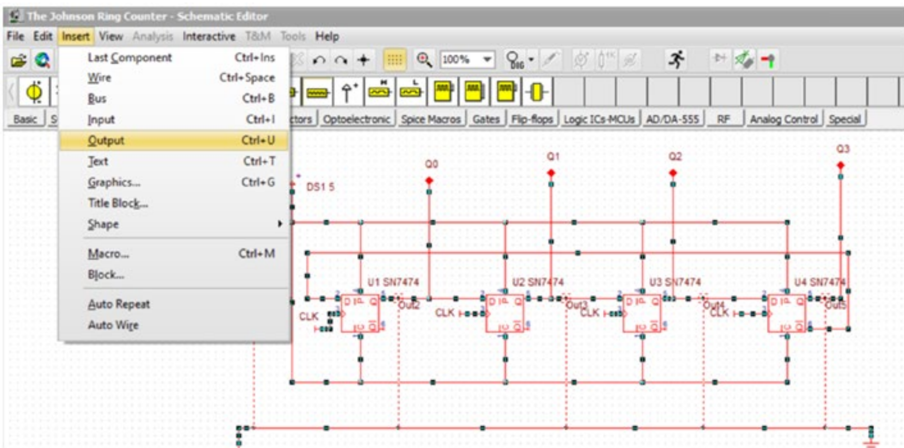


***Figure 8-15.** Inserting Outputs in the Schematic*

Unfortunately, there is an issue when placing the outputs. They may not connect to the points in the circuit. You will know they are connected when the schematic shows the connecting circle at the point of connection. To get the outputs to connect properly, you may need to deselect the output line, then reselect it, and move it slightly to one side. It should then connect correctly. This may need some practice to get it right, but it is worth it.

Once you have all five outputs correctly placed and connected, you should select Analysis from the main menu bar and then Transient from the drop-down menu that appears, as shown in Figure 8-16.

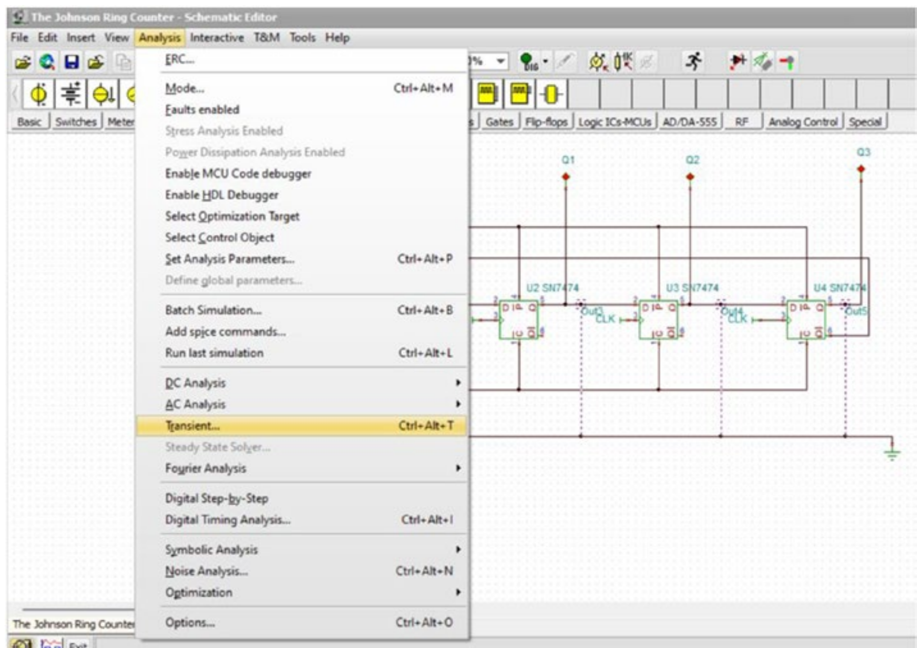


Figure 8-16. Selecting the Transient Analysis

When you select the Transient option, the Transient Analysis window should appear, as shown in Figure 8-17.

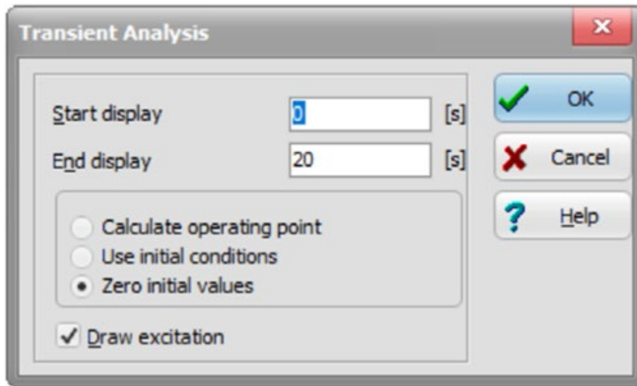


Figure 8-17. *The Transient Analysis setup window*

If you set it up as shown in Figure 8-17, the software will allow us to look at all the outputs we have added up to the first 20s of time. The traces obtained are shown in Figure 8-18.

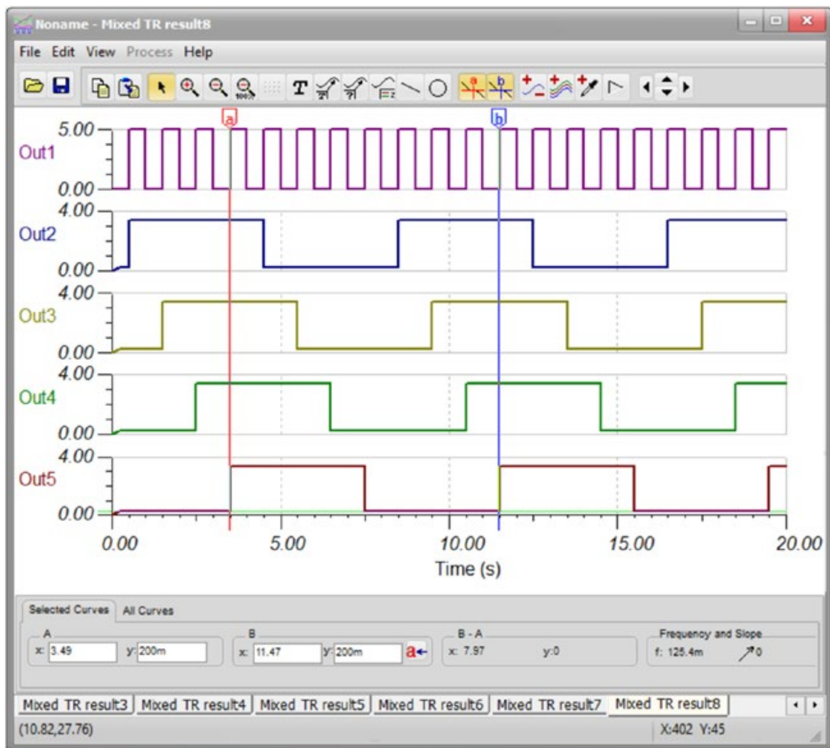


Figure 8-18. *The Traces for All Five Outputs Placed on the Circuit*

Looking at Out5 we can measure the periodic time at 8 sec. This is eight times longer than the input at Out1. This confirms that this is a frequency divide by 8. Also, we can see that all outputs 2–5 have the same frequency.

The aspect we are looking at is the phase difference between the outputs. This is shown in Figure 8-19. We can use the two cursors “a” and “b” to measure the time difference between the start of Out2 and the start of Out3. This is shown to be 1s. As the waveforms cover 360° in 8s, that is, one complete cycle, then in 1 sec the waveform covers 45°. We can see that this phase difference is constant between each trace 2 to 5.

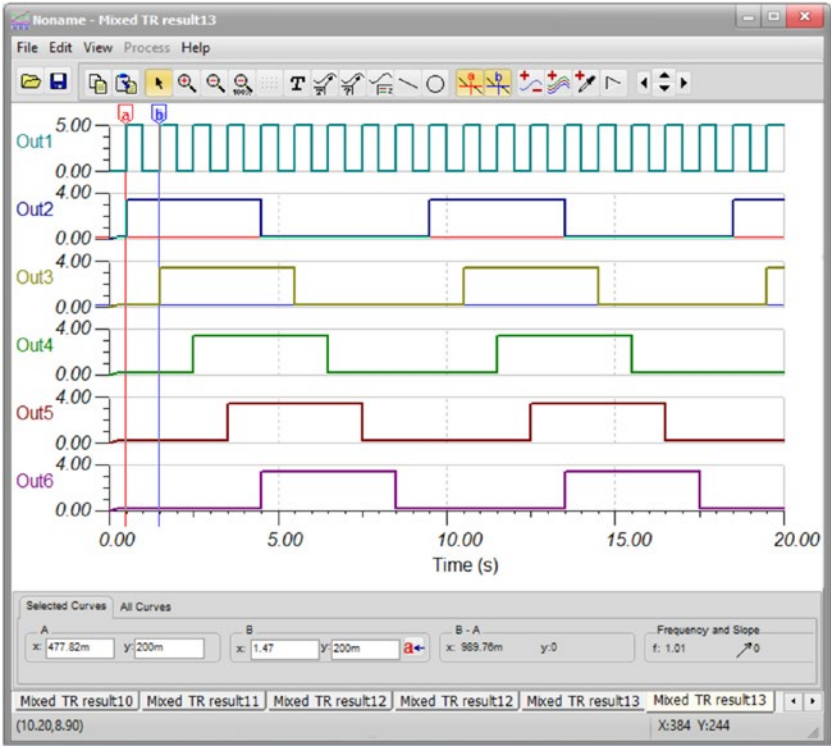


Figure 8-19. *The Traces Showing the One-Second Time Difference*

The actual phase difference between the outputs depends upon how many latches are used in the circuit. With respect to the circuit shown in Figure 8-14, the phase difference between the two outputs is 90°. We will simulate a circuit with five latches in to see what the phase shift is then. The test circuit to see what phase shift can be achieved is shown in Figure 8-20.

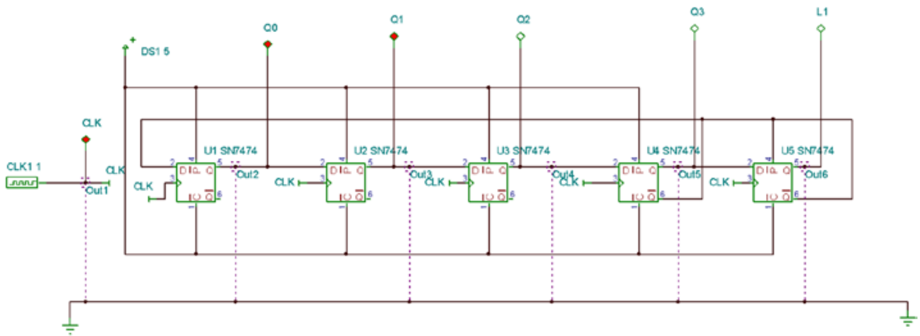


***Figure 8-20.** The Divide by 10 Johnson Ring Counter*

The output traces are shown in Figure 8-21.

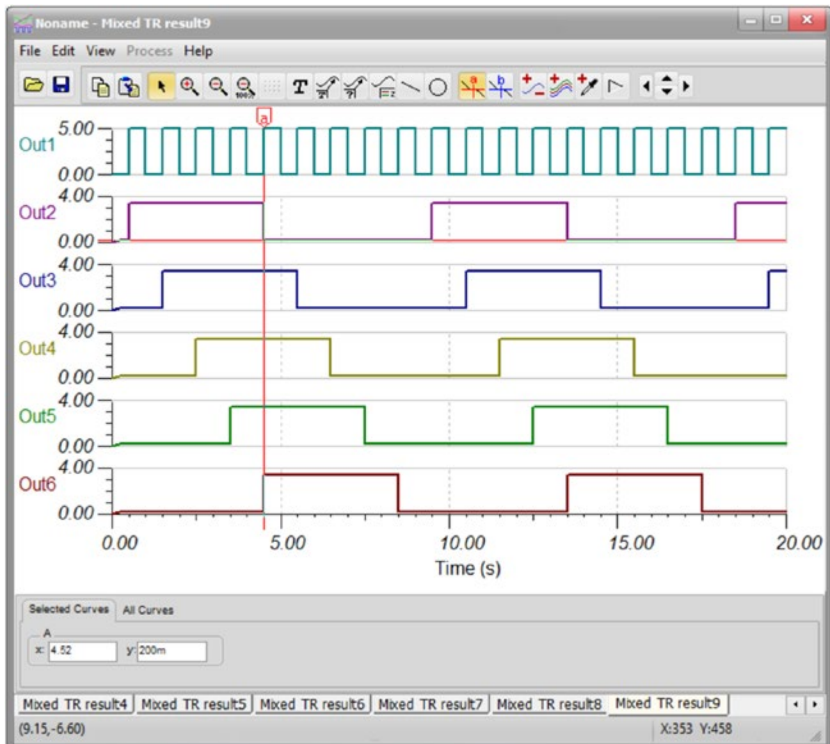


***Figure 8-21.** The Output Traces of the Divide by 10 Counter*

If we compare trace 2 to trace 6, we can see that there is a 180° phase shift between the start of each square wave. The ability to be able to produce a phase shift is useful in engineering especially in control systems.

Summary

In this chapter we have studied some basic registers including the PIPO, SIPO, etc. We have also looked at how we can create and use a ring counter to create a frequency divider and a phase shift circuit. I hope you have found this chapter informative and useful.

In the next chapter, we will look at how a structured design method can be applied to designing some real applications of logic circuits, such as a set of traffic lights and an egg timer.

CHAPTER 9

Designing Some Useful Logic Circuits

In this chapter we will look at how a structured design method can be applied to designing some real applications of logic circuits.

Example 1: A Design Process for a Single Set of Traffic Lights

This design process will start by creating a timing table that can be derived from the following sequence of events:

- The red lamp comes on at time zero.
- Five seconds later the amber lamp comes on as well.
- Two seconds later both the red and amber lamps go out and the green lamp comes on.
- Five seconds after that, the green lamp goes out and the amber lamp comes back on.
- Two seconds later the amber lamp goes out, and the cycle repeats.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_9

The timer is at the heart of this circuit, and we could design a modulo 14 counter using JKs or D-type latches; however, there are a number of counter ICs that are readily available to us that can be configured as a modulo 14 counter. The SN7493 counter is one such device. We will use that counter to create the timing source for the circuit.

The design of this circuit is really only about the output logic as we need to design a circuit that will turn on and off the red, amber, and green lights of the traffic lights at the specified times. We have been given the timing of these events, and so using that specification, we can create a timing table for the lights. This is shown in Table 9-1.

Table 9-1. *The Timing Table for the Traffic Lights of Example 1*

Q_3	Q_2	Q_1	Q_0	Action	Output Requirement
0	0	0	0	Red on	Set red on.
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1	Amber on	Set amber on.
0	1	1	0		
0	1	1	1	Red off, amber off, green on	Reset red and amber and set green on.
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		

(*continued*)

Table 9-1. (*continued*)

Q_3	Q_2	Q_1	Q_0	Action	Output Requirement
1	1	0	0	Green off, amber back on	Reset green and set amber on again.
1	1	0	1		
1	1	1	0	Amber off, cycle repeats	Reset amber and start the sequence again.

From the preceding table, it is clear there are five important times at which the lights undergo a change. Using those five times, the following Boolean expressions, one for each change of state, can be obtained:

$$\text{RedOn} = \overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}$$
$$\text{AmberOn} = \overline{Q_3}.Q_2.\overline{Q_1}.Q_0$$
$$\text{RedOff} = \overline{Q_3}.Q_2.Q_1.Q_0$$
$$\text{AmberOff} = \overline{Q_3}.Q_2.Q_1.Q_0$$
$$\text{GreenOn} = \overline{Q_3}.Q_2.Q_1.Q_0$$
$$\text{GreenOff} = Q_3.Q_2.\overline{Q_1}.\overline{Q_0}$$
$$\text{AmberBackOn} = Q_3.Q_2.\overline{Q_1}.\overline{Q_0}$$
$$\text{AmberOffAgainCycleRepeats} = Q_3.Q_2.Q_1.\overline{Q_0}$$

It is easy to see that each action can be taken from a four-input AND gate. However, before we jump into the circuit construction, it should be appreciated that the condition for each change of state will only remain true for one second of the clock input. However, the lights should not change back to the previous state even though the conditions for them to change have gone. For example, the red lamp comes on when all four Q outputs are at logic "0." It should then stay on for the next five seconds, but the Q output would have changed in the meantime. To achieve this condition of keeping the lights on, even though the conditions have

changed, we will use an SR latch to turn the lamps on and off, that is, set and reset. That is why I have added the extra column in Table 9-1.

There is also the situation that the amber lamp can be set and reset with two sets of output states. This means that the amber can be set with

$$Amber\ On = \overline{Q_3}.Q_2.\overline{Q_1}.Q_0 \text{ or } Amber\ Back\ On = Q_3.Q_2.\overline{Q_1}.\overline{Q_0}$$

It can be reset with

$$Amber\ Off = \overline{Q_3}.Q_2.Q_1.Q_0 \text{ or } Amber\ Off\ Again\ Cycle\ Repeats = Q_3.Q_2.Q_1.\overline{Q_0}$$

This means that the output logic will be quite a complex circuit; however, if you know what is required and take your time checking each stage as you progress through the design, you should be able to produce a working solution. I say test each stage as you progress as you will be very lucky if you can design the whole circuit successfully at your first attempt. That being the case, you are better finding the fault on the smaller sections of the design than on the whole circuit.

Table 9-1 shows us that we should set the red lamp on when all four Qs are at a logic "0" and then reset it, that is, turn the red off, when Q_0, Q_1, and Q_2 are at a logic "1" and Q_3 is a logic "0." The two expressions for these conditions are

$$Set\ Red = \overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}$$

$$Reset\ Red = \overline{Q_3}.Q_2.Q_1.Q_0$$

We can test this part of the lights with the test circuit shown in Figure 9-1.

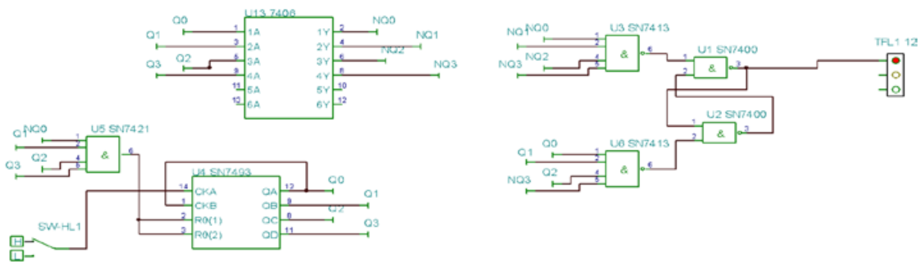


Figure 9-1. *The Test Circuit for Setting and Resetting the Red Lamp*

With this test circuit, we are using the 7493 counter. It needs the Q_0 output to be fed back to the clock B, CKB, input, while the main clock input goes to the CKA input. Also, to make this into a modulo 14 counter, we have applied the output of a four-input AND gate to both the reset inputs R0(1) and R0(2). As this counter does not provide us with the NOT Q outputs, we are using a hex inverter IC, the 7406, to provide the NOT Q logic we need for the circuit.

When we simulate this circuit, we see that the red lamp comes on straight away, as expected. It also turns off when the counter gets to show 7 sec in binary. However, there is a problem, which is that the red lamp comes back on as soon as the count changes to 8 in binary. The reason this happens is that in changing from 7 in binary to 8 in binary, there is an instance when all four Qs are at a logic "0." This is when Q_0, Q_1, and Q_2 all turn off, that is, go to logic "0," just before Q_3 changes from logic "0" to logic "1." There is an instance when all four Qs are at a logic "0." This is called "race hazard," and it is due to the time the gates take to change their logic and the fact that the change has to ripple through from one JK to the next. This time delay is called propagation delay. It is a common problem in logic, and we have to find a workaround. One such workaround would be to simply reset the red lamp when the timer went to 8 sec and not 7. This would mean the expression to reset the red lamp would be

$$Reset\ Red = Q_3.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}$$

The circuit to test this expression is shown in Figure 9-2.

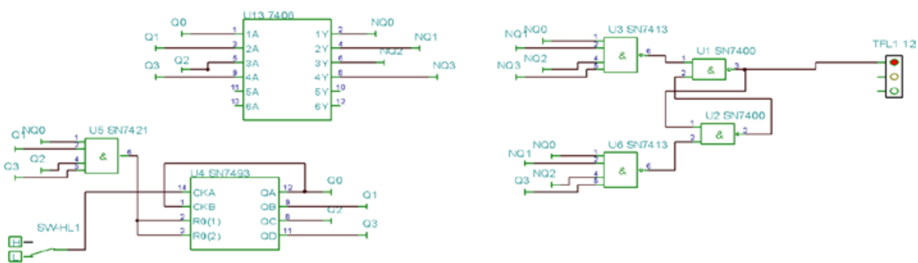


Figure 9-2. *The Race Hazard Avoided by Resetting Red at a Count of 8*

This does overcome the race hazard of the previous circuit. This small change in timing will ripple through the rest of the sequence and change the expressions for all the other timings. This race hazard is a common occurrence in what is termed "asynchronous sequential circuits." An asynchronous sequential circuit is one in which not all, if any, of the actions are synchronized to a clock input. The ripple counter only has the first JK flip flop that has a clock input. In this way then, the ripple counter can be viewed as an asynchronous sequential circuit.

We can see that this simple solution does overcome this race hazard. However, a better solution to the race hazard might be to use a synchronous up counter instead of the simple ripple counter. We will use the 4-bit synchronous counter we looked at in Chapter 8. This will remove the requirement for the hex inverter as we will be able to use the NOT Q outputs from the counter circuit. The test circuit for this is shown in Figure 9-3.

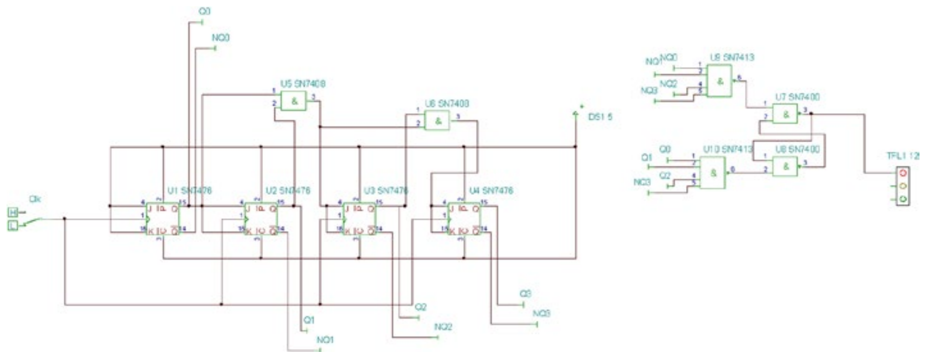


Figure 9-3. *Using a Synchronous Up Counter to Avoid the Race Hazard*

The synchronous up counter does not suffer from the race hazard as all outputs change at the same instance. Therefore, to move on with the circuit, we will use the synchronous counter. Another benefit of using the synchronous counter is that the count starts straight away, whereas with the ripple counter, there is a redundant count right at the start.

Now that we are using the synchronous counter, we can use the original expressions for the set and reset of all lamps. However, we must now create a synchronous modulo 14 counter. This will involve redesigning some of the input logic circuit to accommodate this. We will look at that redesign at the end as shown in Figure 9-8. This is because we don't need the counter to reset yet.

We will now move on to the amber lamp. The initial expressions for this lamp are

$$Set\ Amber = \overline{Q_3}.Q_2.\overline{Q_1}.Q_0$$

$$Reset\ Amber = \overline{Q_3}.Q_2.Q_1.Q_0$$

The test circuit for this action is shown in Figure 9-4.

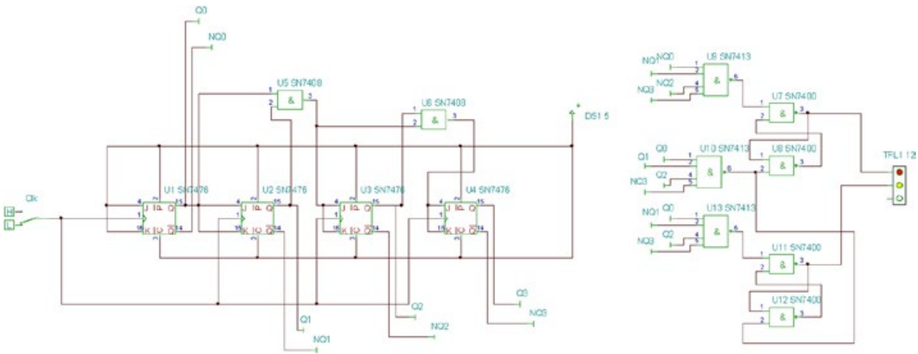


Figure 9-4. *The Test Circuit for Red and Amber*

When we simulate this circuit, we see that it works as expected. Therefore, we can go on and consider the control of the green lamp. The expressions for the green lamp are

$$Set\ Green = \overline{Q_3}.Q_2.Q_1.Q_0$$

$$Reset\ \ Green = Q_3.Q_2.\overline{Q_1}.\overline{Q_0}$$

The test circuit for this is shown in Figure 9-5.

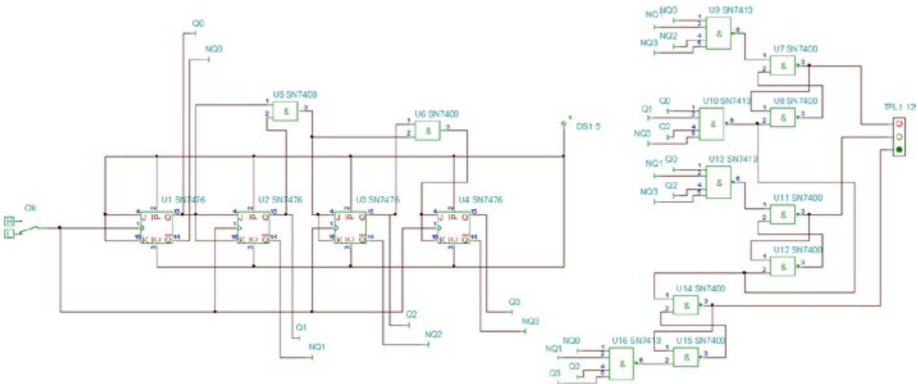


Figure 9-5. *The Test Circuit for the Green Lamp*

To try and make the circuit design clearer, Figure 9-6 shows just the output logic for the control of the three lamps so far.

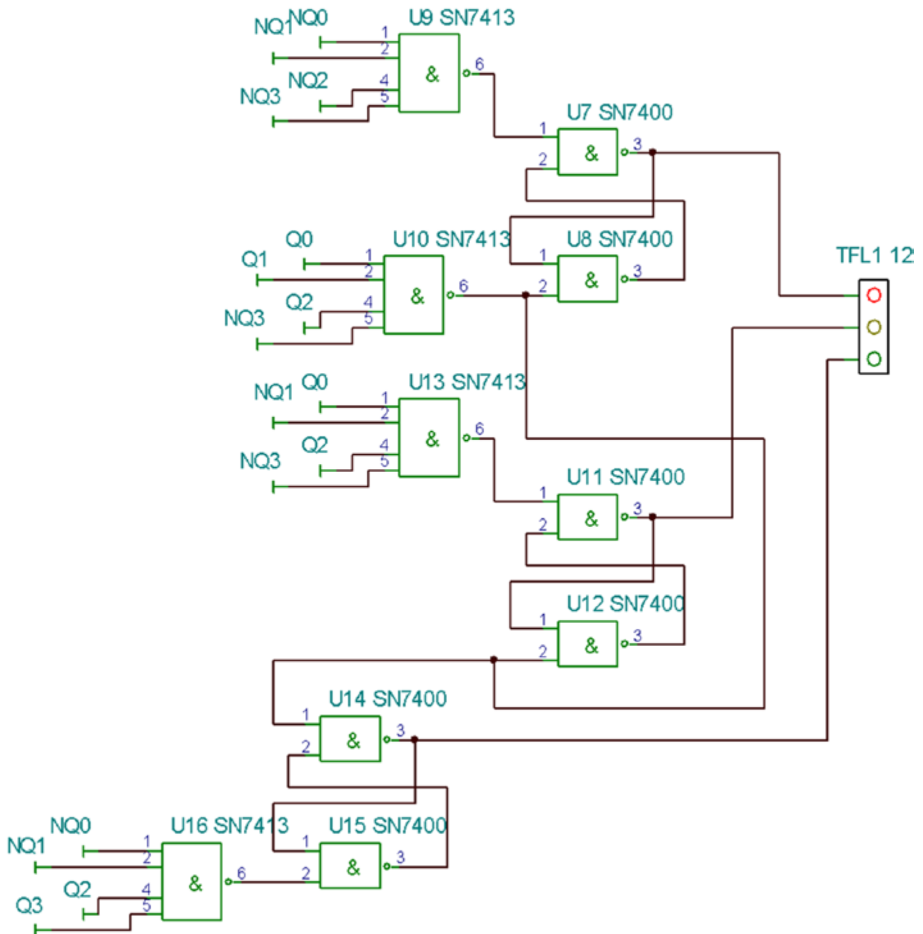


Figure 9-6. *The Output Logic Circuit for the Three Lamps*

Figure 9-6 shows only the output logic section of the traffic light circuit. The synchronous counter is still part of the circuit, but it is not shown here so that we can make the output logic large enough for you to see how it is constructed. We can see that the set input for the green is taken from the reset input of the amber, which in itself is taken from the reset of the

red lamp. This is because these timings are all the same time. When we simulate this circuit, we see that it also works well. All that is left for us to consider is the second timing for setting and resetting the amber lamp. The expressions for these setting and resetting are

$$Set\ Amber\ 2 = Q_3.Q_2.\overline{Q_1}.\overline{Q_0}$$

$$Reset\ Amber\ 2 = Q_3.Q_2.Q_1.\overline{Q_0}$$

However, this is a second way in which we want to control the amber lamp. This means we control the amber lamp by way1 OR by way2, and so we need an OR gate to process this with. The output logic circuit that implements this is shown in Figure 9-7.

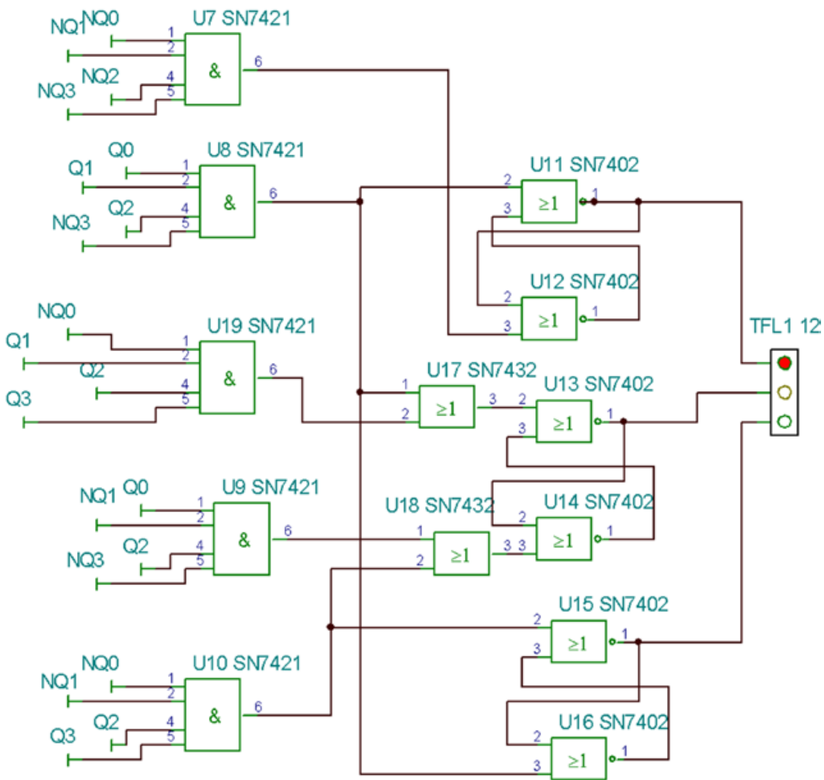


Figure 9-7. *The Final Output Logic Circuit*

Figure 9-7 only shows the output logic of the circuit. The counter circuit is still there, but we need to focus on this output logic. The circuit now uses NOR gates instead of the NAND gates to make the SR latches. This is because these are active high SR latches, whereas the NAND SR latches are active low. We need active high SR latches because the OR gate output goes high, not low, which means we can't use the NAND gates that we used before. The other difference is that the S and R inputs to the SR latches have swapped over. Also, this means we have had to use AND gates to interpret the internal signals derived from the Q outputs of the counter instead of NAND gates as used previously. You might want to try designing the circuit using the initial NAND gates to create the SR latches. I am sure if you do you will see why we have changed to the NOR gates for the SR latch.

Analysis of the Output Logic

It might be useful to analyze how the output logic circuit works. This analysis is with respect to the circuit shown in Figure 9-7.

We will start with the red lamp. The expression to set the red lamp on is

$$Set\ Red = \overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}$$

Therefore, we connect the four inputs to the U7 AND gate to the four NOT Q outputs of the counter. The U7 AND gate will produce a logic "1" when we first turn the circuit on as all Q outputs will be a logic "0" and all NOT Q outputs will be a logic "1." This means all inputs to the U7 AND gate will be a logic "1," and its output will be a logic "1," which will set the SR latch made up of gates U11 and U12. This then turns on the red lamp.

The expression to reset the red lamp is

$$Reset\ Red = \overline{Q_3}.Q_2.Q_1.Q_0.$$

That is why the U8 AND gate has the four inputs $\overline{Q_3}$, Q_2, Q_1, and Q_0 from the counter. This will produce four logic "1"s at the input of U8, and its output will be a logic "1," which will reset the SR latch made up of gates U11 and U12. This then turns off the red lamp.

The first expression to set the amber lamp is

$$Set\ Amber = \overline{Q_3}.Q_2.\overline{Q_1}.Q_0$$

These NOT Qs and Qs are the input to the U9 AND gate. Therefore, this will produce a logic "1." This logic "1" is one of the two inputs to the U18 OR gate; the other input will be a logic "0" at this time. This means the output of this gate will go to a logic "1," and this will set the SR latch made up of gates U13 and U14. This then turns on the amber lamp.

The first expression to reset the amber lamp is

$$Reset\ Amber = \overline{Q_3}.Q_2.Q_1.Q_0$$

This is the same NOT Q and Q inputs to the U8 AND gate. Therefore, we need to connect the input to the U17 OR gate to the output of the U8 AND gate. The other input to this U17 OR gate will be a logic "0" at this time. The output of this U17 OR will be a logic "1," and this will reset the amber lamp the first time, that is, the same time as when the red lamp resets.

The expression to set the green lamp is

$$Set\ Green = \overline{Q_3}.Q_2.Q_1.Q_0$$

This the same expression as that to reset the red and amber lamps. Therefore, we need to connect the input to the U16 NOR gate, the set input for the green SR latch, to the output of the U8 AND gate.

The expression to reset the green lamp is

$$Reset\ Green = Q_3.Q_2.\overline{Q_1}.\overline{Q_0}$$

These NOT Qs and Qs are the input to the U10 AND gate. Therefore, this will produce a logic "1." This logic "1" is the input to the U15 NOR gate, the reset input for the SR latch for the green lamp.

Now we need to set and reset the amber lamp a second time. The expression to set the amber lamp the second time is

$$Set\ Amber\ 2 = Q_3.Q_2.\overline{Q_1}.\overline{Q_0}$$

This is the same expression that resets the green lamp. Therefore, we can connect the second input to the U18 OR gate, which will go to a logic "1" when time = 12 seconds. This enables us to set the amber lamp again this second time.

The expression to reset the amber lamp the second time is

$$Reset\ Amber\ 2 = Q_3.Q_2.Q_1.\overline{Q_0}$$

These NOT Qs and Qs are the input to the U19 AND gate. Therefore, this will produce a logic "1." This logic "1" is the other input to the U17 OR gate, which will produce a logic "1," which will reset the SR latch for the amber lamp.

I hope this analysis does explain how the output logic works, and when you simulate the complete circuit, shown in Figure 9-8, you can see the circuit works as expected.

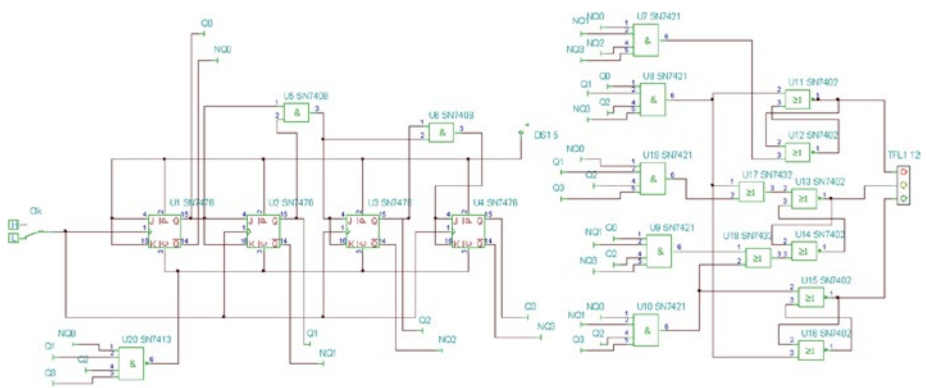


***Figure 9-8.** The Complete Circuit for Example 1*

The only addition we have not discussed is the U20 NAND gate. This is there simply to make the counter clear, that is, reset, when the counter tries to display a value of 15 seconds. The timing for the sequence of the traffic lights stops at 14 seconds. If we didn't use the NAND gate to clear the counter at 15, there would be a one-second delay after the amber had gone out before the red lamp came back on again.

Example 2: An Alternative Single Set of Traffic Lights

In this example we will design the circuit using a state diagram. This approach should follow the design procedure described in the book. It will also align itself better to the diagram of a logic system shown in Figure 5-1 in Chapter 5. The state diagram is shown in Figure 9-9.

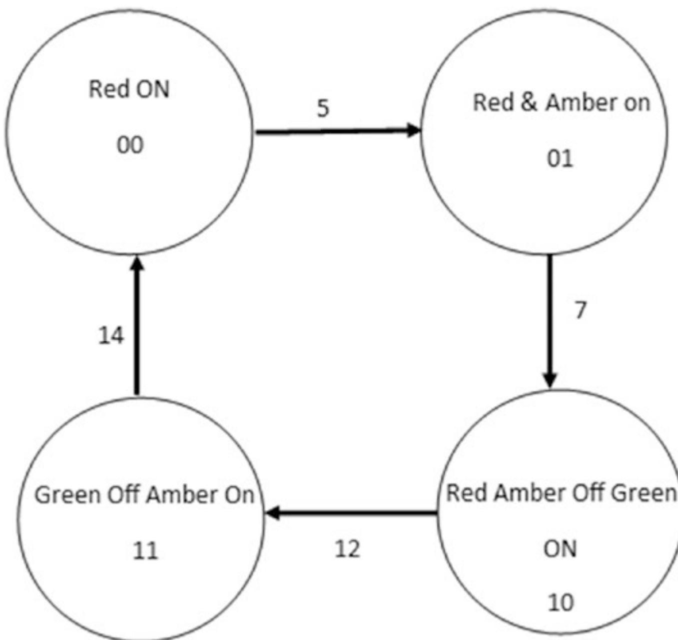


Figure 9-9. *The State Diagram for the Single Set of Traffic Lights*

The state diagram shows the logic values that are required to put the traffic lights in each state. This states that if a binary value is at the specified value, then the traffic lights should be on or off as shown in each state. For example, if the binary value was 10, then the red and amber lights should be off, and the green light should be on. This means that the system will be interrogating this binary number and acting accordingly.

The transition lines on the diagram show the time that has passed between the start of the system, when the binary value is 00, and arriving at the next state. For example, 7 sec after starting the system, the system will arrive at the third state, that is, when the binary value will be 10. This means there will be a timer controlling when we change the binary value for each state. The four states and their times are shown in Table 9-2.

Table 9-2. *The Timing of the Four States*

Time in Seconds	State Binary value
0	00
5	01
7	10
12	11
14	00

We will use a simple binary counter to count the seconds. Therefore, we can relate the state values to the Q outputs of the counter. This is shown in Table 9-3.

Table 9-3. *The Q Outputs*

Counter Outputs				**State Binary Values**			
				Present State		**Next State**	
Q_3	Q_2	Q_1	Q_0	B_1	B_0	B_1	B_0
0	0	0	0	0	0	0	0
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1	0	0	0	1
0	1	1	0				
0	1	1	1	0	1	1	0
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0	1	0	1	1
1	1	0	1				
1	1	1	0	1	1	0	0
1	1	1	1				

We need to appreciate that the circuit will split into two main parts. The first part is how the counter changes the binary values of the states in the system. The second part is how the binary values of the states control the turning on and off of the traffic lights. We will consider how the counter changes the binary values of the states first.

Using Table 9-3 we can see that the binary values change four times. However, we should appreciate that the fourth change is when the counter resets to 0000, which means we may not have to set up a separate control for this time; we can use the same control that resets the counter. This means we only need to consider the following three change conditions and when they occur.

The first change is when the counter reaches a value of 0101, that is, 5 sec after the start. Therefore, the expression for change 1 is

$$Change\,1 = \overline{Q_3}.Q_2.\overline{Q_1}.Q_0$$

The second change is when the counter reaches a value of 0111, that is, 7 sec after the start. Therefore, the expression for change 2 is

$$Change\,2 = \overline{Q_3}.Q_2.Q_1.Q_0$$

The third change is when the counter reaches a value of 1100, that is, 12 sec after the start. Therefore, the expression for change 3 is

$$Change\,3 = Q_3.Q_2.\overline{Q_1}.\overline{Q_0}$$

Now we have to consider how we are going to make these counter values change the binary number. One possible method is related to the fact that the binary values start at 0 and simply increment. This could suggest a second counter being triggered when the first counter reached the appropriate value. This could work, but we need to be aware that the second counter needs to change from high to low to increment its output. Bearing this in mind, we need to subtract 1 from each of the counter values. This means that the change values would be

$$Change\,1 = 0100 = \overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0}$$

$$Change\,2 = 0110 = \overline{Q_3}.Q_2.Q_1.\overline{Q_0}$$

$$Change\,3 = 1011 = Q_3.\overline{Q_2}.Q_1.Q_0$$

We would then need to clock this second counter with either one of the expressions. This means that the expression for this second clock, CKA, is

$$CKA = \left(\overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0}\right) + \left(\overline{Q_3}.Q_2.Q_1.\overline{Q_0}\right) + \left(Q_3.\overline{Q_2}.Q_1.Q_0\right)$$

The actual outputs of the first counter have the letters A, B, C, and D instead of the numbers 3, 2, 1, and 0. The test circuit for this change implementation is shown in Figure 9-10.

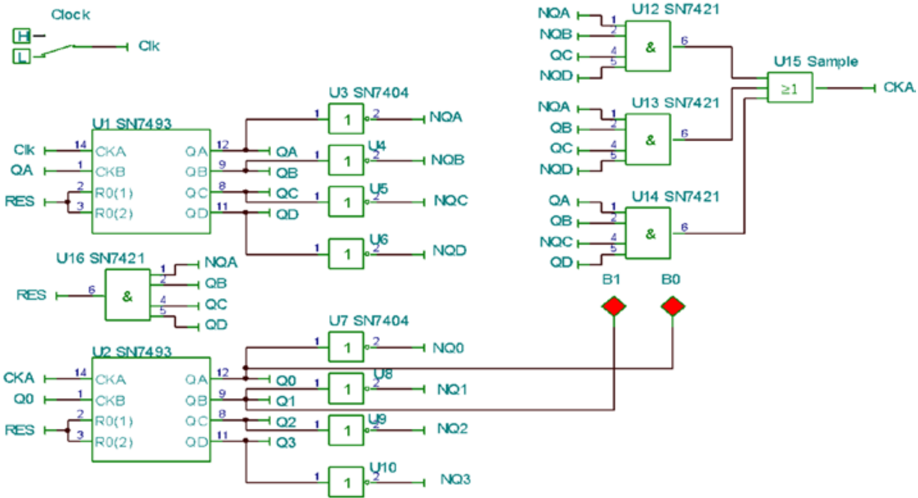


Figure 9-10. The Test Circuit for the Change Operations

Now we can concentrate on the control of the output, that is, the traffic lights. These are controlled by the binary values of the second counter. Using Table 9-4, concentrating on the red lamp, we can see there are two occasions when the red is at a logic "1." This means the expression for the red is

$$Red = \left(\overline{Q_1}.\overline{Q_0}\right) + \left(\overline{Q_1}.Q_0\right)$$

Table 9-4. *The Change of the Traffic Lights*

Second Counter		Traffic Lights		
Q_1	Q_0	Red	Amber	Green
0	0	1	0	0
0	1	1	1	0
1	0	0	0	1
1	1	0	1	0

As an exercise, determine the expressions for the amber and green lights and so confirm that the complete circuit for this set of traffic lights is shown in Figure 9-11.

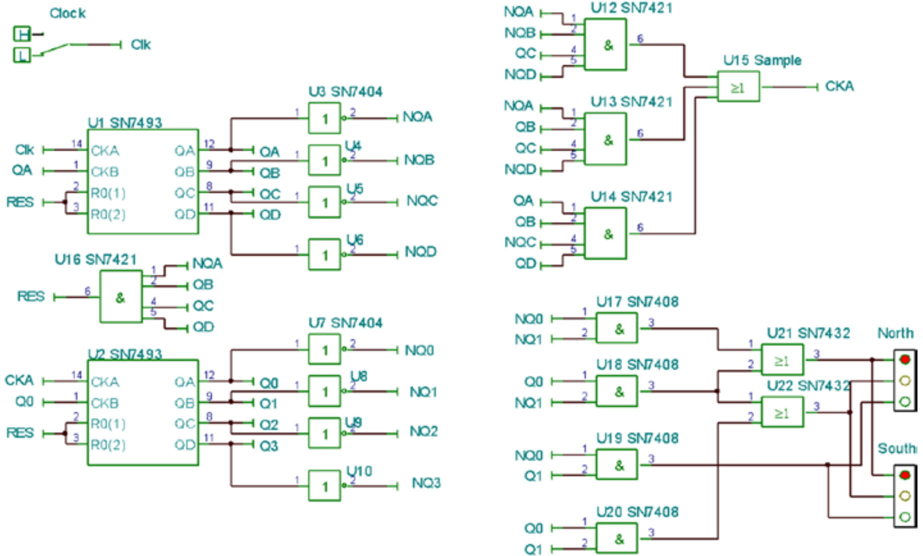


Figure 9-11. *The Complete Circuit for the Traffic Lights*

Example 3: Adding a Pelican Crossing

In the next example, we are going to add a pelican crossing for the single set of traffic lights. The state diagram for this set of lights is shown in Figure 9-12.

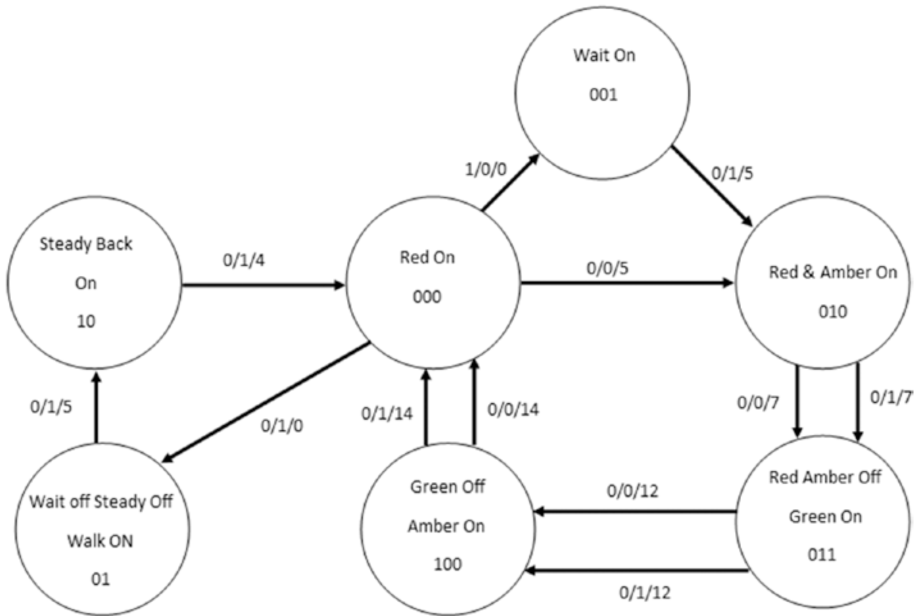


Figure 9-12. *The State Diagram for Exercise 3*

In this circuit there will be an input. This will be the button the user presses to request a time to cross the junction. Therefore, the transition lines will need to reflect this. Also there will be an output that tells the user what to do. This must also be reflected on the transition lines. The timing needs adding to the transition lines, and that is why there are three terms shown on the lines. The first is the state of the input, the second is the state of the output, and the final number is the timing from the start of the system. There are two possible routes around the diagram from Red on to Green off Amber on and then back to Red on: one when the input is 0 and

one when the input is 1. This is why there are two transition lines between those states. The fact that the input is pressed until the wait lamp comes on is shown by there only being a "1" between Red on and Wait on states.

Finally, there are two possible exits from the Red on state. If the wait lamp is 0, then the route out is the normal route. However, if the wait lamp is on, then the route out is the alternative circular route. This pauses the main traffic lights while the system goes through the pelican lights. When finished the system returns to the normal route.

This state diagram is only my initial attempt at describing the process. I am not saying it is the most efficient, but it is one that I can work with. I hope you can follow my thoughts on how the system should work.

The main concept behind my approach is that we have two main timing counters. One controls the count for the normal traffic lights, using timer 1, and one controls the count for the pelican lights, using timer 2. See Figure 9-14. The system must be able to enable and disable the respective timers when required; that is, timer 1 must be disabled when the pelican counter needs to count, and timer 2 must be disabled when the main counter needs to count. The enabling and disabling of the respective timers will be controlled in the most part by the input switch.

We will consider how the first timer controls binary values of the main route of the state diagram. This will show us how we can trigger the counter for the binary values in the state diagram. This is shown in Table 9-5.

***Table 9-5.** The Transition Table for the Binary Values*

Counter Outputs				State Binary Values					
				Present State			Next State		
Q_3	Q_2	Q_1	Q_0	B_2	B_1	B_0		B_1	B_0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	1	0	0	0
0	0	1	1	0	0	1	0	0	0
0	1	0	0	0	0	1	0	0	0
0	1	0	1	0	0	1	0	1	0
0	1	1	0	0	1	0	0	1	0
0	1	1	1	0	1	0	0	1	1
1	0	0	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1	1
1	0	1	0	0	1	1	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	0	0	1	1	1	0	0
1	1	0	1	1	0	0	1	0	0
1	1	1	0	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0

Using Table 9-5 we can see that there are five occurrences, the shaded rows, where there is a change. As with the previous design, these changes will trigger the clock input of the second counter, CKA. Therefore, the expression for the CKA is

$$CKA = \sum 0001, 0101, 0111, 1100, 1110$$

Again, because of the triggering of the second counter, we need to take one from these values. Therefore, the expression for the CKA is

$$CKA = \sum 0000, 0100, 0110, 1011, 1101$$

We don't need to consider the fifth term as we can use the resetting of the timer instead. This means the expression for the CKA is

$$CKA = \left(\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}.\right) + \left(\overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0}\right) + \left(\overline{Q_3}.Q_2.Q_1.\overline{Q_0}\right) + \left(Q_3.\overline{Q_2}.Q_1.Q_0\right)$$

The circuit to test this is shown in Figure 9-13.

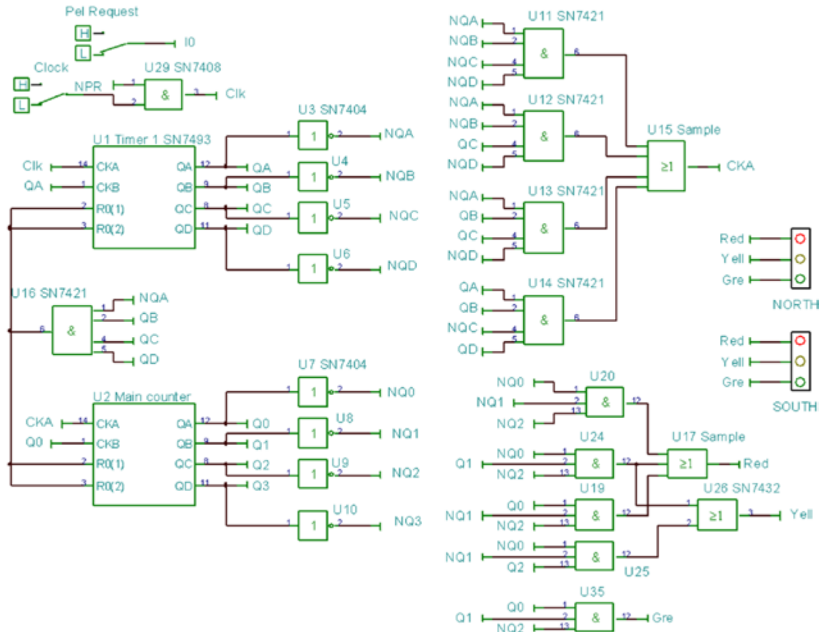


Figure 9-13. *The Test Circuit for the Triggering of the U2 Main Counter*

We can see that the four four-input AND gates that feed the four-input OR gate, U15, do match the expression for the CKA. This circuit works as expected, and so we can move on to design the output logic. To do this we

need to construct the table to show how the traffic lights are related to the binary values. This is shown in Table 9-6.

Table 9-6. *The Table for the Control of the Traffic Lights*

Main Counter U2			Traffic Lights North and South		
Q_2	Q_1	Q_0	Red	Amber	Green
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	0	0	1
0	0	0	1	0	0

We only need to consider the first three outputs of this main counter as it will only count up to a value of 4, that is, 100 in binary. See the state diagram in Figure 9-12. Using this table, we can see there are four unique occurrences for the red being on. Using these occurrences, the expression for the red is

$$Red = \sum 000, 001, 010, 011$$

$$Red = \left(\overline{Q_2}.\overline{Q_1}.\overline{Q_0}.\right) + \left(\overline{Q_2}.\overline{Q_1}.Q_0\right) + \left(\overline{Q_2}.Q_1.\overline{Q_0}\right) + \left(\overline{Q_2}.Q_1.Q_0.\right)$$

As an exercise, determine the expressions for the amber and green lights and so confirm that the complete circuit for the output logic for the set of traffic lights is shown in Figure 9-13, using gates 19, 20, 24, 25, and 35.

The main timer is shown as U1, and this is a basic binary counter using the 7439 IC. The main clock input to the timer is from the U29 AND gate output labeled Clk. The AND gate is used to enable the clock input to be passed on to the Clk line when the second input, the NPR, is a logic “1.”

This NPR will be a logic "1" only when we are not clocking the pelican section of the circuit. This gives us the control over the two timers that we need.

The timer provides the outputs that are fed back to the AND gates U11, U12, U13, and U14, which feed the OR gate, U15, to provide the clock input to the main counter U2.

Both the main timer and the main counter are reset when the timer reaches 14. This is implemented using the AND gate U16.

Figure 9-14 shows the complete circuit for the North-South set of traffic lights with a pelican crossing. This is quite a complex circuit, and it does split into two distinct areas. The main set of traffic lights with the input switch is shown in Figure 9-13. The pelican section of the circuit is shown in Figure 9-15.

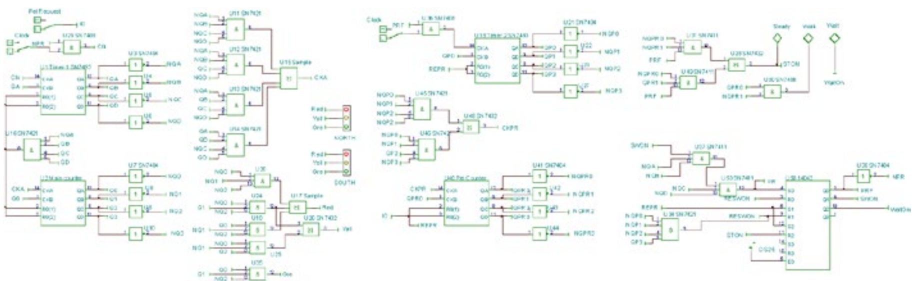


Figure 9-14. *The Completed Circuit for the North-South Traffic Lights with Pelican Crossing*

Figure 9-15 shows the pelican section of the circuit. This can be further split into two sections. The first, in Figure 9-16, shows how timer 2 and the pelican counter are set up. The second, in Figure 9-17, shows how the outputs are controlled by the pelican counter. It also shows how we have used the 14043 IC, U50, to set up some SR latches required to maintain the switching correctly.

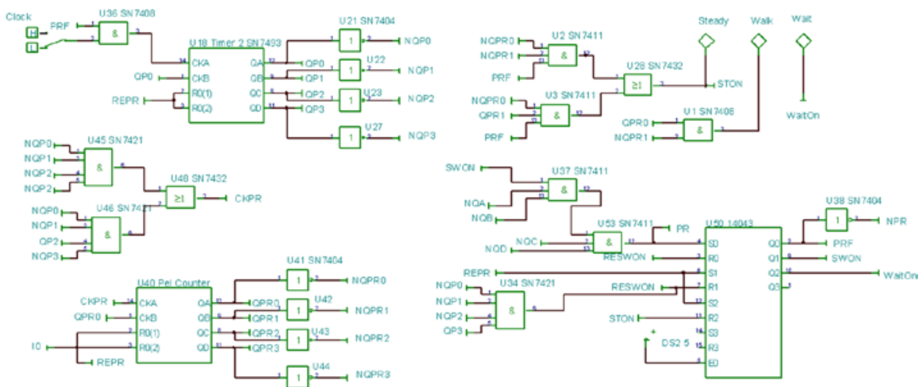


Figure 9-15. *The Pelican Section of the Complete Circuit*

This shows the timer and counter as well as the outputs and inputs for the pelican section of the circuit.

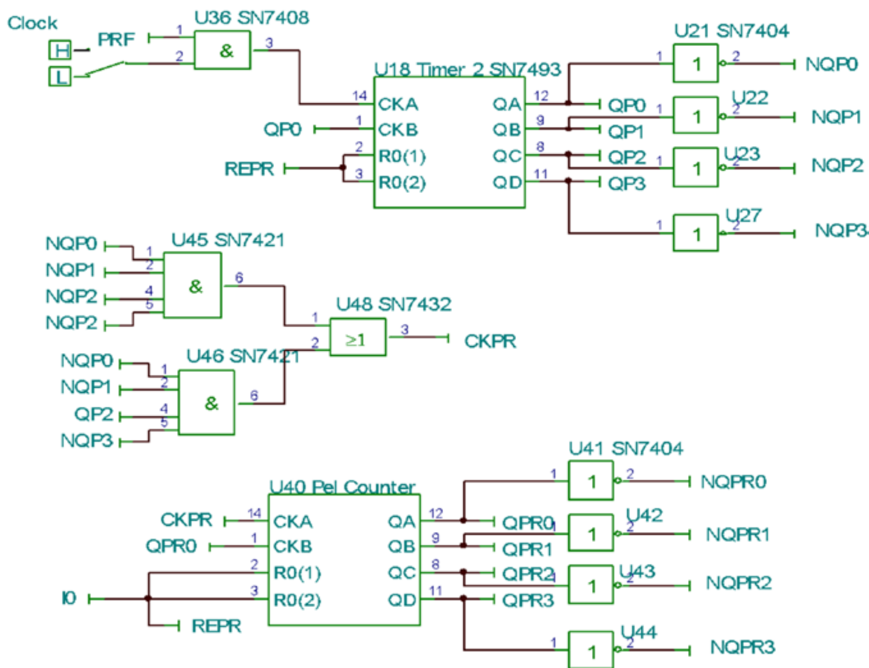


Figure 9-16. *The Timer and Counter for the Pelican Section of the Circuit*

Timer 2, U18, works in the same way as timer 1. The clock input switch works in tandem with the main clock that feeds timer 1. Again, an AND gate, U36, is used to facilitate the enabling and disabling of timer 2 as required.

We can construct a transition table to show how the outputs of this timer facilitate the clock input to the pelican counter, U40. This is shown in Table 9-7.

Table 9-7. *The Transition Table for the Pelican Binary Values*

QPR_3	QPR_2	QPR_1	QPR_0	P_1	P0	P1	P_0
0	0	0	0	0	0	0	1
0	0	0	1	0	1	0	1
0	0	1	0	0	1	0	1
0	0	1	1	0	1	0	1
0	1	0	0	0	1	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	0
1	0	0	0	1	0	0	0

There are only two changes, for the binary values, that are of interest as the last change, when the QPR outputs are 1000, is when the timer and counter should reset. Using those occurrences gives us the following expression for the CKPR, pelican clock input:

$$CKPR = \sum 0000, 0100$$

Therefore, the expression is

$$CKPR = \left(\overline{QPR_3}.\overline{QPR_2}.\overline{QPR_1}.\overline{QPR_0}\right) + \left(\overline{QPR_3}.QPR_2.\overline{QPR_1}.\overline{QPR_0}\right)$$

In Figure 9-16 we can see that these two terms are the inputs to the AND gates U45 and U46. The outputs of these two AND gates feed the two-input OR gate, U48, which provides the clock signal CKPR to trigger the PEL counter, U40, in the circuit.

The outputs of the PEL counter are used to control the outputs of the pelican crossing. This part of the circuit is shown in Figure 9-17.

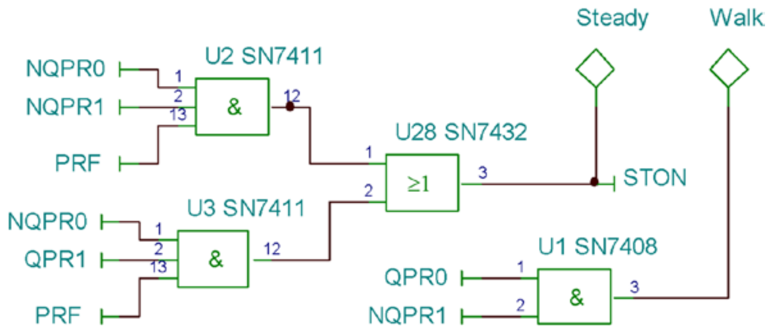


Figure 9-17. *The Output Drive Circuitry for the Pelican Crossing*

We can determine how the outputs of the PEL counter control the lights of the pelican crossing using Table 9-8.

Table 9-8. *The Table for the Control of the Pelican Cross Lights*

Row	PRF	QPR3	QPR2	QPR1	QPR0	Steady	Walk
1	1	X	X	0	0	1	0
2	1	X	X	0	1	0	1
3	1	X	X	1	0	1	0
4	X	X	X	0	1	0	0

To understand how this table works, we should appreciate that the state of the two lights depends upon the four outputs of the PEL counter but also, with the steady light, the logic on the PRF and PR digits. Only the QPR0 and QPR1 outputs of the PEL counter affect the steady and walk lamps as the state diagram only has two states in the pelican crossing route.

The PRF bit is the output of the SR latch that will be set whenever the user presses the input to the lights, the pelican request button. A request to make the traffic lights to go around the pelican crossing states must have happened before these two lights can be activated. Therefore, this bit must be a logic "1" as shown in Table 9-8.

The steady light comes on if we are in the first state of the main loop and the pelican request button has been pressed. The logic of the two outputs of the PEL counter would be 0000 as shown in the state table.

In the next state of the pelican loop, the steady light turns off and the walk light turns on. This is signified by the logic of the counter changing to 0001, which happens when the PEL counter increments.

In the next state, shown in Figure 9-12, the PEL counter has incremented to a value of 0010. This turns the walk off and the steady back on. These conditions are implemented with the logic circuit shown in Figure 9-17.

The final part of the circuit is shown in Figure 9-18. It shows how we have dealt with the issues associated with the switches.

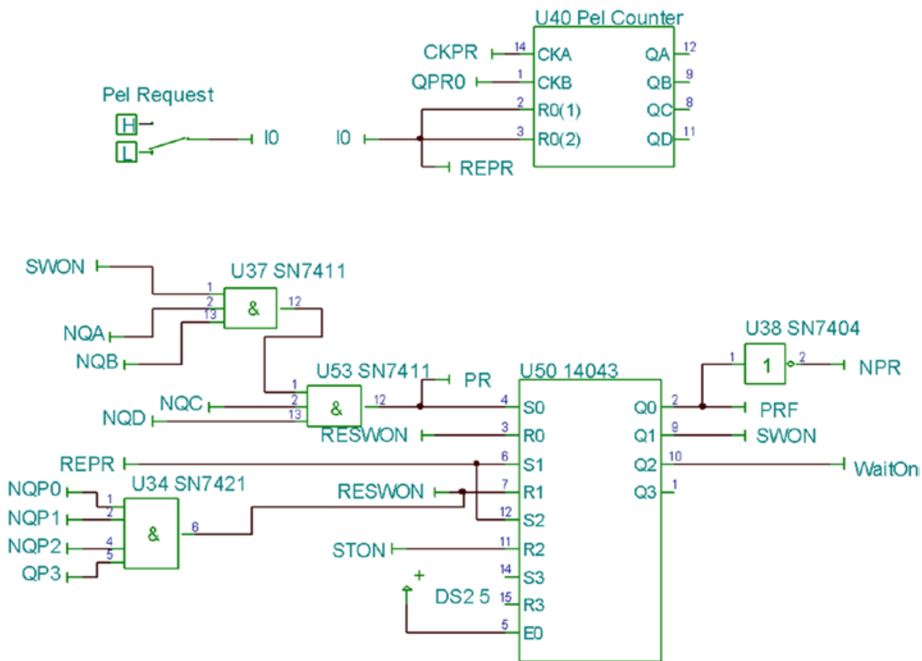


Figure 9-18. The Control of the Input Switches and Other Switching

The first issue to deal with is that the user should not have to keep the pelican request button pressed. They should be able to send it to a logic "1" once and then let it return to a logic "0." It is the basic SR latch that will come to our aid. As the switch goes to a logic "1," then we should use the NOR gate format of the SR latch, as these are active high. We can, and do, use the 14043 IC, which has four active high SR latches in it. We are using the second of the four SR latches to latch the PEL request button. This is shown in Figure 9-18.

Pressing the PEL request button puts a logic "1" on the I0 bit. This, in turn, puts a logic "1" on the REPR bit at the inputs to the PEL counter; see Figure 9-18. Note, in the real circuit, these parts of the circuit are not placed next to each other as they are in Figure 9-18. This is done for convenience only.

The fact that the bit REPR has gone to a logic "1" sets the output of the SR1 latch. This means that the SWON bit, connected to the Q1 output, goes to a logic "1." This means that the REPR bit can now go back to a logic "0" as the user releases the PEL request button.

The REPR bit is also the input to set the third SR latch in the 14043 IC. This will send the logic on the Wait on bit to a logic "1." This is connected to the wait lamp and so it turns on.

If the SWON is a logic "1" when the four NOT outputs of timer 1 are at a logic "1," then this would mean that the circuit has gone back to a red lamp on and recognized that the user had pressed the PEL request button. The system would then turn off the wait lamp and enter into the pelican loop of the state diagram.

The wait lamp is turned off when the third SR latch is reset. The bit that goes to a logic "1" to reset this latch is the STON bit; see the R2 input on the 14043 IC in Figure 9-18. The STON bit goes to a logic "1" when the "steady" lamp is lit; see Figure 9-17.

One major aspect of the circuit is that we have to run it through a pelican crossing routine first. This means that we should press the pelican request button first. Then, because the outputs of the main timer 1 are all at a logic "0," then the circuit can run through the pelican routine when we operate the clock transition from high to low. The pelican routine and the North-South traffic lights will go through the normal routine.

We can test the circuit now by operating the PEL request button at any point in the traffic light routine, except when the red lamp has just come on. Try operating the PEL request switch when the amber light has just come on. When you do that, the wait lamp should come on. Note you only need to press the PEL request switch and then release; you do not need to keep the switch pressed. With the wait lamp still lit, the traffic lights will progress through their normal sequence. When the amber lamp goes out and the red lamp comes back on, the circuit will now move into the pelican sequence. This becomes evident as the wait lamp goes out and the steady lamp comes on. The main timer 1 is now disabled and timer 2 is enabled. This stops the

traffic lights from changing, and the red lamp stays lit. The pelican circuit can now progress through its routine. One second later the steady lamp goes out and the walk lamp comes on. Then, five seconds later the walk lamp goes out and the steady lamp comes back on. A further four seconds later, the steady lamp goes out and timer 2 is disabled. Timer 1 is re-enabled, and the traffic lights start their whole sequence again. I hope you can see that this is what you would expect from the traffic lights with a pelican crossing.

That completes my analysis of the circuit shown in Figure 9-14, the North and South traffic lights with a pelican crossing. I am not saying this is the best logic circuit for this project, but really it is just an example of how to apply what we have studied in this book onto a practical project that we can relate to. The final circuit was only arrived at after a few trial-and-error circuits. You get an idea of a route that might achieve what you are trying to do; then you test it out. The test circuit will either show you that your ideas are good or highlight where your problem is. Then you can hopefully focus on that issue and find a resolution.

When simulating the circuit, we must initiate a pelican request first by switching the Pel Request switch high first. The pelican lights will then go through a complete sequence, after which the normal traffic lights will continue as normal.

An Egg Timer Circuit

This is the next logic circuit we will design. The first step in designing any circuit, indeed any system, is to confirm the sequence of events your design should fulfill. The egg timer should do the following:

- It should be able to count down using a seven-segment display for the following: 10 minutes in units of minutes, tens of seconds, and units of seconds.
- The user should be able to set the length of time in a resolution of 1s.

This would be your first attempt at getting the specification right for the project. You should get your customer to sign an agreement with the proviso that there may be some alterations needed as you progress with your prototype design. You don't want to build your design, and then the customer says, no, that is not what they wanted and they won't pay.

Next, decide on the components you will use for your first prototype. The first component list is the following:

- A 555 timer to create the one-second clock
- Three seven-segment displays
- Three 4-bit down counters that can be loaded with some preset values – try using the 74LS168

Unless you have a great deal of experience in designing systems, it is very unlikely that you know exactly what you need until you start building the circuit. We will build the circuit as we go along and test out some concepts as we come up with them. After this process I am sure you will appreciate what a great help a good ECAD program is in designing your circuits.

The SN74168

We need a counter that can count down from a preset value that we can load the counter with. The SN74161 only counts up. The SN74168 counter can be configured to count up or down. The test circuit for this counter is shown in Figure 9-19.

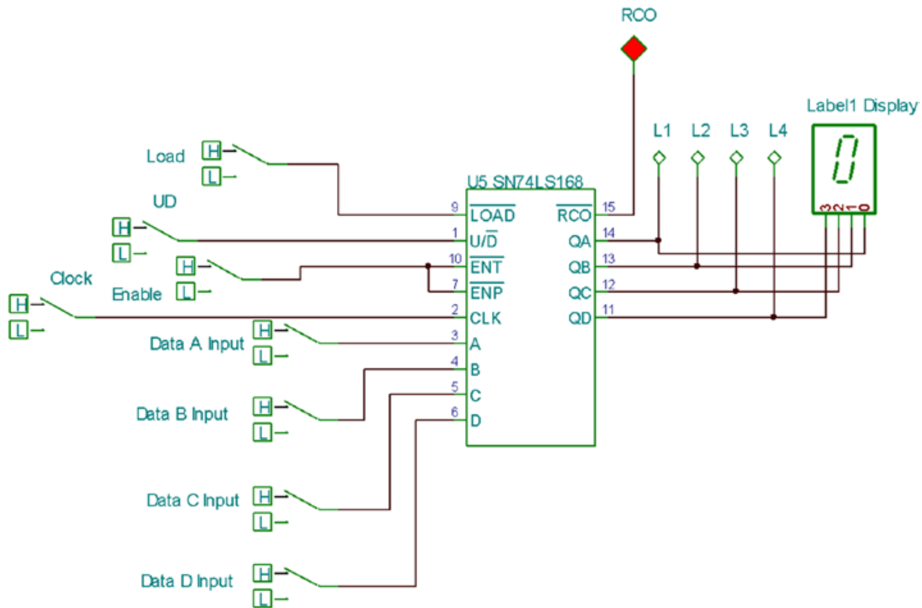


Figure 9-19. *Test Circuit for SN74168*

We will use this test circuit to see how we can control the counter.

There are four data inputs that can be passed on to the four outputs to load the counter with a preset value. We will use this function to set the value to count up from or down from. It is the "Load" input that controls this action. When this switch is sent low, the data at the four inputs is passed on to the output of the counter, when the clock input next goes from low to high.

There is a U/D input that configures the counter to count up, when the input is connected to a logic "1," or count down, when this input is held low.

There are two enable inputs: ENT and ENP. Both inputs are active low. For the basic operation, these two inputs need to be connected to a logic "0." This means we can use these two inputs to act as enable input.

The clock input is used to synchronize the operation of the counter. This input has to go from a logic “0” to a logic “1” for the output to change. This is termed positive or rising edge triggering.

The RCO is an output from the counter that will change from a logic “1” to a logic “0” when the counter has reached the maximum count when counting up or zero when counting down. It returns to a logic “1” when the counter next changes.

A possible test sequence to confirm that the counter operates as stated is as follows:

- Ensure all eight switches are switched to a logic high. The display should be showing “0” and the RCO probe should be on.
- Now take the clock input low, high, and back to low. Nothing should change as the enable input is high.
- Now take the enable input low.
- Now take the clock high and the display should show that it does increase to a “1.”
- Now take the clock low and the display does not change.
- Now take the clock high and the display should change to show a “2.” This confirms that the counter does not change until the enable is at a logic low. It also confirms that the clock must go from low to high to trigger the operation of the clock.
- Now change the U/D input to a logic “0,” that is, a logic low.

- Now send the clock high, then low, then high, and finally back to low. Observe the display each time, and you should see it go to a "1" and then end on "0." This confirms that the counter counts up when the U/D input is high and down when it is at a logic low.
- Now send the Load input to a logic low. Nothing should change.
- Now send the clock high. As soon as the clock changes from low to high, the data at the four inputs, which are all at a logic "1," is passed on to the output, and the display shows the hexadecimal "F," which is 15 in decimal or 1111 in binary.
- Now you send the clock low, high, and back to low. Nothing should change. This is because the Load input is still low. Therefore, instead of the clock decrementing the count and so change the display to "E," the counter simply reloads the output with the data that is still at the input.
- Now send the Load input back high to remove the load signal.
- Now send the clock input high and the counter will decrement by one.
- Now continue operating the clock to decrement the counter until the display shows "1."
- Now at the next decrement, the display should show "0" and the RCO output should go out, that is, go low.
- Now send the clock input high, and the display will show "F" and the RCO output should come back on.

The test sequence for the circuit in Figure 9-19 should enable you to test most, if not all, the functions of the counter.

The first thing we should appreciate is that the counter is normally a BCD counter in that its maximum value is “9.” Only when we load the counter with a higher value than 9 will the counter display A, B, C, D, E, or F.

Another important aspect of this counter we should appreciate is that the RCO output, which goes low and then high when the counter reaches a maximum or 0 output, could be used as the clock input to a counter controlling the next column on a display. Therefore, we can have the first counter counting the units of seconds. Then use the RCO output from the first counter to become the clock input to the second counter, counting the tens display. This concept can be tested with the circuit shown in Figure 9-20.

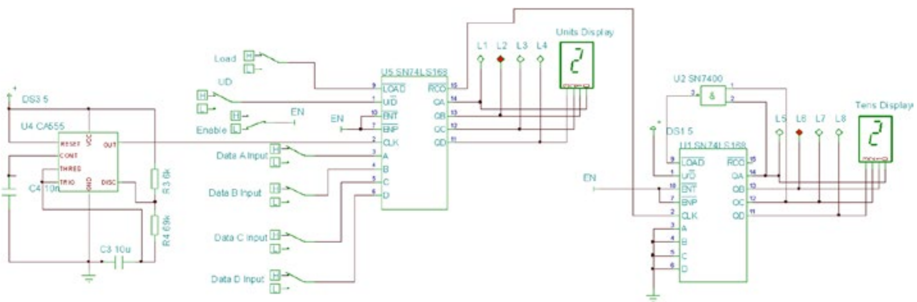


Figure 9-20. The 60-Second Test Circuit

The circuit shown in figure 9-20 shows the tens of seconds seven-segment display and the units of seconds seven-segment display. We are using the 555 timer to generate the one-second clock. We will look at the 555 timer in the next chapter. This is the clock source for the first 4-bit up counter, which is U5 in the circuit. We are using the RCO output from that counter as the clock input to the tens display counter, the U1 logic device in the circuit. When the units counter reaches a count of 9, the RCO output changes from a logic high to a logic low. When the units counter next

reacts to the clock, its display goes back to “0,” and the RCO output returns to its stable state of a logic high. There it stays until the units counter reaches the maximum value of 9 again.

As the RCO output, from the units counter, is fed to the clock input of the tens counter, this transition from high to low and then back to high, when the units counter reaches its maximum of 9, presents the tens counter with the required rising edge trigger to signal a count, and so the display increments from “0” to a “1.” This action repeats every time the units counter reaches a value of 9 and then rolls over back to a value of “0.”

The Load input of the tens counter, the U1 logic device, is fed from the output of a two-input NAND gate. The two inputs to this NAND gate, the U2 logic device, come from the Q_A and Q_C outputs of the tens counter. These two inputs will both go to a logic “1” when the tens counter reaches a value of “5.” This is when $Q_A = 1$, $Q_B = 0$, $Q_C = 1$, and $Q_D = 0$. When this happens, the output of the NAND gate goes low. With any other logic at the two inputs, that is, when the counter has a value of 0, 1, 2, 3, and 4 and 6, 7, 8, and 9, the output of the NAND gate is at a logic “1.” This means that when the counter reaches a value of 5, the Load input goes low. This gets the counter ready to load the output with the data at the four data inputs to the counter. These four inputs are permanently at a logic “0”; they are tied to ground. Therefore, 10 sec later, when the clock input goes low to high, the output of the counter is loaded with the value “0,” and the output of the NAND gate goes high, thus removing the load signal from the tens counter. This means the counter resets to “0” at the 60th count of the one-second counter.

I hope this analysis does help you understand how we can create a 60-second timer. We now need to add the units of minutes counter, and I hope you can see that we need to derive the clock input, to this counter, from the RCO output of the tens of seconds counter. However, before we do that, we should consider how we can load the counter with a preset value and change it to a countdown timer.

To change the counter to a down counter, we need only send the logic on the U/D input to a logic "0." This is best down by tying this input to the ground. This is done in the test circuit shown in Figure 9-21. We will use this test circuit to test out an approach to load the counter with a preset value and start decrementing it from there.

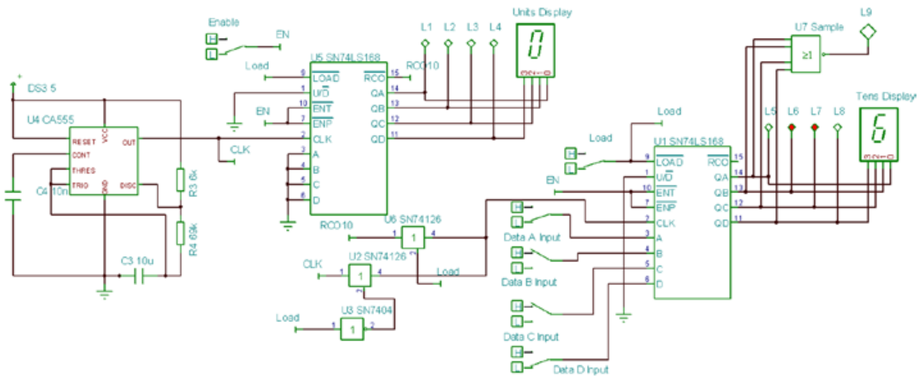


Figure 9-21. *Test Circuit for the 60s Loadable Down Counter*

With the test circuit shown in Figure 9-21, we are loading the counter with a starting value of 60. This is "6" in the tens of seconds counter, U1 logic device, and "0" in the units of seconds counter, the U5 logic device. The main issue with the loading of the counters is that we must send the "Load" input low and keep it low while the clock input goes from low to high. If we derived the clock input to the tens counter from the RCO of the units counter, we would have to keep the "Load" input at a logic "0" for 10 sec as it takes 10 sec after the RCO has gone low to send the RCO high. This means that when loading the counters, we need to feed the clock signal of the tens counter from the output of the one-second 555 timer. This means that the tens of seconds counter needs to change the clock input source from its normal source of the RCO from the units counter to the output of the 555-timer one-second signal. Then after we have loaded the counters with their preset value, we must return the clock input back to the RCO from the units of seconds counter.

To achieve this switching, we are using two tri-state buffers. These tri-state buffers are logic gates that act simply as an electronic switch. They have three terminals, one with a signal input. The second is the output, on which the input signal can be made to come out of the buffer. The third terminal is the gate terminal, which is used to enable the buffer to allow the input to pass through on to the output. If the gate signal is at a logic "1," the buffer allows the input to pass on to the output. However, if the gate signal is at a logic "0," the buffer is disabled and the input is prevented from passing on to the output.

Therefore, when we are loading the counter with the preset value of 60, we need to send the gate signal of the tri state buffer, controlling when we connect the RCO signal to the clock, to a logic "0" to disconnect the RCO signal from the CLK input of the tens of seconds counter. However, for this condition to be successful, we must enable the tri-state buffer connecting the CLK input of the tens of second counter to the 1Hz clock signal at the same time. As we are trying to load the counter at this point in time, then it is simply a matter of sending the load signal directly to the buffer controlling the RCO input, U6 in Figure 9-21, while sending this load signal via a NOT gate to the buffer of the 1Hz clock input; see U2 and U3 in Figure 9-21. In this way we can load the counter with any starting value we want to a definition of one second.

The next thing to consider is how can we select the number of minutes we want the timer to count down from. The simplest way to achieve this would be with the circuit shown in Figure 9-22.

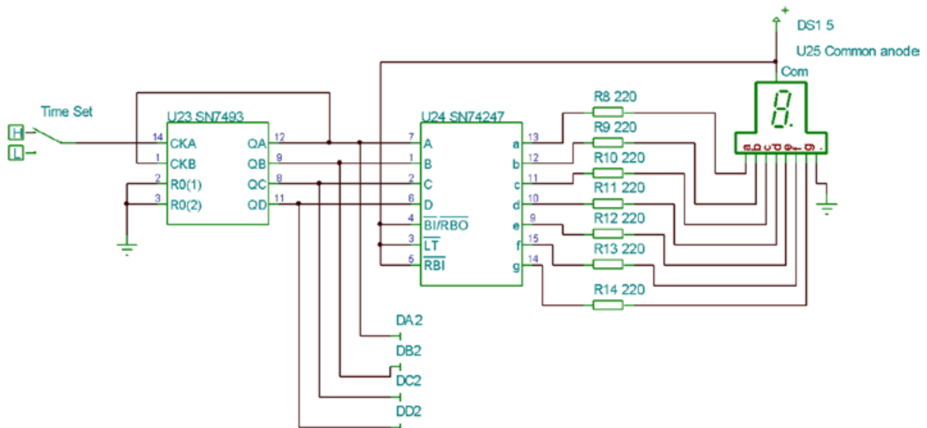


Figure 9-22. *The Minute Number Select Circuit*

This uses a simple counter, which will increment every time the time set switch goes from high to low. To enable the user to see what number the counter is set to, this number appears, in binary format, on the data bus: DA2 to DD2. This data bus connects this counter to the min units counter, U8 in the complete circuit.

We are using the seven-segment driver IC, U24, the SN74247 IC, as we are using a basic seven-segment display. The SN74247 IC is a driver for a common anode, as opposed to a common cathode, which we need as the display we are using is a common anode. The 220Ω resistors are required to limit the current flowing through the LEDs.

We can now look at the complete circuit for the egg timer. This is shown in Figure 9-23.

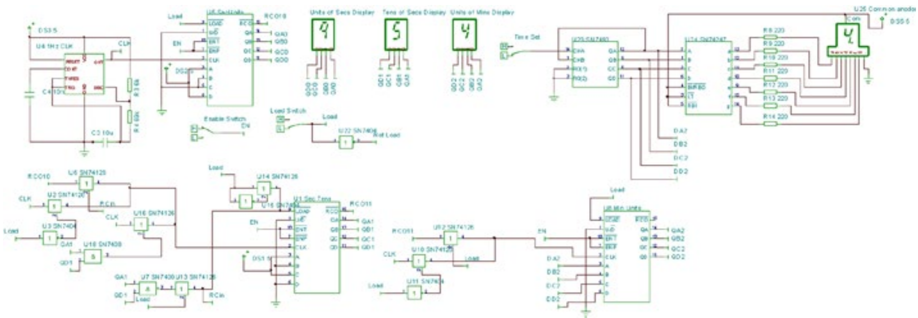


Figure 9-23. *The Complete Egg Timer Circuit*

The first thing we should appreciate is that we have set the time set display to say 4, as in Figure 9-23, but the display on the egg timer goes to 4:59 when we load the egg timer, as shown in Figure 9-23. This is because we have set the data for the sec units and sec tens counters as shown in Figure 9-24.

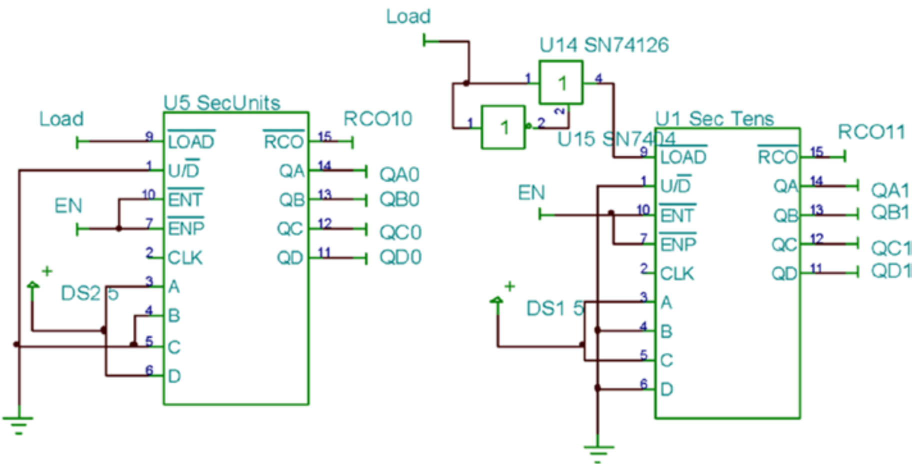


Figure 9-24. *The Sec Units and Sec Tens Counters*

If we look at the A, B, C, and D inputs to the sec units counter, we can see that the inputs D and A are permanently tied to +5V, while the C and B inputs are tied to 0V or ground. This means the data being permanently presented at the inputs is 1001, which is the binary number for 9 in decimal. Similarly, we can see that the permanent data presented to the sec tens counter is 0101, which is binary for 5. This means that whenever we load the counter, the sec units goes to 9 and the sec tens goes to 5. This means that when we want to set the egg timer time, we need to appreciate that the egg timer will add 59 seconds to the time we set the time to.

While setting the time, we need to keep the enable and load switches high. Then when we are happy with the time we want to use, we should switch the load to a logic low. The egg timer should be loaded with the time we have set plus the 59 seconds. We should then switch the load back to a logic high. This is because if we didn't, the timer would continue to reload the timer with the same time.

Now that we have the timer loaded and the load switch back at a logic high, it is time to start the egg timer running. This is done by switching the enable switch to a logic low. When we do this, the egg timer starts decrementing the time displayed. It will start with the sec units, and when the sec units gets to “0,” at the next toggle of the clock at timer 1, the sec units goes back to 9 and the sec tens display is decremented by 1. An issue comes about when both the sec units and the sec tens are at “0.” At the next decrement, the min units should, and does, decrement. Also, the sec units should go to 9 and the sec tens should go to 5. For example, if the display was 5:00, then it should go to 4:59. However, what happens is the display goes to 4:99. To overcome this problem, we have added a third path for the loading of the sec tens counter. This is shown in Figure 9-25.

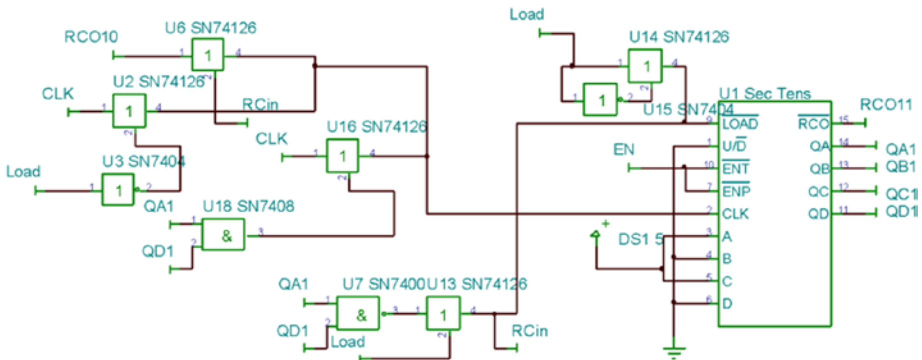


***Figure 9-25.** The Loading Circuit for the Sec Tens Counter*

It is the NAND gate U7 and the tri-state buffer U13 that provide the solution to this problem. The signal to load the counter with the data at its four inputs, A, B, C and D, is the logic on the load pin, pin 9 on the counter, must go to a logic "0." The inputs to the NAND gate U7 are QA1 and QD1. These would go to a logic "1" when the display of the sec tens counter went to a value of 9. This happens, as we have seen, when the sec tens counter goes to 9 and it tries to decrement from 0. When this happens these two inputs to the NAND gate go to a logic "1." This means the output of the NAND gate goes to a logic "0." The output of this NAND gate becomes the input to the tri-state buffer U13. If this buffer is enabled, then this logic 0 could be passed on to its output. This would then be the signal for the sec tens counter to get ready to load what it is at its input, the binary 0101, to its output.

The same data of QA1 and QD1 has been fed to a two-input AND gate U18. The output of this AND gate, which would be a logic "1," has been fed to the gate input of the tri-state buffer U16. This enables this buffer and so connects the main one-second clock signal to the CLK input of the sec tens counter. To make sure this controls the clocking of the counter, we need to disable the tri-state buffer, U6, which is used to connect the RC10 clock to the CLK of the counter. The gate input to that tri-state buffer, U6, is

connected to the output of the NAND gate U7; this is done via the jumper RCin. The logic on the RCin is at a logic "0" because QA1 and QD1 are both a logic "1."

This means that the sec tens counter is counting at a rate of 1Hz. This means that one second of the sec tens counter going to a value of 9, it decrements to a value of 8. This means the TCin bit goes to a logic "1"; this means that the load signal goes from a logic "0" to a logic "1." This completes the load operation, and the output of the sec tens counter goes to 5. At the same time, the main 1Hz clock signal is disconnected from the counter, and the RC10 clock signal is reconnected. This means that the timing of the sec tens counter is controlled by the RC10 output of the sec units counter. The egg timer can now carry on in the correct manner.

This solution may seem to indicate that we have lost a second in the count. However, this is not the case as the sec units counter has been decrementing correctly all the time.

This solution is a good example to show that there is no one design approach that will create a circuit that works perfectly the first time. There are issues that you would not be able to see and understand until you start putting your circuit together. You will then have to think around the problem to come up with possible solutions you can try. The more you think and try out your ideas, the better your experience becomes, and there is no real substitute for experience.

The Practical IC We Have Looked At

The circuits we have simulated so far have been using individual gates, flip flops, etc. This is very useful when we are trying to explain how the circuits work, but when it comes to creating the practical circuits, we find that we can't buy individual gates. The AND gate, and most of the basic logic gates, comes in a quad-dual in-line packaging. That means each 7408 IC comes with four two-input AND gates. The 7400 IC also has four two input

NAND gates. This means that we need to learn how to create circuits using the practical ICs that are at our disposal. This section of the text will study how the most common devices, especially those we have looked at, are arranged and how we need to connect to them to use them properly.

The 7400 Quad-Two-Input NAND Gate

This is an area that I think TINA and indeed most ECAD software packages could improve their representation. TINA does have a representation of how most of the practical ICs look, but they are not really what they actually look like, but they do work and can be used easily. To help explain what I mean, we will look at the 7400 NAND gate we have used in many of our circuits. TINA, and most ECAD software, does not even have a representation of the real ICs of the 7400 NAND gate; it only gives you access to the individual gate. This is true of the other basic logic gates such as AND, OR, NOR, NOT, and EXOR. However, you can make your own representations of the gates using the Macro Wizard available to you in TINA. The process of using this approach is explained in Chapter 11. Figure 9-26 shows the macro I have created to represent the 7400 four-two-input NAND gate.

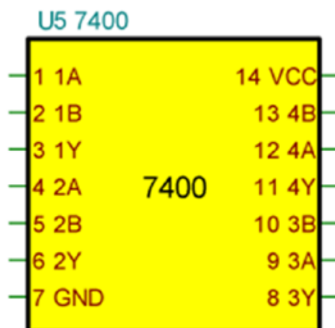


Figure 9-26. *The Macro I Created for the 7400 IC*

Although this is not an exact representation of the actual IC – a better representation is shown in Figure 9-28 – it is good enough for us to learn how we would actually wire up some NAND gates to make an actual circuit. To try and help you appreciate what I am trying to show you, we will look at creating the de-bounce circuit with two cross-coupled NAND gates that we looked at in Chapter 4. This is shown in Figure 9-27.

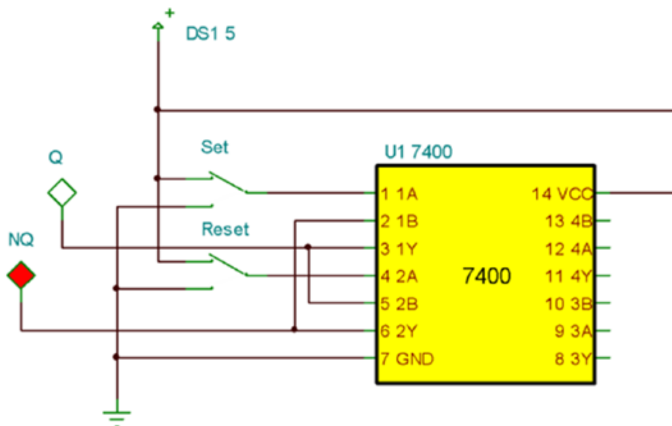


Figure 9-27. *The Practical De-bounce Circuit*

The idea is that this diagram will give you a better idea of how you wire up the circuit using a real 7400 IC. I hope it does go some way to achieving that.

The first thing we need to know, in creating the macro, is what the pins on the IC mean and how to connect the IC into the circuit. The main source for this information will be the data sheet for the device. We will have to learn how to use the information from the data sheet so that we can connect and use the device properly. We will start by looking at the data sheet from the 7400 NAND gate. One of the best diagrams we find from the data sheet is shown in Figure 9-28.

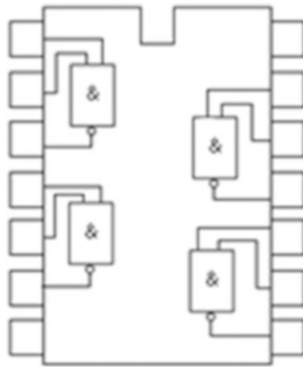


Figure 9-28. *The Pin-Out Diagram for the 7400 Quad-Two-Input NAND Gate*

I feel I should point out that not all data sheets give the same information. In my time working in electronics, I have amassed a large quantity of data sheets, and to use any device, you need to study its datasheet.

The diagram in Figure 9-28 really tells us all we need to know about the IC. It tells which pins are the two inputs for the four NAND gates and which pins are the outputs. It also tells us where to connect the VCC and GND to power the IC. I hope you can see that it was with the knowledge from this diagram that I was able to create my macro representation of the 7400 IC.

The process was to create the macro circuit for the 7400 in the schematics. The macro circuit is shown in Figure 9-29.

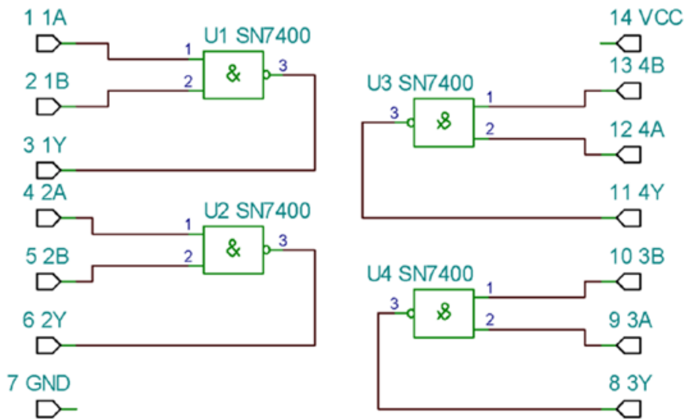
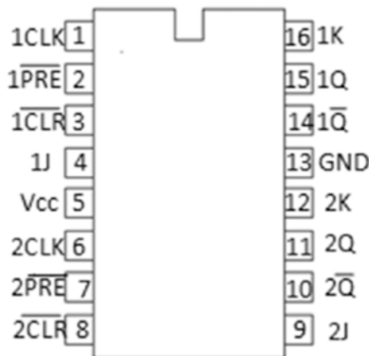


Figure 9-29. *The Macro Circuit for the 7400 IC*

The process is to place 14 macro pins, 7 for each side of the dual in-line IC, and connect up the actual circuit to the pins according to the data sheet. The macro pins are the second option on the specials component group. To complete the process once you have created the macro circuit, you simply follow the Macro Wizard available from the Tools drop-down menu. See Chapter 12 for a fuller explanation.

Counters

I started by looking at the 7400 because it is one of the simplest ICs we have used and it was the first component we used. Following on from the NAND and the NOR gate, we went on to look at the JK. The basic JK flip flop IC is the SN7476. This is another IC for which TINA only shows us a representation of the single JK. Therefore, it would be useful if we could make a macro ourselves. If we look at the data sheet, we can obtain the pin-out diagram as shown in Figure 9-30.



		Inputs			Out-puts	
PRE	CLR	CLK	J	K	Q	$\overline{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H↑	H↑
H	H	⎍	L	L	Q	$\overline{Q}$
H	H	⎍	H	L	H	L
H	H	⎍	L	H	L	H
H	H	⎍	H	L	Toggle	

Figure 9-30. *The Pin-Out Diagram and Function Table for the SN7476 JK Flip Flop*

As with the 7400, TINA does not have its own representation for the 7476. Therefore, I have created my own macro as shown in Figure 9-31.

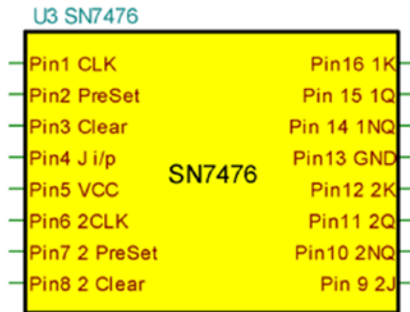


Figure 9-31. *The Macro I Have Created for the SN7476 JK Flip Flop*

I hope you can see how I used the data sheet to come up with the macro shown in Figure 9-31. With devices that are rather more complex than the 7400 NAND gate, the data sheet can give us access to a function table, as well as the pin-out diagram. One such function table is shown in Table 9-9. However, I don't think they give you the clearest idea of how the device functions. I will try to expand upon them with some of the devices we will look at in this chapter. The expansion for the function table for the JK flip flop is shown in Table 9-9.

Table 9-9. *The Expansion of the Function Table for the JK Flip Flop*

$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$
L	H	X	X	X	H	L
Explanation	The PRE SET is an active low input. When this goes low, with the CLR high, the Q will be set to a logic "1" and the $\overline{Q}$ to a logic "0." This action is irrespective of the logic on the clock.					
H	L	X	X	X	L	H
Explanation	The CLR, Clear, is an active low input. When this goes low, with the PRE high, the Q will be set to a logic "0" and the $\overline{Q}$ to a logic "1." This action is irrespective of the logic on the clock.					

(*continued*)

Table 9-9. (*continued*)

$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$
L	**L**	**X**	**X**	**X**	**H!**	**H!**
Explanation	If we try to activate both the PRE and the CLR at the same time, then the flip flop will respond with an indeterminate state. We should avoid doing this.					
H	**H**	**Transition**	**L**	**L**	**Q**	$\overline{Q}$
Explanation	When the clock goes from low to high and back to low, with the J and K both low, the outputs do not change from their previous states.					
H	**H**	**Transition**	**H**	**L**	**H**	**L**
Explanation	When the clock goes from low to high and back to low, with the J high and the K low, then the Q is high and the $\overline{Q}$ is low.					
H	**H**	**Transition**	**L**	**H**	**L**	**H**
Explanation	When the clock goes from low to high and back to low, with the J low and the K high, then the Q is low and the $\overline{Q}$ is high.					
H	**H**	**Transition**	**H**	**H**	**Toggle**	
Explanation	When the clock goes from low to high and back to low, with the J high and the K high, then the Q and the $\overline{Q}$ will alternate between high and low. This is the most useful aspect of the JK flip flop. The output changes every time the clock goes from high to low. This is termed "negative edge triggering."					

If you refer to Chapter 4, you will see that we have simulated most of these actions, and the simulations agree with them. If you create the macro for the SN7476, then you should be able to simulate the circuit shown in Figure 9-32 and so confirm the description in Table 9-9. I know simulation is not as good as experimenting with real-life components, but modern ECAD software is extremely good at replicating the real world and it is much quicker.

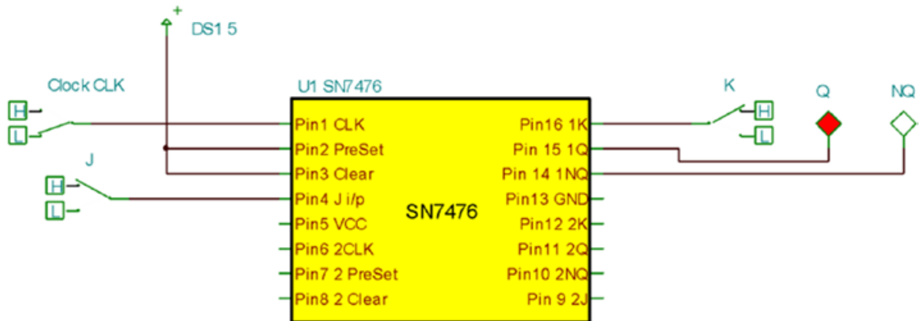


Figure 9-32. *The Test Circuit for the SN7476 JK Flip Flop*

The 7493 Binary Counter

We cascaded four JK flip flops together to create the ripple counter. We could, if we so wished, use two 7476 ICs to make the same ripple counter. Figure 9-33 shows that being implemented.

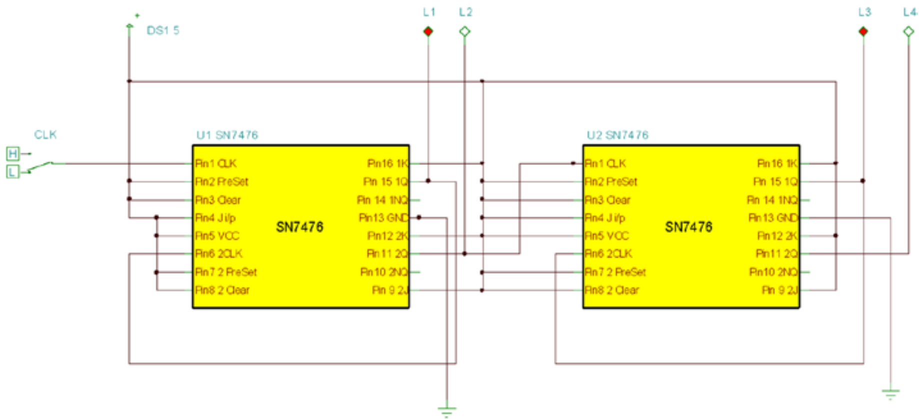


Figure 9-33. *The Ripple Counter with Two 7476 Macros*

However, there is the basic asynchronous binary counter, the 7493 IC, available to use. The data sheet provides us with the pin-out diagram shown in Figure 9-34.

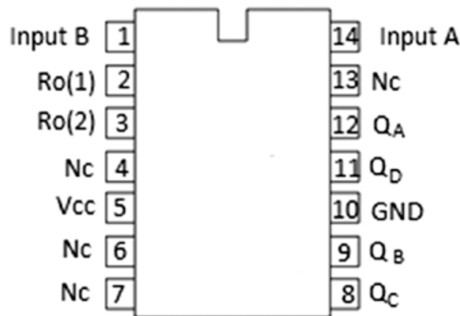


Figure 9-34. *The Pin-Out for the 7493 Binary Counter*

There is also a function table as shown in Table 9-10.

Table 9-10. *The Main Function Table for Counter 7493*

Count	**Outputs**			
	Q_D	**Q_C**	**Q_B**	**Q_A**
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

(*continued*)

Table 9-10. (*continued*)

Reset Inputs		Outputs			
R1(1)	**R0(2)**	**QD**	**QC**	**QB**	**QA**
H	H	L	L	L	L
L	H	COUNT			
H	L	COUNT			
L	L	COUNT			

TINA does provide us with a representation of the 7493 counter. The test circuit, shown in Figure 9-35, uses that, and with this circuit we can test the various configurations for the counter listed here.

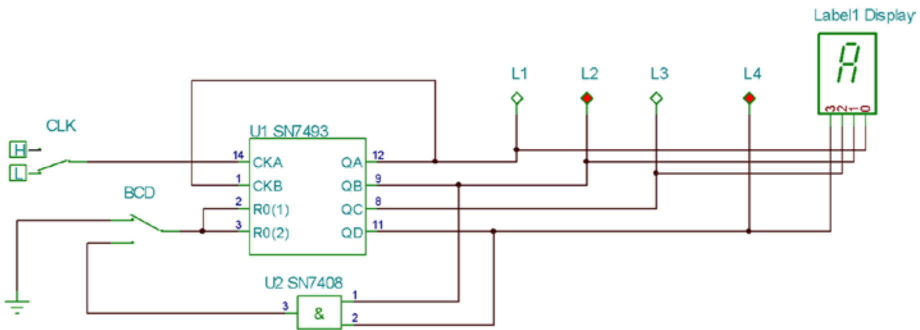


Figure 9-35. *The Test Circuit for the SN7493 Binary Counter*

With the BCD switch connected to ground, the counter can count from 0 to F, that is, in hexadecimal. If we switch it over to connect the output of the two-input AND gate to the two reset inputs, the counter will reset back to 0, depending upon when both inputs go high. With the configuration as shown in Figure 9-35, the counter is a BCD counter in that it counts from 0 to 9. Also, with this arrangement you could change the counter to a modulo 6, 7, etc.

The SN74194 Multifunction Shift Register

This is the last device we will look at; there is not enough room to look at them all. This is a versatile shift register in that it can be used as SISO or PISO, but in both modes, it can shift the data through the register from left to right or from right to left. The test circuit to verify the action of the register is shown in Figure 9-36.

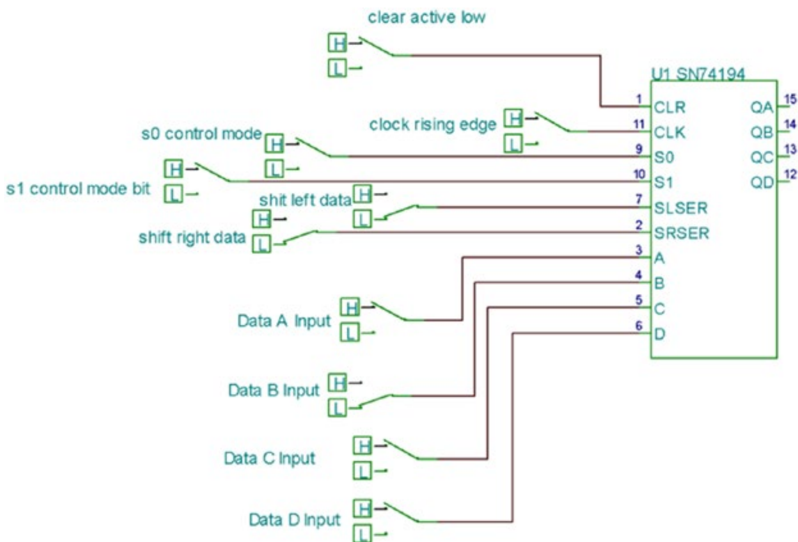


Figure 9-36. *The Test Circuit for the SN74194*

The pin arrangement for the register is shown in Figure 9-37.

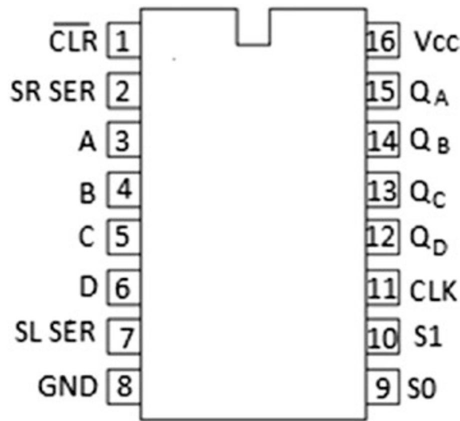


Figure 9-37. *The Pin Layout for the SN74194 Shift Register*

Table 9-11 lists the pin usage for the SN74194 shift register.

Table 9-11. *Pin Usage for SN74194 Shift Register*

Pin	Name	Basic Usage
1	CLR	When this input is sent low, all data in the register is loaded with a logic "0" regardless of any other input.
2	SR SER	This pin has the serial data that is waiting to shift right through the register.
3	A	This is the MSB of the parallel data waiting to be loaded into the register.
4 5 6	B C D	These are the remaining 3 bits of data waiting to be loaded into the register.
7	SL SER	This pin has the serial data that is waiting to shift left through the register.
8	GND	This pin is connected to the ground or 0V.

(*continued*)

Table 9-11. (*continued*)

Pin	Name	Basic Usage
9	S0	These two pins control which mode of operation the register works. See Table 9-12.
10	S1	
11	CLK	The synchronizing clock signal is applied to this pin. See Table 9-13 for maximum clock frequency and typical power dissipation.
12	Q_D	These are the four output pins of the register.
13	Q_C	
14	Q_B	
15	Q_A	
16	VCC	This is where the positive supply is connected to the IC.

The mode of operation is set using the S0 and S1 inputs. The different options are stated in Table 9-12.

Table 9-12. *How the S0 and S1 Inputs Control the Mode of Operation for the Register*

S1	S0	Mode of Operation
0	0	Not used.
0	1	SISO operation. Data shifts right 1 bit at a time when clock goes high. The data on the SR SER bit will be loaded into Q0 when clock goes from low to high.
1	0	SISO operation. Data shifts left 1 bit at a time when clock goes high. The data on the SL SER bit will be loaded into Q0 when clock goes from low to high.
1	1	PISO operation. Mode has to change to SISO with input data at logic 0 for the parallel data to shift right or left.

Table 9-13. *Maximum Clock Frequencies*

Type	Maximum Clock Frequency (MHz)	Typical Power Dissipation (mW)
194	36	195
LS194A	36	75
S194	105	425

All this information is available from the data sheet; however, data sheets are not the easiest to interpret. We can use the test circuit shown in Figure 9-36 to get a clearer understanding of the IC. Carry out the simulation as follows:

- Set the $\overline{CLR}$ input to a logic "1," that is, logic high.
- Set the SR SER and the SL SER inputs to a logic "0."
- Set the S0 and the S1 to a logic "1." This should put the register into parallel mode.
- Ensure the Data A and Data B inputs are a logic "1," with Data C and Data D at a logic "0."
- Now take the CLK input from a logic "0" to a logic "1." This may need you to take it to a logic "0" first.
- Determine that when the clock goes high, the output Qs take on the logic of the Data inputs.
- Now we must take the register back into SISO mode before the next clock transition from low to high.
- Therefore, set S0 to a logic "1" and S1 to a logic "0."
- Now take the clock from low to high for one transition.

- Confirm that the data at the output Qs shifts 1 bit to the right.
- Now take the clock from low to high a further three times.
- Now confirm that the two logic “1”s from the parallel input have completely fallen off at the end of the register. Also, all Qs in the output are at a logic “0.”
- Now take the SR SER input to a logic “1.”
- Now take the clock from a low to a high again and confirm that QA goes to a logic “1.”
- Now take the SR SER input to a logic “0.”
- Now take the clock from low to high a further three times and confirm that the logic “1” shifts right until it falls off at the end.
- Now take the SL SER input to a logic “1.”
- Now take the clock from low to high and confirm QA stays at a logic “0.” This should show that when shifting right the SL SER input has no effect.
- Now set the S1 high and the SO low.

Summary

In this chapter we have tried to apply the processes we have learned into designing some useful logic circuits. I am not trying to say that the designs we have used are the best and most efficient designs, but I have used them to try and give you an idea of how you can approach your designs.

Also, in this chapter we have tried to appreciate the difference between using single logic devices and using real logic ICs. We have learned the value of reading the data sheets for the devices we might use.

Finally, we looked at the complexity of wiring up a real circuit using real ICs. Note I say real but we still used the ECAD software, but I hope it does show you what I mean.

In the next chapter, we will study the 555 timer as up till now we have either used a single switch or a clock source, but in Figure 9-23 we used a 555 timer to provide the 1Hz clock.

CHAPTER 10

Introduction to the 555 Timer

In this chapter we will study the 555 timer. We will learn what it is and how it works. We will then learn how to configure it to operate in a variety of formats.

In Chapter 9 we looked at creating a timer that would time up to ten seconds. In that circuit we used a 1Hz signal source that is available from TINA. However, we need to create our own 1Hz signal source, and we will look at using the 555 timer to do that.

The 555 Timer

This is an integrated circuit, or IC, which can be used as a monostable, that is, a device that has one stable state at its output, either low or high; in this case the stable state is a logic low. It can then be made to switch out of this stable state for a set time, after which it will return back to the one stable state. It can also be used as an astable in that it can be made to produce a square wave at its output. To appreciate how it works, a study of what the eight pins are used for would be helpful. The pin-out is shown in Figure 10-1.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_10

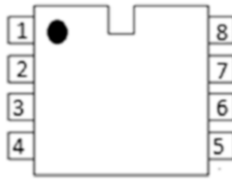


Figure 10-1. *The 555-Timer IC*

The Pins of the 555 Timer

- Pin 1 (Ground)
    - This is used to connect the timer to ground.
- Pin 2 (Trigger Input)
    - This pin is used to make the output come out of its stable state. The output, at pin 3 of the chip, is normally held at ground or logic low. When the input at the trigger pin, pin 2, goes low, that is, reduces to approximately one-third of VCC, the output, at pin 3, goes high, that is, to around VCC – 1.7V. The period of time the output stays high is controlled by the external CR network. However, if the trigger input is held low for a time longer than that period determined by the external CR network, the output will stay high until the voltage at this trigger input is driven high again.
- Pin 3 (Output)
    - This is the output of the chip, and its stable state is a logic low. However, it can be made to switch to a logic "1" for a set period of time, set by the CR network connected to the IC, after which it

will return to the stable state. Alternatively, the timer can be made to produce a square wave at this output pin. The frequency and mark-to-space ratio of the square wave are determined by the CR components around the chip. The 555 timer is capable of delivering 200mA to a load, that is, sourcing 200mA. It can also be used to sink 200mA, that is, providing a path for up to 200mA to flow through the chip to ground.

- Pin 4 (Reset)
    - When the voltage at this pin is driven low, that is, to a voltage between 0V and 0.4V, the output of the timer is made to reset to its stable state. This reset will happen regardless of the states at any of the other pins. If this pin is not to be used as a reset pin, then it can be held high by connecting it to VCC.
- Pin 5 (Control Voltage)
    - This control pin can be used to vary the timing of the device independent of the external CR network. In the monostable mode, this pin can be used to control the width of the pulse independent of the CR network. If we are using the chip in the astable mode, this pin can be used to enable the control of the frequency of the square wave at the output. We will not be using this pin; we will connect it to ground via a capacitor to prevent noise affecting the chip. This is the most common way to use this pin.

- Pin 6 (Threshold Input)
    - When the voltage at this pin rises to around two-thirds of VCC, the output at pin 3 will reset back to its stable state of a logic low. The rate at which the voltage on this pin rises is controlled by the CR network connected to the IC; see Figures 10-2 and 10-5.
- Pin 7 (Discharge Pin)
    - There is an internal transistor connected internally to this pin. When the transistor is turned on, it is used to provide a discharge path to ground for the timing capacitor connected to this pin; see Figures 10-2 and 10-5.
- Pin 8 (VCC Pin)
    - This pin can be connected to a voltage supply from 4.5V to 16V. This makes the timer very adaptable to a range of supply voltages.

The Timer Used as a Monostable

The stable state of the 555 timer is logic low. When triggered the output goes high and stays high until the voltage at the discharge pin, which is connected to the threshold pin, reaches two-thirds of VCC; in this case when VCC = 5V, then the threshold voltage = 3.33V. Figure 10-2 shows a circuit we can use to test the basic operation of the timer.

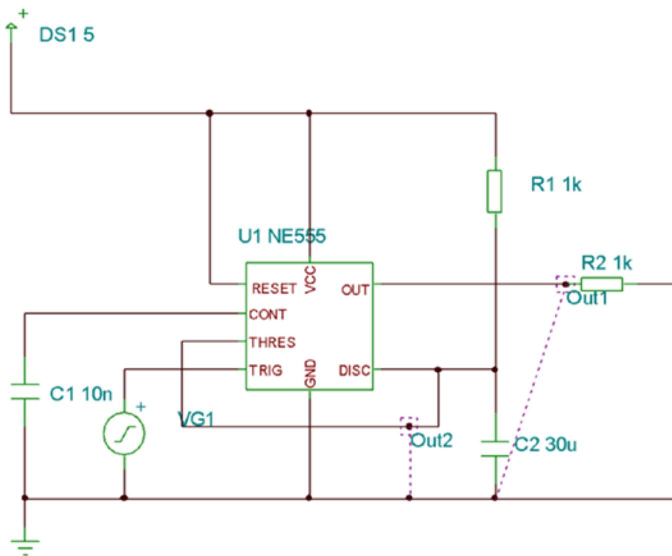


Figure 10-2. *The Basic Monostable*

The basic idea of the monostable circuit is that it has one state that it wants to stay stable in, which is with the output on PIN3 at around 0V. It can be forced out of that state temporarily, by driving the voltage at the "trigger input" down to around 0V. I have chosen 1.5V as the data sheet for the 555 states that this voltage needs to fall to around one-third of VCC; 1.5V is just below this. In this circuit we are using the VG1 voltage source to drive the trigger input down to 0V. To achieve this, the VG1 source starts off at 1.5V. It stays at 1.5V for 5ms and then changes to VCC or 5V. The voltage stays at 5V for the next 50ms. The voltage then repeats the cycle forever. Applying this voltage to the trigger input would force the monostable out of its stable state, which is at around 0V, and send it to around VCC, that is, 5V.

The capacitor C1, connected to the disc and threshold of the monostable, is allowed to charge up toward VCC via R1. When the capacitor reaches around 3.3V, that is, two-thirds of VCC, the output is allowed to return to its stable state of around 0V. The output will remain in this stable state until the trigger input is again forced to go to 0V. In this case, this is at 55ms later.

Figure 10-3 shows the voltage waveforms we are interested in.

- OUT1 is the output, on pin 3, of the timer. We can see that, because the trigger input, VG1, starts off at 1.5V, the output goes to around 3.6V, that is, approximately VCC – 1.7V, as the timer has been forced out of its stable state.
- OUT2 is the voltage at the threshold and disc pins of the timer, pins 6 and 7. The internal transistor at the disc pin is currently turned off, and so the capacitor C1 is allowed to charge up via the resistor R1. This means the voltage at the threshold pin starts to rise as the capacitor charges up. When this voltage gets to 3.32V, as shown in trace Out2 in Figure 10-3, the output of the timer switches back to its stable state, as shown by the Out1 trace in Figure 10-3. At the same time, the internal transistor, in the timer, turns on to provide a path to earth so that the capacitor C1 can discharge. When the transistor turns on, there is very little resistance between the capacitor and ground, and so the capacitor discharges almost instantly.
- VG1 is the voltage applied to the trigger input. This starts off at 1.5V for the first 5ms. It then changes to +5V for the next 50ms. After this the cycle repeats.

Using this triggering action, we can see that the timer does want to stay in the one stable state of 0V at its output. We see that we can force the output to change from this stable state for a set period of time. After this it does return to its stable state, waiting for the trigger to go to around 0V once again. We can see that the timing of this set period is set by the time it takes for the capacitor, C1, to charge up to 3.32V, that is, two-thirds of the VCC. The charging of the capacitor is controlled by the time constant set by the CR combination of R1 and C1. With Figure 10-2 this is set to 1k × 30μF = 30ms. From our work with analogue electronics, we know the voltage across the capacitor, C1, and so the input to the threshold pin follows the expression

$$VC = V\left(1 - e^{-\frac{t}{CR}}\right)$$

Putting the values in, we have

$$VC = 5\left(1 - e^{-\frac{t}{30ms}}\right)$$

We can transpose this expression to determine the time this voltage will take to reach the 3.32V required to send the output back to its stable state of 0V.

The expression for this time is

$$t = -CRln\left(1 - \frac{VC}{V}\right)$$

Putting the values in, we get

$$t = -30mln\left(1 - \frac{3.32}{5}\right) = 32.7ms$$

This means that 32.7ms after the output voltage rose to 5V, it would return to the stable state of around 0V.

If we look at Figure 10-3, we should see that all the preceding predictions are confirmed by all the traces displayed. Also, using the readings of the two cursors, we can confirm that the output voltage stays out of its stable state, at around 5V, for 32.91ms; see cursor A values. This is very close to the time period we have calculated.

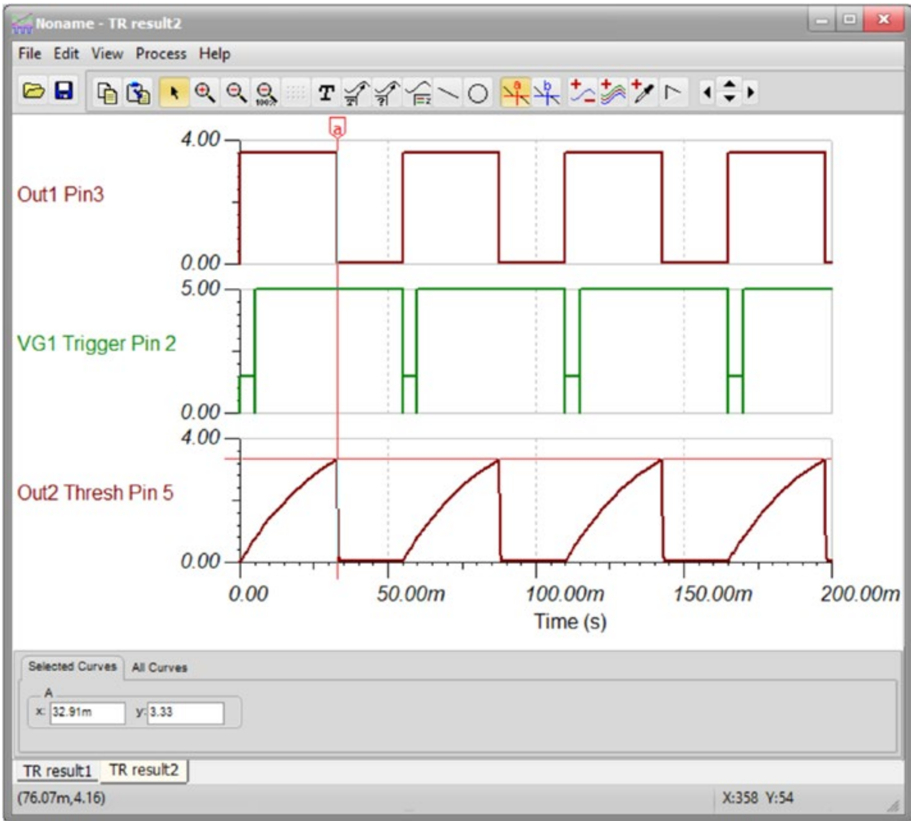


***Figure 10-3.** The Three Waveforms of the 555 Timer Set Up as a Monostable. The CR Setting Equals 30ms*

To further confirm that the circuit works in this way, we can change the capacitor C1 to 10μF. This would change the CR constant to 10ms. Therefore, using the expression to determine the time the output voltage remains high, we get

$$t = -10mln\left(1 - \frac{3.32}{5}\right) = 10.98ms$$

Figure 10-4 shows that this calculation is correct, that is, the A cursor shows the capacitor starts to discharge at a time of 11.16ms.

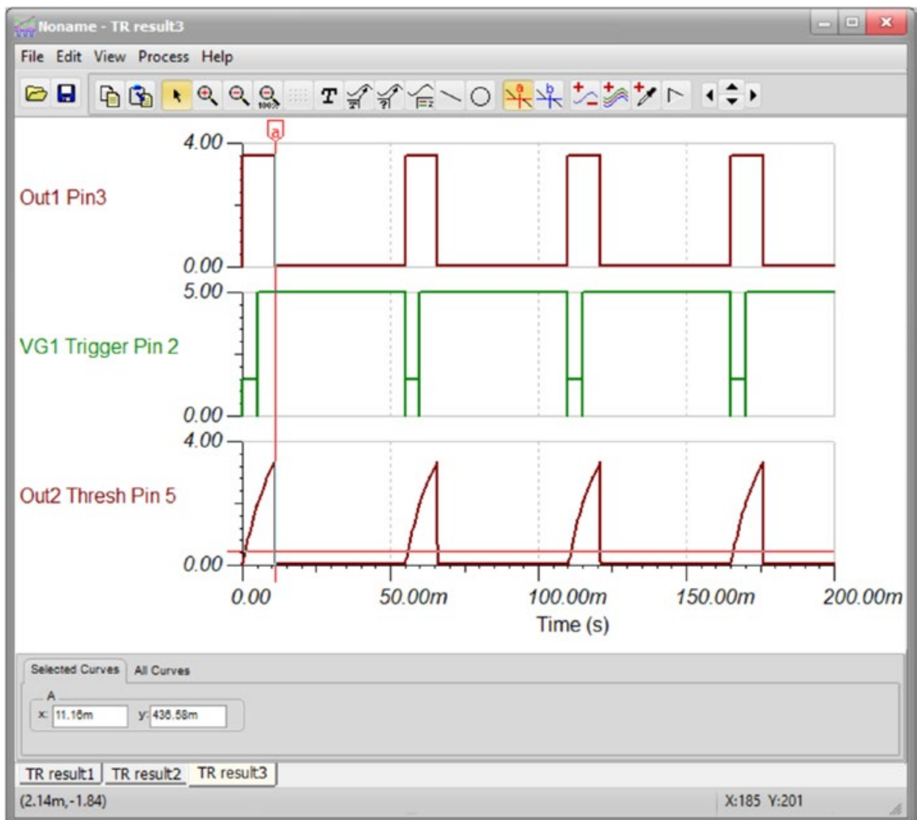


***Figure 10-4.** The Display of the Waveforms with C = 10μF and R = 1kΩ*

The Basic Astable

To change the monostable into an astable circuit, which will use the 555 timer to create a square wave at its output, we simply need to keep triggering the 555 timer, that is, keep sending pin 2, the trigger input, to below one-third of VCC. The simplest way of doing this is to connect the varying voltage across the capacitor C1 to the trigger as well as to the threshold pin. The test circuit for this arrangement is shown in Figure 10-5.

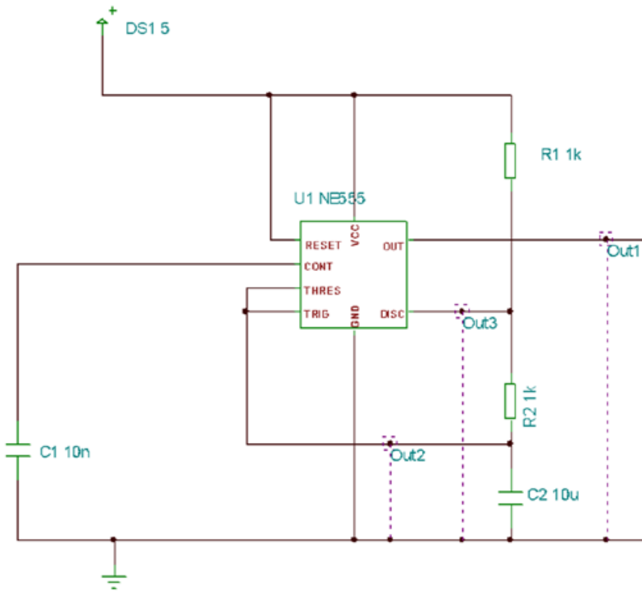


Figure 10-5. *The Basic Astable Test Circuit*

There are now two resistors that control the charging and discharging of the capacitor. As the capacitor discharges via R2 and the internal transistor, which provides the capacitor with a path to ground, then R2 controls the discharge time. Once discharged the internal transistor turns off and allows the capacitor to charge up toward VCC via R1 and R2. Therefore, R1 and R2 control the charge-up time for the capacitor.

The charge-up time of the capacitor controls how long the output is high, that is, the mark time. The discharge time, of the capacitor, controls how long the output returns to its stable state, the logic low or space time. Therefore, R1 and R2 control the mark time, and R2 controls the space time.

Note the capacitor C2 is just to prevent any noise from upsetting the control voltage input pin.

The frequency of the output is set by the following expression taken from the data sheet for the 555 timer:

$$f_o = \frac{1.44}{(R1 + 2R2)C}$$

Therefore, with the circuit values shown in Figure 10-5, the frequency of oscillation is

$$f_O = \frac{1.44}{(1E^3 + 2E^3)10E^{-6}} = 48Hz$$

The data sheet for the 555 timer states that the mark time can be calculated using

$$Marktime = 0.69(R1 + R2)C1$$

Putting the values for the circuit into the expression gives us

$$Marktime = 0.69(1^3 + 1^2)10^{-6} = 13.8ms$$

This is confirmed by the trace shown in Figure 10-6.

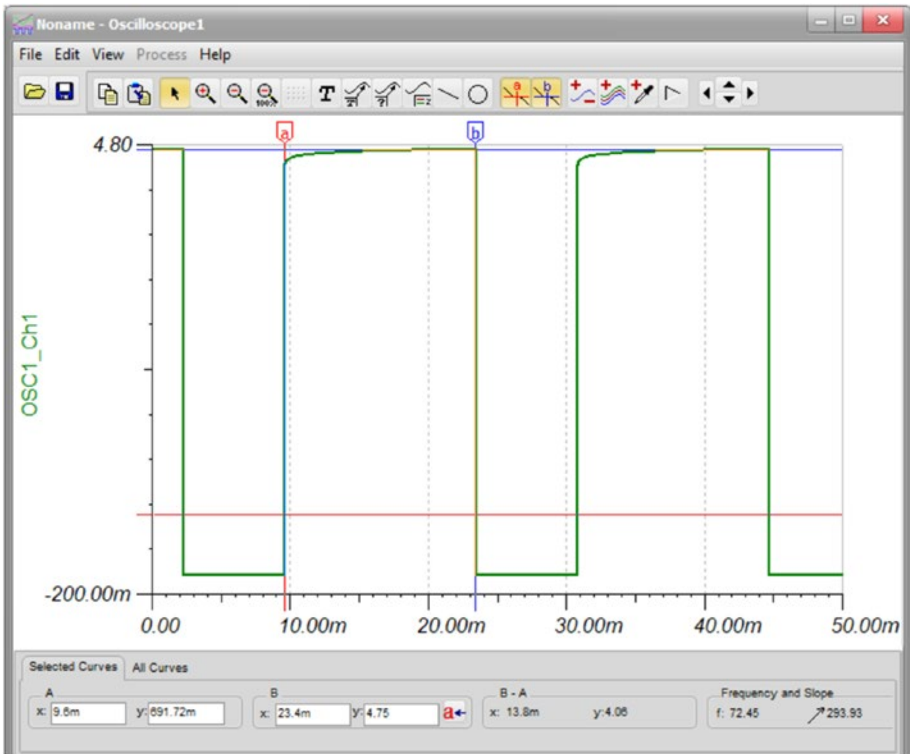


Figure 10-6. *Trace Showing the Mark Time of 13.8ms*

With respect to the space time, using the data sheet, the space time is set by

$$Spacetime = 0.69R2C$$

Putting the values in, we get

$$Spacetime = 0.69 \times 1^{3} \times 10^{-6} = 6.9ms$$

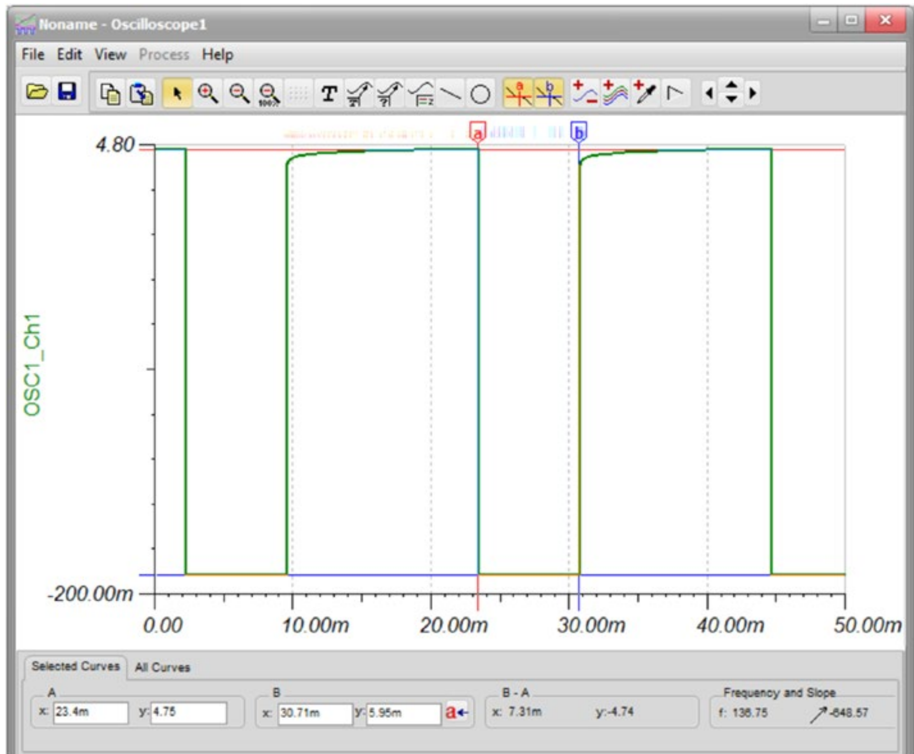


Figure 10-7. *Trace Showing the Space Time*

Figure 10-7 confirms the setting of the space time.

Knowing that the periodic time can be determined by adding the mark and space time together, this gives a periodic time of

$$Periodic\ Time = M + S = 13.8ms + 6.9ms = 20.7ms$$

This agrees closely to the measurement in Figure 10-8.

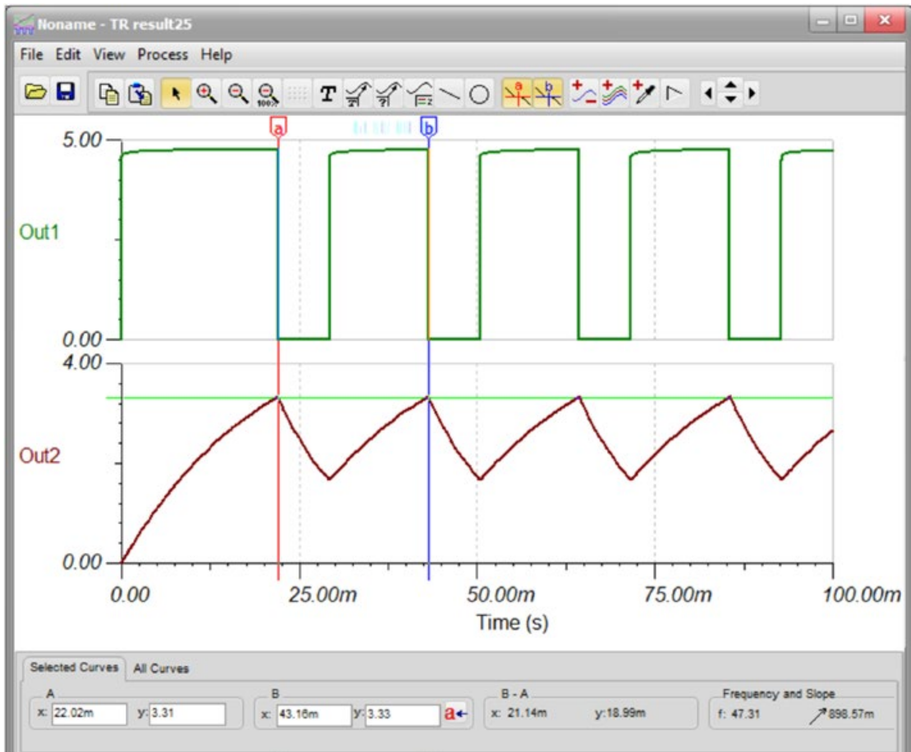


***Figure 10-8.** Trace Showing the Periodic Time for the Square Wave Output*

Figure 10-8 shows the periodic time for the square wave to be 21.14ms. Therefore, using the expression to determine the frequency of a waveform knowing the periodic time "T," we can calculate the frequency to be

$$F = \frac{1}{T} = \frac{1}{21.14^{-3}} = 47.3Hz$$

This agrees closely to the expression for the frequency taken from the data sheet.

In Figure 10-9 we are looking more closely at how the CR network of C1, R1, and R2 controls the 555 timer. If we look at the Out2 trace, which displays the voltage at the threshold and trigger pins, we should be able to see how the timer is triggered to come out of its stable low-voltage state. We know that the voltage has to fall to below one-third of VCC, which for a 5V VCC would be around 1.66V. If we look at cursor "B" on this trace, we can see that at a time of 29.06ms, the voltage falls to 1.67V. This is the voltage the trigger pin has to fall to, to force the timer out of its stable low-voltage state. At the same time, we see the output voltage, on Out1, goes to a logic high of around 3.6V.

If we look at cursor "A," we see that at a time of 21.89ms, the voltage reaches a value of 3.33V. This is the voltage that the threshold pin has to reach before the timer resets. At this point the output switches back to its stable state of a logic low. Also, the internal transistor is on and so allows the capacitor C1 to discharge. The difference with this discharge, with the timer set up as an astable instead of a monostable, is that there is the extra resistance of the R2 resistor within the discharge path. This means that the time constant of the CR discharge is set bey the CR product, that is, around 10ms.

Finally, we can see that once the voltage has fallen to 1.67, the timer switches again as it has been triggered, the voltage goes to a logic high, and the internal transistor turns off. The capacitor is now allowed to charge back towards 5V via R2 and R1, but it is now starting from an initial voltage of 1.67V instead of 0V.

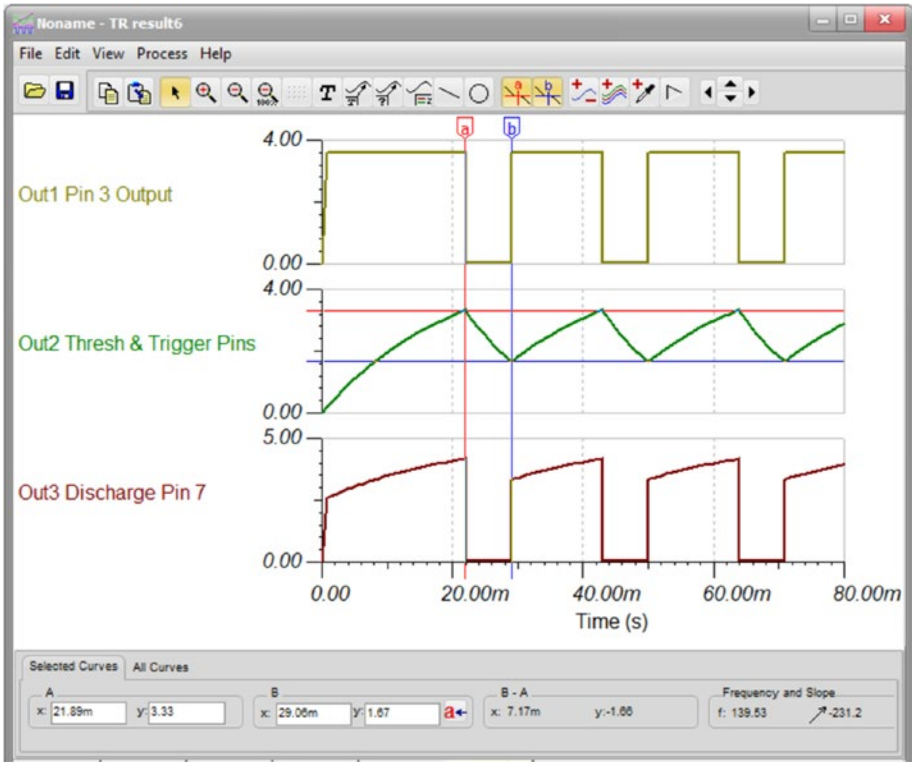


Figure 10-9. *The Threshold and Trigger Voltage Traces*

I hope this analysis does give you some idea of how the 555 timer works.

Creating a 50/50 Duty Cycle Square Wave

It is not essential that the duty cycle of a square wave should be 50/50, but it would be useful if we could do that. To create a 50/50 duty cycle square wave, we need to make the mark time the same as the space time. If we keep the value of the capacitor constant, then it is the resistor values that control these periods. We can rearrange the expressions for mark time and space time to derive expressions for the resistors R1 and R2 as follows:

$$Marktime = 0.69\left(R1 + R2\right)C$$
$$\therefore R1 = \frac{Marktime - 0.69R2C}{0.69C}$$

$$Spacetime = 0.69R2C$$
$$\therefore R2 = \frac{Spacetime}{0.69C}$$

Now, substituting the expression for R2 into the expression for R1, we have

$$R1 = \frac{Marktime - \dfrac{0.69 \times Spacetime \times C}{0.69C}}{0.69C}$$

$$\therefore R1 = \frac{Marktime - Spacetime}{0.69C}$$

This means that it is impossible to get a true 50/50 duty as to make the mark time the same as the space time would result in R1 being 0Ω.

A compromise, which is given in the data sheet, is to get close to 50/50, we must let R1 be between R2/8 and R2/5. To lower the frequency, let R1 be closer to R2/5, and at higher frequencies, let R1 be closer to R2/8. Note this is without changing the value of the capacitor.

Example 1:

When f = 500Hz let space = 1ms, that is, half the periodic time. Therefore, when C = 1μF we have

$$R2 = \frac{Spacetime}{0.69C} = \frac{1E^{-3}}{0.69 \times 1E^{-6}} = 1449.27$$

therefore using

$$R1 = \frac{R2}{5} = \frac{1449.27}{5} = 289.86$$

Figure 10-10 shows the trace of the output from the 555 timer using the component values just calculated. The mark time is shown as 1.21ms, and if you used the simulation to measure the space time, you should get a value of around 1.24ms. This does confirm that we can achieve close to a 50/50 duty cycle, but the output of the 555 timer is not very accurate. However, if your application does not require a very accurate frequency, then this simple timer can be quite useful and easy to set up.

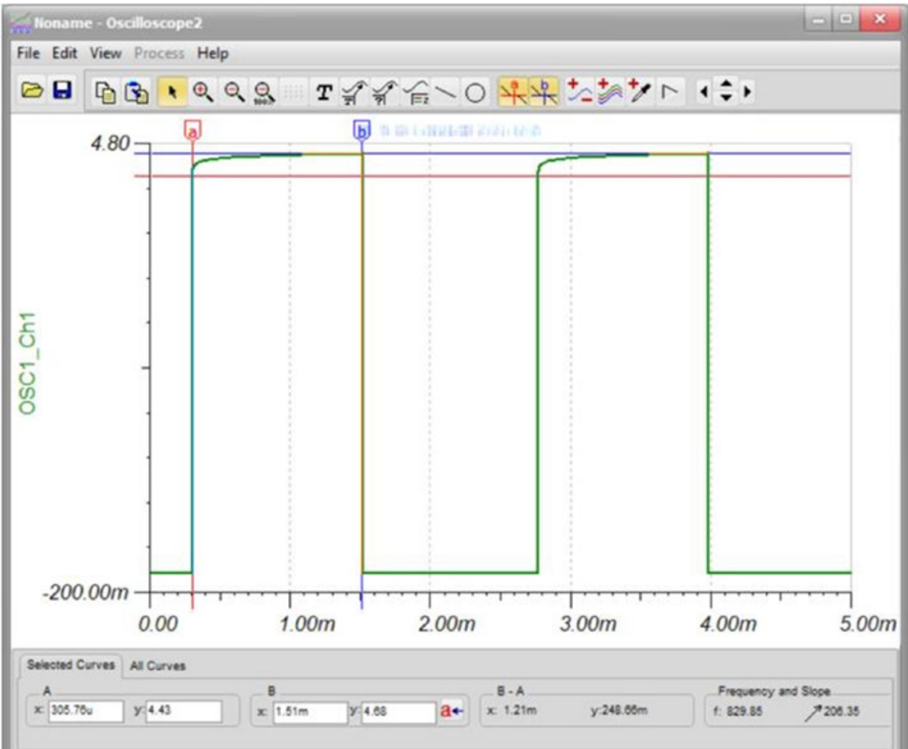


Figure 10-10. *The Trace Showing the Mark Time of the 50/50 500Hz Square Wave*

Example 2:

When f = 50kHz let space = 10μs. Therefore, when C = 1nF we have

$$R2 = \frac{Spacetime}{0.69C} = \frac{10E^{-6}}{0.69 \times 1E^{-9}} = 14.49k$$

therefore using

$$R1 = \frac{R2}{8} = \frac{14.49}{8} = 1.81k$$

A printout of the oscilloscope is shown in Figure 10-11.

I have changed the capacitor value to 1nF as if we kept it at 1μF, the value of the resistors would be too low, that is, R2 = 14.49Ω and R1 = 1.81Ω. This would suggest that at high frequencies use low values for capacitor C1 and for low frequencies use high values of capacitance.

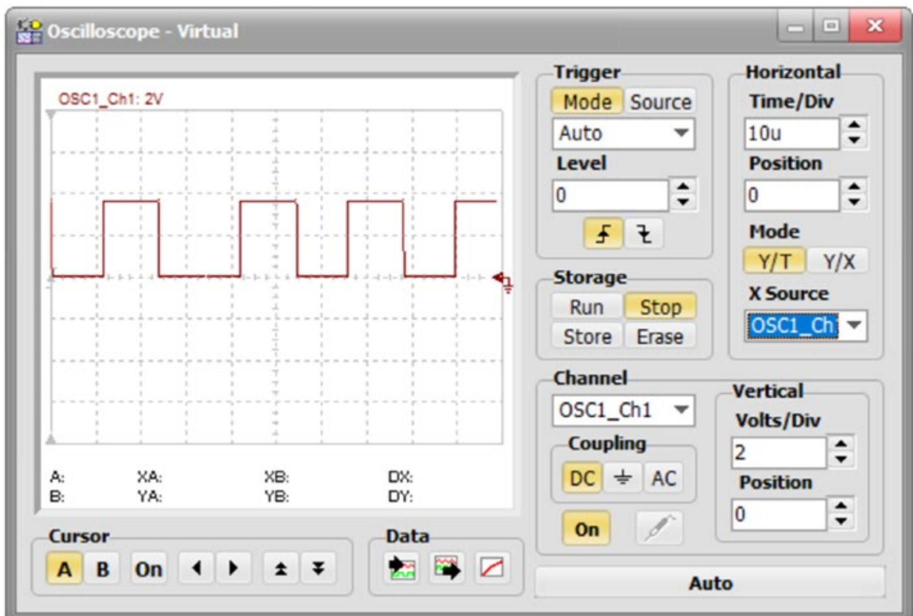


Figure 10-11. *The 50kHz Oscilloscope Trace*

Creating a 1Hz Square Wave

If we want to try and design a digital timer using logic gates, then it would be useful to try and create a 1Hz square wave using the 555 timer. The following is just one possible way of achieving this. As R2 is involved in both CR time constants for the timer, then it might be useful to set this to a reasonable value, such as 50k; then, using the expression for calculating the frequency, determine the value of the resistor R1. Note we should also set the capacitor value, which is used in both time constants, to a reasonable value of 10uF, as the frequency is low at 1Hz. Therefore, using the expression for the frequency

$$f_O = \frac{1.44}{(R_1 + 2R_2)C}$$

we can transpose this for R1 as follows:

$$Cf_0 = \frac{1.44}{R_1 + 2R_2}$$

Therefore, we have

$$R_1 + 2R_2 = \frac{1.44}{Cf_0}$$

Therefore

$$R_1 = \frac{1.44}{Cf_o} - 2R_2$$

Putting the values in gives

$$R_1 = \frac{1.44}{10^{-6} \times 1} - 2 \times 50^3 = 44k\Omega$$

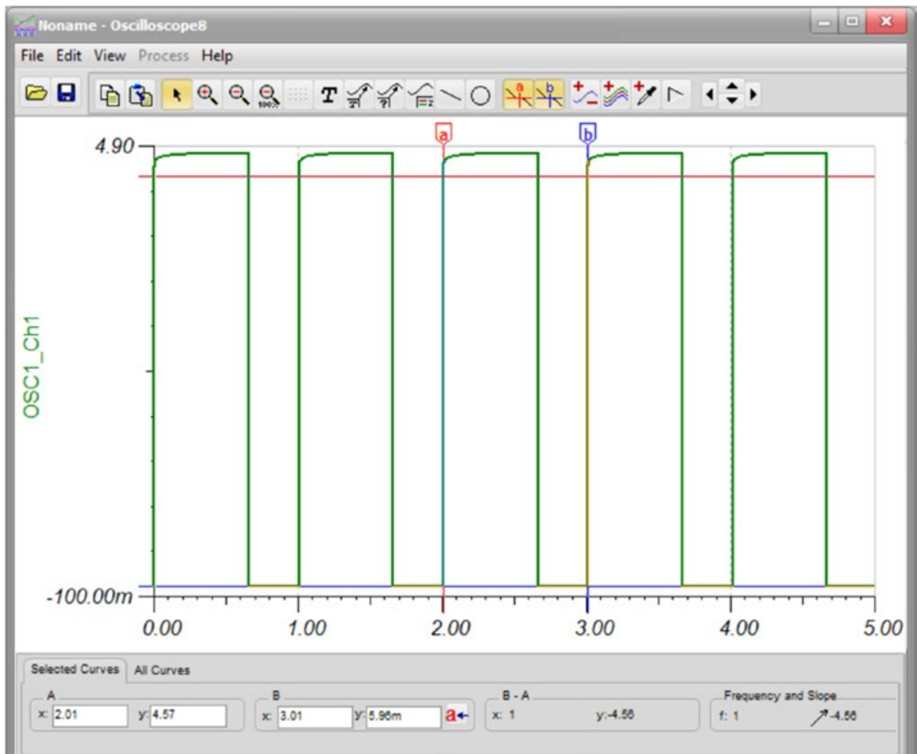


Figure 10-12. *The 1Hz Square Wave Trace*

Figure 10-12 displays the output trace when the timer is set using the components as calculated. The periodic time is shown as 1s. I do not say that this would produce a reliable 1Hz clock, but it does produce a square wave that you could use if accuracy was not your main concern as with our timer. The capacitance has been increased to 10μF to enable sensible values of resistance to be used.

A PWM Application

The speed of a DC motor can be controlled by varying the voltage applied across the motor. The simplest way we can achieve this variation without wasting any power would be to use a pulse width modulated square wave. That is what the PWM stands for.

To appreciate how we can achieve a PWM waveform, we should investigate what a square wave looks like. This is shown in Figure 10-13.

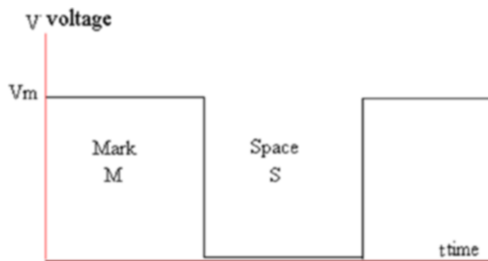


Figure 10-13. *A Simple Square Wave*

The simple square wave has two periods in it. One is called the mark time, which can be referred to as the "on time." This is when the voltage goes to its maximum value. This could be a positive voltage, as shown in Figure 10-13, but it could also be a negative voltage. The other period is called the "space time," which could be referred to as the "off time." This is normally the time period when the voltage is at 0V as shown in Figure 10-13.

All square waves have a mark-to-space ratio, which refers to the amount of time the square wave is high and low. The ratio is normally referred to as the "duty cycle." The simplest of square waves has a mark-to-space ratio of 50/50. This means that the mark time is 50% of the total periodic time for the square wave and the space time has the other 50% of the periodic time. Note the periodic time, normally given the symbol "T," is the time the waveform takes to go through all its possible values, that is, the time the wave takes to go through one cycle.

When a square wave voltage is applied to any device, such as a DC motor or an LED, then that device will respond to the average voltage of the square wave. If we look at the square wave voltage shown in Figure 10-13, then we can determine the average voltage, which I term "Vavge," as follows:

$$Vavge = \frac{VmM}{T} or = \frac{VmM}{M+S} \qquad \text{(Equation 1)}$$

Note "Vavge" is the average voltage, "Vm" is the voltage maximum, "M" is the mark time, "T" is the periodic time for the waveform, and "S" is the space time.

Note $\frac{M}{M+S}$ is termed the duty cycle. When the M time is equal to the space time, then the duty cycle is termed 50/50, and the average voltage is $\frac{Vm}{2}$.

If we now vary the mark time, it should be clear that we would vary the average voltage. Indeed, if the mark time, "M," was the same as the periodic time, "T," this would mean the space time "S" would be zero, and the average voltage would be a maximum of Vm. Conversely, if the space time was equal to the periodic time, then the average voltage would be 0V.

This means that the output voltage can be varied from almost 0 to VCC.

The circuit we can use, for the 555 timer, to create a PWM output is shown in Figure 10-14.

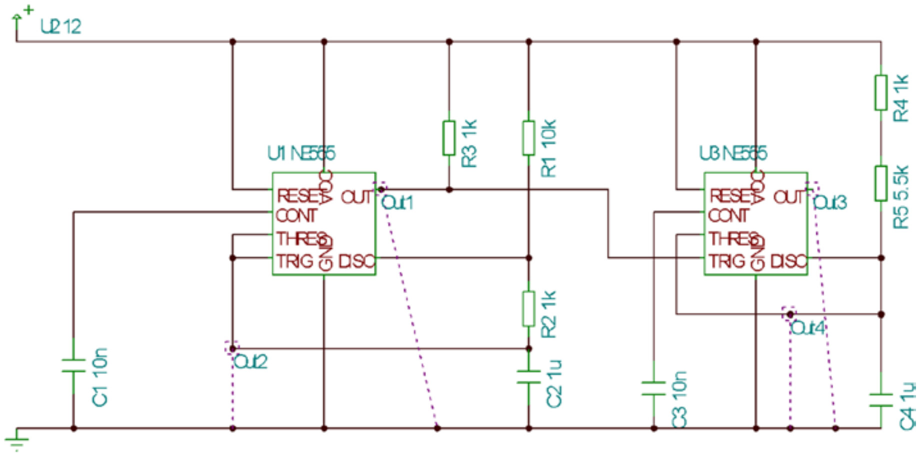


Figure 10-14. *The 555 Timer as a PWM Circuit*

We can simulate this circuit to confirm the circuit works and so provides a PWM at the fixed frequency. The frequency is set, as before, by R_1, R_2, and C_1 using the expression

$$f = \frac{1.44}{(R_1 + 2R_2)C_1}$$

Putting the values in from the circuit shown in Figure 11-13 gives

$$f = \frac{1.44}{\left(10^3 + 2 \times 1^3\right) \times 1^{-6}} = 120Hz$$

Figure 10-15 shows the trace of the output, and the periodic time of the square wave is shown as 8.41ms, that is, cursors B – A.

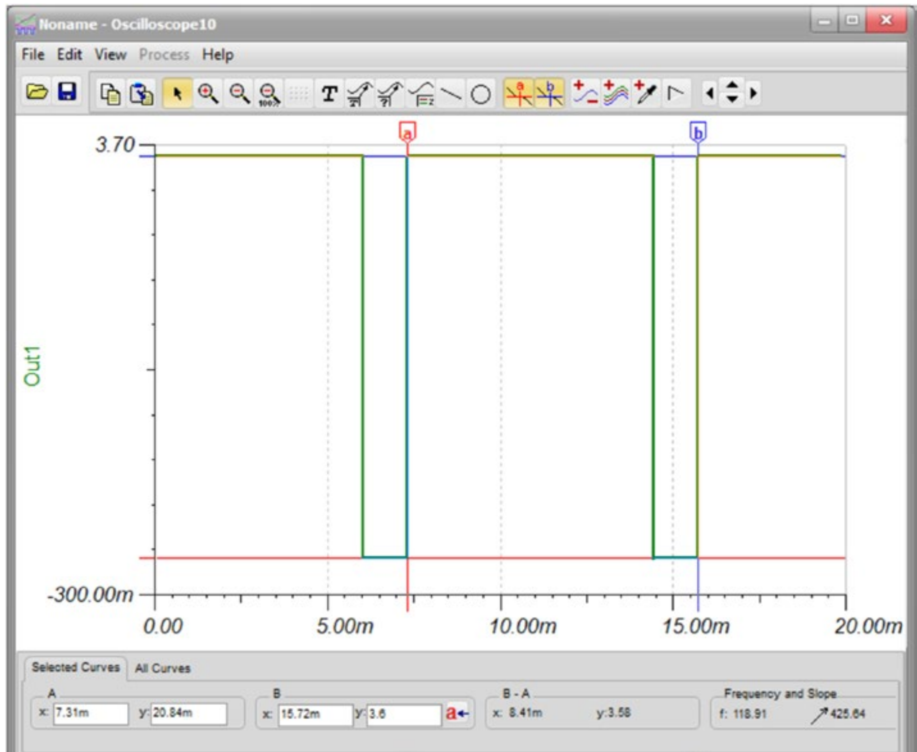


***Figure 10-15.** The Trace Showing the Periodic Time for the Output Is 8.41ms*

Using the periodic time, the frequency can be calculated as follows:

$$f = \frac{1}{T} = \frac{1}{8.41^{-3}} = 118.9Hz$$

This is close to what is expected.

It is important to ensure the time constant set by R_4, R_5, and C_4, which sets the mark time of the square wave, is no more than two-thirds that as set by R_1, R_2, and C_1. The frequency time constant for the $R_1R_2C_1$ is

$$frequency\ time\ constant = (R_1 + R_2)C_1 = 11ms$$

The current mark time constant for the PWM is

$$mark\ time\ constant = (R_4 \times R_5)C_4 = (1^3 \times 5.5^3) \times 1^{-6} = 6.5ms$$

Figure 10-16 shows the output with R_5 set to 5.5kΩ.

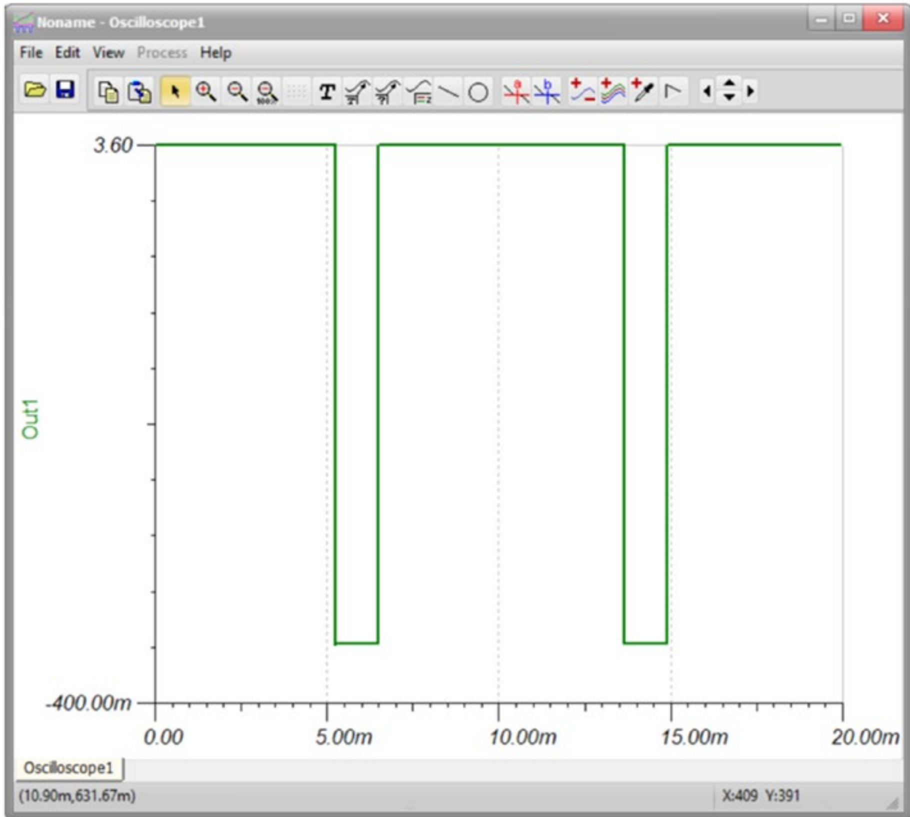


Figure 10-16. *The Output Voltage When R_5 Is 5.5kΩ*

If we reduce R_5 to 2kΩ, this should reduce this time constant to 3ms. Figure 10-17 shows the trace at the output. From it, we can see that the periodic time is still at 8.41ms, and so the frequency has not changed.

However, the mark time has reduced and the space time has increased. This means that the average voltage of the square wave voltage has reduced.

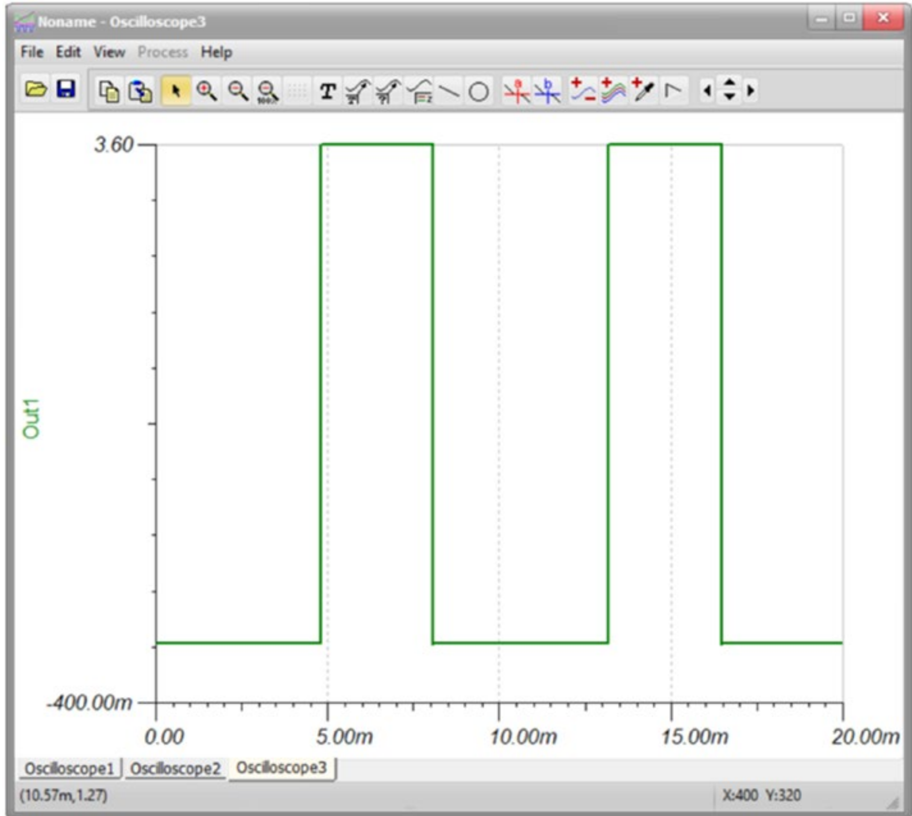


Figure 10-17. *The Output Voltage with the Mark Time Constant at 3ms*

If we increase the value of R_5, the mark time will increase. Figure 10-18 shows the output voltage with R_5 increased.

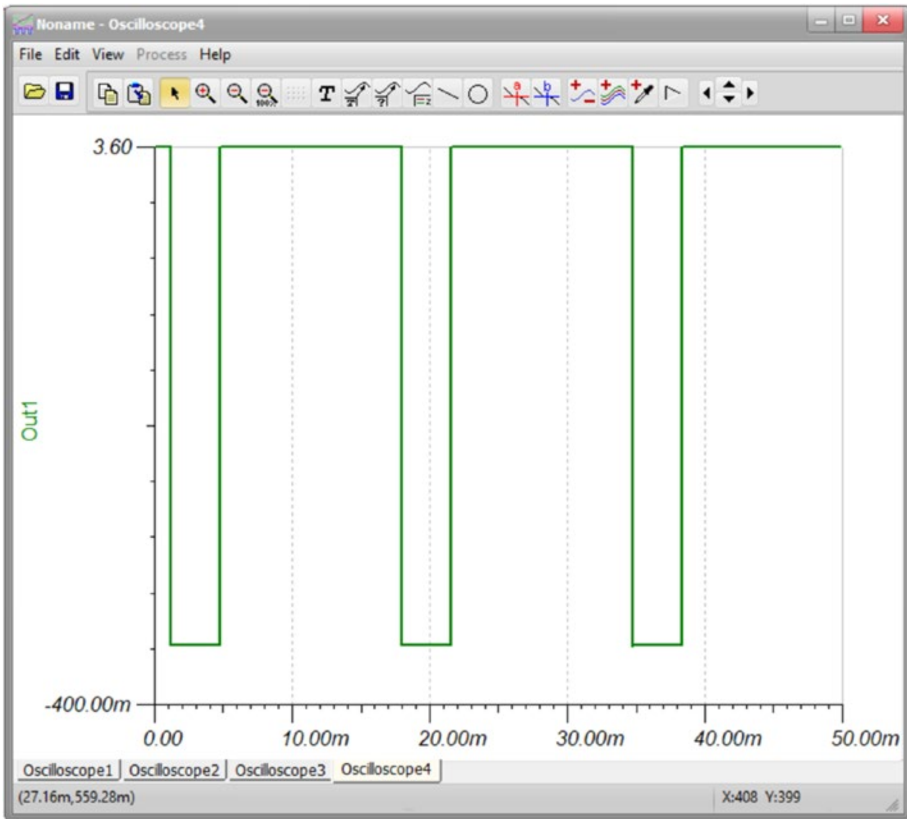


Figure 10-18. *The Output Voltage Is Increased*

In this way it is quite a simple circuit that we can use to vary the speed of a DC motor. You would simply replace R_5 with a variable resistor.

Summary

In this chapter we have studied how the 555 timer works and how it can be configured to be used in a variety of ways.

In the next chapter, we will learn how to use some of the basics of the ECAD software TINA 12. This will help us understand how to carry out some of the simulations used in the book.

CHAPTER 11

Using TINA 12

In this chapter we will look at how to use the ECAD software TINA 12. As we work through the book, we will use this software to simulate the circuits so that we can test the theories that are covered in this book. This chapter will help you learn how to use some of the basic aspects of this software, and we will also look at some of the more specialized aspects that we might use in this book. This chapter is not intended to be a manual for using TINA, but after reading this chapter, you should be able to carry out all the simulations discussed in this book.

What Is ECAD and TINA 12

ECAD stands for Electronic Computer-Aided Design, and TINA 12 is just one of many pieces of software you can buy to use in the design of electrical and electronic circuits. I am not saying it is the best software, as there will be pros and cons for any of the software available to you. However, I have used TINA while teaching electrical and electronic engineering, and I found it more than sufficient for every aspect of what I was trying to teach. At around £100 for the basic software, it is one of the more affordable pieces of software while giving you more than enough to start your career as an engineer. Even the demo version, which is free to download, will be more than enough for you to use alongside this book. With the demo version, you are allowed 31 runs of the software, on separate days, before you need to buy it. With this demo version, there are no restrictions except that you can't save your files.

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_11

Having TINA 12 at your disposal is simply like having a full electrical and electronic lab at your disposal. You can have almost any component you want at any value. You also have any piece of test equipment you could ever want at your disposal. You even have some specialist analysis tool that would be difficult to have anywhere else. However, to be fair, this is only what you would expect from any industrial-standard ECAD software.

The other ECAD software programs I have used are Proteus and Multisim.

These are very good ECAD software, but they do come at a price. It is up to you what software you eventually end up buying. However, as this book uses TINA 12 to simulate all the circuits in this book, I think it would be useful if I showed you how to use the most common aspects of the software.

Running the Software

The software can be downloaded from www.tina.com. You will have to register your request for the demo version, but that should not put you off. Assuming you have successfully downloaded the software, you need to click the TINA icon as shown in Figure 11-1.

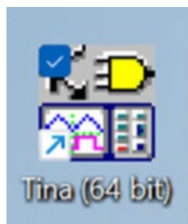


Figure 11-1. *The Program Icon for TINA*

You may be presented with a registration window or run as a demo. You may have to choose to run the software as an administrator. The program should then open up with the main editing window as shown in Figure 11-2.

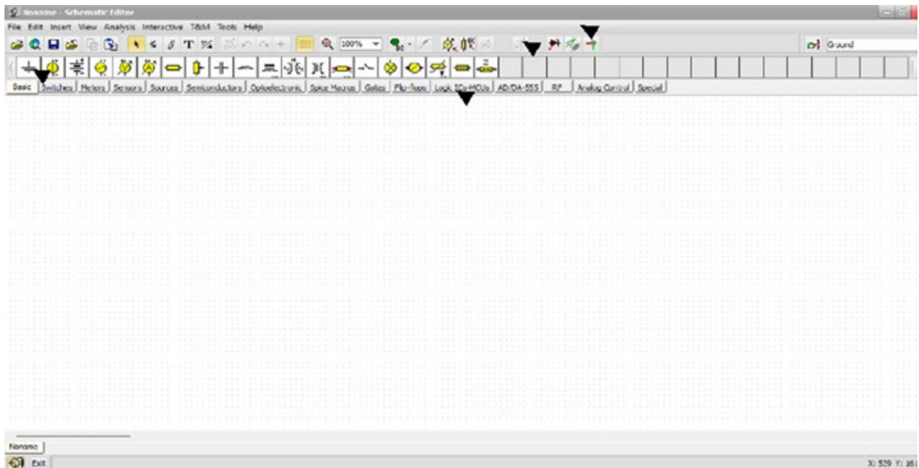


Figure 11-2. *The Opening Screen*

There are four tool bars on the main working window of TINA. These are identified in Figure 11-2 as

- **The main menu bar**: This is where we can select the following options:
    - File, Edit, Insert, View, Analysis, Interactive, ,T&M, Tools, and Help.
- **The main icon bar**: This has some graphical icons for the open, save, and other commands.
- **The current category bar**: TINA splits the different components that you can use into different categories. When a category has been selected, the components in that category are shown in this menu bar.
- **The component categories bar**: This is where TINA allows you to select one of the many different component categories that TINA uses.

Note these are my names that I have given to the menu bars. This is so that I can refer to them in the following text.

Creating Our First Test Circuit

The first circuit we will look at creating is the simple two-input NAND gate. We will use this test circuit to show you how to complete a simple truth table. The first thing we will do is find the NAND gate. This is in the "Gates" component category for TINA. Therefore, we must click the mouse on the word "Gates" on the component categories bar. The current category bar should change to that shown in Figure 11-3.

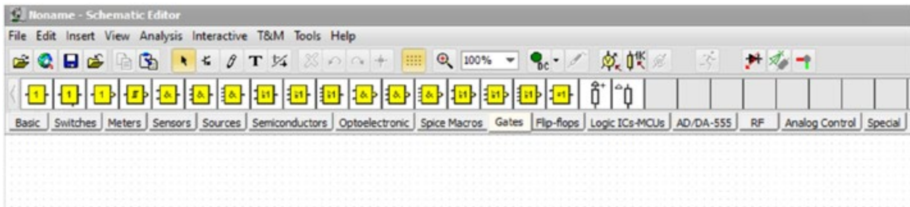


Figure 11-3. *The Gates Current Category Menu Bar*

We are looking for the two-input NAND gate; this is the 11th icon on the current category bar. If you click the mouse on that icon, you will select the NAND gate option from the current category menu bar. If you move the mouse – note you don't have to click and drag the mouse – you will see that the symbol for the two-input NAND gate follows the mouse around the main drawing area. Now, with the mouse in a suitable position, simply click the mouse, and the NAND gate will be dropped in the place the mouse is pointing to when you clicked the mouse. You should see that the NAND gate is still red, which means it is still selected. If the gate turns green, that is, it is deselected, don't worry; it just means you clicked the mouse too many times.

Now we want to add the two inputs. These inputs will come from two switches as we need the inputs to be switched to a logic "1," that is, to a high, or to a logic "0," that is, to a low. TINA allows us to use some special switches for digital circuits or some standard switches. We will use the special switches as we are simulating digital circuits in this book. To select this, we must change the category to that of "Switches." Then select the high-low switch icon, which is the third icon on the current category bar, and place it just to the left of the NAND gate. If we click the mouse on this selection, we can now move a switch to a position close to the NAND gate. As we are using the two-input NAND gate, we need to repeat the procedure to get the second input switch. The two switches, one for each input, are shown in Figure 11-4.

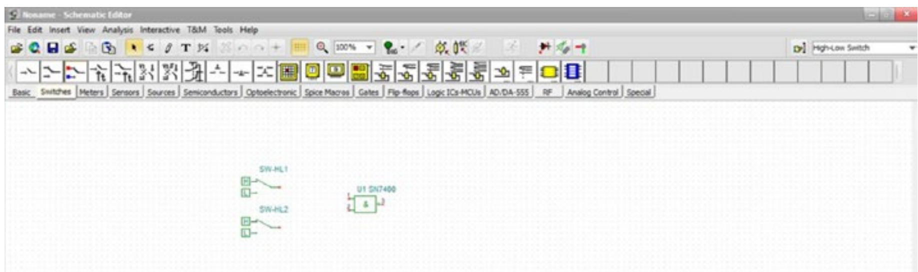


Figure 11-4. *The Two Inputs Added to the Schematic*

TINA does provide us with a grid to help align the components we use. If the grid is not present in the main drawing area, then, if you click the mouse on the word "View," from the main menu bar, you should see a drop-down menu appear. If you now select the word "Grid," the tick should appear, and the grid will be turned on in your main drawing area. This is shown in Figure 11-5.

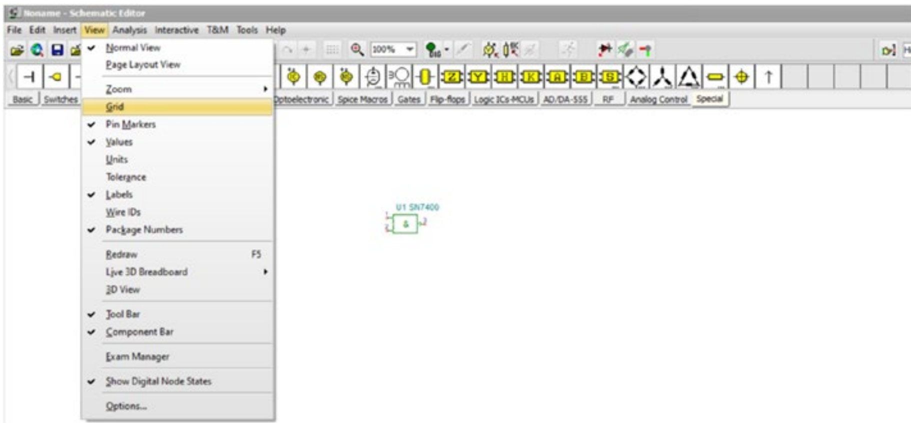


Figure 11-5. *Turning the Grid On*

TINA now gives the switches the default labels of SW-HL1 and SW-HL2. We will change the label SW-HL1 to "Input A," a more sensible name for it. To do this simply double-click the mouse on the switch to select it and open the switch parameters window, shown in Figure 11-6. We can change the label to "Input A" and also change the hotkey to the letter "A." When you first select the parameters window, there will be a small box with a downward arrow alongside the "-" symbol for the "Hotkey." If you click the mouse on that arrow, a selection of letters from the keyboard should appear so that you can select any convenient letter. We should choose "A." Selecting the "A" allows you to operate this switch by tapping the "A" key on your keypad, a very useful option.

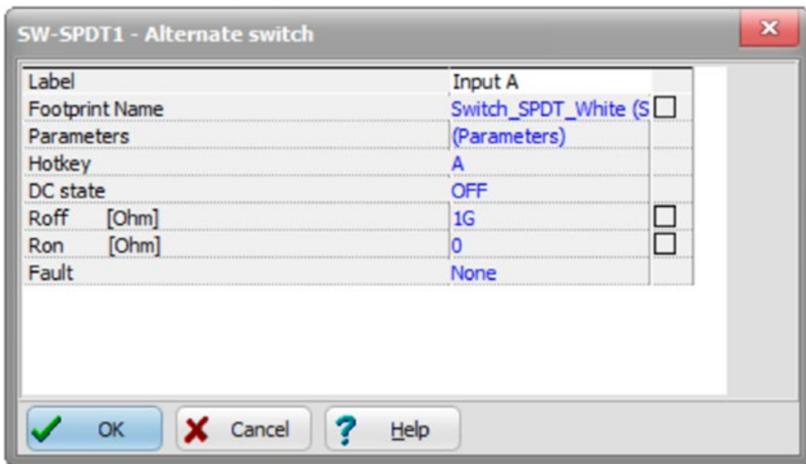


***Figure 11-6.** The Parameters Window for the SPDT Switch*

Once you are happy with your selections, simply click the mouse on the OK button. The window closes, and the switch will reflect your changes. You can now change the parameters for the second switch, the name to "Input B" and the hotkey to "B." The screen should now look like that shown in Figure 11-7.

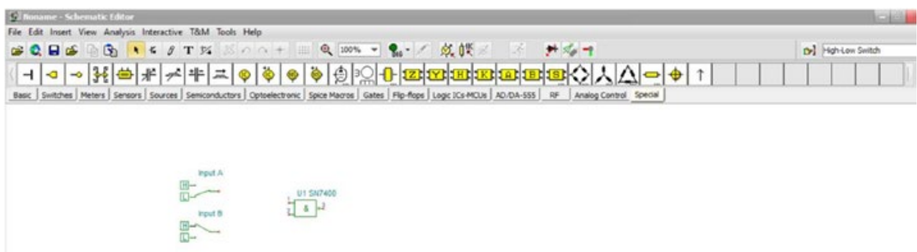


***Figure 11-7.** The Circuit with the Two Input Switches*

Now we need to connect the common terminal to one of the inputs to the NAND gate. At this stage we will simply wire the switch to the NAND gate; later in this chapter, we will use the jumper terminals for more complicated circuits. If we move the mouse to the output from the switch

"A," the mouse will change to a pencil; this is the wire tool. Now click the mouse to select the wire tool at that point. Now, while still keeping the mouse clicked, drag the mouse to the top input of the NAND gate. At that point let go of the mouse, and the wire should connect to the input of the NAND gate. Note, if you let go of the mouse too soon, the wire will end at that point on the screen when you clicked the mouse, so take care. This will take a bit of practice, so be patient and keep trying. When you have connected both switches to the NAND gate, the circuit should look like that shown in Figure 11-8.

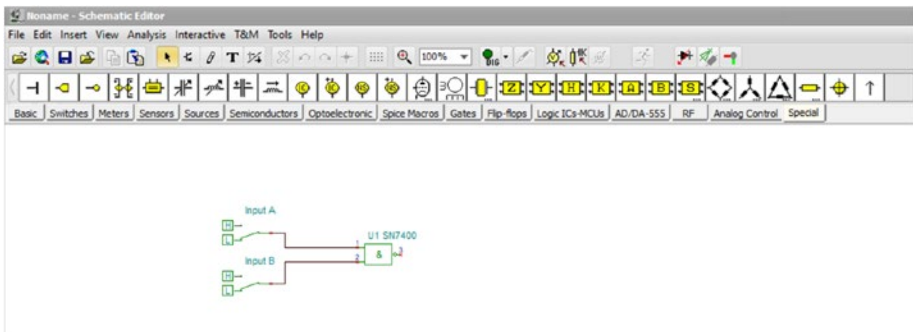


Figure 11-8. *The Two Input Switches Connected to the NAND Gate*

We will use some logic probes to indicate the responses of the circuit to the different input combinations. Then we will record the response of the NAND gate in the truth table shown in Table 11-1. The logic probes are in the "Meters" category of components. Therefore, select the "Meters" tab from the component categories menus bar. The screen will now look like that as shown in Figure 11-9.

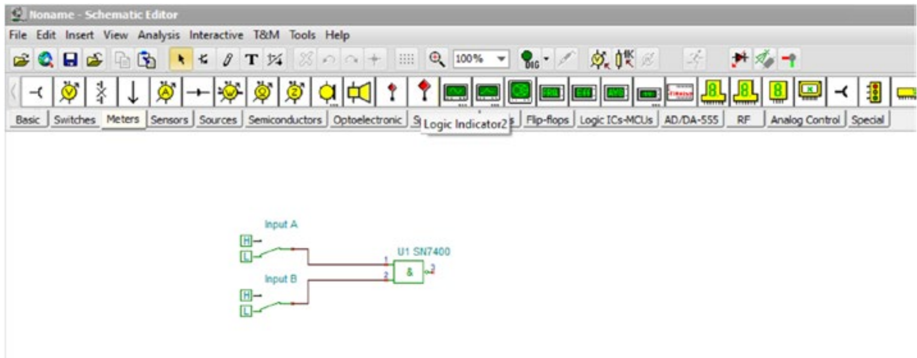


Figure 11-9. *Selecting the Logic Indicators from the "Meters" Category*

We should select the large logic indicator, "Logic indicator2," and place that at the output. We should select two smaller logic probes and use them to indicate the logic at the inputs. After placing the logic indicators and changing their labels, your screen should look like that in Figure 11-10.

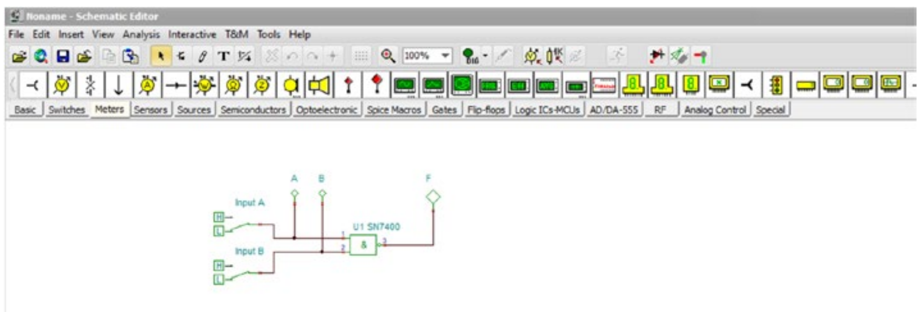


Figure 11-10. *The Completed Circuit Ready to Run the Simulation*

Now we are ready to run the simulation and record the response of the NAND gate. TINA offers a range of simple simulation processes, and the default process is the "DC" option, which is on the second menu bar just to the right of the 100% box. If we click the mouse on the small arrow next to the term "DC," a small drop-down menu would appear. We need to select

the “Dig” option to run a digital simulation of this circuit. This is shown in Figure 11-11.

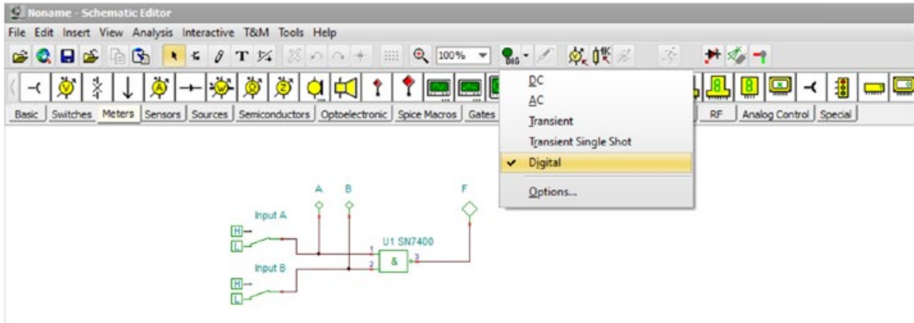


Figure 11-11. Selecting the “Dig” Simulation Option

Now, after ensuring both inputs are switched to low “L,” run the simulation by simply clicking the mouse on the icon “Dig.” The icon will turn green, and the logic indicators will show the response of the circuit. This is shown in Figure 11-12.

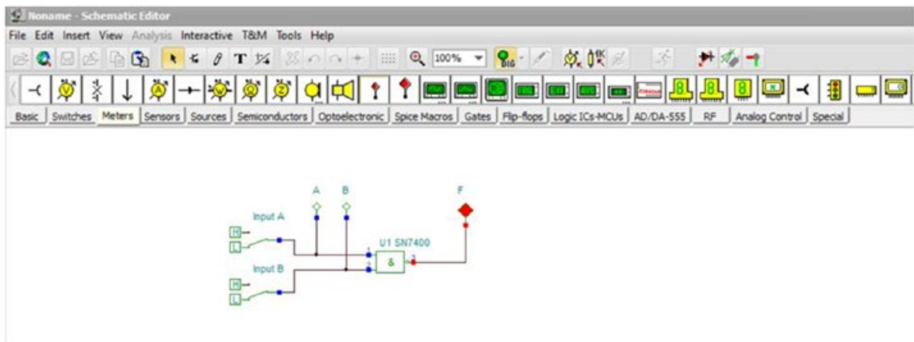


Figure 11-12. The NAND Simulation Running

As there are only two inputs, there will only be four possible combinations of the logic at the inputs. These are shown in Table 11-1.

Table 11-1. *The Truth Table for the NAND Gate*

Input A	Input B	Output F
0	0	1
0	1	1
1	0	1
1	1	0

With both inputs switched to a logic "0," the two small indicators are shown turned off, that is, at a logic "0," while the large indicator, for the output "F," is shown turned on, that is, at a logic "1." Now change the logic of the two inputs as shown in Table 11-1 and record the response of the NAND gate. When we do this, we should see that the result is the same as that shown for the NAND gate response in Table 2-1 of Chapter 2.

This simple example of creating the NAND circuit should help you create the circuits for most of the simulations. The next example, of using the software TINA, should help with some of the more complicated circuits, that is, those that have a large number of wires and use the more practical ICs.

Using a Binary Counter

This next example will be based on the 7493 binary counter. As we know this is a counter; then, the simplest way of finding this component would be to use the "Logic ICs-MCUs" category from the component categories menu bar. The sixth icon in this category is the "Ctr" counter component. If we click the mouse on that icon, a drop-down menu appears from which we can scroll and find the 7493 counter as shown in Figure 11-13.

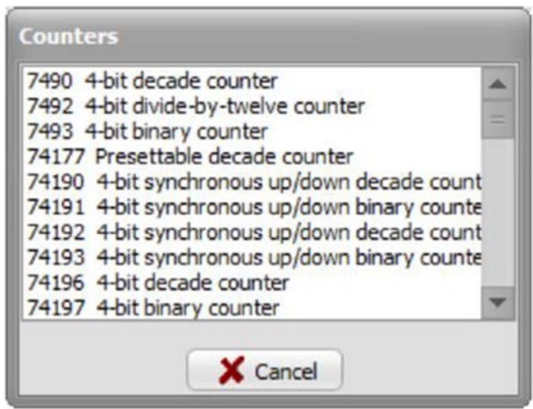


Figure 11-13. *Selecting the 7493 4-Bit Binary Counter*

If we click the mouse on that selection, the counter will become active on the screen. We should place it somewhere convenient on the screen.

Another alternative way to finding a component is that TINA allows us to search using the "Find Component." To select this option, simply click the mouse on the small hand icon at the far right of the "main icon bar," as shown in Figure 11-14.

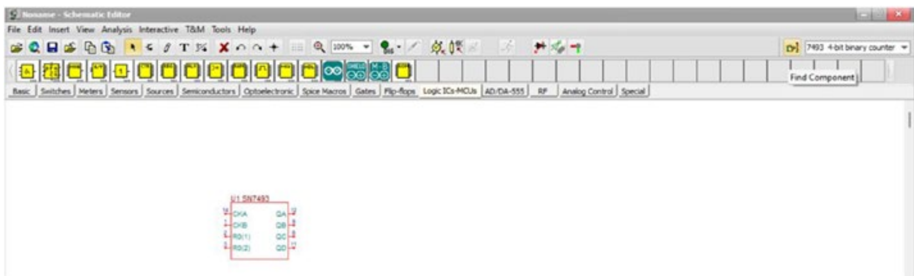


Figure 11-14. *Selecting the Find Component Option from the Second Main Menu Bar*

When you click the mouse on this icon, the pop-up menu will appear. One problem with this approach is that you will need to know the IC number for the item you are searching. As an example, we will use this Find

Component option to find the 7406 chip, which is a hex inverter chip. The Find Component window to find this component is shown in Figure 11-15.

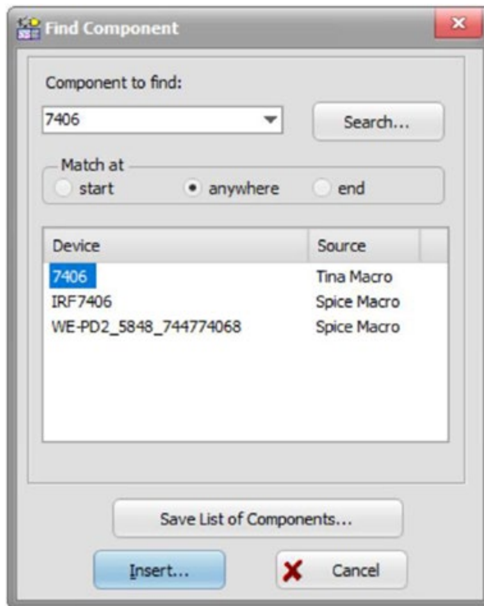


Figure 11-15. *The Find Component Window*

Once you have found the component you are looking for, simply select it and then click the mouse on the button "Insert." The window will close, and the component will be moveable on the schematic.

Using Jumper Terminals

As there will be a lot of separate connections with this circuit, we will use the "jumper" terminals to make the schematic less cluttered and easier to read and use. This "jumper" component is within the "Special" category of components. Therefore, click the mouse on the "Special" tab on the component categories menu bar. The "jumper" is the first component icon, and so click it to drag a "jumper" onto the schematic as shown in Figure 11-16.

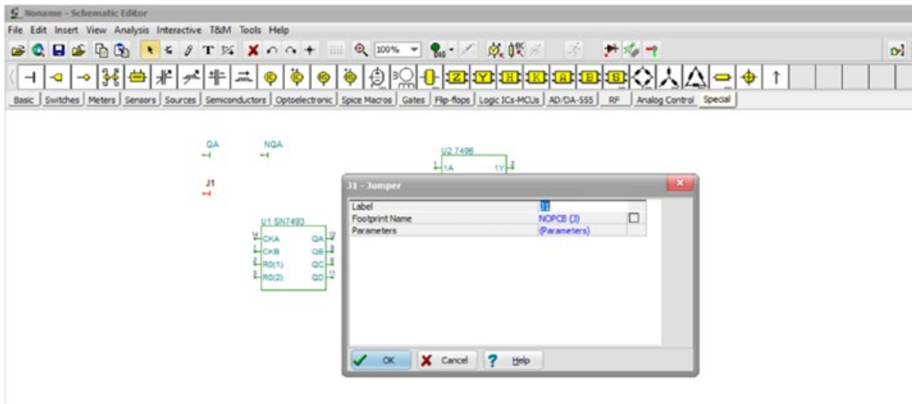


Figure 11-16. *The Schematic with the Jumper Properties Window*

Figure 11-16 shows three jumpers already inserted onto the schematic. It also shows the properties window that you can use to change the label of the jumper. When a schematic has more than one label with the same name, TINA will automatically connect them together electrically without showing the wire on the schematic.

Figure 11-17 shows the schematic fully populated with the components for this example.

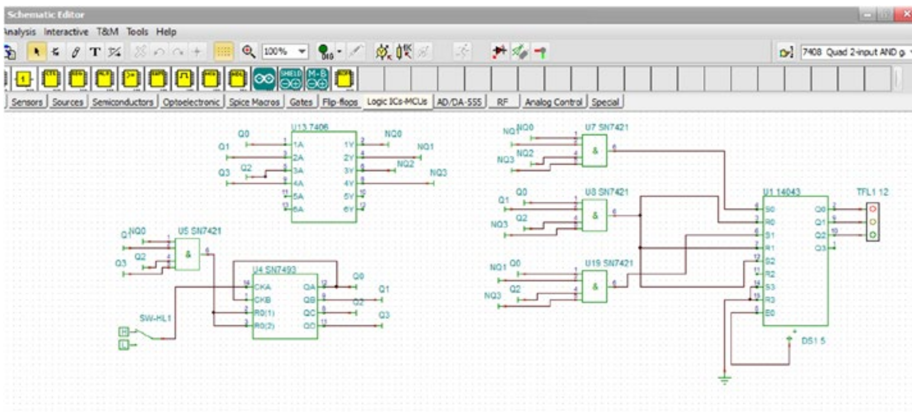


Figure 11-17. *The Completed Circuit for This Example*

I hope you can see that using the jumpers greatly reduces the clutter on the schematic. You can start the simulation by clicking the mouse on the green “Dig” button on the “main icon bar.” The idea of the circuit is that the red lamp comes on at a count of zero with the amber light coming on at a count of 5. Then at a count of 7, the red and amber lamps go out, and the green lamp comes on.

The 7493 binary counter will count the transitions from high to low of the switch “SW-HL1” to control when the lamps come on and off. Therefore, as you click the switch to change from high to low, the count increases. You should see that the lamps turn on and off as expected, but the red lamp comes back on when the count reaches 8. This is an issue with this circuit, and we will look at what is going wrong in Chapter 9.

Creating a Macro for the 7400 IC, a Quad-Two-I/P NAND Gate

We will use the creation of this IC as an example of how to carry out the process of creating what TINA calls a “macro.” The process is to create the actual circuit that is inside the macro, in this case the 7400 IC, as a normal schematic but with addition of some “macro pins.” From the data sheet for the 7400, we know the IC has 14 pins. Therefore, we will start off by creating a new schematic and inserting these 14 macro pins. The macro pins can be found by selecting the "Special” tab on the “component categories bar.”

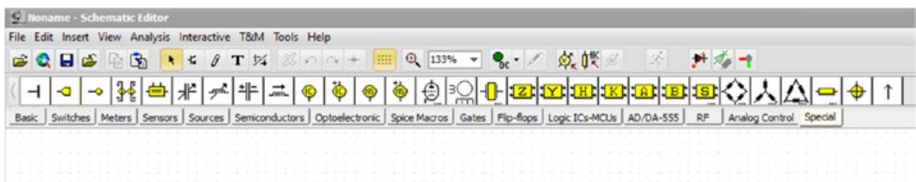


Figure 11-18. *The Special Tab*

Figure 11-18 shows the “Special” tab selected. The macro pin tool is the second item on this tool bar. Figure 11-19 shows all 14 pins inserted onto the schematic.

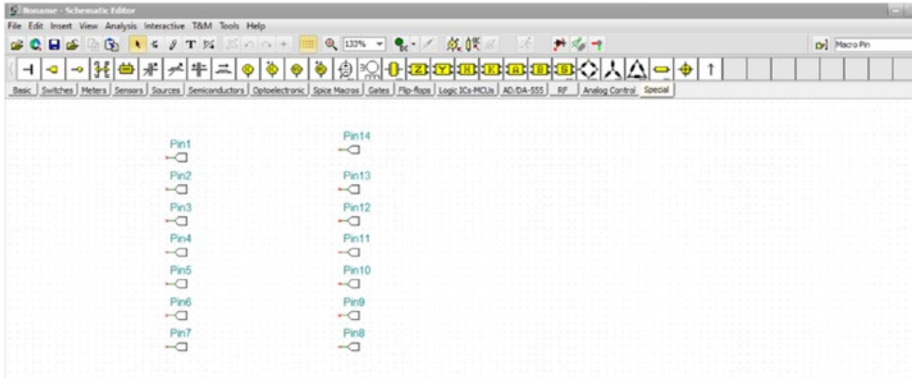


Figure 11-19. *The 14 Marco Pins Added to the Schematic*

We need to make the component reflect the actual IC as closely as possible. Therefore, we need to refer to the data sheet for the IC we are creating, the 7400 in this case. This explains why we have added the 14 pins as two sets of 7. Note also the numbering of the pins follows the data sheet as well.

We should label the pins to show what they connect to inside the IC. This information can be obtained from the relevant data sheet. Figure 11-20 shows the 14 pins correctly labeled.

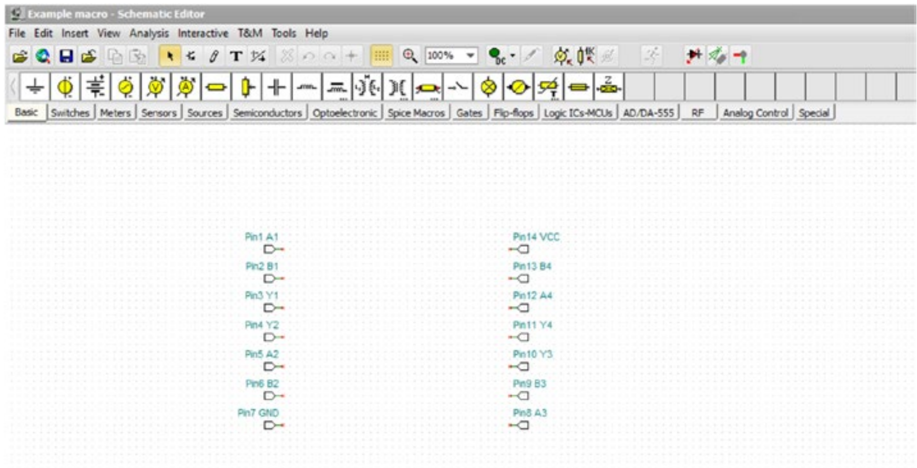


***Figure 11-20.** The Pins Fully Labeled*

Note we have mirrored pins 1–7 so that the connections are all pointing inward. Now it is simply a matter of inserting the individual NAND gates and connecting them to the pins of the macro. The completed circuit is shown in Figure 11-21.

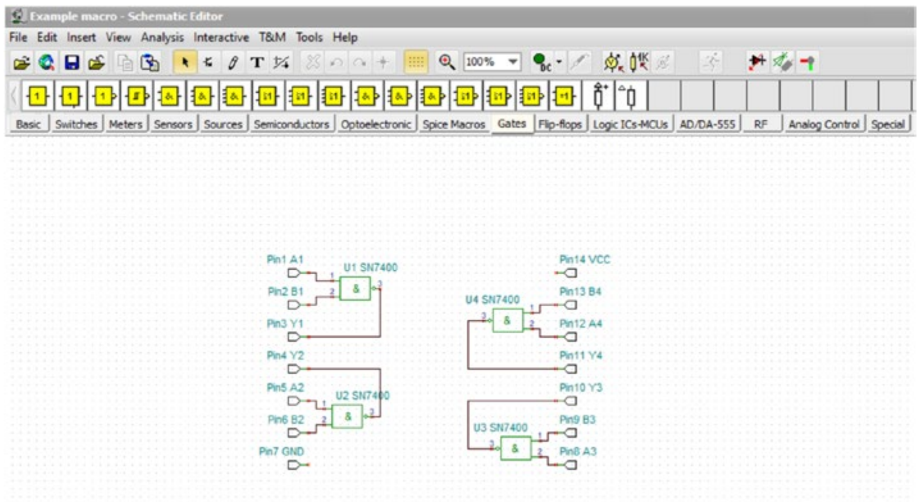


***Figure 11-21.** The Completed Circuit for the Macro*

The GND and VCC pins are only for show as TINA does not use these connections in the model of the individual gates.

Now we need to save this circuit as an actual macro. To do this we need to select the New Macro Wizard option from the drop-down menu that appears when we click the “Tools” option from the main menu bar, as shown in Figure 11-22.

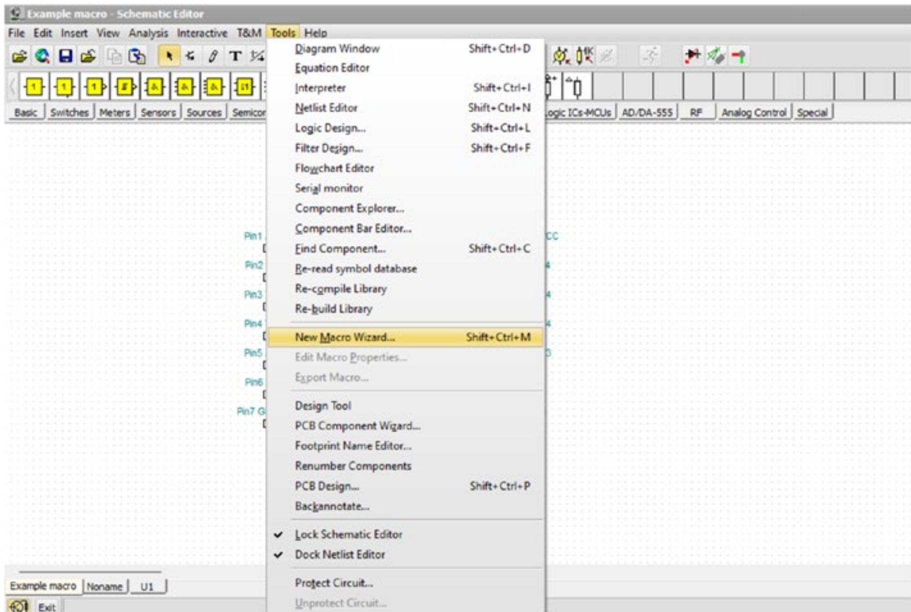


Figure 11-22. *Selecting the New Macro Wizard Option*

When this option is chosen, the New Macro Wizard window appears. We will give the macro a suitable name and select the Current circuit option as shown in Figure 11-23.

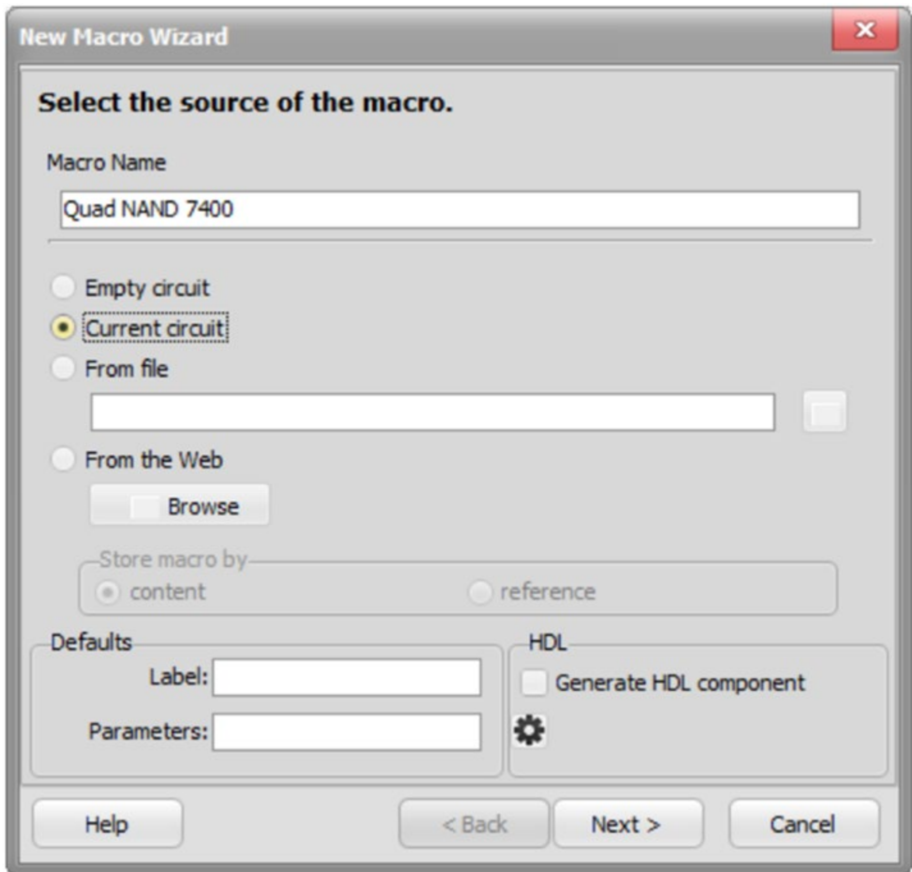


Figure 11-23. *The New Macro Wizard Window*

Once we are happy with our choices, we simply click the Next button to move on to the next window, shown in Figure 11-24.

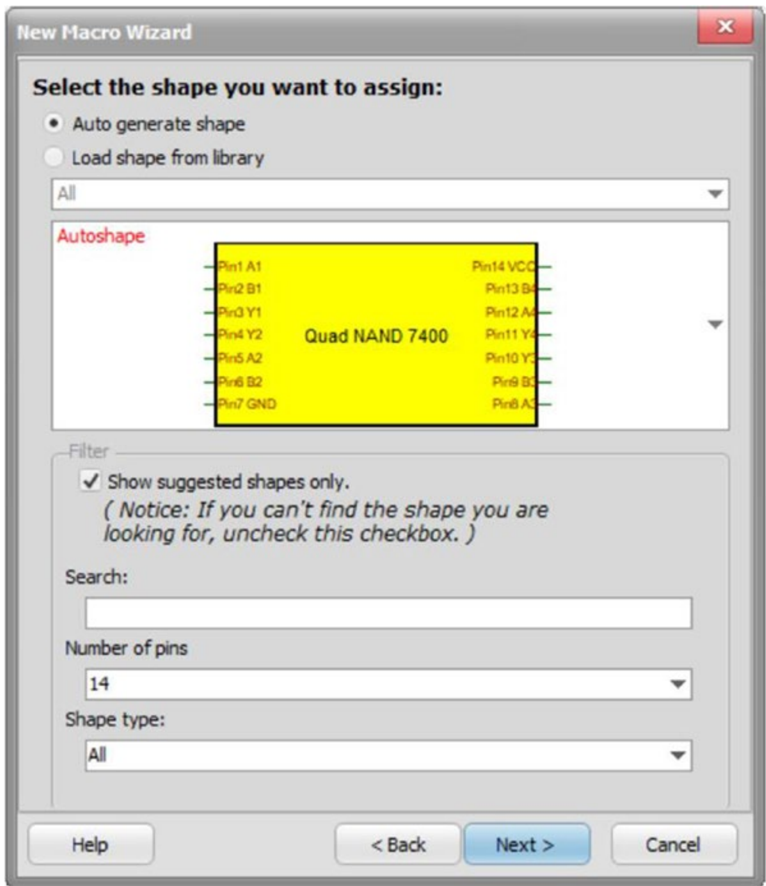


Figure 11-24. *The Second New Macro Wizard Window*

You should select the Auto generate shape option, which is the default option, unless you want to create a more realistic shape for the IC. If you are happy with the shape for the IC, simply click Next, and you will be presented with a window that asks you to give the macro a name and save the file in the macro lib folder.

Once you have saved the macro, you will be given the option to insert it onto the current schematic. At this point don't use that option as we will create a new schematic that will use the macro. This is done next.

Using the Quad NAND 7400 Macro

We will use the macro to create the de-bounce circuit, using two cross-coupled NAND gates, that we looked at in Chapter 4. Therefore, select the New File option from the main menu bar and select the Insert tab on the main menu bar. Now select the Macro option from the drop-down menu that appears. This is shown in Figure 11-25.

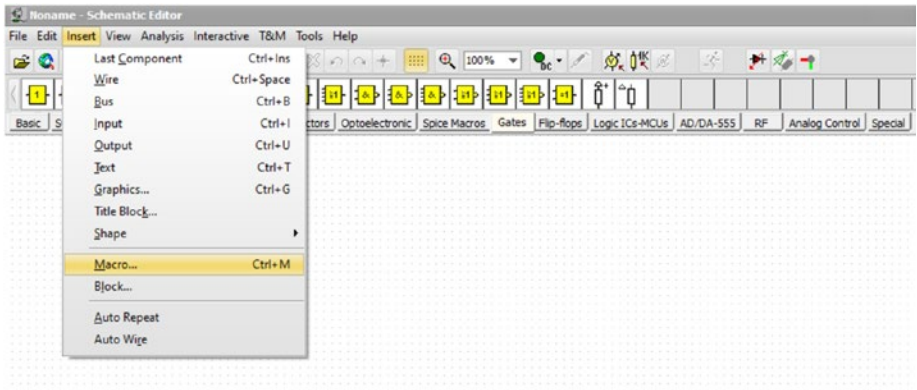


Figure 11-25. *Inserting a Macro onto the Schematic*

This should take you to the folder where you have just saved the macro schematic you have just created. You now need to click the mouse on the correct macro file you want to insert onto the schematic. If you now click the “Open” tab shown in the window, the newly created macro will be inserted onto the schematic. We can now use this macro in just the same way as if we had inserted four 7400 NAND gates onto the schematic. The completed de-bounce circuit using this macro is shown in Figure 11-26.

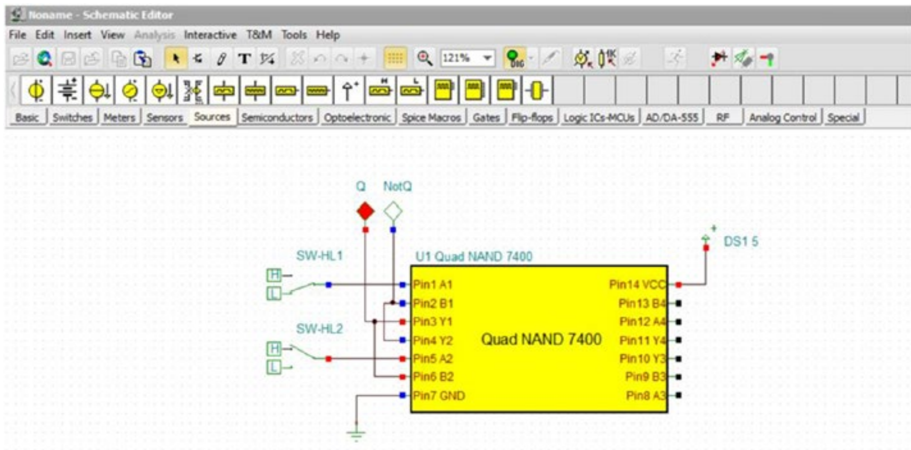


Figure 11-26. *The Completed De-bounce Circuit Using the Macro*

If you simulate this circuit, you should see that it works in the same way as the de-bounce circuit looked at in Chapter 4. The macro-IC can be made to look more like the actual chips, but that would take too long to show you. The idea is to show you how you would use the actual ICs to wire up a practical circuit, and these macros will be sufficient for that.

Summary

In this chapter we have looked at using some options available to you within the TINA 12 ECAD software. We will go through these and some more in the book as we use them. I hope you do find this software useful in your design work and that you do enjoy using it.

Appendix 1

I hope by now you have learned a lot about digital electronics after reading this book. The following is a set of exercises designed to help you put what you have learned into practice. The answers to these exercises, and the exercises given out in the chapters of the book, are given in Appendix 2.

Exercise 1

Use Boolean algebra to simplify the following:

1a

$$F = A.B + A.\overline{B}$$

1b

$$F = \left(A.B\right) + \left(\overline{A}.B\right) + \left(\overline{A}.\overline{B}\right)$$

1c

$$\left(A + B\right).\left(B + \overline{C}\right)$$

1d

$$F = \left(A + D\right) + \left(A + B + C + D\right)$$

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_1

Exercise 2

Draw up the circuits for the original expression and the simplified expression for the tasks in Exercise 1 and so confirm the simplified expressions work.

Exercise 3

Use Karnaugh maps to simplify the following Boolean expressions. In all the questions, the MSB is the "A" variable and the LSB is the "D" variable:

3a

$$F = \left(\overline{A}.\overline{B}.\overline{C}\right) + \left(A.\overline{B}.\overline{C}\right) + \left(\overline{A}.\overline{B}.C\right)$$

3b

$$F = \left(\overline{A}.\overline{B}.\overline{C}\right) + \left(\overline{A}.\overline{B}.C\right) + \left(\overline{A}.B.\overline{C}\right) + \left(A.\overline{B}.C\right)$$

3c

$$F = \left(\overline{A}.\overline{B}.\overline{C}.\overline{D}\right) + \left(\overline{A}.\overline{B}.\overline{C}.D\right) + \left(\overline{A}.\overline{B}.C.D\right) + \left(A.\overline{B}.\overline{C}.\overline{D}\right) + \left(A.\overline{B}.C.D\right) + \left(A.\overline{B}.\overline{C}.D\right)$$

3d

$$F = \left(\overline{A}.B.\overline{C}.\overline{D}\right) + \left(A.B.\overline{C}.\overline{D}\right) + \left(\overline{A}.B.C.\overline{D}\right) + \left(A.B.C.\overline{D}\right) + \left(\overline{A}.B.C.D\right) + \left(A.B.C.D\right) + \left(\overline{A}.B.\overline{C}.D\right) + \left(A.B.\overline{C}.D\right) + \left(A.B.\overline{D}\right)$$

3e

$$F = \left(\overline{A}.\overline{B}.\overline{C}.\overline{D}\right) + \left(\overline{A}.\overline{B}.\overline{C}.D\right) + \left(\overline{A}.B.\overline{C}\right)$$

Exercise 4

Draw up the circuits for the original expression and the simplified expression for the tasks in Exercise 3 and so confirm the simplified expressions work.

Exercise 5

Derive the truth tables and Boolean expressions for the following logic circuits, shown in Figure A-1 and A-2:

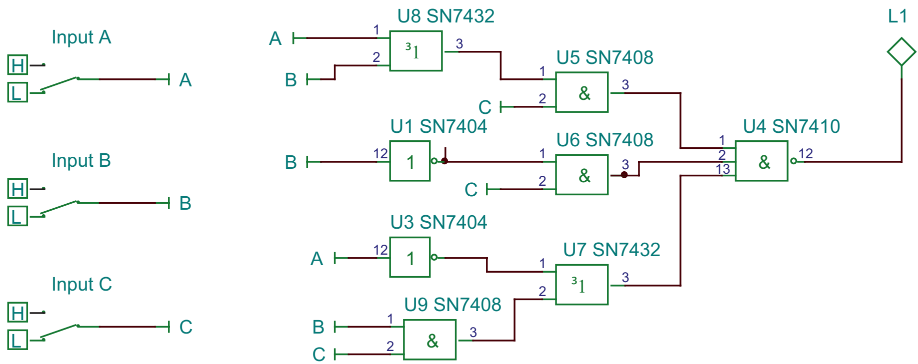


Figure A-1. *Circuit for Q5.1*

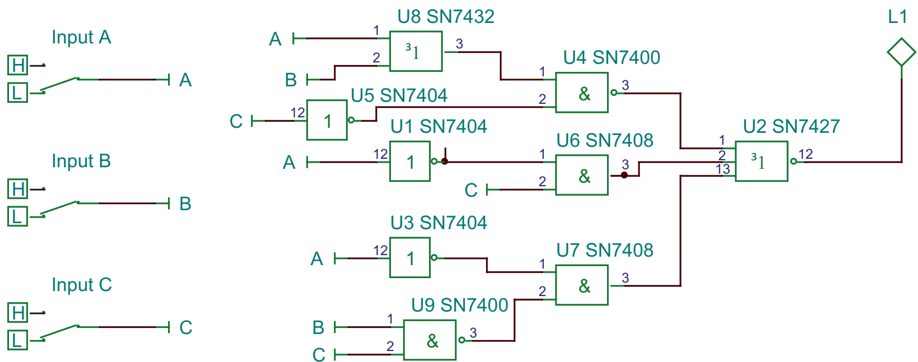


Figure A-2. *Circuit for Q5.2*

APPENDIX 2

Solutions for Exercises in the Chapters

In this part of the appendix, we will look at the possible solutions I have for the exercise set throughout the book.

Chapter 1

Exercise 1

Convert these numbers to binary:

99	is	1100011
255	is	11111111
137	is	10001001

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_1

Exercise 2

20 + 21, 35 + 123, 125 + 75

20 in 8-bit binary is	00010100
21 in 8-bit binary is	00010101
Sum =	**00101001 in decimal = 41**
35 in 8-bit binary is	00100011
123 in 8-bit binary is	01111011
	10011110 in decimal = 158
125 in 8-bit binary is	01111101
75 in 8-bit binary is	01001011
	11001000 in decimal = 200

Exercise 3

128 – 28, 79 – 78, 55 – 5, 251 – 151

128 in 8-bit binary is			10000000
28 in 8-bit binary is	00011100	Invert	11100011
Add 1			00000001
Total			**01100100 in decimal = 100**
79 in 8-bit binary is			01001111
78 in 8-bit binary is	01001110	Invert	10110001
Add 1			00000001

55 in 8 bit binary is 00110111

5 in 8 bit binary is 00000101 Invert 11111010

Add 1 00000001

Total **00110010 in decimal = 50**

251 in 8-bit binary is 11111011

151 in 8-bit binary is 10010111 Invert 01101000

Add 1 00000001

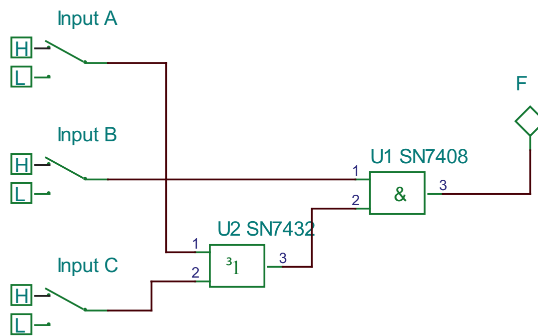
Total **01100100 in decimal = 100**

Chapter 2

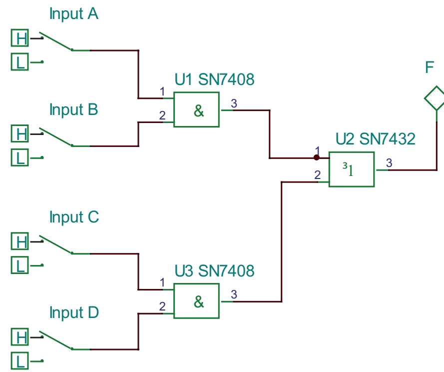
Exercise 1

$$F = \overline{\overline{\left(\bar{A}.B\right) + C}.D}$$

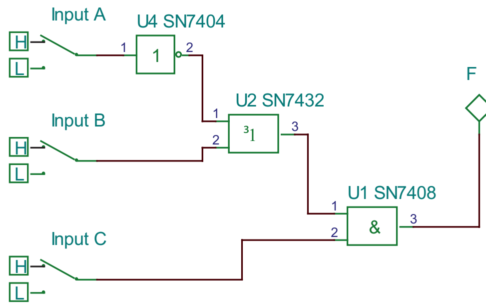
Exercise 2



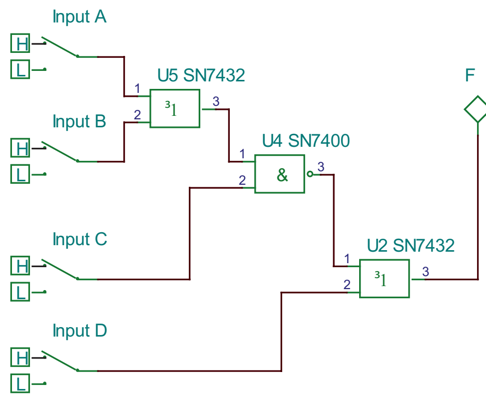
Circuit 1



Circuit 2



Circuit 3



Circuit 4

Chapter 3

Exercise 3.1

1. $F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.C + \overline{A}.\overline{B}.C$
2. $F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C$
3. $F = \overline{A}.B.C.D + \overline{A}.B.\overline{C}.D + A.B.D$

Question 1

$$F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.C + \overline{A}.\overline{B}.C$$

Take $\overline{A}.\overline{B}$ out as a common factor:

$$F = \overline{A}.\overline{B}\left(\overline{C} + C\right) + \overline{A}.B.\overline{C} + \overline{A}.B.C$$

Take $\overline{A}.B$ out as a common factor:

$$F = \overline{A}.\overline{B}\left(\overline{C} + C\right) + \overline{A}.B\left(\overline{C} + C\right)$$

Knowing

$$\overline{C} + C = 1$$

we get

$$F = \overline{A}.\overline{B}\left(1\right) + \overline{A}.B\left(1\right)$$

which means

$$F = \overline{A}.\overline{B} + \overline{A}.B$$

Now take $\overline{A}$ out as a common factor, which gives

$$F = \overline{A}.\left(\overline{B} + B.\right)$$

Therefore

$$F = \overline{A}$$

Question 2

$$F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C$$

Take $\overline{A}.\overline{B}$ as common factors out of the first two terms:

$$F = \left[\overline{A}.\overline{B}\left(\overline{C} + C\right) + A.\overline{B}.\overline{C} + A.\overline{B}.C\right]$$

Knowing $\overline{C} + C = 1$ we have

$$F = \overline{A}.\overline{B} + A.\overline{B}.\overline{C} + A.\overline{B}.C$$

Now take $A.\overline{B}$ out of the last two terms, which gives

$$F = \overline{A}.\overline{B} + A.\overline{B}\left(\overline{C} + C\right)$$

Knowing $\overline{C} + C = 1$ we have

$$F = \overline{A}.\overline{B} + A.\overline{B}$$

Take $\overline{B}$ out as a common factor, which gives

$$F = \overline{B}\left(\overline{A} + A\right)$$

Knowing $\overline{A} + A = 1$ we have

$$F = \overline{B}$$

Question 3

$$F = \overline{A}.B.C.D + \overline{A}.B.\overline{C}.D + A.B.D$$

Take B. D out as a common factor:

$$F = B.D\left(\overline{A}.C + \overline{A}.\overline{C} + A\right)$$

Take $\overline{A}$ out as a common factor:

$$F = B.D\left[\overline{A}\left(C + \overline{C}\right) + A\right]$$

Knowing $\overline{C} + C = 1$ we have

$$F = B.D\left(\overline{A} + A\right)$$

Knowing $\overline{A} + A = 1$ we have

$$F = B.D$$

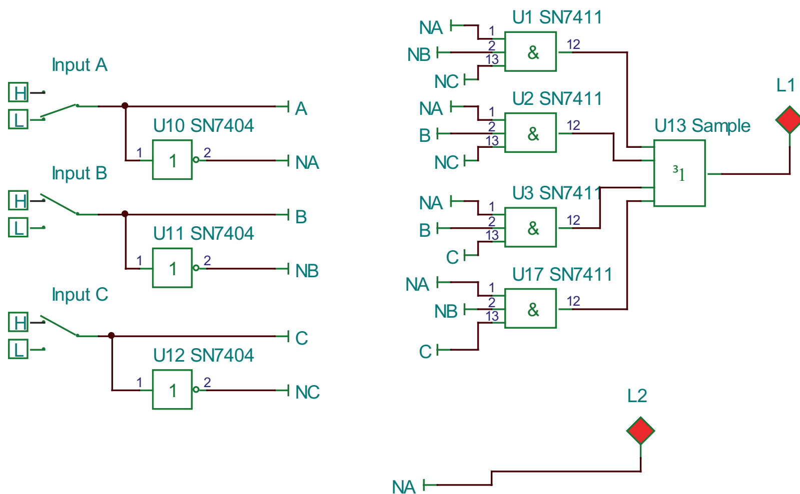
Exercise 3.2

Question 1

$$F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.C + \overline{A}.\overline{B}.C$$

C \ AB	00	01	11	10
0	1	1		
1	1	1		

$$F = \overline{A}$$



Question 2

$$F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C$$

A	0	0	1	1
B	0	1	1	0
C				
0	1			1
1	1			1

$$F = \overline{B}$$

Question 3

$$F = \overline{A}.B.C.D + \overline{A}.B.\overline{C}.D + A.B.D$$

A	0	0	1	1
B	0	1	1	0
D C				
0 0				
0 1				
1 1		1	1	
1 0		1	1	

$$F = B.D$$

Exercise 3.3

1. $F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + \overline{A}.B.C + \overline{A}.\overline{B}.C$

   $F = \sum 000, 010, 001, 011$

   $F = \sum 0, 2, 1, 3$

2. $F = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.\overline{B}.C$

   $F = \sum 000, 100, 001, 101$

   $\sum 0, 4, 1, 5$

3. $F = \overline{A}.B.C.D + \overline{A}.B.\overline{C}.D + A.B.D$

$F = \sum 1110,1010,1111,1011$

$F = \sum 14.10,15,11$

Exercise 3.4

Using the 1st canonical format, we have

$$F = \left(\overline{A}.\overline{B}.C.\overline{D}\right) + \left(\overline{A}.\overline{B}.C.D\right) + \left(A.\overline{B}.\overline{C}.\overline{D}\right) + \left(A.\overline{B}.C.\overline{D}\right) + \left(A.B.\overline{C}.\overline{D}\right) + \left(A.B.C.\overline{D}\right)$$

$$\overline{F} = \sum 0010,0011,1000,1010,1100,1110$$

Using the 2nd canonical format, we have

$$\overline{F} = \left(\overline{A}.\overline{B}.\overline{C}.\overline{D}\right) + \left(\overline{A}.\overline{B}.\overline{C}.D\right) + \left(\overline{A}.B.\overline{C}.D\right) + \left(\overline{A}.B.C.\overline{D}\right) + \left(\overline{A}.B.C.D\right)$$
$$+ \left(A.\overline{B}.\overline{C}.D\right) + \left(A.\overline{B}.C.D\right) + \left(A.B.\overline{C}.D\right) + \left(A.B.C.D\right)$$

$$\overline{F} = \pi(0000,0001,0100,0101,0110,0111,1001,1011,1101,1111)$$

We should simplify using the 1st canonical format, starting with the Karnaugh map:

AB \ DC	00	01	11	10
0 0		**1**	**1**	
0 1				
1 1	**1**	**1**		
1 0	**1**	**1**		

$$F = \left(\overline{A}.\overline{B}.C\right) + \left(A.\overline{D}\right)$$

Chapter 7

Exercise for seven-segment display:

Using the 2nd canonical format, we have

$$\overline{SegA} = \pi\left(0001, 0100, 1011, 1101\right)$$

which means

$$\overline{SegA} = \pi\left(\left(\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0\right) + \left(\overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0}\right) + \left(Q_3.\overline{Q_2}.Q_1.Q_0\right) + \left(Q_3.Q_2.\overline{Q_1}.Q_0\right)\right)$$

Therefore, we have

$$SegA = \left(Q_3 + Q_2 + Q_1 + \overline{Q_0}\right).\left(Q_3 + \overline{Q_2} + Q_1 + Q_0\right).\left(\overline{Q_3} + Q_2 + \overline{Q_1} + \overline{Q_0}\right)$$
$$.\left(\overline{Q_3} + \overline{Q_2} + Q_1 + \overline{Q_0}\right)$$

As there are only four occurrences when there is a logic “0” in the truth table, we should use the 2nd canonical format.

Therefore, we have

$$\overline{SegG} = \pi\left(0000, 0001, 0111, 1100\right)$$

$$\overline{SegG} = \pi\left(\left(\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}\right) + \left(\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0\right) + \left(\overline{Q_3}.Q_2.Q_1.Q_0\right) + \left(Q_3.Q_2.\overline{Q_1}.\overline{Q_0}\right)\right)$$

After inverting each variable and changing the “.” and the “+”, we have

$$SegG = \pi\left(\left(Q_3 + Q_2 + Q_1 + Q_0\right).\left(Q_3 + Q_2 + Q_1 + \overline{Q_0}\right).\left(Q_3 + \overline{Q_2} + \overline{Q_1} + \overline{Q_0}\right).\left(\overline{Q_3} + \overline{Q_2} + Q_1 + Q_0\right)\right)$$

APPENDIX

Exercises

Exercise 1

Use Boolean algebra to simplify the following:

Solution for 1a:

$$F = A.B + A.\overline{B}$$

Take "A" out as a common factor"

$$F = A\left(B + \overline{B}\right)$$

Knowing

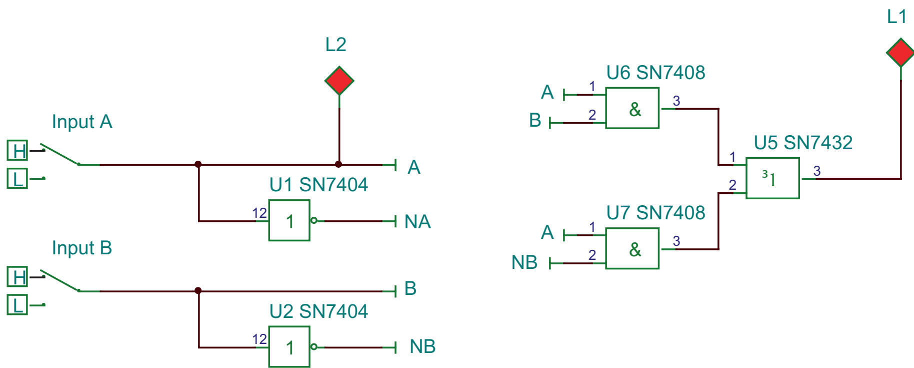
$$B + \overline{B} = 1$$

therefore

$$F = A\left(1\right) = A$$

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7_1

Both circuits for 1a:



Solution for 1b:

$$F = (A.B) + (\bar{A}.B) + (\bar{A}.\bar{B})$$

Take B out as a common factor:

$$F = B.(A + \bar{A}) + (A.B)$$

Knowing $(A + \bar{A}) = 1$

$$F = B + (A.\bar{B})$$

Solution for 1c:

$$(A + B).(B + \bar{C})$$

Expand the brackets:

$$F = (A.B) + (A.\bar{C}) + (B.B) + (B.C)$$

Knowing B.B = B

$$F(A.B) + (A.\bar{C}) + B + (B.C)$$

Now take B out of the last two terms:

$$F = (A.B) + (A.\bar{C}) + B(1 + C)$$

Knowing 1+ C = 1

$$F = (A.B) + (A.\bar{C}) + B$$

Rearrange to

$$F = (A.B) + B + (A.\bar{C})$$

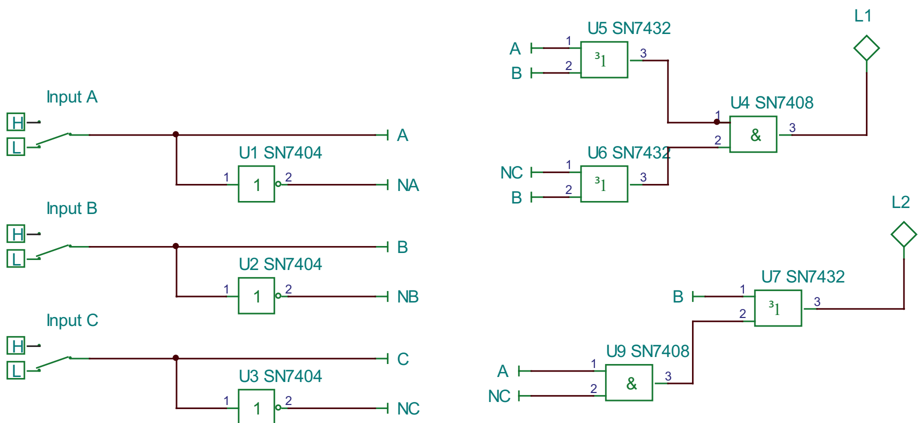
Take B out as a common factor:

$$F = B(1 + A) + (A.\bar{C})$$

Knowing 1 + A = 1

$$F = B + (A.\bar{C})$$

Solution circuit for 1c:



Solution for 1d:

$$F = (A + D) + (A + B + C + D)$$

Expand the brackets, which gives

$$F = (A.A) + (A.B) + (A.C) + (A.D) + (D.A) + (D.B) + (D.C) + (D.D)$$

Knowing A.A = A

$$F = A + (A.B) + (A.C) + (A.D) + (D.A) + (D.B) + (D.C) + D$$

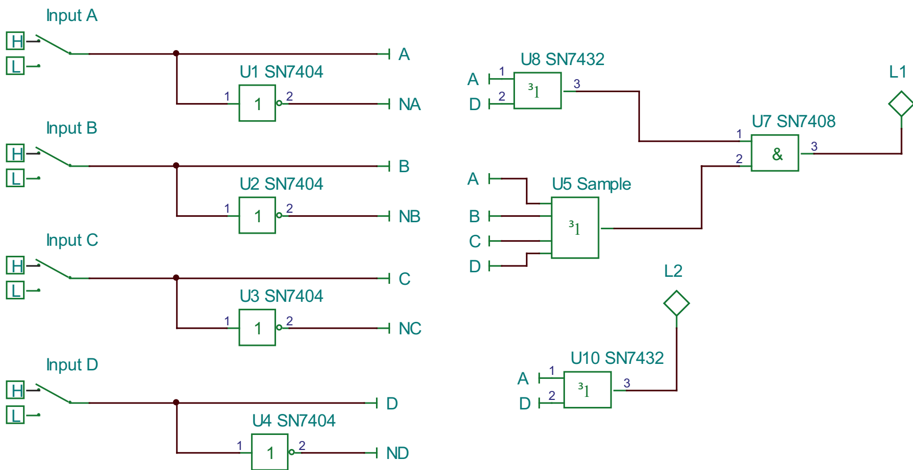
Taking A out as a common factor and D out, we have

$$F = A.(1 + B + C + D) + D.(A + B + C + 1)$$

Knowing 1 + any term = 1

$$F = A + D$$

Solution circuit for 1d:



Exercise 3

Use Karnaugh maps to simplify the following Boolean expressions:

3a

$$F = (\bar{A}.\bar{B}.\bar{C}) + (A.\bar{B}.\bar{C}) + (\bar{A}.\bar{B}.C)$$

3b

$$F = (\bar{A}.\bar{B}.\bar{C}) + (\bar{A}.\bar{B}.C) + (\bar{A}.B.\bar{C}) + (A.\bar{B}.C)$$

3c

$$F = (\bar{A}.\bar{B}.\bar{C}.\bar{D}) + (\bar{A}.\bar{B}.\bar{C}.D) + (\bar{A}.\bar{B}.C.D) + (A.\bar{B}.\bar{C}.\bar{D}) + (A.\bar{B}.C.D) + (A.\bar{B}.\bar{C}.D)$$

3d

$$F = (\bar{A}.B.\bar{C}.\bar{D}) + (A.B.\bar{C}.\bar{D}) + (\bar{A}.B.C.\bar{D}) + (A.B.C.\bar{D}) + (\bar{A}.B.C.D) + (A.B.C.D) + (\bar{A}.B.\bar{C}.D) + (A.B.\bar{C}.D) + (A.B.\bar{D})$$

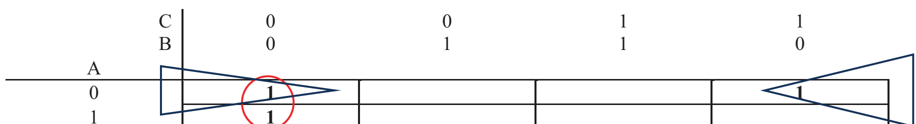
3e

$$F = (\bar{A}.\bar{B}.\bar{C}.\bar{D}) + (\bar{A}.\bar{B}.\bar{C}.D) + (\bar{A}.B.\bar{C})$$

Solution for 3a:

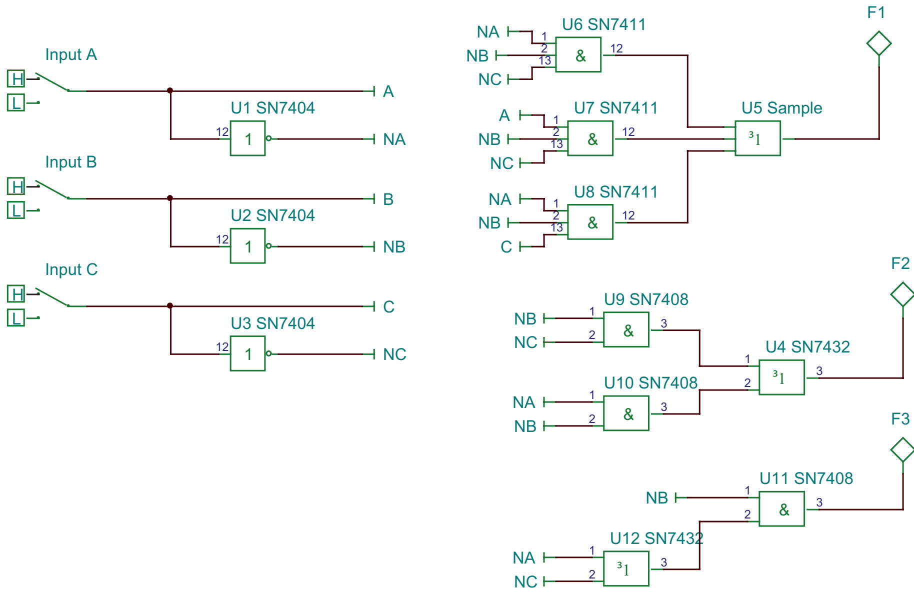
$$F = (\bar{A}.\bar{B}.\bar{C}) + (A.\bar{B}.\bar{C}) + (\bar{A}.\bar{B}.C)$$

$$F = \sum 000,100,001$$



$$F = \left(\overline{B}.\overline{C}\right) + \left(\overline{A}.\overline{B}\right)$$

$$F = \overline{B}.\left(\overline{C} + \overline{A}\right)$$



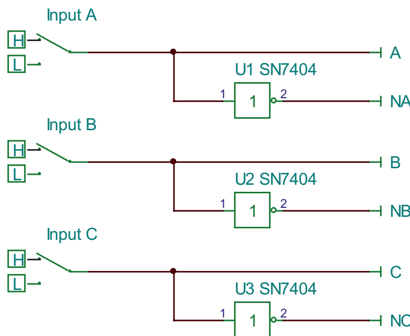
Solution for 3b:

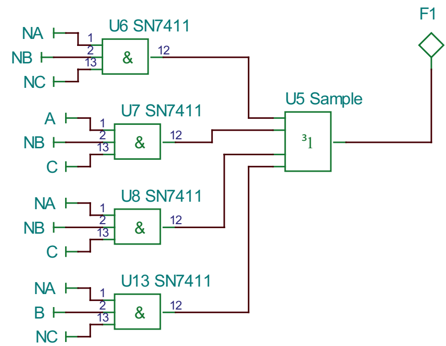
$$F = \left(\overline{A}.\overline{B}.\overline{C}\right) + \left(\overline{A}.\overline{B}.C\right) + \left(\overline{A}.B.\overline{C}\right) + \left(A.\overline{B}.C\right)$$

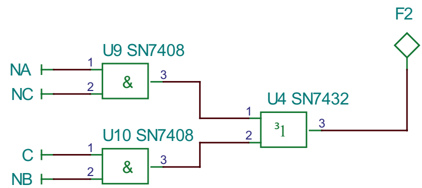
$$F = \sum 000, 001, 010, 101$$

A \ CB	00	01	11	10
0	**1**	**1**		**1**
1				**1**

$$F = \left(\overline{A}.\overline{C}\right) + \left(\overline{B}.C\right)$$



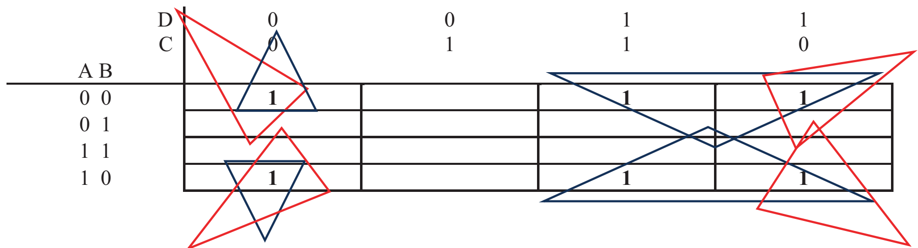




Solution for 3c:

$$F = \left(\bar{A}.\bar{B}.\bar{C}.\bar{D}\right) + \left(\bar{A}.\bar{B}.\bar{C}.D\right) + \left(\bar{A}.\bar{B}.C.D\right) + \left(A.\bar{B}.\bar{C}.\bar{D}\right) + \left(A.\bar{B}.C.D\right) + \left(A.\bar{B}.\bar{C}.D\right)$$

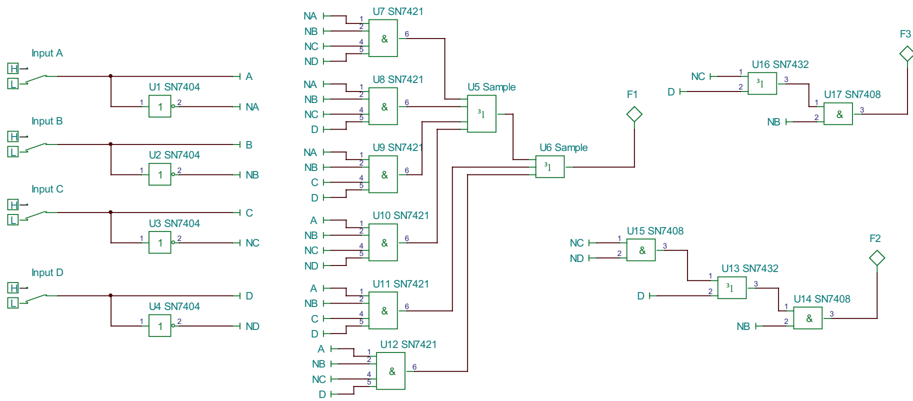
$$F = \sum 0000, 0001, 0011, 1000, 1011, 1001$$



$$F2 = \left(\bar{B}.\bar{C}.\bar{D}\right) + \left(\bar{B}.D\right)$$

$$F2 = \bar{B}.\left(\left(\bar{C}.\bar{D}\right) + D\right)$$

$$F3 = \left(\bar{B}.\bar{C}\right) + \left(\bar{B}.D\right) = \bar{B}.\left(\bar{C} + D\right)$$



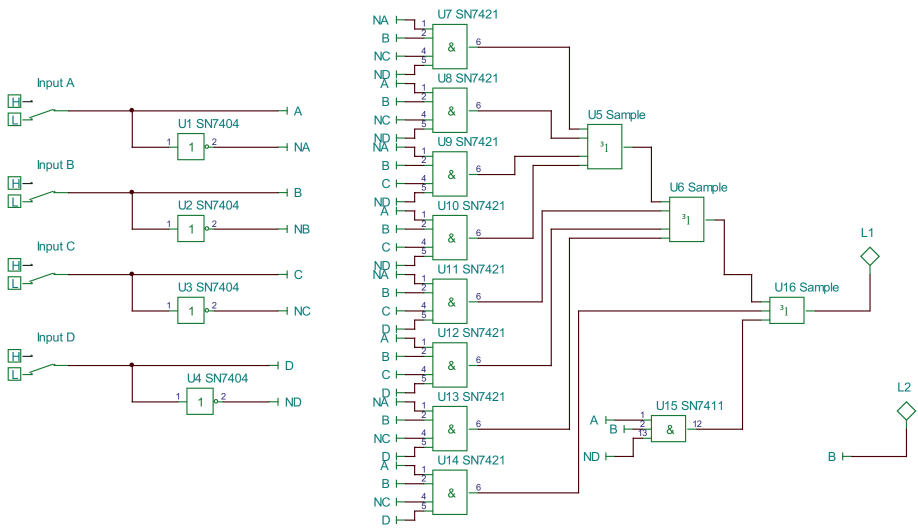
Solution for 3d:

$$F = \left(\bar{A}.B.\bar{C}.\bar{D}\right) + \left(A.B.\bar{C}.\bar{D}\right) + \left(\bar{A}.B.C.\bar{D}\right) + \left(A.B.C.\bar{D}\right) + \left(\bar{A}.B.C.D\right)$$
$$+ \left(A.B.C.D\right) + \left(\bar{A}.B.\bar{C}.D\right) + \left(A.B.\bar{C}.D\right) + \left(A.B.\bar{D}\right)$$

$$F = \sum 0100, 1100, 0110, 1110, 0111, 1111, 0101, 1101, 1110, 1100$$

D C A B	0 0	0 1	1 1	1 0
0 0				
0 1	**1**	**1**	**1**	**1**
1 1	**11**	**11**	**1**	**1**
1 0				

$$F = B$$



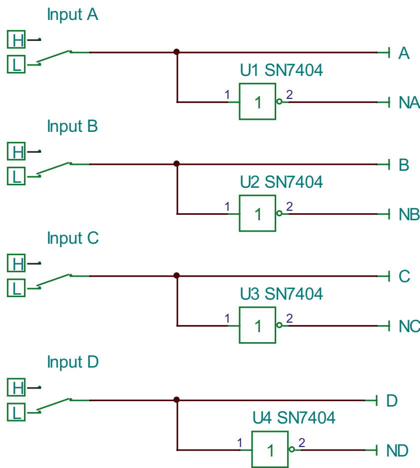
Solution for 3e:

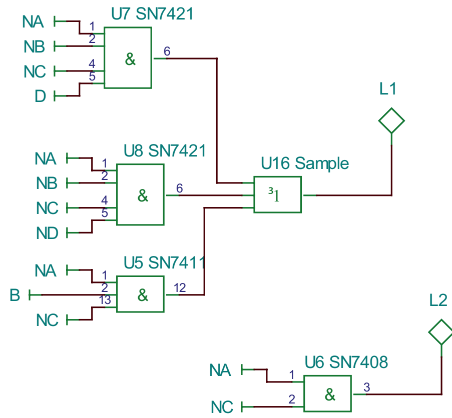
$$F = \left(\bar{A}.\bar{B}.\bar{C}.\bar{D}\right) + \left(\bar{A}.\bar{B}.\bar{C}.D\right) + \left(\bar{A}.B.\bar{C}\right)$$

$$F = \sum 0000, 0001, 010X$$

AB \ DC	00	01	11	10
0 0	**1**			**1**
0 1	**1**			**1**
1 1				
1 0				

$$F = \bar{A}.\bar{C}$$





Exercise 5

Q5.1

$$F = \overline{\left[\left(A+B\right).C\right].\left(\bar{B}.C\right).\left[\bar{A}+\left(B.C\right)\right]}$$

Q5.2

$$F = \overline{\left[\left(A+B\right).\bar{C}\right]+\left(\bar{A}.C\right)+\left(\bar{A}.\overline{B.C}\right)}$$

Index

A

AND function, 16, 28, 39–45, 75–77, 79, 83, 85, 281, 285, 396
AND gate, 11, 18, 28–30, 40, 41, 47, 48, 50, 53, 54, 56, 127, 140, 205, 211, 217, 248, 250, 285, 299–301, 333, 349, 351, 357–359, 370, 371, 373, 391, 402
Asynchronous system, 179, 180

B

Binary-Coded Decimal (BCD), 200, 383, 402
Binary number system, 21
  adding/subtracting numbers, 25, 26
  binary to decimal, converting, 23–25
  decimal to binary, converting, 22, 23
  representing logic, 21
  subtracting binary numbers, 26–28
Boolean algebra, 36
  AND/NAND gates, 39–41
  build logic circuit, example, 54–58
  circuit 1, 47–49
  circuit 2, 50–52
  circuit 3, 52
  definition, 37
  digital gates/circuits, 38
  EXOR gates, 44–46
  expressions, 46
  expressions exercise, 59, 60
  laws
    absorption, 67–69
    associative, 61–63
    commutative, 60, 61
    distributive, 64–66
  logic gates/logic circuits, 37
  NOT gate, 38
  OR/NOR gates, 42–44
Boolean expressions
  idempotent law, 85
  identity law, 83, 84
  inverse law, 81–83
  minimize expressions, 90
  null law, 84, 85
  OR version
    idempotent law, 86
    identity law, 87
    inverse law, 88, 89
    null law, 87, 88
  simplification example, 90–99

© Hubert Henry Ward 2024
H. H. Ward, Maker Innovations Series, *Mastering Digital Electronics*,
https://doi.org/10.1007/978-1-4842-9878-7

C

Combinational logic circuits, 271
- anode seven-segment display, 315
- binary subtractor circuit, 288–291, 293, 294
- cathode seven-segment display, 316–322
- demultiplexer, 302, 303
- digital decoder, 308–312
- digital encoders, 303–305
- digital encoders, application, 306, 308
- 4-bit multiplexer, 298–300, 302
- full adder, 278–280, 282–286
- half adder, 276–278
- seven-segment display, 312–314, 323
- subtract by adding, 294–297
- 3-bit full adder, 287
- tri-state buffer, 273–275

D

De Morgan's theory, 69, 74
- Boolean expression, 70
- example, 70–74, 76
- OR function, NAND gates, 77–79

Denary number system, 21

Digital circuits, 324
- asynchronous/synchronous system, 179
- combinational logic, 177
- design example
  - modulo 6 binary counter, 213, 214
  - modulo 10 counter, 181, 183
  - non-sequencial output, 184–186
  - synchronized sequencial output, 187–191, 193–197, 199
  - synchronized up counter, 200, 201, 203–208, 210, 212
- digital system, 178, 179
- D-type latches
  - D_0 Inputs, 215–218
  - D_1 Inputs, 218, 219
  - D_2 Inputs, 220
- JK flip flop, 222
- ripple counter, 180
- sequential logic, 178

Diode-Resistor (DR), 1

Diode-Resistor (DR) logic, 7
- basic symbol, 6
- circuit 1, 7–10, 12, 13
- circuit 2, 10, 12
- circuit 3, 14–16, 18

Distributive law, 64

D-type latch, 139, 326

E

Electronic Computer-Aided Design (ECAD) software, 1, 6, 20, 135, 136, 138, 155, 158, 392, 399, 408, 436–438

EXOR gate, 34, 35, 44–46, 277, 281–285

F, G

Field-effect transistors (FETs), 2, 19, 20, 28
50/50 duty cycle square wave, 424–427
555 timer
  astable circuit, 418, 420, 422–424
  50/50 duty cycle square wave, 424–427
  formats, 409
  IC, 409, 410
  monostable, 412–415, 417
  1Hz square wave, 428, 429
  pins, 410, 411
  PWM application
    frequency, 432
    output voltage, 436
    periodic time, 433, 434
    square wave, 430
    Vavge, 431
"4000" series, 19–20

H

Hexadecimal number system, 232, 314

I

Inclusive OR gate, 43, 285
Integrated circuit (IC), 1, 157, 409
Inverse Law, 81–83, 88–89, 94, 95, 97

J

JK flip flop, 138, 139, 154–176, 184, 198, 200, 202, 213, 222, 224–232, 246, 269, 325, 326, 352, 395–399

K

Karnaugh maps
  definition, 100
  example, 100, 101, 106–110, 133, 135, 136
  logic, 102, 103
  minterms/maxterms, 112
  2nd canonical format, 113–120
  simplification examples, 110, 111, 121–131

L

Logic circuits
  counters, 395, 397, 399
  egg timer, 378, 379
  example
    alternative single set, traffic lights, 360–363, 365
    output logic, 357, 358, 360
    pelican crossing, 366, 367, 369–378
    traffic light design process, 347–349, 351–355, 357
  practical ICs, 392
  7400 NAND gate, 392–395

Logic circuits (*cont.*)
7493 binary counter, 399, 400, 402
SN74168, 379, 380, 382–391
SN74194 multifunction shift register, 403, 406, 407
Logic gates
classification, 2
CMOS, 4
families, 3
fan-out/fan-in, 3, 4
"4000" series, 19, 20
IC, 1
laws, 5
TTL *vs*. CMOS, 2, 3
unused inputs, 4

M

Microprocessor-based systems, 26
Moore's and Mealy representations, 269

N

NAND function, 16, 39–42, 281
NAND gate, 30
clocked
master-slave clocked SR, 153–157
operation, 149–151
truth table, 152
de-bounce circuit, 144, 145
D-type latch, 170, 171
indeterminate condition, 148, 149
JK flip flop
cacade, 169
circuit working, 162
logic, 163, 164
oscilloscope, TINA, 167
test circuit, 166
test simulation, 168
timing waveforms, 170
transition table, 174, 175
truth table, 165, 173
logic circuits, 176
NOR gates, SR latch, 145–147
SR latch, 139–144
T latch, 172
NOR gate, 32, 33, 42–44, 53, 79, 145–148, 175, 357–359, 376, 395
NOT function, 13, 20
NOT gate/inverter gate, 13, 35, 36, 38–39, 48, 71, 76, 77, 127, 155, 156, 186, 282, 302, 322, 333, 386

O

OR function, 11, 12, 43–45, 77–79, 105, 127, 281, 285
OR gate, 31–32, 43, 44, 47, 48, 50, 54, 57, 58, 71, 127, 215, 217, 248, 250, 285, 299, 322, 356–359, 369, 371, 374

P, Q

Parallel In Parallel Out (PIPO), 332, 346
Parallel In Serial Out (PISO), 324, 330–332, 403, 405
Programmable Logic Array (PLA), 20

R

Relay Logic, 5

S

Serial In Parallel Out (SIPO), 324, 332–333
Serial In Serial Out (SISO), 329–330
Shift registers
  D-type latch, 325, 326, 328, 329
  4-bit/SISO, 329, 330
  frequency divider, 336, 337, 346
  Johnson ring counter, 335
  Johnson ring counter, divide by 4, 338, 339
  phase shift, latches, 339–341, 343–345
  PIPO, 332
  PISO, 330–332
  ring counter, 334, 335
  SIPO, 332, 333
State diagrams
  binary counter, 242, 244, 245
  bit stream monitor, 252–254, 256, 258
  D_0 expression, 264–267
  D_0 inputs, 236, 237, 258, 259
  D_1 expression, 259–261
  D_1 input, 237, 238
  D_2 input, 238–241
  D-type latches, 235, 247–249, 251, 252
  example, 262–264
  exercise, 267, 268
  internal logic elements, 223
  JK flip flop, 224, 225
  JK flip flop state table, 226–231
  Moore's/Mealy representations, 269
  sequencial digital logic circuits, 231
  state table, 245, 246
  synchronized binary counter, 232, 234, 235

T

TINA 12
  binary counter, 447–449
  ECAD, 437, 438
  first test circuit, 440–446
  "jumper" terminals, 449–451
  quad NAND 7400 macro, 457, 458
  running software, 438, 439
  7400 IC, quad-two-I/P NAND gate, 451–456
T latch, 139, 172, 175

Transistor-Transistor Logic (TTL), 1–5, 7, 9, 17, 20, 21, 28, 30, 36, 303
Tri-state buffer, 273–275, 331, 333, 386, 390

U, V, W, X, Y, Z

Universal Asynchronous Receive and Transmit IC (UART), 330