Global Text

Introduction to Digital Logic with Laboratory Exercises

Introduction to Digital Logic with Laboratory Exercises

James Feher

Editor-In-Chief: James Feher
Associate Editor: Marisa Drexel
Proofreaders: Jackie Sharman, Rachel Pugliese

For any questions about this text, please email: drexel@uga.edu

The Global Text Project is funded by the Jacobs Foundation, Zurich, Switzerland

Table of Contents

About the author and reviewers

Author: James Feher

Jim currently teaches Computer Science at McKendree University in Lebanon, Illinois, USA. He is a huge open-source software proponent. His research focuses on the use of open source software in the areas of hardware, programming and networking. His hobbies include triathlon, hiking, camping and the use of alternative energy. He lives with his wife and three kids in St. Louis, Missouri where he built and continues to perfect a solar hot water heating system for his home.

Reviewer: Andrew Van Camp

Professor Van Camp is a retired electronics professor. In addition, he has extensive experience working and consulting in industry. He currently resides in central Missouri where he continues his consulting for industry.

Reviewer: Kumud Bhandari

Kumud graduated from McKendree University with degrees in computer science and mathematics. He has interned at the University of Texas and the Massachusetts Institute of Technology. He currently is employed as a researcher with Argonne National Laboratory.

Preface

This lab manual provides an introduction to digital logic, starting with simple gates and building up to state machines. Students should have a solid understanding of algebra as well as a rudimentary understanding of basic electricity including voltage, current, resistance, capacitance, inductance and how they relate to direct current circuits. Labs will be built utilizing the following hardware:

- breadboards with associated items required such as wire, wire strippers and cutters
- some basic discrete components such as transistors, resistors and capacitors
- basic 7400 series logic chips
- 555 timer

Discrete components will be included only when necessary, with most of the labs using the standard 7400 series logic chips. These items are commonly available and can be obtained relatively inexpensively. Labs will include learning objectives, relevant theory, review problems, and suggested procedure. In addition to the labs, several appendices of background material are provided.

Format for each chapter

Each chapter is a combination of theory followed by review exercises to be completed as traditional homework assignments. Full solutions to all of the review exercises are available in the last appendix. Procedures for labs then follow that allow the student to implement the concepts in a hands on manner. The materials required for the labs were selected due to their ready availability at modest cost. While students would gain from just reading and completing the review exercises, it is recommended that the procedures be completed as well. In addition to providing another means re-enforcing the material, it helps to develop real world debugging and design skills.

This manual concentrates on the basic building blocks of digital electronics: logic gates and memory. It focuses on these items from the ground up. The reader will first see how logic gates can be constructed from transistors and then how digital logic functions are constructed using those gates. The concept of memory is then introduced through the construction of an SR latch and then a D flip-flop. A clock is created to be used in a basic state machine design that aims to combine logic circuits with memory.

Target audience

This text will be geared toward computer science students; however it would be appropriate for any students who have the necessary background in algebra and elementary DC electronics. Computer science students learn skills in analysis, design and debugging. These skills are also used in the *virtual* world of programming, where no physical devices are ever involved. By requiring the assembly and demonstration of actual circuits, students will not only learn about digital logic, but about the intricacies and difficulties that arise when physically implementing their designs as well.

Global Text Project

***Education is the most powerful weapon you can use to change the world* - Nelson Mandela**

The goal of this text is to allow more students to gain access to this material by publishing it in the Creative Commons as well as specifying inexpensive materials to be used in the labs. For this reason the author chose to work with the Global Text project to develop this text. The Global Text Project will create open content electronic textbooks that will be freely available from a website. Distribution will also be possible via paper, CD, or DVD. The

goal of the Global Text Project initially is to focus on content development and Web distribution, and work with relevant authorities to facilitate dissemination by other means when bandwidth is unavailable or inadequate. The goal is to make textbooks available to the many who cannot afford them.

Acknowledgments

A work such as this would not be possible without the help of many. First, I would like to thank the Global Text Project for their vision of providing electronic textbooks for free to everyone. Marisa Drexel, Associate Editor at the Global Text Project provided countless suggestions and helpful hints for the document and for the creation of the document using OpenOffice. Andrew Van Camp II, retired professor of electronics provided excellent suggestions for technical review of the content. Kumud Bhandari, currently a research aide at Argonne National Laboratory, also provided technical review of the material. My students Evan VanScoyk, Samantha Barnes, and Ben York all provided helpful corrections and review as well as countless diagrams found in the document. I would like to thank all of the countless open-source developers who produced such fine software as GNU/Linux, OpenOffice, Gimp, and Dia which were all used to create this document. I am grateful to McKendree University for providing support in the form of a sabbatical to allow me to complete this work. And I certainly wish to thank Sandy who provided excellent review suggestions, support and an extremely patient ear when I ran into trouble trying to incorporate a new feature from OpenOffice or attempted to edit a particularly tricky graphic.

0. Introduction

It is nearly impossible to find a part of society that has not been touched by digital electronics. Obvious applications such as computers, televisions, digital video reorders and countless other consumer electronics would not be possible without them. The Internet is run on a system of computers and routing equipment built with digital electronics. Yet even outside of some of these obvious applications we find that our cars and utilitarian home appliances such as microwaves, washers, dryers, coffee makers and even refrigerators are all increasingly being designed with digital electronic controls. You likely carry some sort of device designed with them with you nearly all your waking hours whether it is a watch, cell phone, MP3 player or PDA. Indeed, digital electronics provide the foundation upon which we build the infrastructure of modern society.

You no doubt have heard stories about some of the first computers. Machines built with mechanical relays and vacuum tubes that filled entire rooms. In the 1940s John Bardeen, Walter Brattain and William Shockley developed the first transistor; it allowed computers to be built cheaper, smaller and more reliable than ever before. The integrated circuit, a single package with several transistors along with other circuit components, was developed in the late 1950s by Jack Kilby at Texas Instruments. This helped to further advance the digital revolution. Advances then became so common that in the 1960s Gordon Moore, co-founder of Intel Corporation, proposed his famous law stating that the capacity of computers we use would double every two years. This observation has held up since then, even being amended to doubling every eighteen months.

The quad core microprocessors of today contain millions of components, but the basic building blocks are digital logic functions combined with memory. Despite the fact that many of these devices are tremendously complex and require vast amounts of engineering in their design, they all share the ubiquitous bit as their fundamental unit of data. In essence it all starts with TRUE and FALSE or 0 and 1. And so the next chapter starts with the simplest of logic devices, the inverter, built with a single transistor. You then continue your journey into the world of digital electronics by examining the NAND and NOR gates. Remember, the digital revolution would not be possible without these simple devices.

1. The transistor and inverter

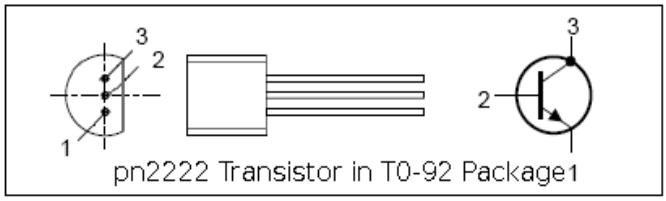
Learning objectives:

- Use the digital trainer and breadboard.
- Assemble a circuit.
- Build a logic circuit with discrete components.

The transistor

A transistor is a three-terminal device that can be used as an amplifier or as a switch. When the transistor is used as an amplifier, it is working in analog mode. When it is being used as an electronic switch, it is functioning in digital mode. The transistor will only be used in digital mode in these labs, which means the transistor will either be on or off. The terms ground, low, zero, zero volts, open switch, and dark lamp are all equivalent to the boolean value false. Likewise five volts, high, one, closed switch, and lit lamp (light-emitting diode, LED), are equivalent to the boolean value true. We will use false (F or 0) and true (T or 1) when speaking of the logical states in this manual. Modern computers contain millions of transistors combined together in digital mode to create advanced circuits.

PINS	COMPONENT
1	Emitter
2	Base
3	Collector



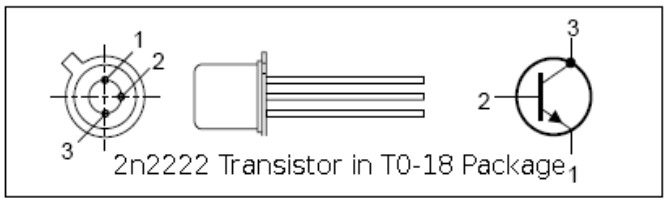


Exhibit 1.1: Common NPN transistors

Transistors are three pin devices that are similar to valves for controlling electricity. The amount of current that can flow between the collector and emitter is a function of the current flowing through the base of the transistor. If no current is flowing through the base of the transistor, no current will flow through the collector and emitter. With the transistor operating in digital mode, it will be configured to carry the maximum (if on) or minimum (if off) amount of current from the collector to the emitter that the circuit will allow.

The transistor used in this lab, the pn2222 or 2n2222, is an NPN, bipolar junction transistor which is sometimes referred to as a BJT. Other types of transistors exist, and while they differ in how they function, they are used in a similar manner in digital circuits. In this lab, a single transistor will be used to create an inverter. The principles used to build this inverter could be applied to other circuits with other types of transistors. Pinouts of the two types of transistors most likely to be used in these labs are shown in Exhibit 1.1.

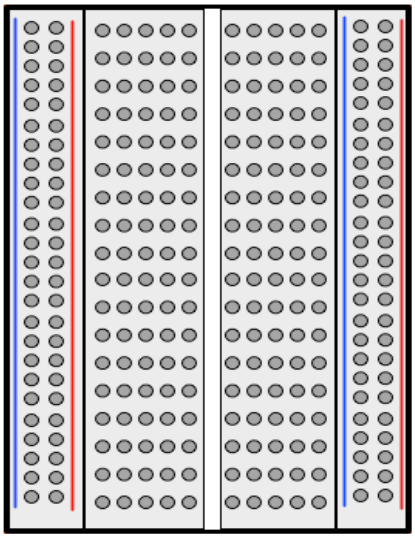


Exhibit 1.2: Breadboard

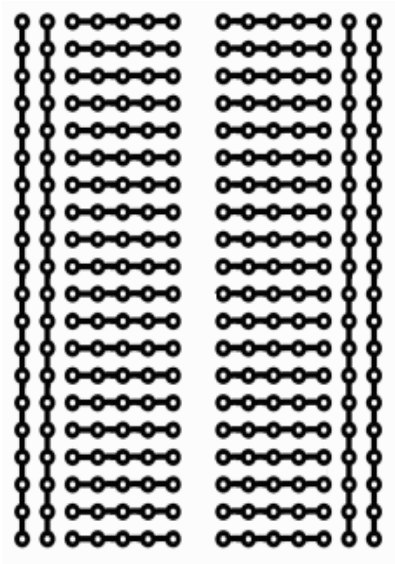


Exhibit 1.3: Common connections

The breadboard

In order to build the circuit, a digital design kit that contains a power supply, switches for input, light emitting diodes (LEDs), and a breadboard will be used. Make sure to follow your instructor's safety instructions when assembling, debugging, and observing your circuit. You may also need other items for your lab such as: logic chips, wire, wire cutters, a transistor, etc. Exhibit 1.2 shows a common breadboard, while Exhibit 1.3 shows how each set of pins are tied together electronically. Exhibit 1.4 shows a fairly complex circuit built on a breadboard. For these labs, the highest voltage used in your designs will be five volts or +5V and the lowest will be 0V or ground.

A few words of caution regarding the use of the breadboard:

- Keep the power off when wiring the circuit.
- Make sure to keep things neat, as you can tell from Exhibit 1.4, it is easy for designs to get complex and as a result become difficult to debug.
- Do not strip more insulation off of the wires used than is necessary. This can cause wires that are logically at different levels to accidentally touch each other. This creates a short circuit.
- Do not push the wires too far into each hole in the breadboard as this can cause two different problems.
  - The wire can be pushed so far that only the insulation of the wire comes into contact with the breadboard, causing an open circuit.
  - Too much wire is pushed into the hole; it curls under and ends up touching another component at a different logical level. This causes a short circuit.
- Use the longer outer rows for +5V on one side and ground on the other side.
- Wire power to the circuit first using a common color (say red) for +5V and another (black) for ground.
- Always make sure to have a clearly documented circuit diagram before you start wiring the circuit.

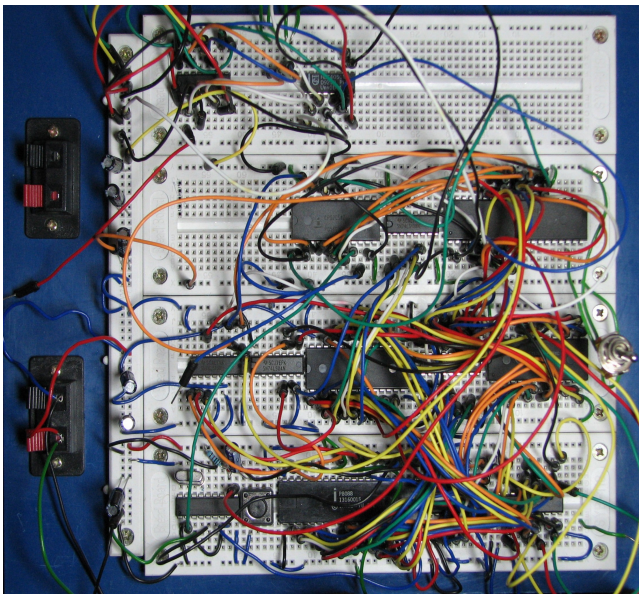


Exhibit 1.4: Complex circuit

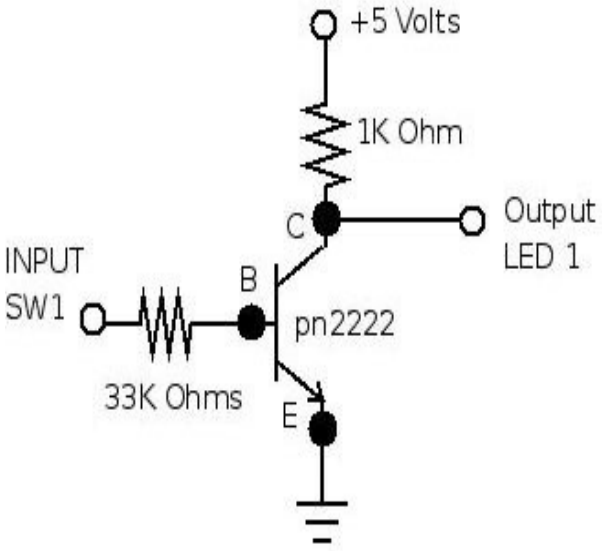


Exhibit 1.5: Inverter circuit

The inverter

The inverter, sometimes referred to as a NOT gate, is a simple digital circuit requiring one transistor and two resistors. The circuit should be connected as in Exhibit 1.5. Make sure to start with a neat diagram in your lab notebook before you start constructing your circuit! The input is connected to a switch and the output connected to an LED. The two resistors are current limiting resistors and are sized to insure that the circuit operates in digital mode. If the inverter circuit is altered slightly with the addition of another transistor placed in series with the current one, it results in one more input and the creation of a NAND gate. Likewise, if another transistor is added in parallel with the transistor in the inverter circuit a NOR gate can be built. These two gates are discussed at greater length in the next chapter.

Review exercises

1. Sketch your breadboard. Make sure to indicate which portions of the board are electrically connected in common.
2. Construct a truth table for an inverter with **x** being the input and **!x** being the output.
3. Using the color codes, determine the value of each of the resistors. Hint: You may need to review Appendix B if you are unfamiliar with using resistors.
   (a) red, orange, red
   (b) brown, black, orange
   (c) orange, orange, orange
   (d) brown, black, green
4. What is the symbol used for electrical ground or zero volts?
5. Construct a truth table for a NAND gate.
6. Construct a truth table for a NOR gate.

1. The transistor and inverter

Procedure

1. Write the prelab in your lab notebook for all the circuits required in the steps that follow.
2. Obtain instructor approval for your prelab.
3. Draw a diagram of the inverter circuit.
4. With the power off on your digital trainer, construct your inverter. Upon completion of the circuit, you may wish to have your instructor examine it before turning the power on.
5. Turn power on for your circuit and verify the proper operation of the inverter.
6. Demonstrate the proper operation of the inverter for your instructor.
7. Using a 7404 series logic chip, connect one of the inverters to demonstrate its operation. Note that Appendix A contains descriptions of the 7400 series chips used in the labs, including the 7404 inverter chip.

Optional exercises

1. Draw a diagram of a NAND inverter circuit using two NPN transistors.
2. Construct the NAND circuit.
3. Verify proper operation of the NAND gate.
4. Demonstrate the proper operation of the NAND for your instructor.

2. Logic gates

Learning objectives:

- Use 7400 series chips in designing digital logic functions.
- Draw complete circuit diagrams.
- Construct and debug digital logic circuits using 7400 series chips.

History of logic chips

Logic gates could be constructed from transistors and resistors just as the inverter was constructed in the last lab. However, using discrete transistors to build logic gates can be time consuming and prone to problems as increasing the number of connections also increases the possible points of failure. Before the advent of the transistor, and today in certain industrial applications, logic gates are created using mechanical relays. Mechanical devices suffer from similar problems along with the added complication that such devices generally cannot be switched from one state to another quickly enough for modern computer applications. The introduction of the integrated circuit in the late 1950s aimed at placing many individual circuit components in a single package that had all of the connections self-contained in silicon. This revolutionized the computing industry and has led to CPUs today that contain millions of components in a single chip.

You will use 7400 series logic chips in this manual. This series of chips has been manufactured since the 1960s. These chips were used to design and build computers during that time; however, they are rarely used in computers built today. Despite this, they still have many uses (in addition to just teaching students digital logic). They are still produced, easy to obtain and are fairly inexpensive. The chips come in various packages, but the package used in these labs is a dual in-line package, otherwise know as a DIP as shown in Exhibit 2.1. In order to determine the polarity of the chip, a notch is put on one side of the chip. From a top view, pin one is on the left of the notch with other pins numbered sequentially in a counter clockwise manner. Chips may also have a dot placed near pin one. Pinouts of the chips that will be used in the labs can be found in Appendix A.

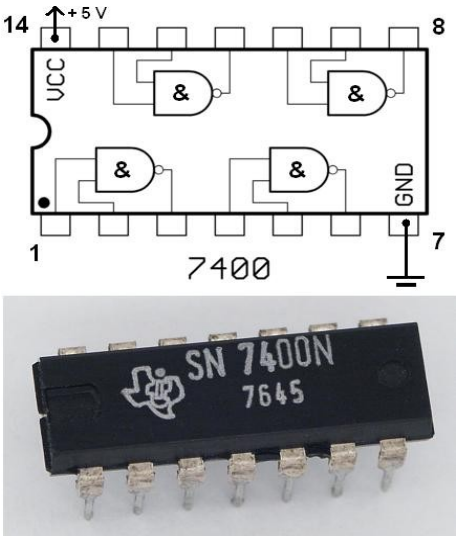


Exhibit 2.1: 7400 NAND DIP

Chips in the 7400 family are constructed using a variety of different circuit configurations that all have different properties. Some utilize BJT and others, field effect transistors (FETs). The different series (C, HC, L, S, LS, etc. within the 7400 family) are designed with such considerations as the need for low power consumption, switching speed, or reliability under stressful environments that might be incurred in military applications. Consult Appendix E for families that are appropriate for these labs.

Logic symbols

As mentioned in the previous lab, NAND and NOR gates can be constructed with fewer components than AND and OR gates. For this reason, the inverter, NAND and NOR make up four of the seven chips used in all of the labs. Symbols used to represent the NAND, NOR, AND, OR and inverter or NOT are provided along with the truth tables for the NAND and NOR. The truth tables have "0" representing false and "1" representing true. A circuit that can be used to create a NAND gate using two transistors is shown in Exhibit 2.7. Circuit configurations for NAND gates provided by the 7400 series chips, while logically equivalent, vary from this design.

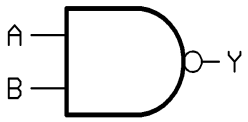


Exhibit 2.2: NAND

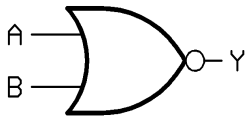


Exhibit 2.3: NOR

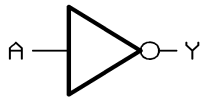


Exhibit 2.4: Inverter

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Table 1: NAND table

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Table 2: NOR table

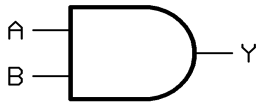


Exhibit 2.5: AND

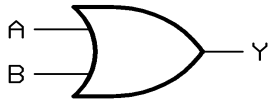


Exhibit 2.6: OR

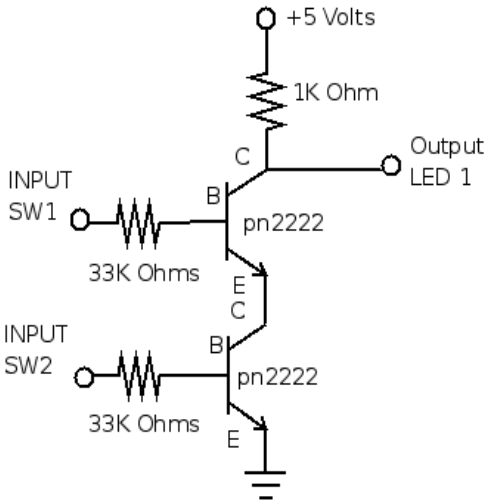


Exhibit 2.7: NAND circuit

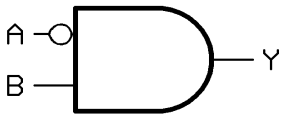


Exhibit 2.8: A' AND B

Notice that only the small circle is used to indicate the inversion of the AND to produce the NAND instead of using the full inverter symbol in Exhibit 2.2. This shorthand is often used at the input of a gate, shown in Exhibit 2.8 which is equivalent to (A' AND B).

Since the NAND gate is used more often, how do you obtain a simple AND or OR gate? One way would obviously be to simply combine a NAND gate along with an inverter as in Exhibit 2.9. While this works, as each chip contains more than one gate, if an extra NAND is available, it may be more advantageous to use a spare gate rather than to use an entirely new chip as in Exhibit 2.10.

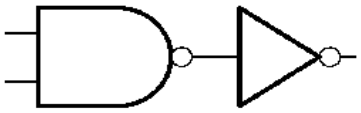


Exhibit 2.9: NAND inverter yields AND

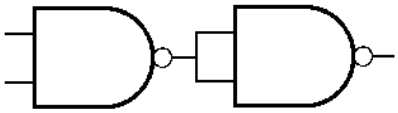


Exhibit 2.10: NAND NAND to yield AND

Logical functions

Exhibit 2.11 demonstrates how to implement a simple logical expression using the gates provided. Make sure to use only those gates that are provided in your kit when designing your circuit. This diagram implements the function f(A,B,C) = AB + BC. Since there are three inputs to this function, there are eight possible logical input conditions as shown in the truth table.

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

Table 3: AB + BC

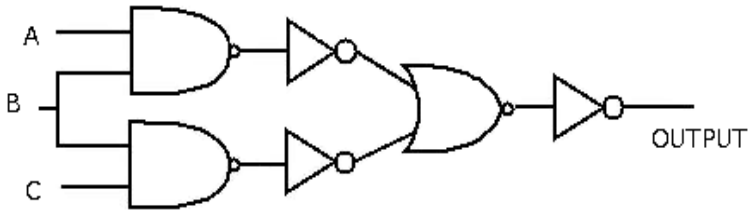


Exhibit 2.11: AB + BC

When building a logical circuit, it is important to document the circuit diagram as shown above. However, even this diagram could be made clearer for those attempting to build and debug the circuit. Exhibit 2.12 yields a much more detailed description of how the circuit should be built.

You should include a diagram for every circuit that you build in your lab notebook and you should follow the format in Exhibit 2.12. Let us examine the type of information contained here. First, chips are labeled as IC1, IC2 and IC3. Then a legend is included that specifies the type of chip for each of the IC or integrated circuits. The IC numbers should appear in the order that they will appear in your breadboard from left to right or top to bottom, depending upon how the breadboard is configured in your digital trainer. Second, the pins used for each connection on the chip are also given, which makes connecting the circuit possible without having to continually consult the datasheet for that logic chip. Third, the switches and LEDs are labeled in the order that they are used for the respective inputs and outputs. All of this

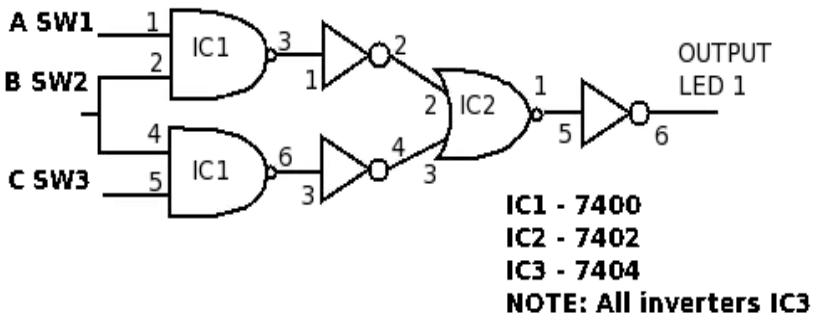


Exhibit 2.12: Detailed wiring diagram for AB+BC

makes it much easier to construct and demonstrate the circuit. But above all, the greatest benefit comes if the circuit does not work and needs to be debugged! In this case, with all of the pins clearly labeled on your diagram, it is much easier for someone to examine your circuit, compare it to your diagram, trace the various connections and hopefully find and correct any problems in the circuit.

> LAB NOTEBOOK TIP: In addition to the circuit diagram, always put a truth table in your lab notebook to make it easier to debug and test the operation of your circuit.

This circuit would require three different 7400 series logic chips and ten different connections, yet if designed with individual transistors using the inverter from the last lab, as well as the NAND circuit shown in Exhibit 2.7, this would take nine different transistors, fifteen resistors, and many more connections than if just the chips were used. It is no wonder that the decrease in complexity of digital circuits that followed the introduction of the 7400 series chips led to a revolution in the computing industry!

Let us examine one more simple circuit. This one is used to implement an exclusive or (XOR), which is represented by the symbol ⊕ in logical expressions. The truth table for A XOR B follows along with the gate used to represent it in circuit diagrams. As no XOR chip is provided in the kit, in order to implement this circuit, the XOR must be built by examining the truth table to find the resulting logical function, A'B + AB'. The circuit diagram for the XOR is shown in Exhibit 2.14. Remember, a diagram such as this should be included in your lab manual to ease construction and debugging of the circuit.

A	B	⊕
0	0	0
0	1	1
1	0	1
1	1	0

Table 4: XOR table

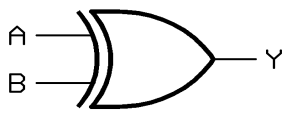


Exhibit 2.13: XOR

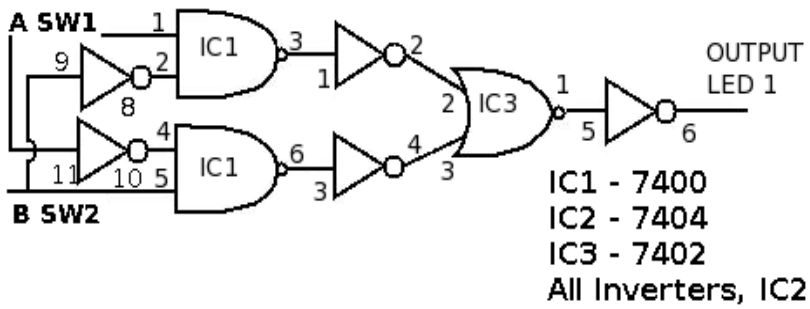


Exhibit 2.14: Circuit diagram for XOR

We will discuss how to build more complicated circuits in the next chapter, as well as how to logically simplify the functions with Boolean algebra. Both circuits designed in this chapter can be simplified significantly with the use of De Morgan's law, also discussed in the next chapter.

Review exercises

1. If a logic function has three inputs, how many rows must the truth table have to contain all possible states? Justify your answer.
2. Repeat the last problem for five inputs.
3. For the following functions, construct a truth table and draw a circuit diagram.
   (a) y(A,B) = (AB)' + B'
   (b) y(A,B,C) = (A + B)' C
   (c) y(A,B,C) = (AC)' + BC
   (d) y(A,B,C) = (A ⊕ B)C'
   (e) y(A,B) = A' + B
   (f) y(A,B,C) = ((A+B)'(B+C)')'
4. For 3(e) of the previous exercise, design the circuit using 7400 series chips listed in Appendix A. Label the pinouts on the circuit diagram. Make sure to label all of the pinouts, just as in Exhibit 2.14.
5. Repeat exercise 4 using 3(f).

Procedure

1. Write the prelab in your lab notebook for all circuits required in the steps that follow.
2. Obtain instructor approval for your prelab.
3. Assemble one single NAND gate from a 7400 chip and verify its operation.
4. Assemble one single NOR gate from a 7402 chip and verify its operation.
5. Build the circuit required for Exercise 4 from the review exercises. Make sure to have your instructor verify that your circuit works correctly before moving on.
6. Build the circuit required for Exercise 5 from the review exercises.

Optional procedure

1. Design, construct, and verify the operation of the circuit from Exercise 5 using only NAND gates.

3. Logic simplification

Learning objectives:

- Use reduction techniques to obtain minimal functional representations.
- Design minimal three and four input logical functions.
- Build and debug three and four input logical functions.

De Morgan's laws

As you observed in the previous lab, managing the number of connections (or wires) in your circuit can become a challenge. This challenge seems to increase exponentially as the number of components in the circuit increases. In order to keep your breadboard as neat as possible and your design as simplified as possible, it is often advantageous to spend time examining the logical function for ways to reduce the complexity of the final design. Reducing the number of gates in a circuit will generally lead to a reduction in the number of connections, resulting in a simpler circuit. Designs with fewer connections and parts have fewer possible points of failure. Less complex circuits are generally easier and cheaper to build and debug. In this chapter, techniques will be introduced that can help to implement complex circuits in the least complex manner possible.

It is often possible to implement logical functions correctly in many different ways. The first step in obtaining a logically minimal expression should be a clear understanding of the rules of Boolean algebra listed in Appendix D. De Morgan's laws in particular can be very helpful when attempting to simplify circuit design. De Morgan's laws are listed below.

(AB)' = A' + B' **(A+B)' = A'B'**

Given these two equations, it is easy to see the alternate symbols that are sometimes used for the AND and OR gates listed in and . Applying De Morgan's laws to the functions listed yields the following.

(A' + B')' = (AB')' = AB (A'B')' = ((A + B)')' = A + B

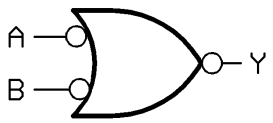


Exhibit 3.1: Alternate AND symbol

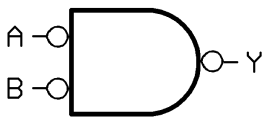


Exhibit 3.2: Alternate OR symbol

An example of using De Morgan's laws for simplification can be found by examining the logical function: AB + BC from the previous chapter. This function can actually be implemented with just three NAND gates and one 7400 chip. Examining the equation AB + BC below and applying De Morgan's law demonstrates that the expression can be implemented with only NAND gates.

AB + BC = ((AB + BC)') ' *Double Negative*

= ((AB)' (BC)')' *De Morgan's law*

Notice that the first expression exactly matches the function that was built in the previous chapter using two NANDs, one NOR and three inverters. The new circuit shown in Exhibit 3.3 implements the same expression with just three NAND gates. This results in a design using only one 7400 series chip and fewer connections that still

yields the same result. Designs with fewer chips and wires generally take less time to build, resulting in less expensive, more robust circuits. Similarly, the circuit that implements the XOR from the last chapter could be built with just NAND gates, however as five gates would be required, it still would use two chips, one 7400 and a 7404.

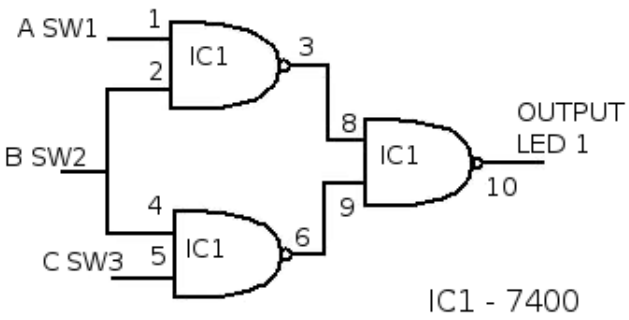


Exhibit 3.3: AB + BC (NANDS only)

Karnaugh maps

Karnaugh maps or K-maps for short, provide another means of simplifying and optimizing logical expressions. This is a graphical technique that utilizes a sum of product (SOP) form. SOP forms combine terms that have been ANDed together that then get ORed together. This format lends itself to the use of De Morgan's law which allows the final result to be built with only NAND gates. The K-map is best used with logical functions with four or less input variables. As the technique generally becomes unwieldy with more than four inputs, other means of optimization are generally used for expressions of this complexity. While it can be more instructive for students to use Boolean algebra reduction techniques, when minimizing gate circuit; it is less obvious for students to recognize when they have reached the simplest circuit configuration. One of the advantages of using K-maps for reduction is that it is easier to see when a circuit has been fully simplified. Another advantage is that using K-maps leads to a more structured process for minimization.

In order to use a K-map, the truth table for a logical expression is transferred to a K-map grid. The grid for two, three, and four input expressions are provided in the tables below. Each cell corresponds to one row in a truth table or one given state in the logical expression. The order of the items in the grid is not random at all; they are set so that any adjacent cell differs in value by the change in only one variable. Because of this, items can be grouped together easily in rectangular blocks of two, four, and eight to find the minimal number of groupings that can cover the entire expression. Note that diagonal cells require that the value of more than two inputs change, and that they also do not form rectangles.

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C'D' 00				
C'D 01				
CD 11				
CD' 10				

Table 5: 4 input K-map

	A' 0	**A** 1
B' 0		
B 1		

Table 6: 2 input K-map

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C' 0				
C 1				

Table 7: 3 input K-map

Examine the expression f(A,B,C) = ABC + ABC' + A'BC + A'BC'. As listed, it requires four three-input AND gates, one four-input OR gate and several inverters. The truth table is copied over to the eight cell K-map below. Notice the square of ones in the center of the K-map. These cells all share the fact that they are true when B is true. And indeed, the expressions shown below are equivalent.

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

Table 8: f(A,B,C)

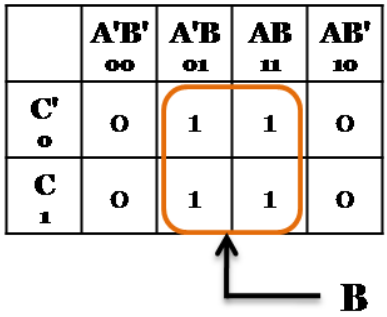


Exhibit 3.4: K-map of f(A,B,C)

ABC + ABC' + A'BC + A'BC'	= AB(C + C') + A'B(C + C')	Distributive Property
	= AB + A'B	C + C' is always true
	= (A + A')B	Distributive Property
	= B	A + A' is always true

Of course, implementing the logical expression B is much simpler than the previous expression! Although rules of logic applied above yield the same result, it is often much easier to note the groupings that result in minimal expressions using the graphical representation of the K-map.

Let us examine the equation g(A,B,C,D) given in the truth table in Table 7 with the associated K-map. The expression contains three different terms: A'B', AC, and ABC'D circled in Exhibit 3.5. However, this is not the minimal expression because not all of the largest possible groupings are included. In order to obtain the largest groupings, it is often necessary to overlap some of the terms. This just causes certain terms to be included in more than one grouping as shown in Exhibit 3.6. Notice term ABCD which is actually included in two different groupings, ABD and AC, which is perfectly acceptable. Using the new groupings, we obtain the minimal SOP expression g(A,B,C,D) = A'B' + AC + ABD. This expression contains the same number of groupings or products, but one less term in one of the products. In this case ABC'D from Exhibit 3.5 is replaced with ABD in Exhibit 3.6 yielding a simpler expression. While other techniques exist for finding minimal expressions, with some practice, the K-map can be used effectively for expressions with four or less inputs.

Not selecting the largest grouping is a very common error to those just beginning to use K-maps. Remember, always select the largest grouping possible, even if it results in some terms being double covered. Larger groupings result in simpler expressions.

3. Logic simplification

A	B	C	D	g
0	0	0	0	1
0	0	0	1	1
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	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 9: g(A,B,C,D)

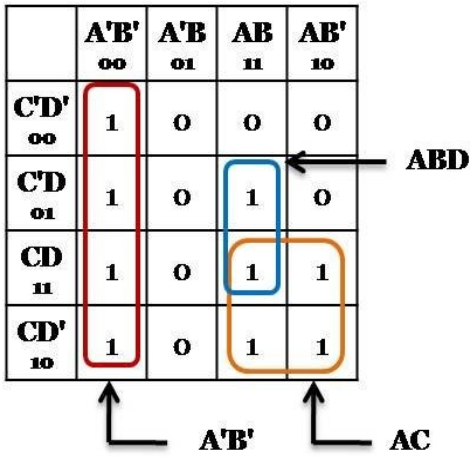


Exhibit 3.6: K-map of g(A,B,C,D)

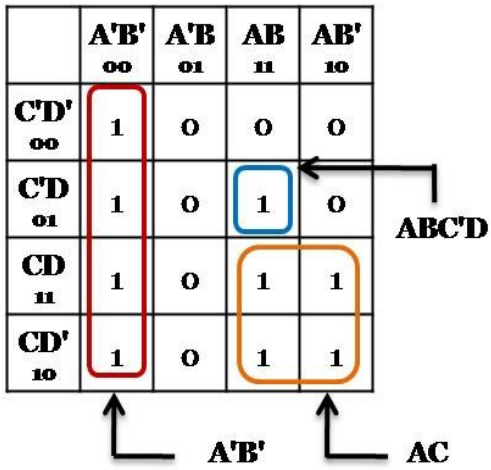


Exhibit 3.5: K-map of g(A,B,C,D)

In summary, the procedure for using K-maps to find minimal logical expressions is given below.

1. Construct the K-map corresponding to the truth table.
2. Circle any 1 that is NOT adjacent (isolated) to any other 1.
3. Find any 1 that is adjacent to *only* one other. Then circle these pairs, even if one in the pair has already been circled.
4. Circle any group of eight (octet), even if a 1 in the group has already been circled.
5. Circle any group of four (quad) that contains one or more one 1 that is not already circled.
6. Make sure that every 1 is circled.
7. Form the OR sum of the terms generated by each grouping.

The following example goes through all the steps in order to find the minimal expression for h(A,B,C,D). First, the truth table given in Table 8 is transcribed to fit into the K-map given in Table 5.

A	B	C	D	g
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Table 10: h(A,B,C,D)

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	1	1	0
C'D 01	1	1	1	1
CD 11	0	1	0	0
CD' 10	0	0	0	1

Table 11: h(w,x,y,z)

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	1	1	0
C'D 01	1	1	1	1
CD 11	0	1	0	0
CD' 10	0	0	0	1

Table 12: Step 3

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	1	1	0
C'D 01	1	1	1	1
CD 11	0	1	0	0
CD' 10	0	0	0	1

Table 13: Step 2

In step 2, above, the 1 in the bottom right is shaded.

In step 3, to the left, the pair of two 1s in the second column is shaded. Note that the bottom item A'BCD dictates that this group is circled. The top item, A'BC'D has many different adjacent elements, but the first 1 only has one adjacent element. For step 4, no groups of eight exist, so there is no table. For step 5, two groups of four exist, C'D and BC'.

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	1	1	0
C'D 01	1	1	1	1
CD 11	0	1	0	0
CD' 10	0	0	0	1

Table 14: Step 5

Note that both of these groupings cover elements already covered from step 2 and that both share the group of two, BC'D. This overlap is shaded in green. This is not only perfectly acceptable, but required to obtain the minimal expression. Now, all of the 1s are covered, yielding the minimal solution.

$$h(A,B,C,D) = AB'CD' + A'BD + BC' + C'D$$

Circuit design, construction and debugging

While these techniques are useful in minimizing the logical expression, ultimately the circuits still need to be constructed. As the complexity of the circuits increases, it is important to note some of the techniques that can be useful in building a complete working circuit.

> DESIGN TIP: The time spent in the design stage can pay **huge** dividends later! Mistakes made at the beginning of the design phase carry through the entire process and can consume countless hours trying to debug the final product.

- Start by making sure that the circuit minimization was correct and copied in your lab notebook. The truth table is helpful when testing the final circuit. Building the wrong circuit serves no purpose at all.
- Verify that the pinouts selected are proper for each gate and chip; these are helpful when debugging as well as when building the circuit. Again, time spent here helps cut down on the construction and debugging later.
- Remember the tips given in the chapter "The transistor and the inverter" regarding the use of the breadboard.
- Keep connecting wires neatly and avoid unnecessarily long loops of wire, yet do not spend excessive time cutting wires that are *exactly* the proper length between spans. It may feel like a work of art, but in the end you want a neat circuit that works properly.

If your circuit does not work properly:

- Attempt to reason out the problem. Does the circuit act reliably?
    - Does it always produce the same wrong result? If so, then the error is likely in the logic.
    - If it yields different results at different times, a loose connection is very likely. If two output lines are connected together (which should never be done), it can also result in unpredictable outputs.
- Test each component of the circuit independently. For example, if you have the expression AB' + ABCD + ABC' built with NAND gates and inverters, first test that the input and output of (AB')' is working correctly. Then move onto each succeeding term.
- Verify the circuit has power and ground to all of the appropriate pins for each chip.
- Verify that all of the pins are connected properly.
    - Make sure that they follow what is specified in your circuit diagrams.
    - Make sure that none of the output pins are tied together. If each of the output pins were to obtain a different value, this could result in a logic high being tied directly to a logic low level. At best, this can result in an indeterminate value. This will result in further problems if this output is then used as an input for another gate.
- Remember that often things do not work the first time when you build them.

> DEBUGGING TIP: **Do not allow yourself to get frustrated!** This is easier said than done, but getting upset does not serve any purpose in effective troubleshooting.

If you have done all of the above and the circuit still does not work:

- Return to the design phase and verify that your minimization and pinouts are correct.

- Sometimes errors come from the components or equipment themselves. Errors such as those listed below can occur, but are *very* rare. These should be considered as a last resort and other causes of error should be investigated before looking for the following errors:
  - A pin on a DIP can become bent and curl under the chip so that it does not get inserted into the breadboard. This is difficult to see without taking the chip out and examining its legs.
  - In general, solid state devices are very reliable when operated under proper temperature ranges, but very occasionally a chip may be faulty.
  - Connecting wires can be split inside of the insulation. When this occurs, the insulation will cause the wire to look as though it is intact, but if the copper is in two pieces inside the insulation, current will not flow and the wire will actually be open.
  - Faulty test equipment can adversely effect the circuit being tested and lead one to believe a circuit is malfunctioning when it is not, or give you other false information that leads you down the wrong path in your reasoning.
- Ask for help from fellow classmates and your instructor.
- Take a break and come back to the problem. No one works at their best when they are totally aggravated.

Review exercises

1. Design a 4-input NAND gate using two 2-input NAND gates and one 2-input NOR gate. Hint: Use DeMorgan's law.
2. What are the possible groupings in a 4-input K-map? Sketch their shapes.
3. Construct a truth table for the following functions:
   (a) f(A,B,C) = AB + A'BC' + AB'C
   (b) g(A,B,C) = A'C + ABC + AB'
   (c) h(A,B,C,D) = A'BC' + (A $\oplus$ B)C + A'B'C'D + ABCD
   (d) j(A,B,C,D) = A'C'D' + C'D + CD
4. Construct the K-map for each of the functions from the previous problem and determine the minimal expression for each.
5. For 3(b), design the circuit for the minimal SOP expression found in problem 4 using just NAND gates and inverters. Label the pinouts on the circuit diagram.
6. For 3(c), design the circuit for the minimal SOP expression found in problem 4 using just NAND gates and inverters. Label the pinouts on the circuit diagram.
7. Given each of the K-maps, determine the minimal expression associated with it.

(a)

	A' 0	**A** 1
B' 0	1	1
B 1	1	0

(b)

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C' 0	1	1	1	1
C 1	0	1	0	0

3. Logic simplification

(c)

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	1	1
C'D 01	1	1	1	0
CD 11	0	0	1	1
CD' 10	0	0	1	1

(d)

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	0
C'D 01	1	1	0	1
CD 11	0	1	0	1
CD' 10	1	1	0	0

Procedure

1. Write the prelab in your lab notebook for all the circuits required in the steps that follow.
2. Obtain instructor approval for your prelab.
3. Build the circuit required for Exercise 5 from the review exercises.
   (a) Make sure to test each of the portions of the expression independently. Meaning, test the output of each of the first level NAND gates to verify that each works before testing the final output.
   (b) Demonstrate the working circuit for your instructor.
4. Repeat the steps from the last procedure for Exercise 6 of the review exercises.

4. More logic simplification

Learning objectives:

- Review all possible K-map groupings.
- Use “don't care” conditions in minimization.

Additional K-map groupings

Some of the rectangular groupings allowed for Karnaugh maps, such as the one in Exhibit 4.1, are not obvious. Cells on borders actually are adjacent to cells on the opposite border, which produce groupings that may not appear continuous. This grouping of two cells actually forms a rectangle represented by B'C', even though this rectangle is split.

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	1	0	0	1
C 1	0	0	0	1

B'C'

Exhibit 4.1: K-map grouping

The possibilities for non-obvious groups increase for K-maps with four-input functions. Exhibit 4.2 shows B'D, a four cell square grouping that is split on the two side borders. In Exhibit 4.3, the eight cell rectangular grouping D' is shown. One of the most non-obvious four cell groupings that contains all four corners is shown in Exhibit 4.4. The interested reader can verify that the minimal expressions for Exhibit 4.2, 4.3 and 4.4 are B'D+A'D+A'B'C, D'+AB'+A'C' and B'D'+A'BD+A'CD respectfully.

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	0
C'D 01	1	1	0	1
CD 11	1	1	0	1
CD' 10	1	0	0	0

B'D

Exhibit 4.2: 4-element group

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	1	1
C'D 01	1	1	0	1
CD 11	0	0	0	1
CD' 10	1	1	1	1

D'

Exhibit 4.3: 8-element group

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	0	0	1
C'D 01	0	1	0	0
CD 11	1	1	0	1
CD' 10	1	0	0	1

B'D'

Exhibit 4.4: Four corner group

Input placement on K-map

All of the K-maps shown so far have had the input variables A and B set along the top with the input variables C and D along the side. This does not need to be the case, but it is the convention used here. In addition, the inputs have used the gray code 00 -> 01 -> 11 -> 10, which does not need to be the case either. For example, the input sequence could have been 00->10->11->01 while still only changing one input at a time. Although altering these conventions will still lead to the exact same minimal expressions, it is discouraged because when verifying results, it can often lead to confusion. By altering the convention, you could cause those trying to assist you to spend extra time when examining your work. The following example illustrates how the same representation will be obtained despite the ordering of the input variables. In Exhibit 4.5 the same function is represented as in Exhibit 4.3. In this case, the region highlighted for D' does not span two boundaries, while the grouping for A'C' does in this format. Again, it can be shown that the same minimal expression is obtained: D' + A'B + A'C'.

	D'C' 00	D'C 01	DC 11	DC' 10
A'B' 00	1	1	0	1
A'B 01	1	1	1	1
AB 11	1	1	0	0
AB' 10	1	1	0	0

D'

Exhibit 4.5: 8-input K-map grouping

Don't care conditions

While all input cases for a logical function must be considered, in an actual design it often occurs that certain cases never exist. For instance, a particular counter that cycles through the states zero through five would never reach states six (110) and seven (111). In such cases, it can be advantageous to fill the spots with a don't care condition (*d*). The don't care can then be included with a grouping if it helps to minimize the final logical representation, otherwise it can be treated as false. Consider the example in Exhibit 4.6. If only the ones are grouped, the minimal expression is C'D' + A'BC' + BD'. However, if the don't care conditions are allowed to be grouped with ones, the resulting minimal expression is B + C'D'.

Remember that the presence of a don't care condition does not require that this cell be covered in the final output. Exhibit 4.7 demonstrates this case. Note, two of the don't cares are included to yield a minimal representation of C'. The don't care along the bottom is not included at all.

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	1	1
C'D 01	0	1	d	0
CD 11	0	d	d	0
CD' 10	0	1	1	0

C'D'

B

Exhibit 4.6: Don't care conditions

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	1	d	d	1
C 1	0	d	0	0

C'

Exhibit 4.7: Don't care not covered

Review exercises

1. Given each of the K-maps, determine the minimal SOP expression. *d* represents a don't care condition.

(a)

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C' 0	*d*	0	1	1
C 1	1	0	0	*d*

(b)

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C' 0	0	0	1	1
C 1	1	*d*	*d*	1

(c)

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C'D' 00	0	0	0	1
C'D 01	0	0	1	1
CD 11	1	0	1	1
CD' 10	1	0	0	1

(d)

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C'D' 00	1	1	0	1
C'D 01	0	1	0	0
CD 11	0	1	0	0
CD' 10	1	1	0	1

(e)

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C' 0	1	1	0	1
C 1	0	1	1	0

(f)

	A'B' 00	**A'B** 01	**AB** 11	**AB'** 10
C'D' 00	1	1	0	0
C'D 01	1	1	1	1
CD 11	0	0	0	0
CD' 10	1	1	0	0

2. For the functions listed below, construct a K-map and determine the minimal SOP expression.
   (a) f(a,b,c) = a'b'c' + a'bc' + abc' + abc
   (b) g(a,b,c) = ab'c' + abc' + abc + don't cares(a'bc + ab'c)
   (c) k(a,b,c,d) = abc'd + ab'c'd + a'bc'd + a'b'cd' + don't cares(a'b'cd+ a'bcd + ab'cd + abcd)
   (d) m(a,b,c,d) = a'b'cd' + a'bcd' + abc'd' + abcd' + ab'c'd' + ab'cd' + don't cares(a'bc'd + abc'd)

Procedure

1. Write the prelab in your lab notebook for all the circuits required in the steps that follow.
2. Obtain instructor approval for your prelab.

4. More logic simplification

3. Build the circuit required for Exercise 2(b) from the review exercises.
4. Demonstrate the working circuit for your instructor.
5. Repeat the steps from the last procedure for Exercise 2(c) from the review exercises.

5. Multiplexer

Learning objective:

- Use the multiplexer to implement complex logical functions.

Background on the "mux"

A multiplexer, often just called a mux, is a device that can select its output from a number of inputs. This device is useful in computer systems that use a bus architecture, where several devices share the same communication path. A 2-to-1 multiplexer is shown in Exhibit 5.1. In this case the two inputs are D0 and D1. If the select line is low, then the output will reflect the state of D0. Likewise, if the select line is high, the output is the state of D1. Hence, the output is switched between two different devices connected to D0 and D1 using the select line. In this way, only one device will be active or connected to the bus at any given time.

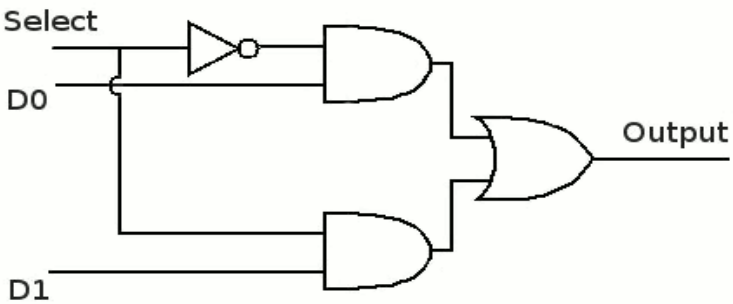


Exhibit 5.1: 2-to-1 multiplexer

With an increase in the number of select lines, multiplexers allow for more than just two input lines. If two select lines are used, then the output can be selected from four different inputs forming a 4-to-1 mux. The 74151 provided in your kit is an 8-to-1 mux that uses three select lines to chose from 8 different input lines. A diagram of the 74151 chip is given in Appendix A. The 8-to-1 multiplexer can be used to take a byte of parallel data on the input lines and determine which of the input lines to display at the output. This is useful with bus architectures in order to convert the parallel data that most often comes in bytes into a serial stream of bits.

Using a multiplexer to implement logical functions

Another use for the mux is to implement fairly complicated logic functions without the aid of other logic gates. As an example, examine the following function along with its K-map, and the resulting minimal SOP expression.

$$g(a,b,c) = a'b'c' + a'bc + ab'c' + ab'c + abc'$$
$$= a'bc + b'c' + ac' + ab'$$

In order to implement the circuit of this function for even the minimal SOP representation, five NAND gates are required. However, a single mux can be used to implement the same expression. The key is to use the input variables for the function as the input for each select line and set the data lines to the value for each of the corresponding outputs. Note that the value of data lines D0, D3, D4, D5, and D6, which also are found on pins 4, 1, 15, 14, and 13 are set to high with the remaining data lines set low. In this manner, any three input logical functions

can be built with a single mux. Note that as mentioned previously, the strobe pin is tied low and the order of the inputs from the function differ from the order of the input lines for the 74151 chip.

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	1	0	1	1
C 1	0	1	0	1

Table 15: g(a,b,c)

A	B	C	f
0	0	0	1
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	0

Table 16: g(a,b,c)

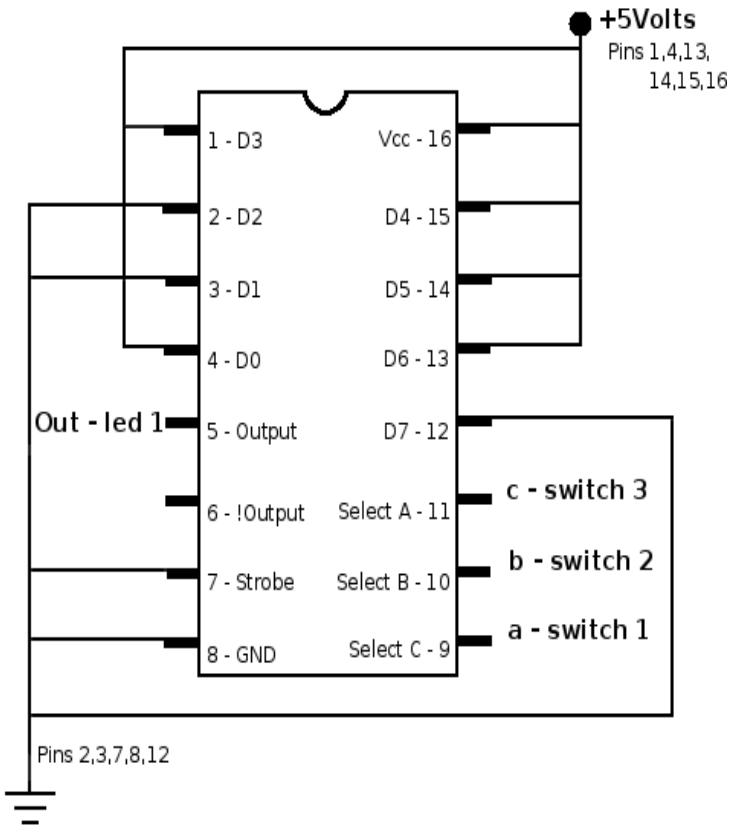


Exhibit 5.2: Circuit for g(a,b,c)

When used in this manner, the 74151 is often referred to as a boolean function generator. This circuit could be even more flexible if the data input lines, D0 through D7, could be changed. The function that the multiplexer implemented could be changed while the circuit is running with the use of memory chips. This change stores temporary values for the input lines to create a truly programmable boolean function generator.

> When using the 74151 multiplexer:
> (1) Make sure to properly select the strobe line.
> (2) Note that values chosen for A, B, and C may differ from those given in the truth table in Appendix A. Appendix A assumes that C is the most significant input line, which may not be the case in your design.

Just as this method of using an 8-to-1 mux can be used to implement any 3-input function with just one chip, any 4-input function can be built with a 16-to-1 mux. However, the kit provided with this lab only contains the 8-to-1 mux. This can present a problem when a complex four input function would require several different 7400 series chips to implement, such as the function h(a,b,c,d) found in the K-map and truth table that follow. Two different minimal SOP expressions exist for this function. See below.

h(a,b,c,d) = a'bc' + a'b'c + acd' + ab'c'd + a'*c'd'*

h(a,b,c,d) = a'bc' + a'b'c + acd' + ab'c'd + a'*bd'*

Either of the terms at the end of each expression could be used to obtain a minimal expression. Yet, either would require four 3-input NAND gates, one 4-input NAND gate and one 5-input NAND gate, assuming that your kit even provided NAND gates with four or five inputs.

It may not be obvious how to use the multiplexer in cases such as this to implement the function. One approach would be to use two mux chips along with some additional gates. One trick is to use two 8-to-1 multiplexers along with one 2-to-1 mux as shown in Exhibit 5.1. Each half of the function is implemented with an 8-to-1 mux and the output of each is selected using the remaining input as the select line for the 2-to-1 mux. Luckily, a simple trick can be used with an 8-to-1 mux. First take the function given in the K-map for h(a,b,c,d) produce the truth table, but add one column for the multiplexer input of each data element.

	a'b' 00	**a'b** 01	**ab** 11	**ab'** 10
c'd' 00	1	1	0	0
c'd 01	0	1	0	1
cd 11	1	0	0	0
cd' 10	1	0	1	1

Table 17: h(a,b,c,d)

Each of the two rows in the sixth column now represent one of the input lines. Instead of the input lines taking just true or false to implement the truth table directly, the input lines will take the value of true, false, d, or d'. In this way, only one multiplexer needs to be used along with possibly one inverter gate. As a, b, and c are used to select the data line, each set of two rows that share the same input values for a, b, and c are grouped together in the table. Then by comparing the output value of h for these two rows, it can be determined what value the data line should take. For example, since h matches input d for the first two rows, the input value for D0 should be tied to input d. The circuit that implements h(a,b,c,d) is given in Exhibit 5.3. It is assumed that the inverse of the input d is available somewhere in the circuit, if not, an inverter would need to be added to this circuit.

a	**b**	**c**	**d**	**h(a,b,c,d)**	**Input**
0	0	0	0	0	D0=d
0	0	0	1	1	D0=d
0	0	1	0	0	D1=0
0	0	1	1	0	D1=0
0	1	0	0	1	D2=1
0	1	0	1	1	D2=1
0	1	1	0	0	D3=d
0	1	1	1	1	D3=d
1	0	0	0	0	D4=d
1	0	0	1	1	D4=d
1	0	1	0	1	D5=d'
1	0	1	1	0	D5=d'
1	1	0	0	1	D6=1
1	1	0	1	1	D6=1
1	1	1	0	0	D7=0
1	1	1	1	0	D7=0

Table 18: h(a,b,c,d)

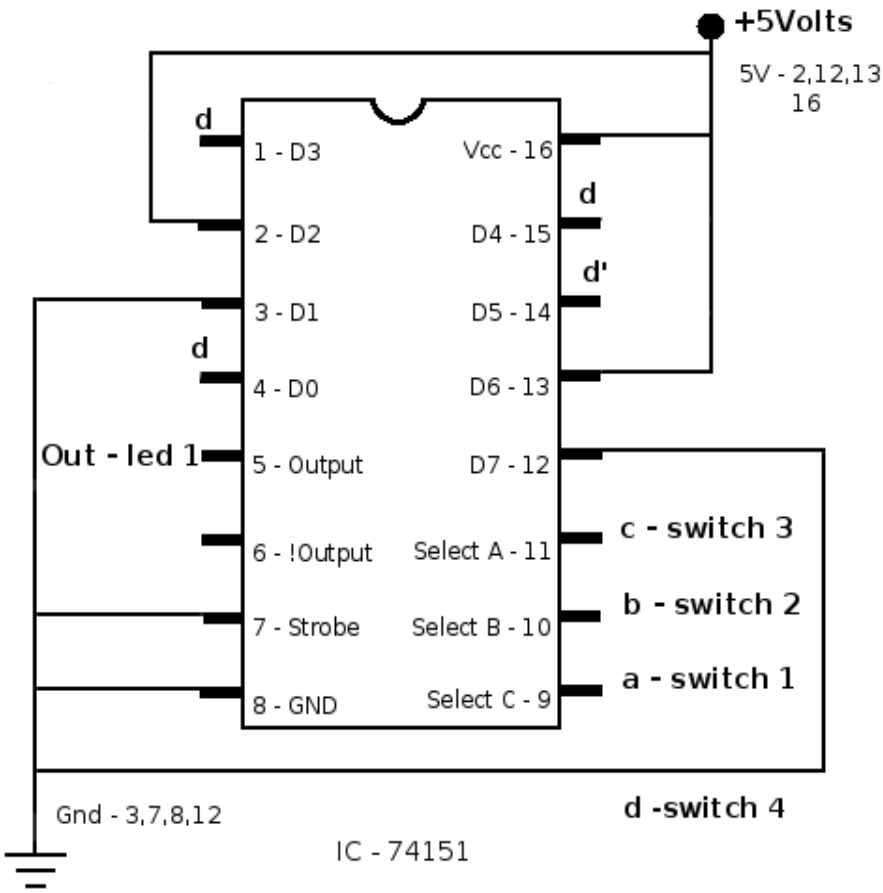


Exhibit 5.3: h(a,b,c,d) implemented with 8-to-1 mux

As the mux can implement logical functions directly from the truth table without the need for any logic minimization, it is often tempting to use the mux to implement every function and simply skip the minimization techniques described earlier. Resist this temptation! Often the minimal SOP implementation will require few gates resulting in a simple design without a mux. In addition, when different functions are required for a given circuit, if only multiplexers were used, a mux would be needed for each and every function. However, the minimal SOP expressions for the different functions will sometimes share common logical terms. Examine the two functions below that are required for a given circuit.

$f(x,y,z) = x'yz$

$g(x,y,z) = z' + x'yz$

They share the term x'yz, and this part would only need to be built once and could be used for both functions, saving gates. Sharing of terms in this manner is not possible when using the mux to implement functions. So in order to insure that the simplest circuit is designed to implement the function, the logic minimization techniques described earlier should be examined first before resorting to the mux to implement a function.

Review exercises

1. Construct the truth table and K-map for each of the following functions and determine the minimal SOP expression.

(a) f1(a,b,c) = a'b'c' + a'bc' + a'bc + ab'c'

(b) f2(a,b,c) = a'b'c + a'bc + abc' + ab'c

(c) f3(a,b,c,d) = a'b'c'd' + a'bcd + abcd + ab'c'd' + ab'c'd

(d) f4(a,b,c,d) =a'b'c'd' + a'bc'd + abcd + a'b'cd' + a'b'cd + a'bcd' + ab'c'd

2. Design the implementation of expression 1(b) using an 8-to-1 mux.
3. Design the circuit that will implement 1(d) using an 8-to-1 mux chip along with any necessary circuitry.
4. Examine the following four-input functions and design a circuit that will implement each.

(a) g1(a,b,c,d) = a'b'c'd + abcd + a'bcd + a'bc'd + ab'c'd + a'b'cd + abc'd + ab'cd

(b) g2(a,b,c,d) = a'bc'd + a'b'cd' + ab'cd

(c) g3(a,b,c,d) = abc'd' + abc'd + abcd + abcd' + a'bc'd + a'bcd

(d) g4(a,b,c,d) = a'bc'd' + abc'd' + abcd' + ab'cd' + a'bc'd + abc'd + abcd + ab'cd

Procedure

1. Write the prelab in your lab notebook for all the circuits required in the steps that follow.
2. Obtain instructor approval for your prelab.
3. Build the circuit required for Exercise 2 from the review exercises. Demonstrate the working circuit for your instructor.
4. Repeat the steps from the last procedure for Exercise 3 from the review exercises.

6. Timers and clocks

Learning objectives:

- Review relation between time and frequency.
- Construct timer and clock circuits.
- Produce a timing digram for a circuit.

Timing in digital circuits

Timing circuits are often required for various applications. One may need to measure the length of time that a given switch has been on or off. As will be seen in future labs, for more complicated circuits, a clock is often necessary to synchronize the various components. While many different ways exist to build timing circuits, the 555 timer chip has proven to be an industry standard for this purpose.

555 timer

The 555 timer chip was first manufactured in the early 1970s and continues to be used in electronic devices. The detailed circuit diagram seen in Appendix A for this integrated circuit contains two diodes, many resistors and over twenty transistors. All of this is contained in one small dual inline package that can be used in timing and clocking circuits. It is important to note that propagation delays caused by the time it takes for signals to travel through the circuit components prevent it from being used in circuits requiring fast switching times. In this case, fast is considered a few µseconds. The propagation delay varies slightly depending upon the version of the 555 being used. This limitation prevents the 555 from reaching speeds necessary for modern computer systems. However, many applications have less rigorous requirements for which the 555 timer has proven to be the component of choice. Due to mass production, this chip is widely available at a modest price.

Timers

A timing circuit using the 555 timer is found in Exhibit 6.1. This circuit is also called a one-shot because it will work once for every time it is triggered properly. After being triggered, it turns on for the specified time and then returns to its stable off state. It is also often said to be operating in monostable mode because it only has one stable state, when its output is low, ground or off.

The circuit is triggered with a voltage below (1/3)Vcc (Vcc is the supply voltage for the circuit), at which time the capacitor labeled C begins charging through the resistor labeled R. At the time when the voltage on the capacitor reaches (2/3)Vcc, the output will turn low. The voltage across the capacitor is given below. See Appendix B for more information regarding resistors and capacitors.

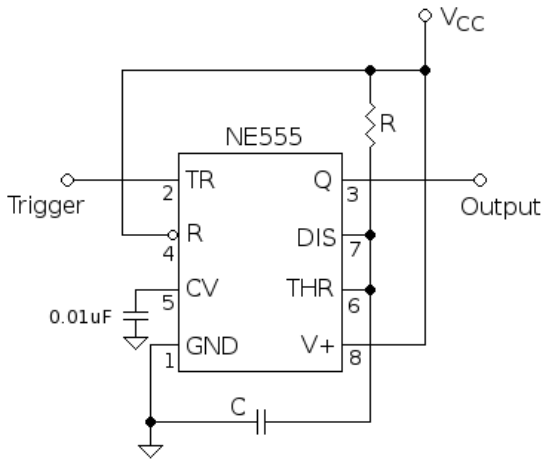


Exhibit 6.1: Timer circuit

$$V(t) = Vcc(1 - e^{-(t/rc)})$$

Setting V(t) equal to (2/3)Vcc and solving for t yields the time when the output will go low (assume three digits of accuracy).

$$t = 1.10(RC)$$

Note that the values for resistors and capacitors often vary with a tolerance of ±5 per cent and ±10 per cent respectively. Hence, the time of the timer may not exactly match the calculated value. When it is critical for the application to have a very specific time, either the components used must be measured to insure that they match the time needed or a variable resistor can be used so that it can be adjusted once the circuit is built.

Clocks

Just as the drummer in a band helps to keep the rest of the members synchronized, so does the clock in a circuit. A clock is used to synchronize a circuit that contains different components that have different propagation delays. Synchronization is required because signal changes take time to travel through a circuit. Internal inductance and capacitance found in the wires of the circuit and the components themselves cause delays. In order to insure that each transition or change has fully propagated through the circuit, the clock can only switch as fast as it takes the slowest part of the circuit to fully register each change. Modern processors have clocks that operate in the gigahertz range and are built with the use of crystals. The 555 timer chip cannot be clocked that fast due to the internal propagation delay within the transistors in the chip, but it can provide a reliable clock pulse for applications that do not require that speed.

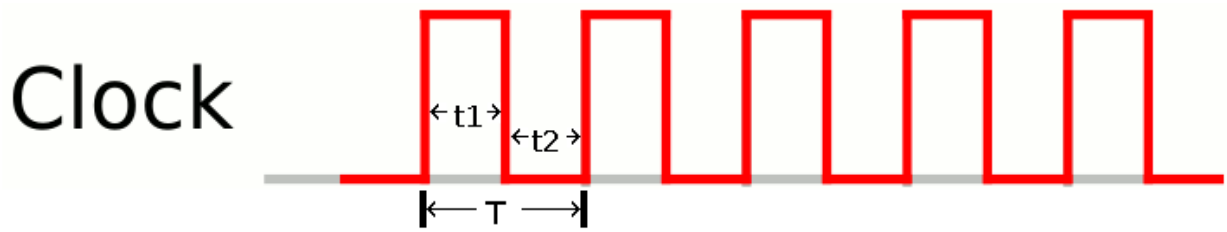


Exhibit 6.2: Clock waveform

Clock speeds are given in terms of frequency which uses the unit hertz; this stands for cycles per second. So if a clock is said to have a frequency of 200 megahertz, it transitions from logic high to logic low 200,000,000 times in one second! Another measure often associated with a clock is its period, which is the time it takes for the full clock cycle. The period of the 200 megahertz clock is 5 nanoseconds.

$$T = 1/f$$

Mathematically, period (T) and frequency (f) are related inversely. The clock waveform given in Exhibit 6.2 illustrates an idealized waveform. In reality the transitions from low to high or high to low take some time and are not instantaneous as those shown. As another example, a 5 gigahertz clock has a period of 1/5,000,000,000 seconds, which is 0.0000000002 seconds or 0.2 nanoseconds. The clocks built for these labs will be much slower than this. The fastest clock will have a period of one second.

Exhibit 6.3 shows a clock circuit using the 555 timer. When configured in this manner, it is said that the timer is operating in astable mode. This means that there is no stable state for the circuit; it just continues to oscillate, going from low to high and back again. In this case, the trigger is tied to the voltage across the capacitor, so that the circuit is triggered by itself. The capacitor is charged through the series combination of R1 and R2 and discharged through R2. The capacitor charges to 2/3*Vcc and then discharges to 1/3*Vcc repeatedly. Using the same method given in the previous section, the times to charge and discharge the capacitor along with the equations for the period and frequency are listed below.

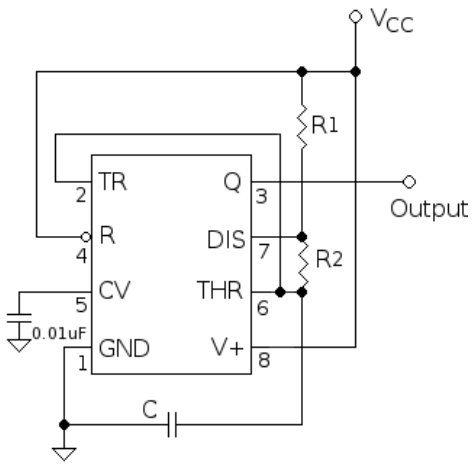


Exhibit 6.3: Clock Circuit

t1 = 0.693(R1 + R2)C	*charge time*
t2 = 0.693(R2)C	*discharge time*
T = t1 + t2 = 0.693(R1 + 2*R2)C	*period*
f = 1/T = 1.44/((R1 + 2*R2)C)	*frequency*

Note that the accuracy of the values of the resistors and capacitors will affect the actual values for the frequency of the clock. Also, this clock will not have a symmetric waveform as it will be charging (on) for a longer time than it will be discharging (off).

> When measuring the frequency of the clock, count the time for ten full clock pulses and then divide this number by ten to determine the period. This will reduce the effect of timing errors introduced by those taking the measurements.

Timing diagrams

The graph of the logical transition for a circuit is given in a timing diagram. Timing diagrams provide a visual trace of the circuit functionality. They can also be helpful in determining the maximum possible delay for a given circuit which can then be used to determine the fastest frequency in which the circuit can be clocked. The diagrams display each value in one of three different states: logic high, logic low, and indeterminate. The indeterminate state would occur when a given state cannot be guaranteed to be either high or low. Indeterminate states are usually shown as gray areas that span the entire region from low to high for the duration of the indeterminate period. The transition edges are often not shown to be totally vertical, as they are in Exhibit 6.2. This is to illustrate the point that changes in output are not instantaneous due to delays caused by transition times as well as internal inductance and capacitance in the circuits.

The timing diagram shown in Exhibit 6.5 is for the circuit found in Exhibit 6.4. This circuit has three extra points listed: A, B, and C to determine the intermediate states of each of the gates for a given transition. In this case, values for D0 is assumed to be logic high and D1 is assumed to be logic low with the SELECT line making a transition from logic low to logic high. A is the output of the inverter, B the output of the top AND gate, and C the output of the lower AND gate. The circuit is assumed to be in a stable state with the inputs SELECT, D0, and D1 at logic low, high, and low prior to time zero. Assume that the manufacturer specifies that each gate will have a maximum delay of 10.0 nanoseconds. This may vary depending upon the logic family used, so the data sheet should be consulted for verification when determining the maximum delay for a given circuit. Notice that once the SELECT

line is brought low, A, B, and the Output all assume an intermediate value as there is no guarantee of how fast the transition will occur. Once at 10.0 nanoseconds, the output of the inverter can be verified to have gone low and the state for A is listed as low. This transition value then ripples through the other gates as the top AND gate now takes another 10.0 nanoseconds to insure that its output has changed from high to low. Output changes may occur faster than the times listed, however as that cannot be guaranteed, the slowest time must be used to determine the fastest frequency in which a circuit can be clocked.

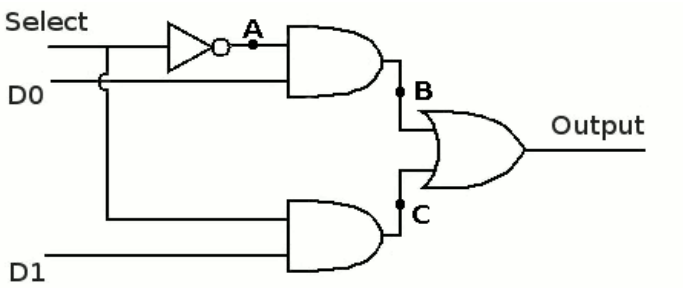


Exhibit 6.4: 2x1 Multiplexer

If this circuit were to be clocked, since the maximum delay for the entire circuit is 30.0 nanoseconds, this would also be the smallest allowable value for the period of the clock, which would yield a maximum frequency of 33.3 Mhertz. In these labs, the circuits will be clocked at a slow enough rate that delays on the order of nanoseconds will not impact the circuits. However, for circuits where speed is essential, detailed analysis such as this is critical to insure that the circuit is clocked as fast as possible while still allowing enough time for the circuit to stabilize.

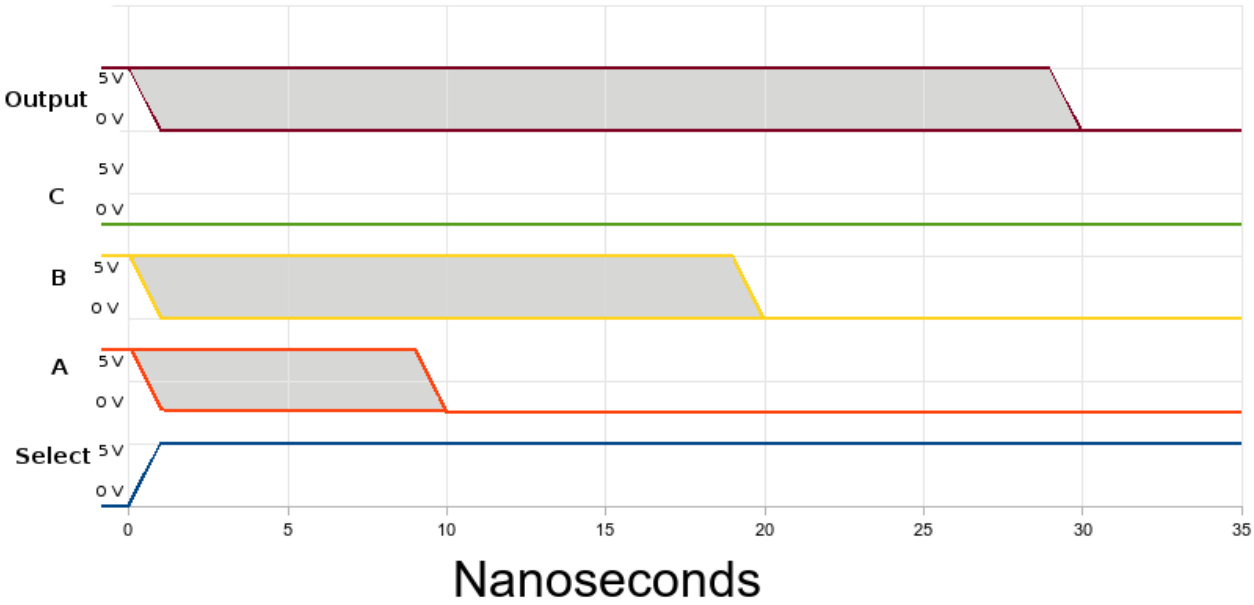


Exhibit 6.5: Timing diagram

Accuracy of answers

As this chapter involves answers that go beyond the simple binary, true or false format, a brief discussion of the accuracy of the numbers follows. When answers are provided, it is beneficial to know how accurate those answers are. The precision of any measurement is dependent upon the accuracy of the device that is used to perform the measurement. For example, one would not expect to obtain measurements within thousandths of a second using an ordinary wristwatch or within thousandths of a millimeter using a standard ruler. Once the accuracy of the measurements used is understood, it is important to remember the rules that apply to the number of significant digits for any calculation.

- Trailing zeros are significant to the number.
- Use all digits when performing calculations and round only for the final answer.
- When numbers are multiplied or divided, the final answer has the same number of significant digits as the number with the smallest amount orf significant digits in the calculation.
- In this book, the formulas are provided using three digits of accuracy. It may be the case that fewer digits can be obtained for a given measurement or that the components used may only be known within one digit of accuracy. In these cases, the final answers should be rounded accordingly.

As mentioned, the tolerances of the components will cause deviation of the measured answer from the theoretical answer. The tolerance of the resistors used in these labs is ±5 per cent while the capacitors have a tolerance of ±10 per cent. This means that for a 1000 ohm resistor, that resistor is guaranteed to be between 950 and 1050 ohms.

$$1000 - 0.05(1000) < \text{actual value} < 1000 + 0.05(1000)$$

Likewise, a 1 mircofarad capacitor is guaranteed to be between 0.9 μF and 1.1 μF

$$1 - .1(1) < \text{actual value} < 1 + .1(1)$$

This may cause the measured answer to differ quite a bit from the answer calculated using the formulas. In addition, when the values of the resistors and capacitors are multiplied together, as is the case with the formulas above for the timer and clock, these tolerances are compounded. For example, assume that a 100,000 ohm resistor is combined with a 100μF capacitor to produce a time of 10.1 seconds.

$t = 1.10(RC) = 1.10*100{,}000*0.0001 = 11.0$ seconds.

However, if we take the worst case for each value, we can see that the answer will actually be within ±15 per cent.

$$1.10(95{,}000)(0.00009) < \text{actual value} < 1.10(105{,}000)(0.00011)$$

$$9.41 < \text{actual value} < 12.7$$

For this reason, it should not be assumed that the final values for the clock and timer will match exactly the values calculated theoretically. The tolerances of the components used will often mean that the theoretical value of the clock or timer may only have one significant digit of accuracy. When the accuracy of the timer or clock is important, either components must be measured before being used to insure their values, or components with smaller tolerances should be used (which is more costly), or resistors with adjustable values (potentiometers) can be used and adjusted after the circuits are built. Of course adjusting the potentiometers is time consuming and thus costly.

Review exercises

1. What is the period in seconds of the clock with the given frequencies?
   a. 6.00 Ghertz
   b. 10 Mhertz
   c. 6000 RPM (NOTE: 60 seconds are in each minute)
2. For the given period, determine the frequency of the clock in Hertz
   a. 10.0 μsec
   b. 0.0500 nanoseconds
   c. 1.00 milliseconds

3. Assume delay for each logic gate is 10.0 nanoseconds for the circuit in Exhibit 3.3 and that input values of A is low and B and C are all at logic high. Draw a timing diagram for a transition at time zero that takes input for C from logic high to logic low. List input A, B, C, and Output as well as values for pins 3, 6, and 10.
4. If the delay for each logic gate is 10.0 nanoseconds, what is the maximum frequency that the circuit from Exhibit 2.14 can be reliably clocked in order to insure proper operation?
5. A 100 μF capacitor is used to build timers. Three timers are to be built with times of 1, 5 and 10 seconds.
   a. What resistors should be chosen to obtain the times provided?
   b. Assuming that you are limited to choosing the values provided in the lab, which resistors should be chosen to come as close to the desired values as possible? Recall that when resistors are added in series, the total resistance is the sum of the resistors.
   c. Draw a schematic of the 5-second timer.
   d. Given that capacitors have a tolerance of + -10 per cent and resistors have a tolerance of + -5 per cent, what range of values could you expect for your timer?
6. A 100 μF capacitor is used to build clocks. Two clocks are to be built with periods of 1 and 5 seconds.
   a. Using values of resistors provided in your lab, pick two resistors that yield periods as close to those desired as possible.
   b. What is the time on and time off for each of the clocks during one period?
   c. Draw a schematic of the 5-second clock.

Procedure

1. Write the prelab in your lab notebook for all the circuits required in the steps that follow. Include all necessary equations and calculations.
2. Obtain instructor approval for your prelab.
3. Build and test the 5-second timer from Exercise 5 above .
   a. How different is the measured value from the calculated value?
   b. Demonstrate the timer for your instructor.
4. Repeat Procedure 3 for the 10-second timer from Exercise 5 above.
5. Build and test the 1 second clock from Exercise 6.
   a. How different is the measured value from the calculated value?
   b. Demonstrate the clock for your instructor.
6. Repeat Procedure 5 for the 5 second clock from Exercise 6 above.

7. Memory

Learning objectives:

- Review differences between logic circuits and persistent memory.
- Review properties for the S-R latch and D flip-flop.
- Construct a circuit using a flip-flop.

Memory

You have often heard the phrase: "In order to know where you are going, you need to know where you have been." While all the circuits discussed in previous chapters are very useful, many applications quite simply cannot be implemented without the use of memory to "remember where they have been". Modern computer systems employ a wide array of different memory storage methods that have different properties. Non-volatile memory used for secondary storage such as hard drives, CD-ROM drives, or solid state memory (i.e. an Secure Digital or SD card) retains its value after power is shut off. Volatile, dynamic random access memory (RAM) loses its value when power is shut off and also must have its values continually refreshed with external circuitry. Static, volatile random access memory such as that found in cache memory and CPU registers cannot retain its value when power is not provided, yet it does not need to be refreshed. This chapter will focus upon the static, volatile, electronic memory listed last.

All of the logic circuits built in the previous sections are known as **combinatorial logic** circuits. They depend only upon the state of their inputs at any given time and do not take into account anything that has happened in the past. Often it is necessary for the output of a circuit to take past values into account. Logical circuits that take past output values along with present inputs into account to compute the output values are known as **sequential logic** circuits. In order to determine the next state of an output, the previous state must be known. Memory is used to store the history of the state(s) of a digital circuit for use in sequential circuits. An example of a sequential logic circuit would be a counter. A computer is nothing more than an advanced sequential logic circuit with memory to store data, programs, and references to the state of programs currently being run.

SR latch

Two NOR gates can be configured using feedback to produce one bit of memory. The configuration given in Exhibit 7.1 is known as an SR latch. The S, SET and R, RESET are the inputs and the Q output is provided along with its inverse. The S input is used to set or turn on the latch by setting the output Q high and inverse low. Similarly, the R input is used to reset or turn off the latch by resetting the output to low and the inverse to high. Once the latch is set, Q remains at a logic high while both input lines are off. Similarly, once the latch is reset, the Q output will be set to logic low and will remain that way while both input lines are off. In this way, the latch can store one bit of information indefinitely, or at least as long as it has power supplied to it. The NOR SR latch has active high inputs, meaning that if either input is brought high, it will force a corresponding output condition. Note that setting both input values high must be avoided in order to retain the output values as opposite to each other. Latches can also be built using NAND gates, but the set and reset lines operate in a slightly different manner under this configuration. The transitions for these latches are examined in more detail in the exercises.

S	R	Q
0	0	Q (does not change)
0	1	0
1	0	1
1	1	state not used

Table 19: NOR SR latch

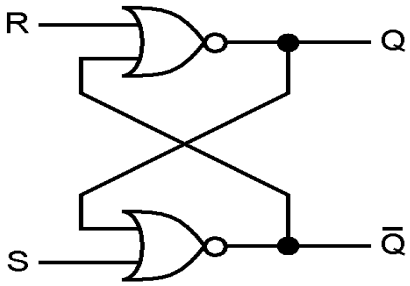


Exhibit 7.1: SR latch

The NAND based SR latch is an active low device with a default state of logic high for both S and R inputs. The S and R input values are brought low to change the state. Just as the NOR based SR latch should not have both input values turned high simultaneously, the S and R for a NAND based SR latch should not be brought low at the same time.

S	R	Q
0	0	state not used
0	1	1
1	0	0
1	1	Q (does not change)

Table 20: NAND SR latch

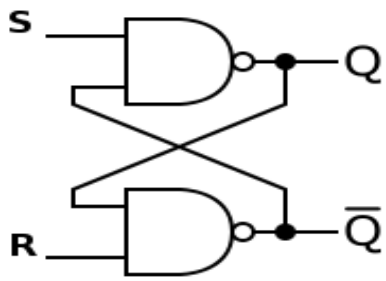


Exhibit 7.2: SR latch

Flip-flops

A flip-flop is a latch that has been modified to work with the use of a clock. Clocks are used to synchronize the timing for different components in a circuit. The output of the flip-flop will only change when the clock signal is in a given state, such as high. Exhibit 7.3 is a D flip-flop that will only change when the clock, C in the figure, is high. Some flip-flops are designed to examine the inputs when the edge of the clock makes a transition from low to high, called rising or positive edge triggered flip-flops. Similarly, negative edge triggered flip-flops can be designed that only examine inputs when the clock makes a high to low transition. The time in which the inputs can affect a change on the output is reduced with a rising or falling edge triggered device. The speed with which a flip-flop can be clocked is determined by the maximum delay from the gates that are used to construct the device. For this reason, the input to the gate should be stable prior to the clock transition and the time before the next clock pulse should last long enough for the output state to stabilize. Manufacture specifications for the device being used should be consulted to determine the maximum clock speed.

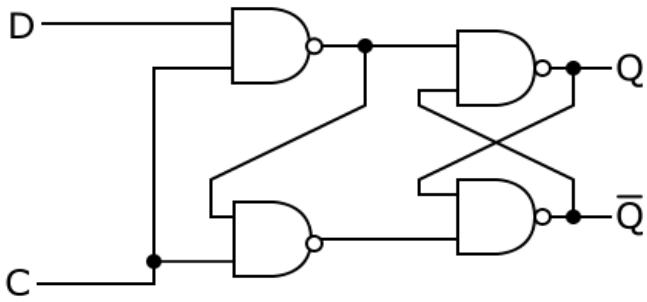


Exhibit 7.3: D flip-flop

Since these labs only use clocks with periods no faster than 1 second, the circuits designed never approach the limits of the maximum clock speed. Exhibit 7.4 uses two D flip-flops. The output of the first is used as the input of the second creating a master-slave arrangement. This results in a positive edge triggered flip-flop.

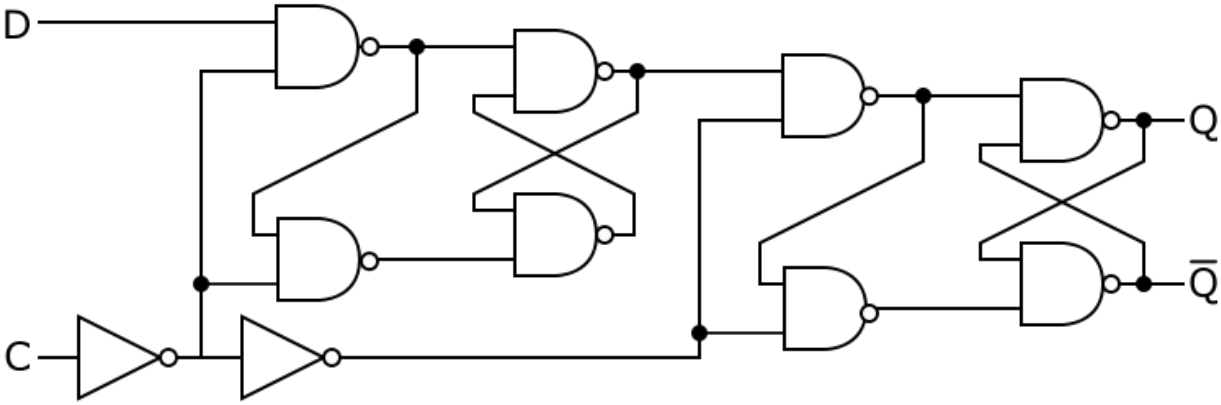


Exhibit 7.4: Positive edge triggered D flip-flop

Circuitry can be added to produce JK, T, or D flip-flops. The JK flip-flop, like the SR latch has two inputs, however all four states are valid for the JK flip-flop. The T is known as a toggle flip-flop because if the input is high, the state of the output toggles. This means that when clocked with an input of one and a current state of high, the output goes low and if it was low, it goes high. The D flip-flop output follows the value of the input while enabled or when clocked, otherwise it remains in the memory state. Both the T and D have only one data input. The tables below list the input of the flip-flop along with the present state, Q, and then the next state, Q_N. The circuit for a rising edge triggered D flip-flop is provided below. JK flip-flops are very common in many designs. For the sake of simplicity, only the D flip-flop will be used for the designs in this text.

J	K	Q	Q_N
0	0	0	0 - unchanged
0	0	1	1 - unchanged
0	1	0	0 - reset
0	1	1	0 - reset
1	0	0	1 - set
1	0	1	1 - set
1	1	0	1 - toggle
1	1	1	0 - toggle

Table 21: JK flip-flop

T	Q	Q_N
0	0	0 - unchanged
0	1	1 - unchanged
1	0	1 - toggle
1	1	0 - toggle

Table 22: T flip-flop

D	Q	Q_N
0	0	0 - off
0	1	0 - off
1	0	1 - on
1	1	1 - on

Table 23: D flip-flop

Exhibit 7.5 shows the symbolic representation of the D flip-flop used for circuit diagrams. The rectangle shown is commonly used for latches and flip-flops. Also note the bubble in front of the CLEAR line to indicate that the device can be set to low or "cleared" when this line is set low; for normal operation the CLEAR should be left high.

Some devices also come with a PRESET line that can be used to set or turn on the output in much the same manner. These lines can be used to load the flip-flops with specific values, especially when the unit is first powered on. The clock line has a small triangle that denotes that the device is triggered with a rising edge. Falling edge-triggered devices will have a small bubble preceding the triangle. For these labs, the 74175, which provides four rising edge triggered D flip-flops on a single chip, is recommended. Schematics of the 74175 can be found in Appendix A.

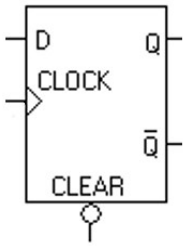


Exhibit 7.5: D flip-flop symbol

Review exercises

1. Use the SR latch from Exhibit 7.1. Assuming the values in the table represent values that have just occurred, determine the stable values for the outputs QN and QN'. Recall that the NOR gate is an active high gate, meaning any time either of the input values is high the output is low. The first, fourth, and sixth rows of the table are done for you. The truth table for the NOR is provided.

S	R	Q	Q'	Q_N	Q_N'
0	0	0	1	0	1
0	0	1	0	?	?
0	1	0	1	?	?
0	1	1	0	0	1
1	0	0	1	?	?
1	0	1	0	1	0
1	1	0	1	?	?
1	1	1	0	?	?

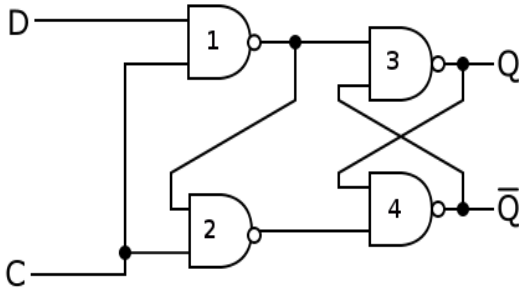
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

2. As an example, output for the first row is traced:
   a. S is 0 and Q' is 1, therefore QN stays 0.
   b. R is 0 and QN is 0, therefore QN ' stays 1
   c. Stable because Q and Q' retain values.
3. For the fourth row, the outputs toggle:
   a. R is 1, so QN' must be 0.
   b. S is 0 and QN is 0, so QN ' is 1.
   c. Stable. R is 1, Q is 0 and not affected by Q'. With S and Q 0, Q' stays
4 Tracing the sixth row yields the following:
   a. S is 1, so QN' must be 0.
   b. R is 0 and QN' is 0, so QN is 1.
   c. Stable as S is 1, QN' stays 0. With R and QN' 0, QN stays 1.To start tracing, recall that if any of the input values to a NOR are 1, the output must be 0.

5. Repeat exercise 1 with the latch from Exhibit 7.2 by determining the stable states of all 8 rows of the truth table from the previous problem. While values for QN and QN ' are provided in rows 1, 4 and 6 for the last problem, you must work all 8 rows for this problem. Remember that the NAND gate has an output of 1 if either of the input values of the gate is 0.

6. Using the D flip-flop below, determine the stable output of each of the NAND gates labeled 1 through 4 when the values for D, C, and Q first occur. The following trace for the first row serves as an example.
   a. Remember the NAND is an active low device, meaning the output will be 1 if either input is 0 (low).
   b. D and C are 0, so NAND1 and NAND2 will be 1.
   c. NAND3 is 0 and NAND2 is 1, making NAND4 1.
   d. NAND1 is 1 and NAND4 is 1, so NAND3 will stay 0.
   e. Stable as neither NAND3 or NAND4 change state.

D	C	Q	Q'	1	2	3/Q_N	4/Q_N'
0	0	0	1	1	1	0	1
0	0	1	0	?	?	?	?
0	1	0	1	?	?	?	?
0	1	1	0	?	?	?	?
1	0	0	1	?	?	?	?
1	0	1	0	?	?	?	?
1	1	0	1	?	?	?	?
1	1	1	0	?	?	?	?



7. Use the data given for the 74175 in Appendix A to determine the value of the output Q after a rising clock edge has been received by the clock pin.

D	CLEAR	Q
0	0	?
0	1	?
1	0	?
1	1	?

Procedure

1. Write the prelab in your lab notebook for all the circuits required in the steps that follow. Include all necessary equations and calculations.
2. Obtain instructor approval for your prelab.
3. Construct an SR latch using NOR gates. Verify its operation and demonstrate the circuit for your instructor.
4. Construct one bit of memory using one D flip-flop from a 74175 chip. Verify its operation and demonstrate the circuit for your instructor.

8. State machines

Learning objectives:

- Construct state transition diagrams.
- Relate the number of memory bits required for a given state machine.
- Build four-state state machines.

What is a state machine?

A state machine, often referred to as a finite state machine is a sequential logic circuit that has a finite number of defined states that can be represented. A state machine requires the use of memory to store the state of the machine. Combinatorial logic is used to combine the values of the present state along with inputs to the system to determine the next state of the system.

An example of a simple state machine could be a counter that counts from from 0 to 1 to 2 to 3 and back to 0. In this case, the state machine does not have any input at all, it uses the past state and increments the value of every clock cycle. An example of a complex state machine would be a computer. In this case, the computer can have many different inputs and has many different states. Input data can come from the keyboard, network, mouse, memory, etc., while the state would normally be associated with the address in memory of the program being run. In this text, the state machines will be like the counter just described but certainly nothing as complex as a computer.

State machines are used in more than just computers. Any process that can be defined with a given predictable algorithm can often be represented by an electronic state machine. For example, a coffee vending machine could be automated with a state machine. The states would be: waiting for correct change, select options such as cream or sugar, drop cup, dispense coffee, and dispense options. Inputs could include the cream button, sugar button, correct change indicator, and a timer to determine how long to fill the cup with coffee.

State transition diagrams

A state transition diagram is a graphical representation of the state machine. Exhibit 8.1 shows the state transition diagram for a counter that starts at 0 and goes up through 3 and then back again to 0. This machine has no input, transitioning from one state to the next at every clock pulse. Each state is marked with a circle that contains the value of the state. The arrows represent the transitions from one state to the next. The state machine shown in Exhibit 8.2.a is also a four state counter, but it uses one input. The input, labeled x, determines whether the counter continues to increment the count. When x equals 0, the counter counts and when x equals 1, it remains in the current state. The convention followed here is as follows: state values are listed inside of each state bubble and input values that determine the transition are listed next to each arrow. If the state

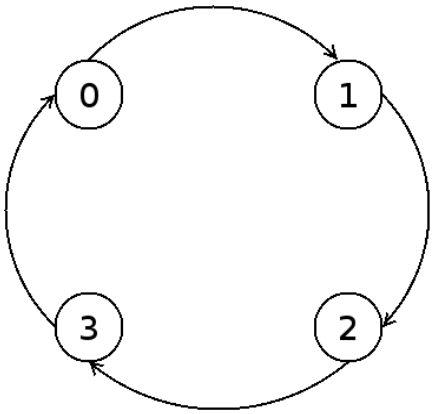


Exhibit 8.1: Four state counter

will transition regardless of any input, then no input will be listed next to that arrow. A timing diagram for this four state counter is given in Exhibit 8.2.b. This assumes that the final circuit is clocked at 1.00 seconds and that the rising edge triggered flip-flops are used. Note that the values of each bit, D1, the most significant bit, and D0, the least significant bit, only change on the rising edge of the clock, while the input is free to change at any time. This diagram serves a slightly different purpose than the timing diagram shown in a previous chapter. While the previous diagram was used to determine maximum possible delays for a circuit, this one is used to illustrate the traversal of the machine through the various states. The timing diagram, like the state diagram can be helpful when attempting to verify the operation of a constructed circuit.

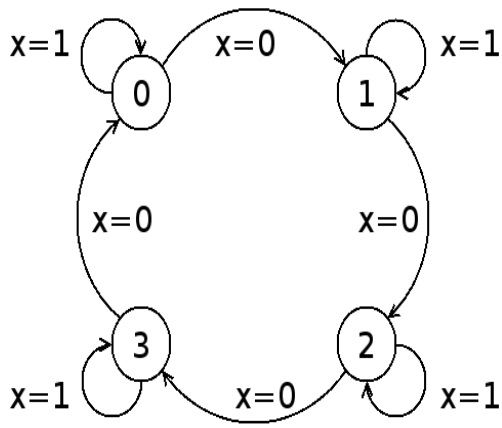


Exhibit 8.2.a: Four state counter with input

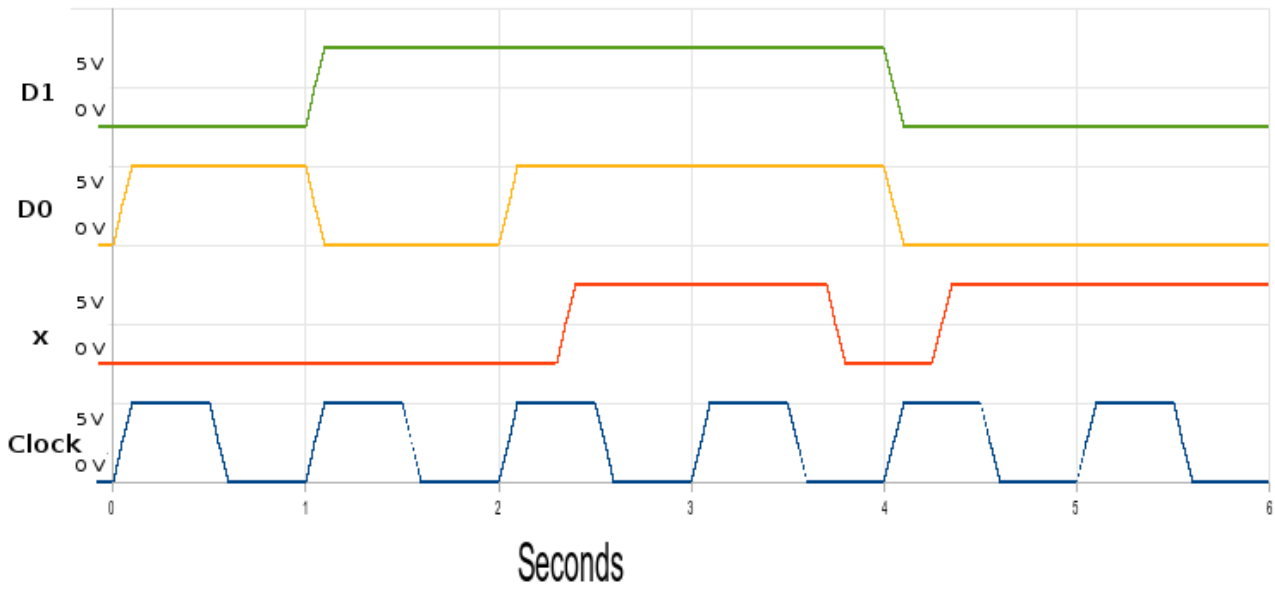


Exhibit 8.2.b: Timing diagram for four state counter

State machine design

In order to design a state machine one would need to recognize the inputs of the system, the states, and how it transitions from one state to the next. This is graphically represented with a state transition diagram. Then, the transition diagram should be used to create a truth table that has the inputs to the system and current state values as inputs in that table. The output of the truth table is the next state of the system. Combinatorial logic is used to implement the functions required to obtain the next state values for the state machine. All of the Boolean logic minimization techniques used in earlier chapters are used at this stage. As memory is used to store the states, the output or next state that results from the truth table is used as the input to the flip-flop storing the state values.

Finally, the flip-flops will need to be clocked. In these labs we want to observe the states, so the clock used has a slow period, such as 2 seconds, and a frequency of ½ hertz.

Example 1: Four state counter

The following steps outline the design of a four state counter with no input. The four state counter is re-labeled in Exhibit 8.3 to show the values taken for the required two bits of memory labeled Q1 and Q0. The values of Q1 and Q0 are given as well, which happen to follow the binary equivalent of the value of the counter. The table below shows how the present state, given by Q1 and Q0, transition at the next clock signal to the next state, given by Q_{1N} and Q_{0N}.

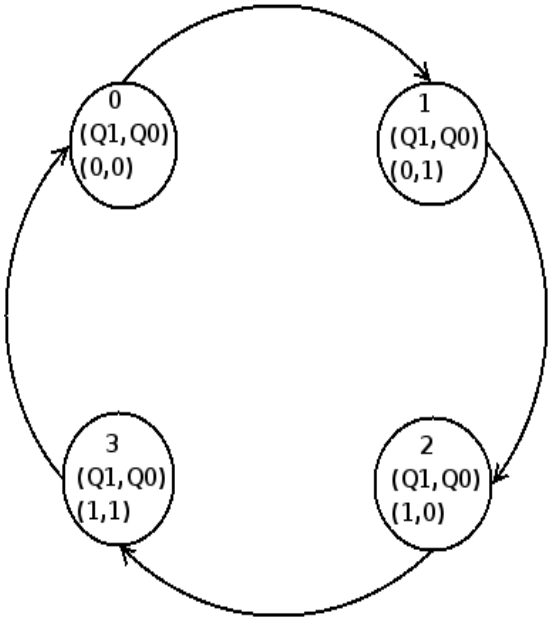


Exhibit 8.3: Four state counter with states

Q_1	Q_0	Q_{1N}	Q_{0N}
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Table 24: Truth table

The next step is to determine the functions that represent values of the next state, Q_{1N} and Q_{0N}. As these functions only have two variables, they are fairly easy to determine without the use of complex boolean algebra or K-maps. Q_{0N} is just the inverse of Q0. Q_{1N} is the Exclusive OR of the two inputs. As the logic kit does not contain an Exclusive OR gate, the equivalent logic using AND and OR gates is given along with the equivalent logic using NAND gates only. The resulting circuit is shown in Exhibit 8.4.

$Q_{0N}(Q1,Q0) = Q0'$

$Q_{1N}(Q1,Q0) = Q1 \oplus Q0$

$= Q1Q0' + Q1'Q0$

$= ((Q1Q0')'(Q1'Q0)')'$

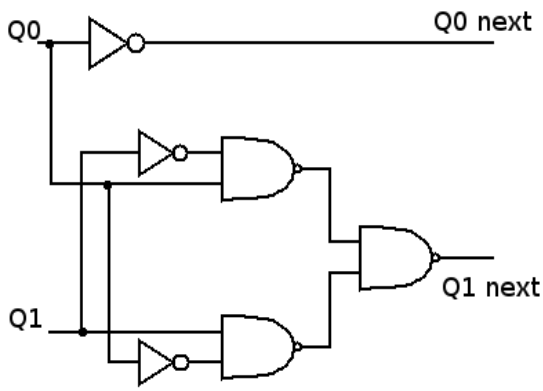


Exhibit 8.4: Counter Logic

Now, in order to create a fully functional circuit, memory needs to be included. In this case, two D flip-flops from a 74175 chip will be used. Because the 74175 chip provides the output Q as well as its inverse, the design can be simplified by eliminating the inverters from the diagram in Exhibit 8.4. The full schematic of the circuit is shown in Exhibit 8.5 along with pinouts for each chip. A switch can be used to clock the circuit for test purposes. A clock, such as one designed with a 555 timer from the previous chapter, should be used in any final design.

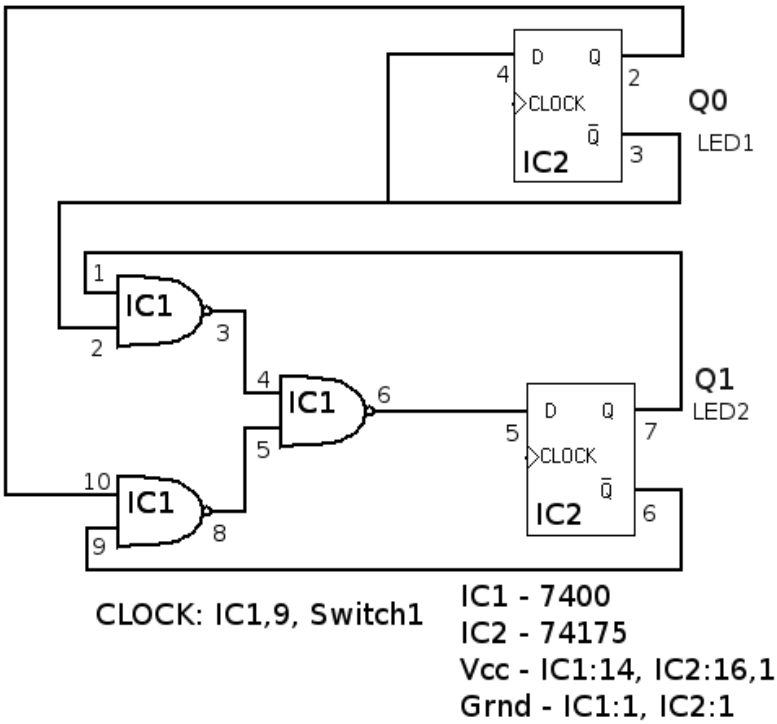


Exhibit 8.5: Counter Circuit

Example 2: Four state counter with input

The four state counter given in Exhibit 8.2.a introduces a complexity by adding an external input. The state transition diagram is redrawn in Exhibit 8.6 with the states labeled in binary, Q1 being the most significant bit. The truth table using the three items as input: x, Q_1 and Q_0 , and the output given by the next state values Q_{1N} and Q_{0N} is given in Table 25.

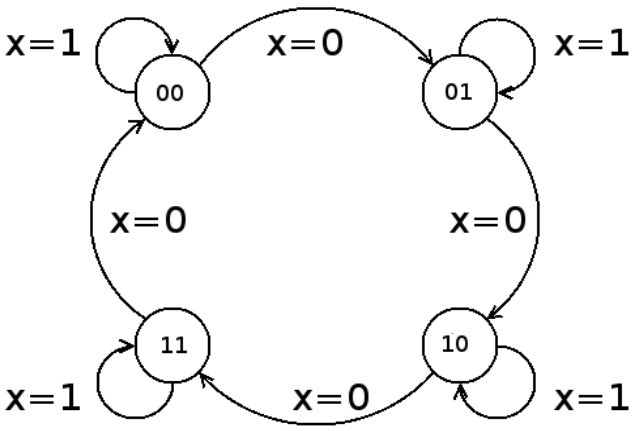


Exhibit 8.6: Four state counter with input

x	Q_1	Q_0	Q_{1N}	Q_{0N}
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Table 25: Counter truth table

The values of the outputs for Q_{1N} and Q_{0N} are then listed in K-maps to determine the minimal SOP expressions. Equivalent expressions using only NAND gates are given.

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	0	1	0	1
x 1	0	0	1	1

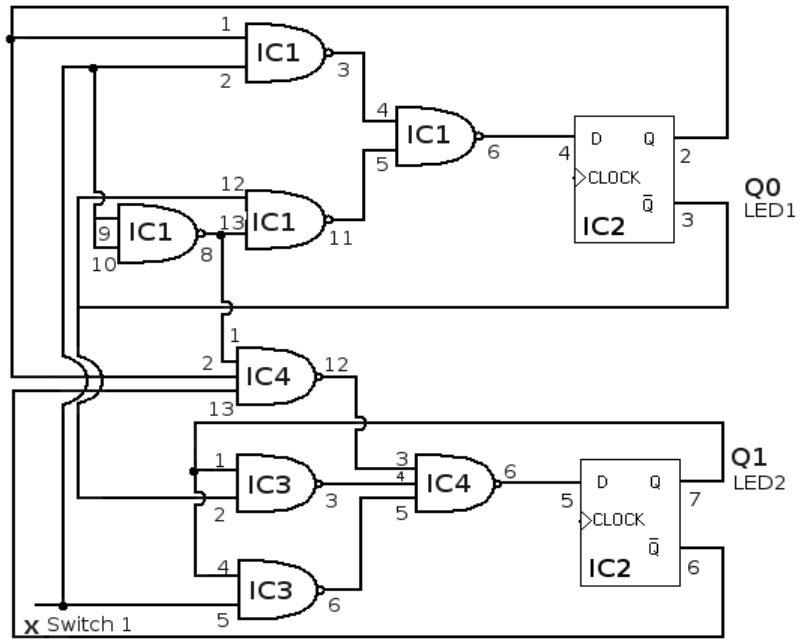
Table 26: $Q_{1N}(x,Q1,Q0)$

$$Q_{1N} = x'Q1'Q0 + xQ1 + Q1Q0'$$
$$= ((x'Q1'Q0)' (xQ1)' (Q1Q0')')'$$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	1	0	0	1
x 1	0	1	1	0

Table 27: $Q_{0N}(x,Q1,Q0)$

$$Q_{0N} = xQ0 + x'Q0'$$
$$= ((xQ0)' (x'Q0')')'$$



CLOCK: IC2,9, Switch 2

IC1, IC3 - 7400
IC2 - 74175
IC4 - 7410
VCC - IC1, IC2, IC4:14,
IC2:1,16
Grnd - IC1, IC2, IC3, IC4: 1

Exhibit 8.7: Circuit diagram for four state counter with input

The logic is then implemented using the 7400 series chips, as shown in Exhibit 8.7. The output of the logic is used to feed the input of each D flip-flop and the output of each flip-flop is used as input for the logic. Note that the CLEAR line for the 74175 must be tied to Vcc. The CLEAR line can be used on power up to clear or set the flip-flop value to logic zero. However, if the line is kept low, the value of the flip-flop will always remain at logic low. The CLEAR can be left to float, however this makes the flip-flop susceptible to fluctuations in electrical noise. The use of the CLEAR line will be discussed in more detail in the next chapter. For now the CLEAR will just be tied to logic

high. For testing purposes, a switch can be used for the clock. However, make sure to read the next section regarding debounced switches before using a switch for this purpose.

Debounced switches

One word of caution is in order when using switches as the clock source. As a switch is a mechanical device, they can suffer from bounce. Bounce occurs when the metal contacts strike each other and "bounce" before they come to rest. When this occurs, it can look like the switch changes state multiple times even though it has only gone from open to closed. Switches come in a variety of configurations. Two common versions are the single pole double throw, SPDT or the single pole single throw, SPST shown below.

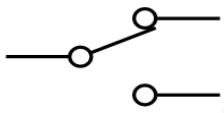


Exhibit 8.8: SPDT switch

Exhibit 8.9: SPST switch

Different approaches exist for "debouncing" switches. Software can be used to test the output of the switch and insure that only one change is registered, instead of the multiple changes that can occur with bouncing. Two common hardware approaches are provided for both types of switch. The values for resistors and capacitors shown should be chosen so that the time is as long as the system bounce is expected to last. Values of 100KΩ for the resistors and 0.1μF for the capacitor would provide a pulse of 1.1 msec, which should be sufficient to debounce the switch in Exhibit 8.11. Of course other circuits can be used to debounce switches and adjustments may need to be made to the values of the components to suit the application. The logic kit provided should have at least two debounced switches.

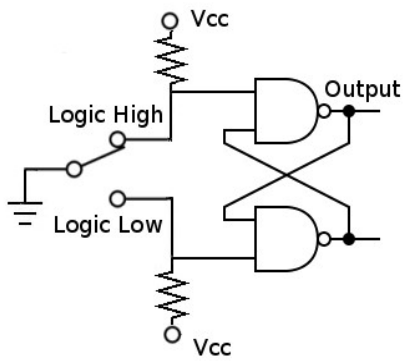


Exhibit 8.10: Debounced SPDT switch

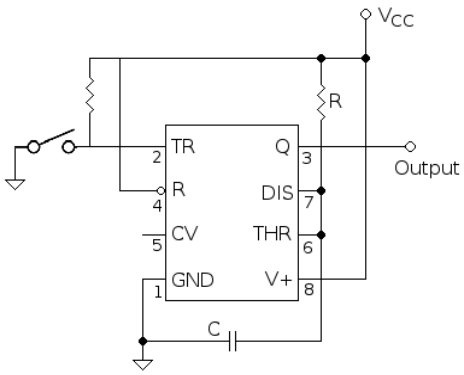


Exhibit 8.11: Debounced SPST switch

Review exercises

1. How could using a regular switch as the clock source affect the operation of the counter?
2. Draw a timing diagram for the machine that uses the state transition diagram found in Exhibit 8.3. Assume that the machine will use a clock with a period of 1.00 seconds, that the flip-flops used for the design are rising edge triggered and that the machine is in state 01 prior to time zero and that the machine goes through 4 clock pulses.
3. Draw the state diagram for a four state counter with one input where the counter counts up in binary when the input is low and counts in reverse when the input is high.
4. How many D flip-flops are required for the counter from Exercise 3?
5. Determine the logic required for the input of the four state counter from problem 3 and draw a circuit diagram with pinouts.
6. Draw the state diagram for a three-state state machine that counts from 00 → 01 → 10 → 00 etc. as long as the input is low. When the input is high, the counter does not count and stays at its current state.
7. How many D flip-flops are necessary for the counter from the previous problem? Are all of the possible states for the flip-flops used? If not which ones are not?

Procedure

1. Write the prelab in your lab notebook for all the circuits required in the steps that follow. Include all necessary equations and calculations.
2. Obtain instructor approval for your prelab.
3. Build and demonstrate the successful operation of the four state counter found in Exhibit 8.5. Attempt to clock the circuit with both a regular switch and debounced switch. Note the difference in performance.
4. Build and demonstrate the successful operation of the four state counter from Exercise 5 of the review exercises.

Optional

1. Build and demonstrate the successful operation of the four state counter from Exercise 6 .

9. More state machines

Learning objectives:

- Relate number of states to required amount of memory.
- Insure state machines do not enter illegal states.

How many bits of memory does a state machine need?

The amount of memory or number of flip-flops required for a state machine is directly related to the number of states in the state transition diagram. The number of possible states that can be represented increases by a power of two for each new bit of memory added. One bit of memory can represent 2^1 or two states, two bits can represent up to 2^2 or four different states, and three bits up to 2^3 or eight different states. To reduce the complexity of the design, use the fewest number of flip-flops that would still accomplish the task successfully. If the design required a number of states that is not a power of two, then the smallest number of bits raised to the power of two that is greater than the number of states required should be used. As an example, if three states were required, two bits would be needed, or if six states were required, three bits would be needed.

$$\text{Number of states} \leq 2^{\text{number of bits}}$$

What are unused states?

A machine that visits the following states in the order listed, 000 → 001 → 011 → 111 → 110 → 100 → 000, will require three bits of memory. What becomes of the unused states, 010 and 101? Several approaches are common when dealing with the unused states. Note that any legal state moves to another legal state, never visiting the unused states. Because the unused states are never visited, these could be considered as don't care conditions in the K-maps for the input to the flip-flops. This can reduce the complexity of the design. In addition, PRESET and CLEAR lines for flip-flops can be used to insure not only that the state machine enters a legal state when powering on, but it also insures that it powers up in a specific initial state.

Using PRESET and CLEAR pins

As the system powers up, logic levels cannot always be guaranteed. What if during this time, the system happened to enter one of the unused states? Depending upon the logic that was used, the system may then transition into one of the legal states, or it may get stuck indefinitely in one of the illegal states. In order to guarantee that the system does not enter an illegal state as the system powers up, the CLEAR lines can be held low temporarily to insure that the memory bits are set to zero or logic low on power up. Exhibit 9.1 has an RC circuit that can be used to power on to keep the CLEAR line low long enough to insure that the bits are set to zero. When power is first turned on, the capacitor will be uncharged and must charge through the resistor. If the time constant, RC, is set at several clock cycles, then the state machine will be guaranteed to start with all of

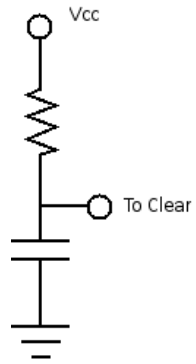


Exhibit 9.1: RC for Power On

the memory bits at zero. The 74175 quad D flip-flop in the logic kit does not offer a PRESET pin. However the same type of RC circuit can be used for other flip-flops that do.

Assigning unused states to the system

Powering up is not the only time the machine can enter an unused state. At times large transient spikes can occur during storms or when powering on or off other equipment that can cause logic levels to change unpredictably. In cases such as this, the machine can still enter a state that was not planned. Even the RC circuit connected to CLEAR or PRESET pins cannot rescue the state machine in this case. To address this, the designer should add the additional states to the state transition diagram and simply have them transition to a legal state. In this way, even if for some reason a circuit enters an illegal state, it will quickly shift to one that is allowed.

Adding the extra states as well as the RC circuit does indeed complicate the circuit, however for a final design that will be used in production, it provides assurance that the circuit will perform reliably even when the unexpected occurs.

Example 1: Three state counter

The three state counter in Exhibit 9.2 counts up when the input is high and counts down when the input is low. Two flip-flops will be needed to implement this machine which means that four states can be represented by those two bits. The state 11 is not used in this design. What would happen if for some reason, the machine would enter the state 11? The effect of entering this unused or illegal state cannot be known until the circuit implementation is finalized. Instead of waiting to see what happens after the design is completed, it is best to incorporate this state early on in the design phase. Two approaches will be investigated. The first will shift the state 11 to the legal 00 state on the next clock cycle. The next approach will be to place don't care conditions for the state 11 and then examine the next state that would follow depending upon the simplest design that results from using the don't care conditions.

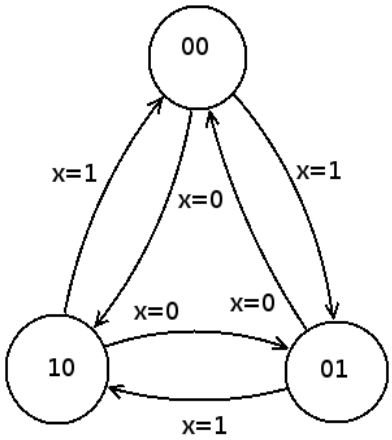


Exhibit 9.2: 3 state counter

Approach 1: 11 → 00

The resulting state transition diagram assuming that state 11 transitions to 00 on the next clock cycle is given in Exhibit 9.3. In all of the cases that follow, unused states will be shown as dotted circles in the state transition diagram. Since state 11 will move to state 00 regardless of the input value, it is not written on the diagram. From the K-maps given below, the next state values for Q_1 and Q_0 are listed as Q_{1N} and Q_{0N}. It is left as an exercise for the reader to determine the circuitry required to implement this state machine.

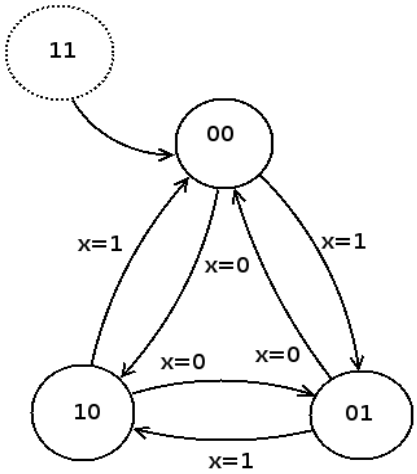


Exhibit 9.3: 3 state counter with unused state

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	1	0	0	0
x 1	0	1	0	0

Table 28: $Q_{1N}(x,Q_1,Q_0) = Q_{1N} = x'\,Q_1'Q_0' + xQ_1'Q_0$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	0	0	0	1
x 1	1	0	0	0

Table 29: $Q_{0N}(x,Q_1,Q_0) = xQ_1'Q_0' + x'Q_1Q_0'$

Approach 2: Using don't care conditions

The next approach instead places don't care conditions for the state 11 as seen in the K-maps below. By selecting the minimal expressions, the next states for Q1 and Q0 can be found. The resulting expressions can be found to be less complex than those from the first approach.

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	1	0	d	0
x 1	0	1	d*	0

Table 30: $Q_{1N}(x,Q_1,Q_0) = xQ_1'Q_0' + x'Q_0$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	0	0	d**	1
x 1	1	0	d	0

Table 31: $Q_{0N}(x,Q_1,Q_0) = xQ_1'Q_0' + x'Q_1$

The resulting state transition diagram is given in Exhibit 9.4.a. Using the don't care conditions does simplify the logic. Notice that the unused state now goes to two different states depending upon the value of the input. The don't care condition labeled as d*, which is xQ_1Q_0 is grouped with the term $xQ_1'Q_0$. This results in a simpler grouping, xQ_0, but it does now cause the machine to transition from 11 to 10 when the input x is a logic high. Similarly, the term d** now causes the state 11 to transition to 01. The remaining don't cares are not contained in a group, so the they will transition to 0 at the next state. It is left to the designer to carefully examine the requirements of the final circuit to determine if indeed these are don't care conditions. If so, then the transition diagram should be updated to reflect their use in the logic simplification.

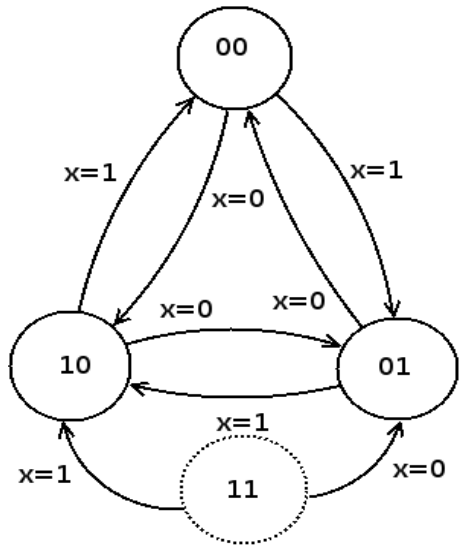


Exhibit 9.4.a: 3 state counter with don't cares

A sample timing diagram that starts on the unused state 11 and cycles through this new diagram in Exhibit 9.4.a is given in Exhibit 9.4.b. Notice that the first transition at time 0 is to the state 01. From there the counter counts in reverse as the input is low, transitioning at time 1 to state 00, at time 2 to state 10, and then back to state 01 at time 3. Somewhere between time 3 and 4 input x goes high, but the state does not change until the next rising clock edge at time 4. From that point on, with the input high, the counter counts up. This assumes that the circuit will use rising edge triggered flip-flops.

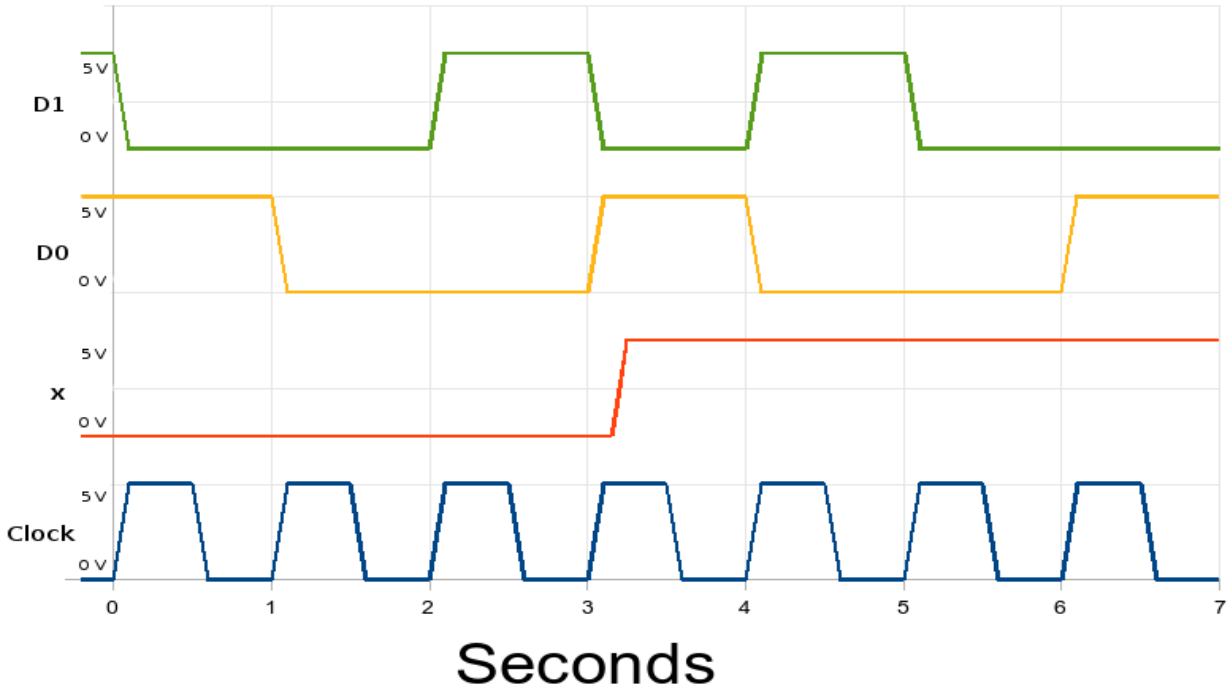


Exhibit 9.4.b: Timing diagram for 3 state counter

Example 2: Five state machine

The five state machine shown in Exhibit 9.5 has two different loops. One of the loops transitions between 000 and 111 while the other goes from 001 to 010 to 100. This leaves three possible states that are unused. The truth table that follows uses don't care conditions for unused states 011, 101, and 110 given as d1, d2, and d3 respectively. The resulting K-maps that follow can be used to determine the minimal expressions. If the unused states were to immediately go to next state 000, then the minimal expressions can be shown as those listed below. It is left as an exercise to draw the new state transition diagram for this design.

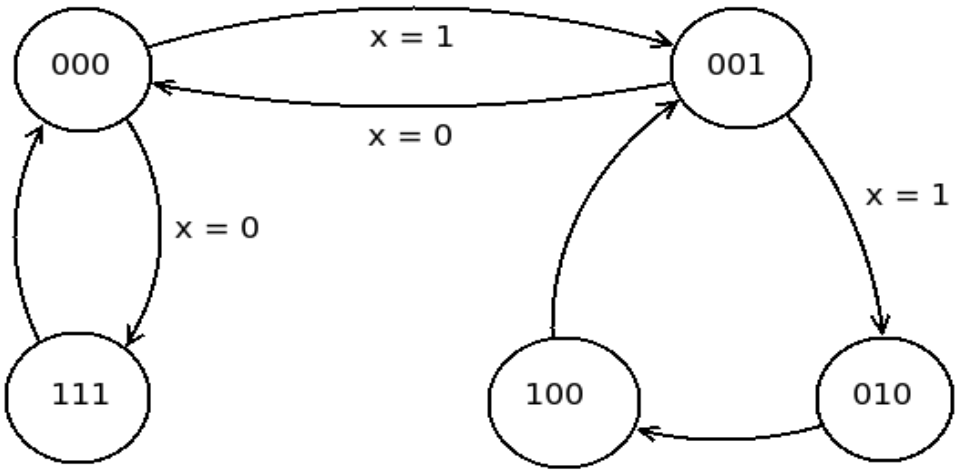


Exhibit 9.5: Five state counter

$Q_{2N} = x'Q_2'Q_0' + Q_2'Q_1Q_0'$

$Q_{1N} = x'Q_2'Q_1'Q_0' + xQ_2'Q_1'Q_0$

$Q_{0N} = Q_1'Q_0'$

Now, if the don't care conditions are used in the design for the minimal expressions, the complexity of the results is reduced.

$Q_{2N} = x'Q_2'Q_0' + Q_2'Q_1$ or $x'Q_2'Q_0' + Q_1Q_0'$

$Q_{1N} = x'Q_2'Q_1'Q_0' + xQ_2'Q_0$ or $x'Q_2'Q_1'Q_0' + xQ_1'Q_0$

$Q_{0N} = Q_1'Q_0'$

The results for Q_{1N} Q_{2N} have two equally minimal forms. The state transition diagram that uses the first minimal form is given in the Exhibit 9.6. Notice that the unused state 011 goes to the legal state 100 if the input is logic high and another unused state, 110 when the input is a logic low. To trace where the external states will go, examine d1 which corresponds to unused state 011. For Q_{2N}, d1 is part of group $Q_2'Q_1$. Q_{2N} will be 1 regardless of the input at the next state. d1 is only grouped if x is 1 for Q_{1N} and not at all in Q_{0N}.

x	Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	1	d1	d1	d1
0	1	0	0	0	0	1
0	1	0	1	d2	d2	d2
0	1	1	0	d3	d3	d3
0	1	1	1	0	0	0
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	d1	d1	d1
1	1	0	0	0	0	1
1	1	0	1	d2	d2	d2
1	1	1	0	d3	d3	d3
1	1	1	1	0	0	0

Table 32: Truth table for 5 state machine

It is left to the designer of the machine to determine if these transitions are acceptable given the specifications for the product.

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
$x'Q_2'$ 00	1	0	d1	1
$x'Q_2$ 01	0	d2	0	d3
xQ_2 11	0	d2	0	d3
xQ_2' 10	0	0	d1	1

Table 33: Q_{2N}

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
$x'Q_2'$ 00	1	0	d1	0
$x'Q_2$ 01	0	d2	0	d3
xQ_2 11	0	d2	0	d3
xQ_2' 10	0	1	d1	0

Table 34: Q_{1N}

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
$x'Q_2'$ 00	1	0	d1	0
$x'Q_2$ 01	1	d2	0	d3
xQ_2 11	1	d2	0	d3
xQ_2' 10	1	0	d1	0

Table 35: Q_{0N}

9. More state machines

It should be noted that all of the designs shown in this text have used only the D flip-flop. However, it can often be the case that another type can result in a simpler design. JK flip-flops can be used to produce ripple counters with minimal extra circuitry. The JK flip-flop does have two inputs, so the resulting logic minimization must be done for both the J and the K input, doubling the number of K-maps required. In order to reduce the required number of parts for the logic kit, only the D flip-flop was used. Designers should become familiar using all of the different types of flip-flops so that they can be assured that they have chosen the one that truly results in a minimal design.

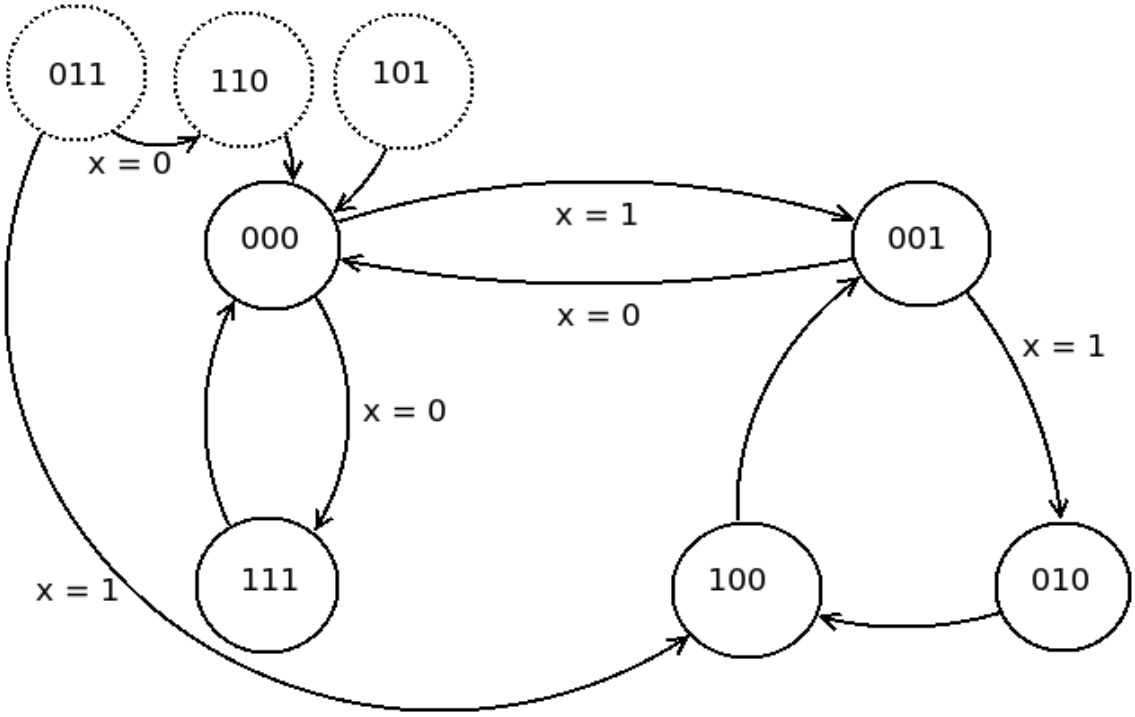


Exhibit 9.6: Five state counter with unused states

Review exercises

1. A state machine requires 7 different states. How many flip-flops are required for this machine?
   (a) If a machine has no external inputs, what size is the K-map for one of the required inputs?
   (b) If the machine has one external output, how large is the K-map for one of the flip-flop inputs?
   (c) If the design were to use JK instead of D flip-flops, how many next state inputs must be determined?
2. Repeat Exercise 1 for a state machine with 14 states.
3. Draw six clock pulses of the timing diagram for the machine that uses the state transition diagram found in Exhibit 9.6. Assume that the clock for the machine has a period of 1.00 seconds, that the machine is in state 011 prior to time zero and that input x is kept at logic high the entire time.
4. A state machine traverses the states listed in this order 000 → 001 → 011 → 111 → 110 → 100 → 000. There is no external input.
   (a) Draw the state transition diagram for this machine.
   (b) What are the unused states?
   (c) Modify the diagram if the unused states transition to 000.

(d) Assuming a state machine were to be built using D flip-flops, determine the value of the next state for each of the flip-flops.

5. The two bit sequence 00 → 01 → 11 → 10 → 00 is a Gray code. Gray codes only have one bit change for each transition.

(a) Sketch the state transition diagram for the 3 bit Gray code: 000 → 001 → 011 → 010 → 110 → 111 → 101 → 100 → 000 … .

(b) Assuming a state machine were to be built using D flip-flops, determine the value of the next state for each of the flip-flops.

6. A two bit counter is to be built that will count forward, 00 → 01 → 10 → 11 → 00, when a logical input is set high and counts in reverse order when it is low.

a. Draw the state transition diagram for this state machine.

b. Assuming a state machine was to be built using D flip-flops, determine the value of the next state for each of the flip-flops.

7. A two bit counter is to be built that will count forward, 00 → 01 → 10 → 11 → 00, when a logical input is set high and as a Gray code when it is low (00 → 01 → 11 → 10 → 00).

(a) Draw the state transition diagram for this state machine.

(b) Assuming a state machine was to be built using D flip-flops, determine the value of the next state for each of the flip-flops.

Procedure

1. Write the prelab in your lab notebook for all the circuits required and the steps that follow. Include all necessary equations and calculations.
2. Obtain instructor approval for your prelab.
3. Your instructor will pick one or more state machines from the various examples from the review exercises for you to build and demonstrate.

10. What's next?

Hopefully, this introduction has whetted your appetite for this fascinating subject. Modern technology simply would not be possible without the advances and applications of this subject in the world in which we live. All of the sequential circuits shown in the chapters “State machines” and “More state machines” are synchronous, meaning they use a clock. However, sequential circuits designed without clocks, known as asynchronous circuits, can be designed. As the clock can often insert added delay for the faster components in the circuit, asynchronous circuits can usually be designed that will respond even faster than synchronous circuits. Timing issues become critical in this case, and the resulting timing analysis can become so complicated that asynchronous circuits are often not chosen over their synchronous counterparts. However, for circuits that require the fastest speed possible, often asynchronous circuits are considered.

In addition, while the circuits designed in these labs all used discrete components, for circuits that are used in applications today, nearly all of the components are fabricated on a single chip. Either Programmable Logic Devices (PLDs) can be used to fit entire state machines on a single chip or custom chips can be fabricated for a specific task. Very large-scale integration (VLSI) techniques are used to design entire systems on a single chip; a CPU with cache memory and a graphics processing unit would be an example. Complexities that require additional analysis are when the size of the transistors is decreased, speeds of the circuits are increased, and the desired power consumption is lowered. Hardware description languages such as Verilog can even be used to synthesize and test circuit performance virtually in software before constructing a single device.

Any one of these areas can provide a wealth of challenging problems to tackle. It is the hope of this author that the foundation gained from this text will prove useful as you use technology and design applications that require digital logic.

Appendix A: Chip pinouts

7400

Inputs		Output
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

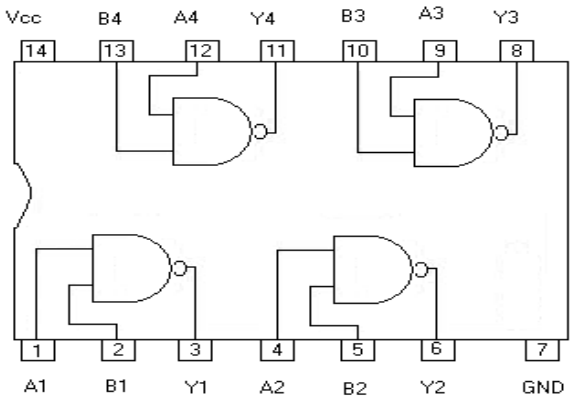


Exhibit A.1: 7400

7402

Inputs		Output
A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

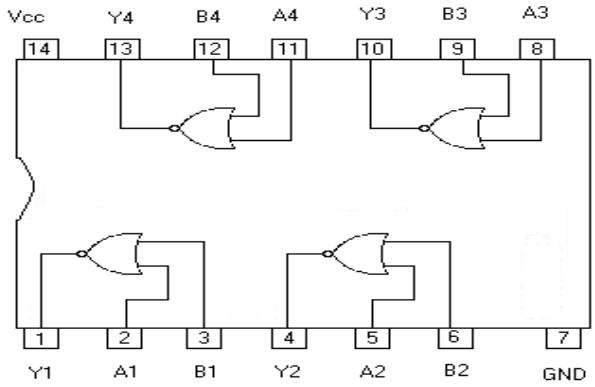


Exhibit A.2: 7402

7404

Inputs	Output
L	H
H	L

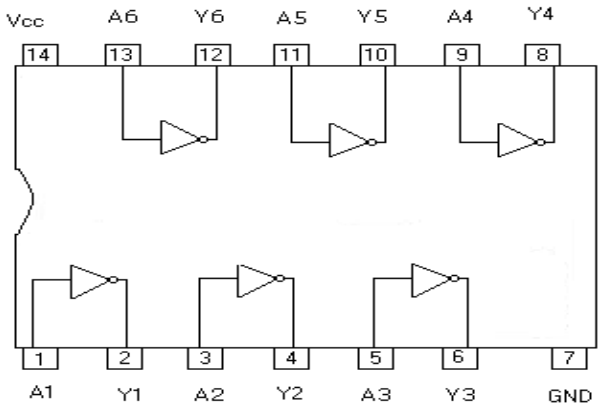


Exhibit A.3: 7404

Appendix A: Chip pinouts

7410

Inputs			Output
A	B	C	Y
X	X	L	H
X	L	X	H
L	X	X	H
H	H	H	L

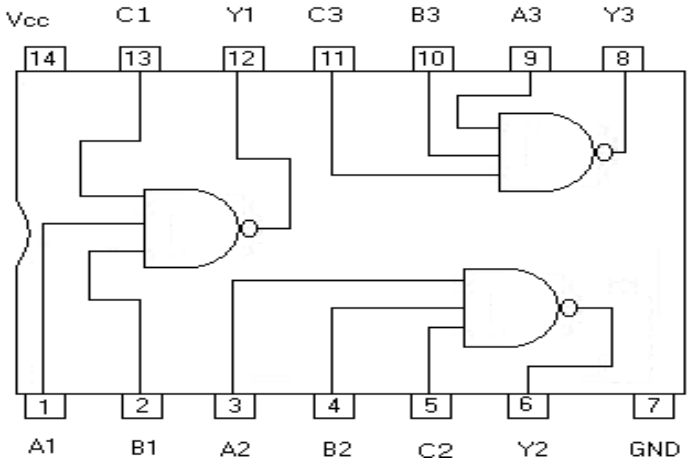
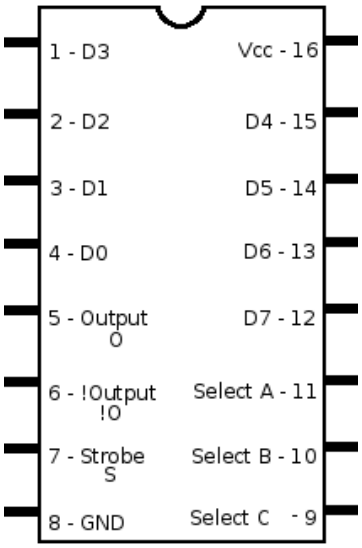


Exhibit A.4: 7410



C	B	A	S	!O	O
X	X	X	1	1	0
0	0	0	0	*D0'*	D0
0	0	1	0	*D1'*	D1
0	1	0	1	*D2'*	D2
0	1	1	1	*D3'*	D3
1	0	0	0	*D4'*	D4
1	0	1	0	*D5'*	D5
1	1	0	1	*D6'*	D6
1	1	1	1	*D7'*	D7

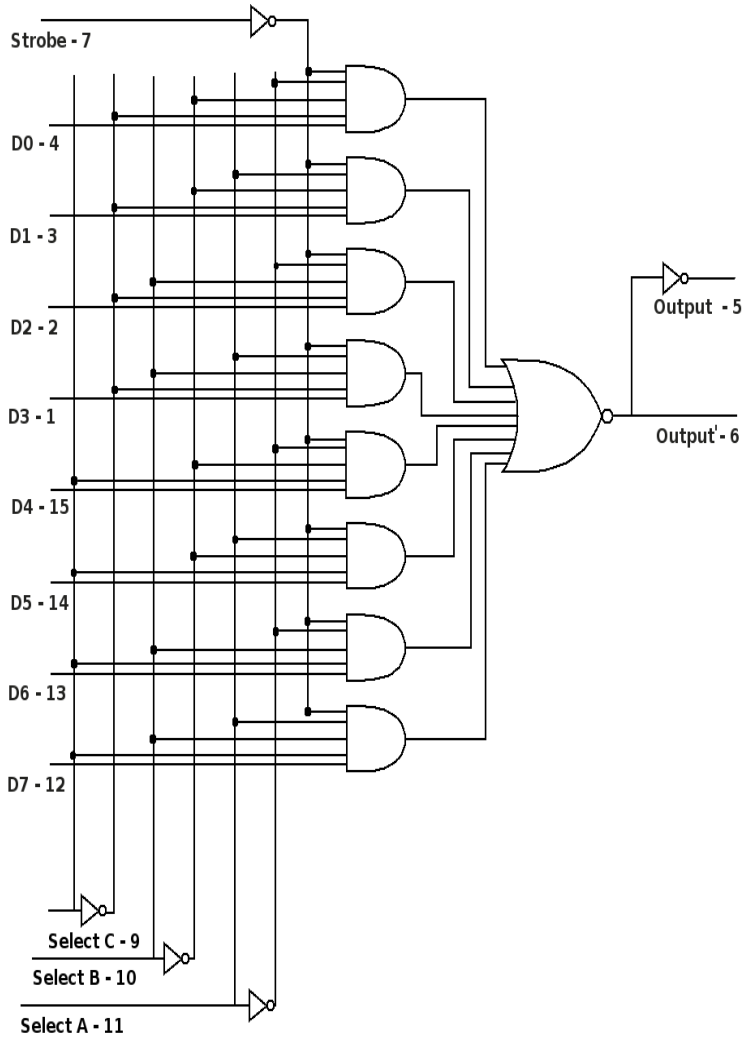
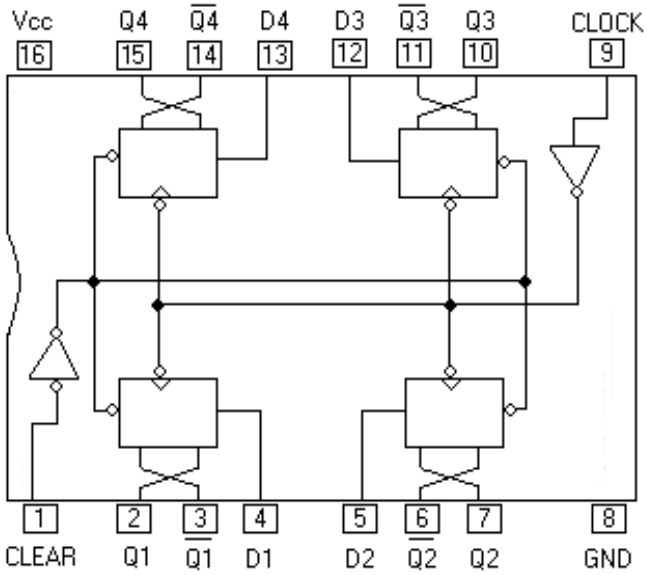


Exhibit A.5: 74151

74175



Inputs			Outputs	
Clear	Clock	D	Q	$\overline{Q}$†
L	X	X	L	H
H	↑	H	H	L
H	↑	L	L	H
H	L	X	Qo	$\overline{Qo}$

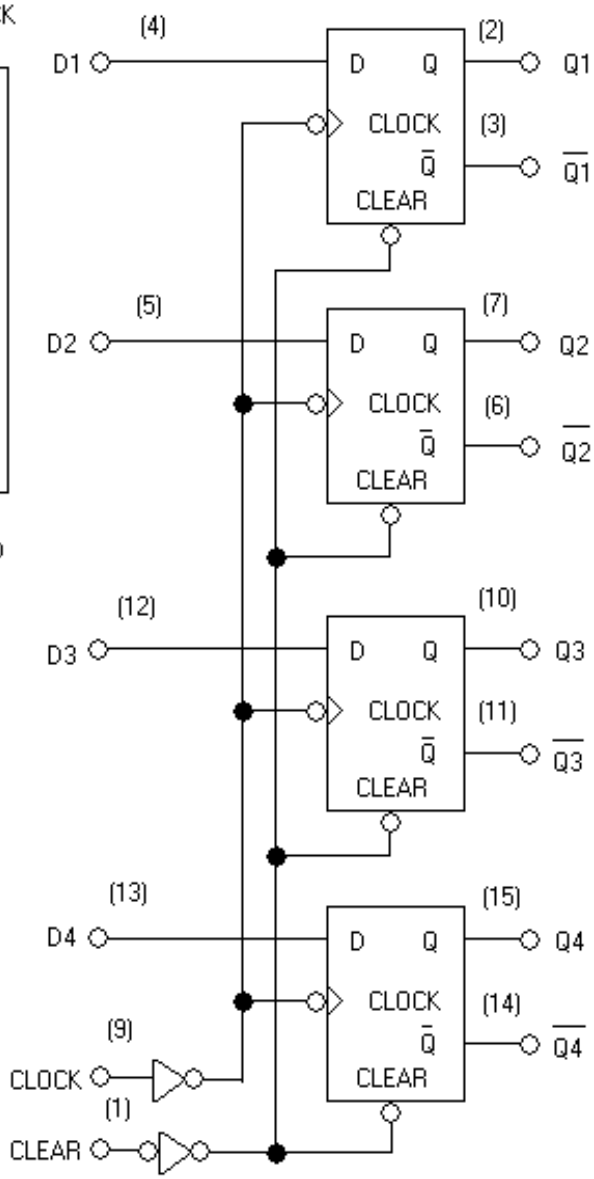
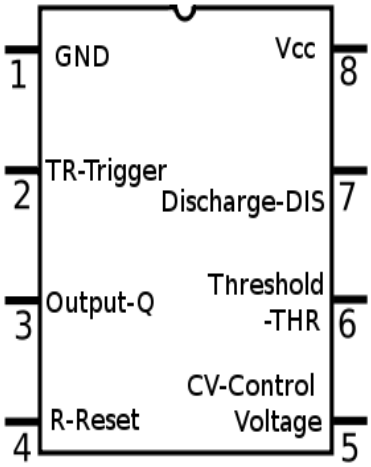


Exhibit A.6: 74175

Appendix A: Chip pinouts

555 Timer



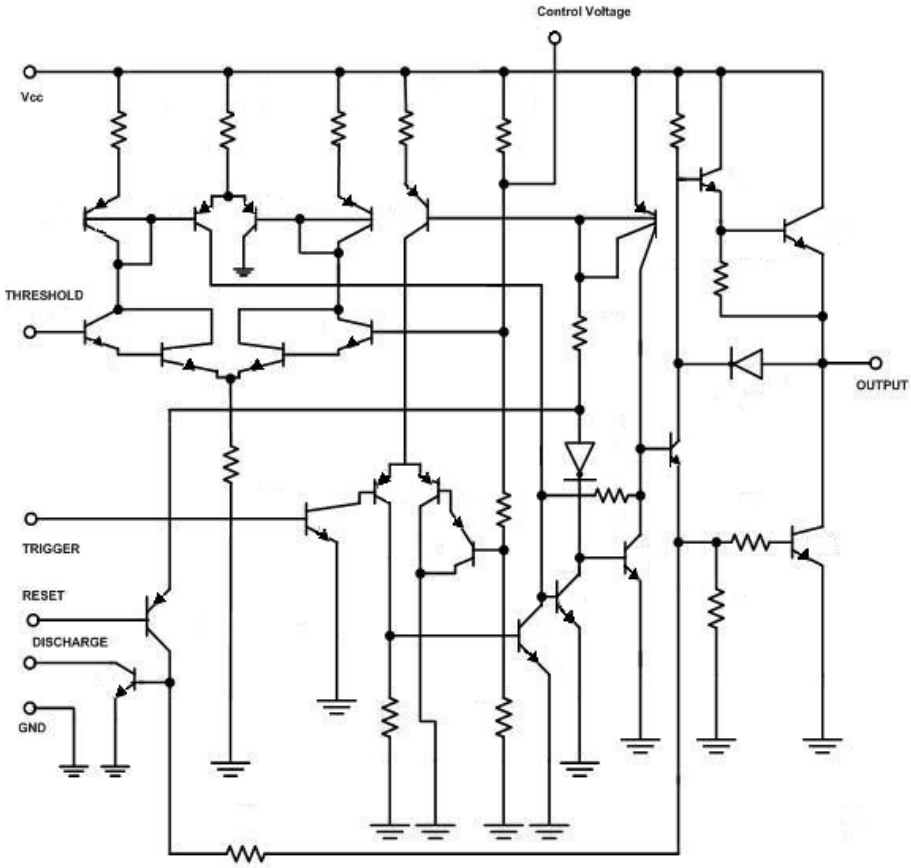


Exhibit A.7: 555 timer

Appendix B: Resistors and capacitors

Resistors

Resistors are electronic components that obey Ohm's law: Voltage across a resistor is equal to the current through the resistor times the resistance of the device.

$$V = I * R$$

Resistance is measured in ohms ( ). Current and voltage are related by the resistance of the object, if voltage is kept constant and resistance rises, current will fall. Likewise if resistance decreases, more current will flow, meaning the measure of the current will rise. While many devices have resistance, including the wire used in these labs, the only resistance that we will be concerned with in this manual is the resistance attributed to actual resistors. Manufactured resistors come in various forms, however those used here will be standard ¼ watt resistors that follow the conventional color code that describes their value.

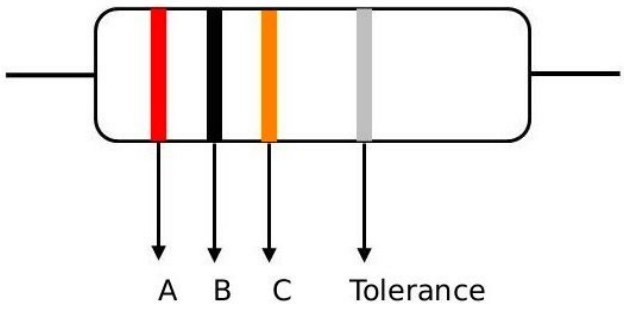


Exhibit B.1: Sample Resistor

COLOR	VALUE	MNEMONIC
Black	0	***B**etter*
Brown	1	***B**e*
Red	2	***R**ight*
Orange	3	***O**r*
Yellow	4	***Y**our*
Green	5	***G**reat*
Blue	6	***B**ig*
Violet	7	***V**enture*
Gray	8	***G**oes*
White	9	***W**est*

Table 36: Color Codes

Each resistor has four colored stripes as shown in the figure above. Each stripe corresponds to a number as shown in Table 36. The formula for the value of each resistor is listed below.

Generic Formula: A B x 10^C

Which for this case yields: 2 0 x 10^3 or 20,000 Ω.

The first two stripes indicate the numerical value of the resistance, the third the exponent of ten which will be multiplied by the numbers from the first two stripes, and the fourth a tolerance of the resistor. The diagram above illustrates how the first three stripes are used to calculate the value of the resistor as well as the diagram below. The mnemonic is often suggested as a means of remembering the color

code. The tolerances will not be utilized in this lab manual. Another example is provided in Exhibit B.2. Applying the formula to obtain the value for this resistor is left as an exercise for the reader.

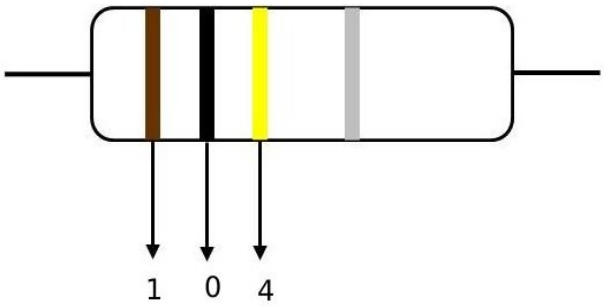


Exhibit B.2: 100,000 Ohm Resistor

Capacitors

In direct current circuits, capacitors can be thought of as charge storage devices. Electrolytic capacitors will be used in these labs. Electrolytic capacitors appear to look like a tiny aluminum can with two wires. Be cautious when connecting the electrolytic capacitors as they have a polarity. Insure that the negative terminal of the capacitor is connected properly or the capacitor can malfunction and in some cases explode! The unit of measurement for capacitors is the Farad. Capacitors with higher Farad measurements can store more charge at a given voltage.

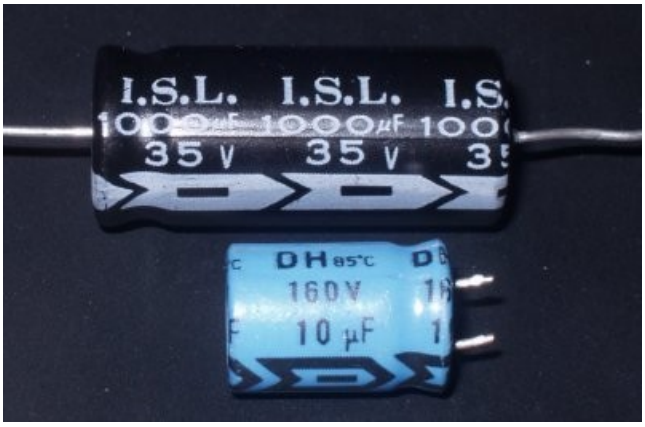


Exhibit B.3: Capacitors

Appendix C: Lab notebook

The lab notebook should be a bound notebook, much like a standard composition notebook. The lab notebook is used to document the experiment or lab procedure. Notebooks can serve many purposes: for the author to review the material, for someone else to replicate the procedure, or even as a legal document for use in patent or court proceedings. The notebook for these experiments will be informal, in that the student will hand write all of the content in the notebook. Do not misinterpret the meaning of informal, because the work should still be neat, legible, well organized, and complete. What follows are some guidelines that should be used to document the labs from this text. Of course your instructor may add or delete from this list.

The lab notebook should:

- be bound
- have two to three pages at the front dedicated to a table of contents
- have numbered pages to use in the table of contents (you may number them yourself)

Each lab should contain:

- name of lab
- your name
- partner(s) name(s)
- date
- brief objective of lab (no more than two sentences)
- equipment list required
- pre-lab including:
    - any necessary diagrams
    - any necessary equations and calculations
    - approval of instructor before you begin the lab exercise
- results and observations
- conclusion

Make sure that you:

- Do not erase any items. Cross them out and redo the work.
- Write only on the right side of each page. This leaves you room to include any corrections.

While following these guidelines certainly makes it easier for your instructor to review your work, that is not its main purpose. Keep in mind, someone should be able to understand what you did and even replicate your work given your lab notebook. Your lab notebook can be a helpful document for you. In industry, it can also be a helpful document for others.

Appendix D: Boolean algebra

Commutative law:

$x + y = y + x$

$xy = yx$

Associative law:

$x + (y + z) = (x + y) + z$

$x(yz) = (xy)z$

Distributive law:

$x(y + z) = xy + xz$

$x + (yz) = (x + y)(x + z)$

Absorption:

$x + (xy) = x$

$x(x + y) = x$

De Morgan's law:

$(x + y)' = x'y'$

$(xy)' = x' + y'$

Other laws and properties:

$(x')' = x$

$x + 1 = 1$

$(x)0 = 0$

$x + 0 = x$

$(x)1 = x$

$x + x' = 1$

$(x)x' = 0$

$(x)x = x$

$x + x = x$

Appendix E: Equipment list

Quantity	Item	Description
1	Digital Trainer	See detailed description below.
2	pn2222 transistors	Other general purpose npn transistors may be substituted.
2	1KΩ 1/4 watt resistors	
2	33KΩ 1/4 watt resistors	
2	4.7KΩ 1/4 watt resistors	
2	100KΩ 1/4 watt resistors	
4	7400	Quad 2 input NAND, see note regarding 7400 series chips
4	7402	Quad 2 input NOR
2	7404	6 inverters
3	7410	3, 3 input NAND
3	74151	8 input multiplexer
2	74175	Quad D flip-flop with CLEAR
2	555 timer	
1	100 μFarad capacitor	
1	0.01 μFarad capacitor	

Digital trainer

A digital trainer is a single purpose unit that contains several features that facilitate the construction and testing of digital circuits. Digital trainers can be constructed, but can be found as a unit for a reasonable price. A digital trainer should include:

- A breadboard
- A 5V power supply which regulates within ±0.25V of 5V
- 8 LEDs that are wired to turn on with logic 1 and off with logic 0
- 6 SPDT switches that are wired to logic high (5V) or logic low (0V)
- 2 SPDT debounced switches (consult Exhibit 8.10 if constructing a digital trainer)

7400 series families

Several of the 7400 series families are acceptable for use with these labs. The LS (Low Powered Schottky), ALS (Advanced Low Powered Schottky) or HC (High speed CMOS) are all widely available, relatively inexpensive and will all perform acceptably.

Appendix F: Solutions

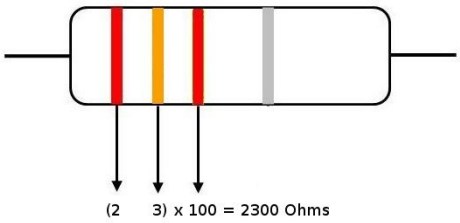
Chapter 1 review exercises

2. Exhibit 1.3 contains the diagram illustrating the commonly connected pins on the breadboard.
3.

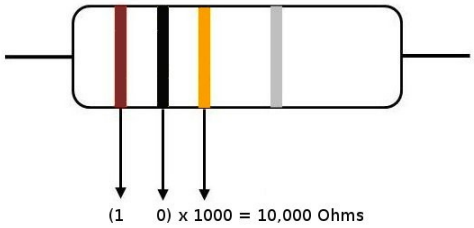
x	x!
0	1
1	0

4. Resistor color codes are explained in detail in Appendix B.

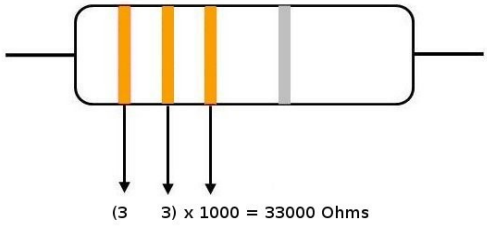
a.



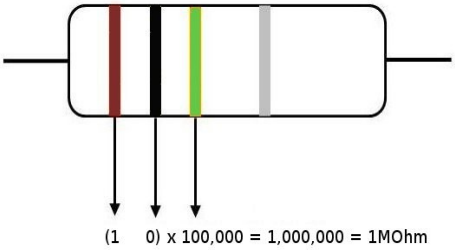
b.



c.



d.



5. The ground symbol is given here. 

6. The NAND is the opposite of the AND gate. The function has two different variables, each with two distinct answers (T-1 or F-0), so there should be four (2^2) different possibilities for the function.

A	B	(AB)'
0	0	1
0	1	1
1	0	1
1	1	0

7.

A	B	(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

Chapter 2 review exercises

1. A logic function with three inputs has eight rows because each of the three inputs has two possibilities. (number of possible outcomes for each input)$^{\text{(number of inputs)}}$ = 2^3
2. A function with five inputs will have 2^5 or 32 different rows.
3. Truth tables follow. It is often easier to obtain the final result if some of the intermediate values that might be necessary are obtained first. For example, in 3.a. the third column is AB, the fourth (AB)' and the fifth is B'. These are then used to obtain the final result.

a. y(A,B) = (AB)' + B'

A	B	AB	(AB)'	B'	y
0	0	0	1	1	1
0	1	0	1	0	1
1	0	0	1	1	1
1	1	1	0	0	0

b. y(A,B,C) = (A+B)'C

A	B	C	A+B	(A+B)'	y
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

c. y(A,B,C) = (AC)' + (BC)

A	B	C	A+B	(A+B)'	y
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

It may not always be necessary to write every intermediate step. In this case, (AC)' is written directly instead of first writing (AC) and then the inverse. If you find this confusing, make sure not to skip steps like this. Note that many different functions can yield the same result. For example, (AB'C)' is equivalent to the function above.

d. y(A,B,C) = (A⊕B)C'

A	B	C	A⊕B	C'	y
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	0	1	0
1	1	1	0	0	0

e. y(A,B) = A' + B

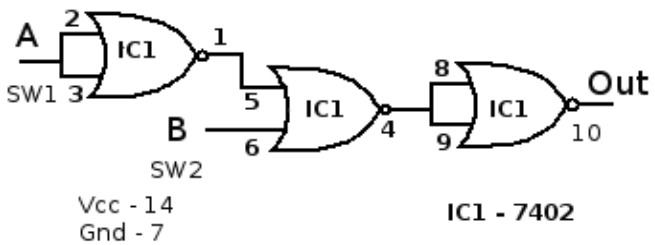
A	B	A'	y
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

f. y(A,B,C) = ((A+B)'(B+C)')'

A	B	C	A+B	B+C	(A+B)'	(B+C)'	(A+B)'(B+C)'	y
0	0	0	0	0	1	1	1	0
0	0	1	0	1	1	0	0	1
0	1	0	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1
1	0	0	1	0	0	1	0	1
1	0	1	1	1	0	0	0	1
1	1	0	1	1	0	0	0	1
1	1	1	1	1	0	0	0	1

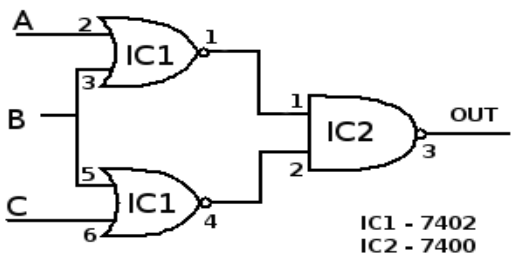
Another example of a logic function with a different equivalent, (A + B + C).

4. Solution with pinout below.



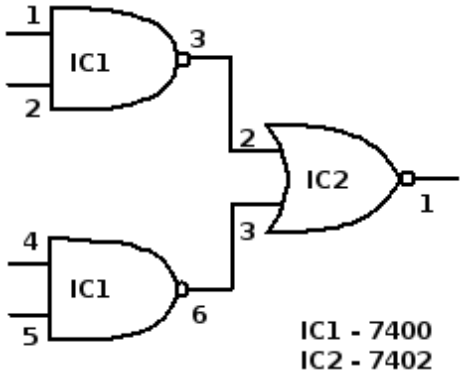
5. Solution with pinout below. It is optional to label Vcc and Gnd on the diagram. Most often for a chip, the Vcc is the upper most right pin and the Gnd is the bottom left, however the chip pinout should always be consulted.

6.



Chapter 3 review exercises

1.



(AB)' + (CD)'

)'(AB)'' (CD)''

(AB)(CD)

ABCD

Original Circuit

De Morgan's law

Double negatives cancel

Parenthesis not necessary

2. Singletons have only one element. Doubles are 2x1 rectangles. Groups of four take two forms, a 4x1 rectangle or a 2x2 square. Finally groups of eight take the form of 4x2 rectangles. Rectangles and squares can be split across borders; further illustrations of this can be found in the next chapter. Example groupings are shown below.

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	0
C'D 01	0	0	0	1
CD 11	0	1	0	0
CD' 10	1	0	0	0

Three single groups

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	0
C'D 01	0	0	1	1
CD 11	1	0	0	0
CD' 10	1	0	0	0

Two 2x1 double groupings

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	0	1	1
C'D 01	1	0	1	1
CD 11	1	0	0	0
CD' 10	1	0	0	0

Two groupings of four

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	0	0	1
C'D 01	1	0	0	1
CD 11	0	0	0	0
CD' 10	0	0	0	0

Group spanning boundary

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	0	0	1
C'D 01	0	0	0	0
CD 11	0	0	0	0
CD' 10	1	0	0	1

Four corner group

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	1	1
C'D 01	1	1	1	1
CD 11	0	0	0	0
CD' 10	0	0	0	0

Group of eight

3. Truth tables follow.

a. f(A,B,C) = AB + A'BC' + AB'C

A	B	C	AB	A'BC'	AB'C	f
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	1	0	1
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	1
1	1	0	1	0	0	1
1	1	1	1	0	0	1

b. g(A,B,C) = A'C + ABC + AB'

A	B	C	A'C	ABC	AB'	g
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	0	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	0	1	1
1	1	0	0	0	0	0
1	1	1	0	1	0	1

c. h(A,B,C,D) = A'BC' + (A ⊕ B)C + A'B'C'D + ABCD

A	B	C	D	A'BC'	(A⊕B)	(A⊕B)C	A'B'C'D	ABCD	h
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0	1
0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	1
0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	1	1	0	0	1
0	1	1	1	0	1	1	0	0	1
1	0	0	0	0	1	0	0	0	0
1	0	0	1	0	1	0	0	0	0
1	0	1	0	0	1	1	0	0	1
1	0	1	1	0	1	1	0	0	1
1	1	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	1	1

d. (A,B,C,D) = A'C'D' + C'D + CD

A	B	C	D	A'C'D'	C'D	CD	j
0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	1
0	0	1	0	0	0	0	0
0	0	1	1	0	0	1	1
0	1	0	0	1	0	0	1
0	1	0	1	0	1	0	1
0	1	1	0	0	0	0	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	1
1	0	1	0	0	0	0	0
1	0	1	1	0	0	1	1
1	1	0	0	0	0	0	0
1	1	0	1	0	1	0	1
1	1	1	0	0	0	0	0
1	1	1	1	0	0	1	1

4. Minimal expressions given for each map. Notice that quite often, the terms in the original are not found at all in the minimal SOP (Sum Of Products) expression.

   a. Original expression: f(A,B,C) = AB + A'BC' + AB'C

      Minimal expression: f(A,B,C) = BC' + AC

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	0	1	1	0
C 1	0	0	1	1

   b. Original expression: g(A,B,C) = A'C + ABC + AB'

      Minimal expression: g(A,B,C) = AB' + C

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	0	0	0	1
C 1	1	1	1	1

   c. Original expression: h(A,B,C,D) = A'BC' + (A ⊕ B)C + A'B'C'D + ABCD

      Minimal expression: h(A,B,C,D) = A'B + A'C'D + BCD + AB'C

      Minimal expression: h(A,B,C,D) = A'B + A'C'D + ACD + AB'C

      More than one minimal expression exists. In these cases, more than one correct answer exists.

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	1	0	0
C'D 01	1	1	0	0
CD 11	0	1	1	1
CD' 10	0	1	0	1

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	1	0	0
C'D 01	1	1	0	0
CD 11	0	1	1	1
CD' 10	0	1	0	1

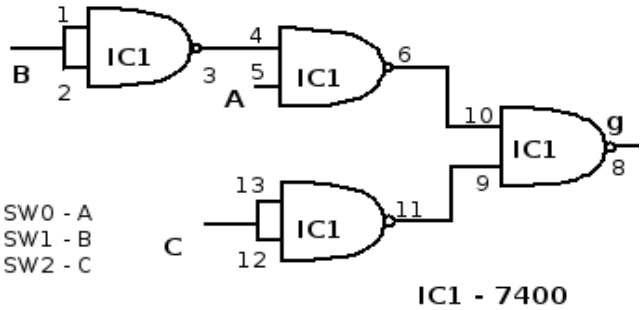
   d. Original expression: j(A,B,C,D) = A'C'D' + C'D + CD

      Minimal expression: j(A,B,C,D) = D + A'C'

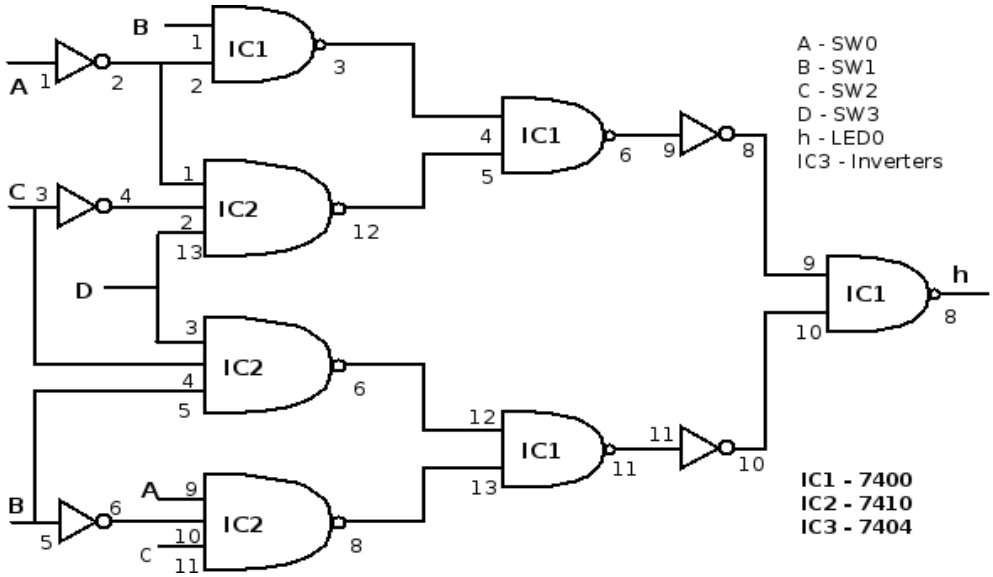
	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	0	0
C'D 01	1	1	1	1
CD 11	1	1	1	1
CD' 10	0	0	0	0

5. g(A,B,C) = AB' + C

   This circuit was designed using only NAND gates. This allows the circuit to be implemented with just one chip. DeMorgan's law was used to avoid needing a NOR gate. In addition, an inverter was avoided by using the remaining NAND gate left on the chip to invert input A.



6. h(A,B,C,D) = A'B + A'C'D + BCD + AB'C



7. As the logic kit does not contain a four input NAND gate, combinations of three and two input NANDs are used. The following justification shows that this is indeed a correct implementation.

   a. Minimal expression: A' + B'

   [(A'B)' (A'C'D)']' ' [(BCD)' (AB'C)']' ')'

   [(A'B)' (A'C'D)']' " + [(BCD)' (AB'C)']' "

   [(A'B)' (A'C'D)']' + [(BCD)' (AB'C)']'

   [(A'B)" +(A'C'D)"] + [(BCD)" + (AB'C)"]

	A' 0	A 1
B' 0	1	1
B 1	1	0

   A'B + A'C'D + BCD + AB'C'

b. Minimal expression: C' + A'B

Direct implementation from circuit

De Morgan's law

Double negatives cancel

De Morgan's law

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	1	1	1	1
C 1	0	1	0	0

Double negatives

c. Two different Minimal expressions exist for this problem.

i. Minimal expression: C'D' + A'C' + BC' + AC

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	1	1
C'D 01	1	1	1	0
CD 11	0	0	1	1
CD' 10	0	0	1	1

ii. Minimal expression: C'D' + A'C' + AB + AC

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	1	1
C'D 01	1	1	1	0
CD 11	0	0	1	1
CD' 10	0	0	1	1

d. Three different Minimal expressions exist for this problem. See below.

i. Minimal expression: A'C'D + A'BD + A'CD' + AB'D

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	0
C'D 01	1	1	0	1
CD 11	0	1	0	1
CD' 10	1	1	0	0

ii. Minimal expression: B'C'D + A'BD + A'CD' + AB'D

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	0
C'D 01	1	1	0	1
CD 11	0	1	0	1
CD' 10	1	1	0	0

iii. Minimal expression: A'C'D + A'BC + A'CD' + AB'D

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	0
C'D 01	1	1	0	1
CD 11	0	1	0	1
CD' 10	1	1	0	0

Chapter 4 review exercises

1. Minimal SOP expressions

a. Minimal expression: B' + AC'

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	d	0	1	1
C 1	1	0	0	d

b. Minimal expression: A + C

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	0	0	1	1
C 1	1	d	d	1

c. Notice that this solution has one of the groupings that spans the boundaries (B'C).

Minimal expression: AB'+ AD + B'C

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	1
C'D 01	0	0	1	1
CD 11	1	0	1	1
CD' 10	1	0	0	1

d. This expression includes the four corner grouping (B'D').

Minimal expression: B'D' + A'B

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	0	1
C'D 01	0	1	0	0
CD 11	0	1	0	0
CD' 10	1	1	0	1

e. Two different Minimal expressions exist for this problem.

Minimal expression: B'C' + A'C' + BC.

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	1	1	0	1
C 1	0	1	1	0

Minimal expression: B'C' + A'B + BC

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	1	1	0	1
C 1	0	1	1	0

f. Minimal expression: C'D + A'D'

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	1	0	0
C'D 01	1	1	1	1
CD 11	0	0	0	0
CD' 10	1	1	0	0

2. K-maps with minimal SOP expressions

a. Minimal expression: AB + A'C'

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	1	1	1	0
C 1	0	0	1	0

b. Notice that not all don't care conditions need to be covered.

Minimal expression: A

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	0	0	1	1
C 1	0	d	1	d

c. Minimal expression: BD + AD + A'B'C

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	0	0
C'D 01	0	1	1	1
CD 11	d	d	d	d
CD' 10	1	0	0	0

d. Minimal expression: CD' + AD'

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	1	1
C'D 01	0	d	d	0
CD 11	0	0	0	0
CD' 10	1	1	1	1

Chapter 5 review exercises

1. K- map and minimal SOP expressions

a. f1(a,b,c) = a'b'c' + a'bc' + a'bc + ab'c'

Minimal expression: f1(a,b,c) = a'b + b'c'

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	1	1	0	1
C 1	0	1	0	0

a	b	c	f1
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

b. f2(a,b,c) = a'b'c + a'bc + abc' + ab'c

Minimal expression: f2(a,b,c) = a'c + b'c +abc'

	A'B' 00	A'B 01	AB 11	AB' 10
C' 0	0	0	1	0
C 1	1	1	0	1

a	b	c	f1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

c. f3(a,b,c,d) = a'b'c'd' + a'bcd + abcd + ab'c'd' + ab'c'd

Minimal expression: f3(a,b,c,d) = b'c'd' + ab'c' + bcd

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	0	0	1
C'D 01	0	0	0	1
CD 11	0	1	1	0
CD' 10	0	0	0	0

a	b	c	d	f3
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

d. f4(a,b,c,d) = a'b'c'd' + a'bc'd + abcd + a'b'cd' + a'b'cd + a'bcd' + ab'c'd

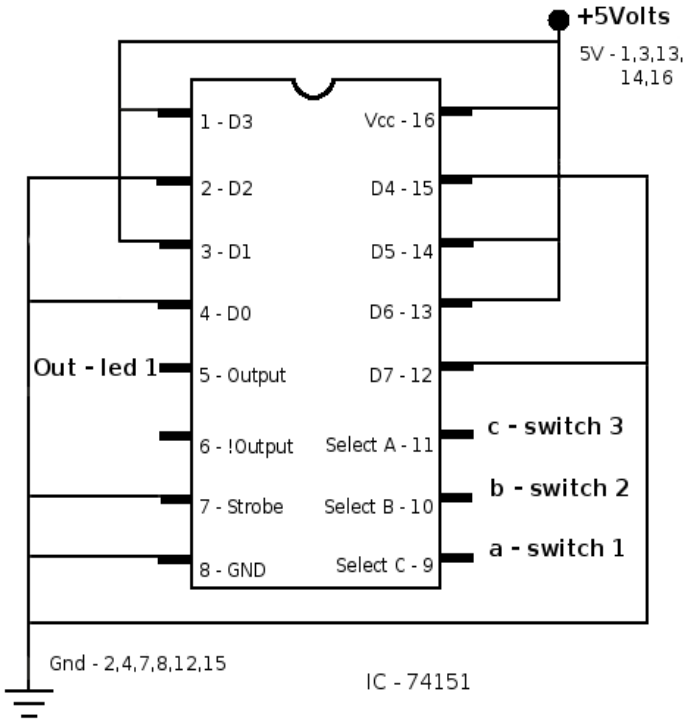
Minimal expression: f4(a,b,c,d) = a'b'c + a'cd' + a'b'd' + abcd + a'bc'd + ab'c'd

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	1	0	0	0
C'D 01	0	1	0	1
CD 11	1	0	1	0
CD' 10	1	1	0	0

a	b	c	d	f4	Mux In
0	0	0	0	1	d'
0	0	0	1	0	d'
0	0	1	0	1	1
0	0	1	1	1	1
0	1	0	0	0	d
0	1	0	1	1	d
0	1	1	0	1	d'
0	1	1	1	0	d'
1	0	0	0	0	d
1	0	0	1	1	d
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	d
1	1	1	1	1	d

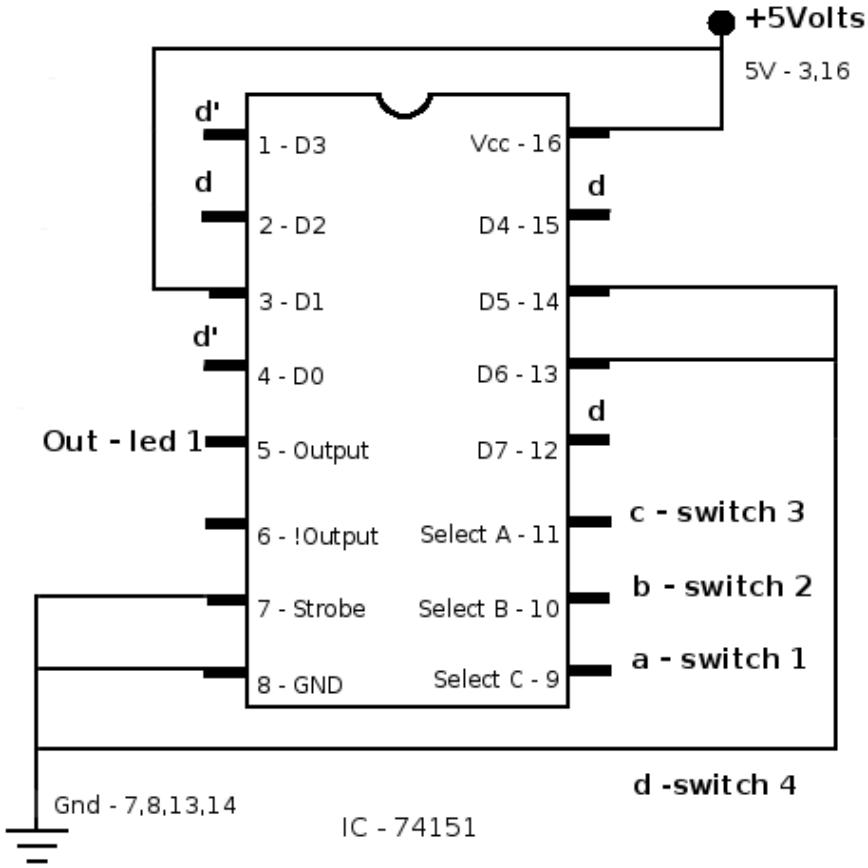
The truth table also shows the inputs required for the multiplexer which will be used later when implementing the function with a mux.

2. f2(a,b,c) = a'b'c + a'bc + abc' + ab'c



3. f4(a,b,c,d) = a'b'c'd' + a'bc'd + abcd + a'b'cd' + a'b'cd + a'bcd' + ab'c'

Examine the truth table from previous problem to understand why input values are chosen.



4. Circuit design

a. g1(a,b,c,d) = a'b'c'd + abcd + a'bcd + a'bc'd + ab'c'd + a'b'cd + abc'd + ab'cd

Minimal expression: g1(a,b,c,d) = d

When the K-map is filled out, it can be seen that the minimal solution is simply d. No logic is needed at all! Hopefully, you did not try to write the truth table and implement it with a multiplexer. This illustrates why even though a multiplexer can implement any circuit, the logic should be analyzed first.

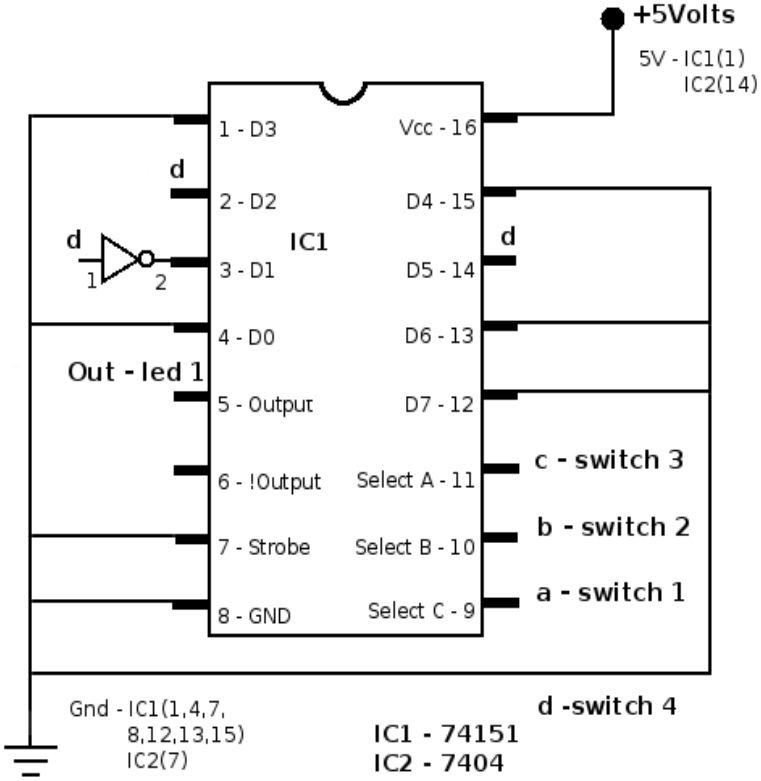
	a'b' 00	**a'b** 01	**ab** 11	**ab'** 10
c'd' 00	0	0	0	0
c'd 01	1	1	1	1
cd 11	1	1	1	1
cd' 10	0	0	0	0

b. g2(a,b,c,d) = a'bc'd + a'b'cd' + ab'cd

For this problem, first the K-map shows that this is the minimal expression. Then the truth table is constructed to determine the input values for an 8-to-1 mux implementation.

	a'b' 00	**a'b** 01	**ab** 11	**ab'** 10
c'd' 00	0	0	0	0
c'd 01	0	1	0	0
cd 11	0	0	0	1
cd' 10	1	0	0	0

a	**b**	**c**	**d**	**g2**	**Mux Input**
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	d'
0	0	1	1	0	d'
0	1	0	0	0	d
0	1	0	1	1	d
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	d
1	0	1	1	1	d
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0



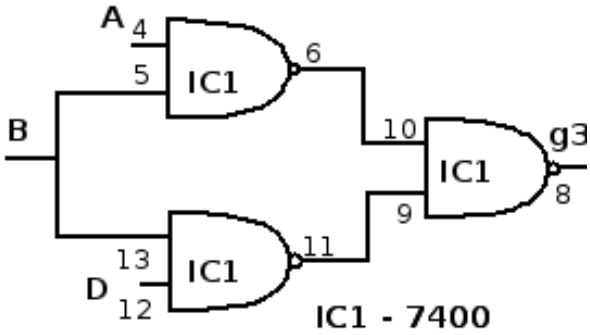
c. g3(a,b,c,d) = abc'd' + abc'd + abcd + abcd' + a'bc'd + a'bcd

Minimal expression: ab + bd

The Minimal expression is the column ab and the middle square bd. This can be implemented with a single 7400 chip with one NAND gate left over.

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	0	1	0
C'D 01	0	1	1	0
CD 11	0	1	1	0
CD' 10	0	0	1	0

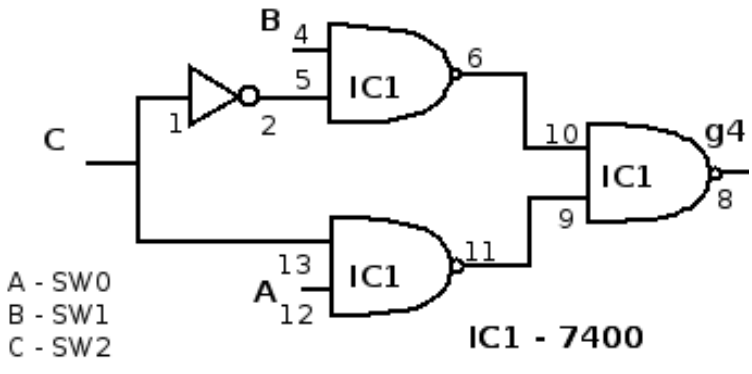
SW0 - A
SW1 - B
SW2 - D



d. g4(a,b,c,d) = a'bc'd' + abc'd' + abcd' + ab'cd' + a'bc'd + abc'd + abcd + ab'cd

Minimal expression: bc' + ac

	A'B' 00	A'B 01	AB 11	AB' 10
C'D' 00	0	1	1	0
C'D 01	0	1	1	0
CD 11	0	0	1	1
CD' 10	0	0	1	1



Chapter 6 review exercises

1. Periods in seconds of the clock with given frequencies
   a. T = 1/*f*

      T = 1/(6,000,000) = 0.000000167 sec or 167 nsec

   b. T = 1/(10,000,000) = 0.0000001 sec or 100 nsec

   c. *f* = 6000 cycles/min * 1min/60sec = 100 Hz

      T = 1/100 = 0.01 sec or 10.0 msec

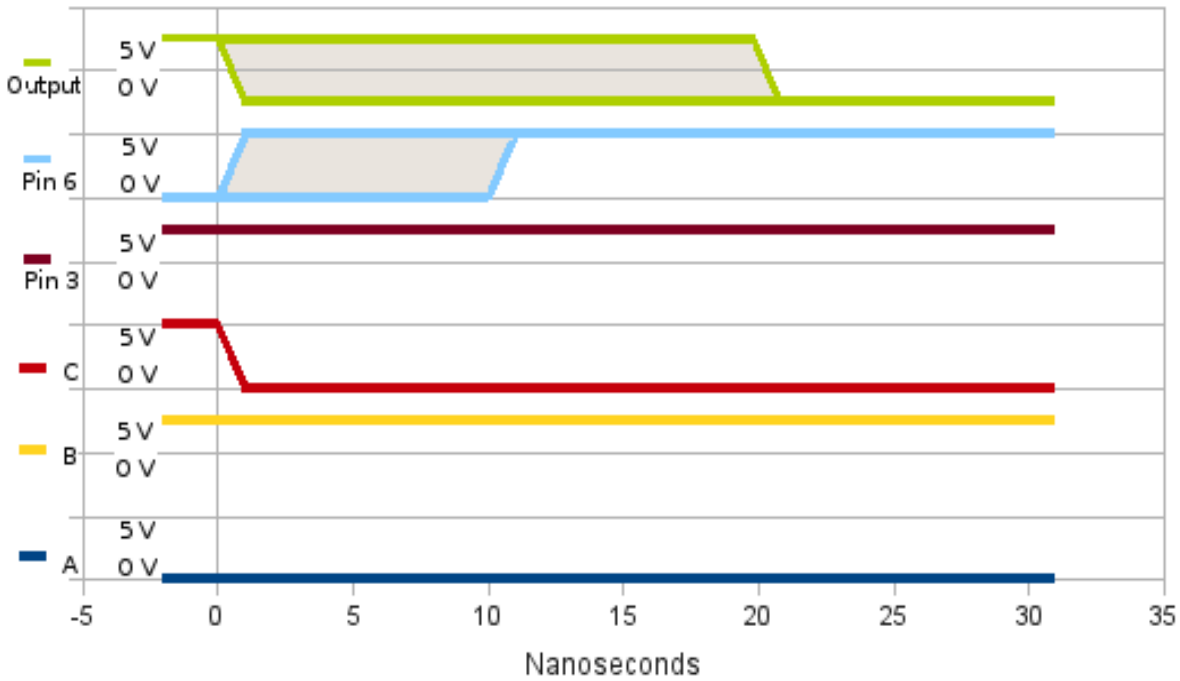
2. Frequency of clock in Hertz
   a. *f* = 1/T

      f = 1/(.00001) = 100 Khz

   b. *f* = 1/(0.00000000005) = 20.0 GHz

   c. *f* = 1/(.001) = 1000 Hz

3. Notice that Pin 3 never changes, although the state for Pin 6 and Pin 10 (the output) are indeterminate until it can be verified that the logic has successfully traveled through the required logic gates.



4. The logic circuit from Exhibit 2.14 has eight logic gates. Many of these gates are in parallel, such as the first two inverters or the two NAND chips from IC1. The longest path for the logic to travel is what determines the maximum frequency that the clock can be traveled.
   So the longest delay is: (10nsec * 5) = 50 nsec
   $f = 1/T = 1/(0.00000005) =$ 20.0 MHz
5. Timers with times of 1, 5, and 10 seconds
   a. Recall that the timer has a delay of: t = 1.10(RC) Solving for R yields: R = t/(1.10C)
      The required values for R are found in the table, along with those that are easiest to obtain using the resistors from the lab kit.

t – desired	R desired	R for lab	t - actual
1.0 sec	9100 Ω	9400 Ω	1.0 sec
5.0 sec	45000 Ω	42400 Ω	4.7 sec
10. sec	91000 Ω	100000 Ω	11 sec

   b. The first R is obtained by putting two of the 4.7 K resistors in series. The second is by putting two 4.7K resistors in series with a 33K resistor.

c. The schematic should look identical to Exhibit 6.1 with the appropriate values for R and C.

d. Last, for the values chosen, the span for the times is calculated below.

i. *1 second timer*: 1.10(0.95 * 9400)(0.9 * 100u) < actual < 1.10(1.05 * 9400)(1.1 * 100u)

.89 < actual value < 1.2

ii. *5 second timer*: 1.10(0.95 * 42400)(0.9 * 100u) < actual < 1.10(1.05 * 42400)(1.1 * 100u)

4.0 < actual value < 5.4

iii. *10 second timer:* 1.10(0.95 * 100000)(0.9 * 100u) < actual < 1.10(1.05 * 100000)(1.1 * 100u)

9.4 < actual value < 13

6. Recall that the period of the clock is given by:

T = t1 + t2

= time on + time off

= 0.693(R1 + R2)C + 0.693(R2)C

= 0.693(R1 + 2*R2)C

a. If R1 and R2 are both 4.7K resistors for the first clock and R1 is 4.7K and R2 is 33K for the second, the resulting times are:

T(1sec) = 0.693(4700 + 4700)0.0001 + .693(4700)0.0001

= 0.651 + 0.326

= .98 seconds

T(5sec) = 0.693(33000 + 4700)0.0001 + .693(33000)0.0001

= 2.61 + 2.29

= 4.9 seconds

b. Time on for the 1 second clock is 0.65 seconds and off is 0.33, while time on for the 5 second clock is 2.6 seconds and off is 2.3 seconds.

c. The schematic will look exactly like Exhibit 6.3 with the appropriate R and C values inserted.

Chapter 7 review exercises

1. Recall, if either of the input values are 1, the output of the gate is 0. While the output values of Q and Q' may change, the input values of S and R will not for this table. So, for any row that has S set to 1, the corresponding value for Q_N' must be 0 and likewise if R is 1, Q_N must be 0. Using this, some values can immediately be determined with this information. As the output values may change, the remaining next state values require more examination.

S	R	Q	Q'	Q_N	Q_N'
0	0	0	1	*?*	*?*
0	0	1	0	*?*	*?*
0	1	0	1	**0**	*?*
0	1	1	0	**0**	*?*
1	0	0	1	*?*	**0**
1	0	1	0	*?*	**0**
1	1	0	1	**0**	**0**
1	1	1	0	**0**	**0**

Row 1: Q' is 1, causing Q to be 0 leaving Q' 1.
Row 2: Q is 1, causing Q' to be 0 leaving Q 1For rows 1 and 2, the state of Q and Q' does not change.
Row 3 & 4: Q_N and S are 0, causing Q_N ' to be 1. Q_N' at 1 means Q_N is 0.For rows 3 and 4, the latch is reset.
Row 5 & 6: Q_N' and R are 0, causing Q_N to be 1. Q_N at 1 means Q_N' is 0. For rows 5 and 6, the latch is set.
Row 7 & 8: Q_N and Q_N' are not inverse values of each other, which explains why these states are not used for the latch. Final stable values are provided in the second truth table.

S	R	Q	Q'	Q_N	Q_N'
0	0	0	1	*0*	*1*
0	0	1	0	*1*	*0*
0	1	0	1	**0**	*1*
0	1	1	0	**0**	*1*
1	0	0	1	*1*	**0**
1	0	1	0	*1*	**0**
1	1	0	1	**0**	**0**
1	1	1	0	**0**	**0**

2. For the SR latch constructed with NAND gates, recall that the NAND gate will have an output of 1 if either of the input values is 0. In this manner, some of the next state values may be determined immediately.

Now, the remaining undetermined rows are examined.

S	R	Q	Q'	Q_N	Q_N'
0	0	0	1	**1**	**1**
0	0	1	0	**1**	**1**
0	1	0	1	**1**	*?*
0	1	1	0	**1**	*?*
1	0	0	1	*?*	**1**
1	0	1	0	*?*	**1**
1	1	0	1	*?*	*?*
1	1	1	0	*?*	*?*

Rows 1 & 2: Both Q_N and Q_N' are 1, not inverses of one another.These states are not used.
Rows 3 & 4: Q_N is 1 and R is 1, so Q_N' will be 0. These are the set states.
Rows 5 & 6: Q_N' is 1 and S is 1, so Q_N will be 0. These are the reset states.
Row 7: S and Q' are 1, so Q_N will stay 0. Q_N is 0 and R is 1, so Q_N' stays 1
Row 8: R and Q are 1, so Q_N' will stay 0. Q_N' is 0 and S is 1, so Q_N stays 1.Rows 7 and 8 are t stable states where the output values do not change.

Final values are provided in the second truth table.

S	R	Q	Q'	Q_N	Q_N'
0	0	0	1	**1**	**1**
0	0	1	0	**1**	**1**
0	1	0	1	**1**	0
0	1	1	0	**1**	0
1	0	0	1	0	**1**
1	0	1	0	0	**1**
1	1	0	1	*?*	*?*
1	1	1	0	*?*	*?*

3. As the NAND gate is an active low gate, meaning if either input is 0, the output will go high, some of the values of the table can be determined immediately (these are bolded).
NAND1 can be determined by D and C (italic).
Where the values of NAND1 and C are known, the value of NAND2 can be determined (highlighted in yellow).
Where NAND1 or NAND2 are known to be 0, the corresponding gates NAND3 and NAND4 must be 1 (shown in light blue).

D	C	Q	Q'	1	2	3/Q_N	4/Q_N'
0	0	0	1	**1**	**1**	?	?
0	0	1	0	**1**	**1**	?	?
0	1	0	1	**1**	0	?	1
0	1	1	0	**1**	0	?	1
1	0	0	1	**1**	**1**	?	?
1	0	1	0	**1**	**1**	?	?
1	1	0	1	*0*	1	1	?
1	1	1	0	*0*	1	1	?

Now treat NAND1 as the S input and NAND2 as the R input to the NAND SR latch (NAND3 and NAND4) and use the work from the previous problem.

Rows 1 & 5: Similar to row 7 from problem 2.

Rows 2 & 6: Similar to row 8 from problem 2.

The states for Rows 1, 2, 5 and 6 do not change.

Row 3: Similar to row 5 from problem 2.

Row4: Similar to row 6 from problem 2.

Rows 3 and 4 correspond to the reset state.

Row 7: Similar to row 3 from problem 2.

Row 8: Similar to row 4 from problem 2.

Rows 7 and 8 correspond to the set state.

D	C	Q	Q'	1	2	3/Q_N	4/Q_N'
0	0	0	1	**1**	**1**	**0**	**1**
0	0	1	0	**1**	**1**	**1**	**0**
0	1	0	1	**1**	**0**	**0**	**1**
0	1	1	0	**1**	**0**	**0**	**1**
1	0	0	1	**1**	**1**	**0**	**1**
1	0	1	0	**1**	**1**	**1**	**0**
1	1	0	1	**0**	**1**	**1**	**0**
1	1	1	0	**0**	**1**	**1**	**0**

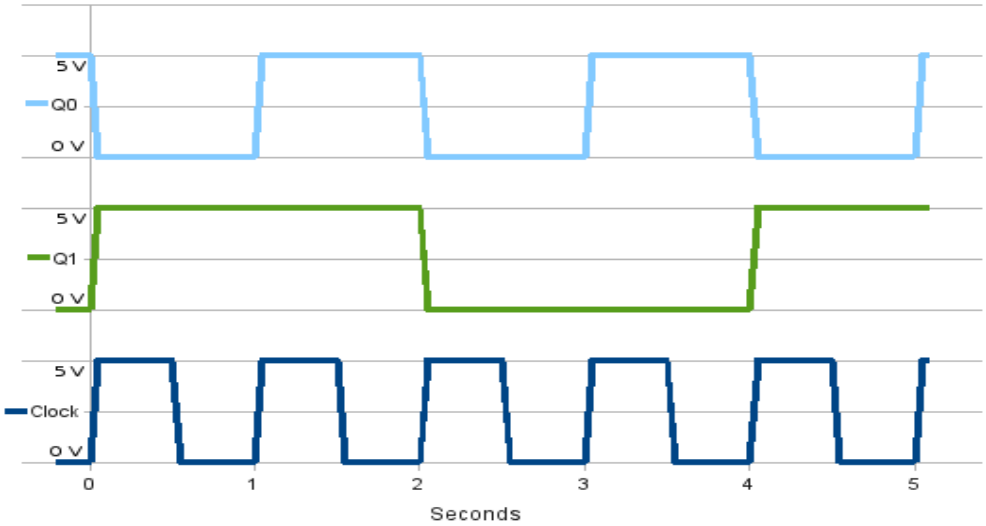
Note than when C is low, the state of the flip-flop can never change. Also, due to the addition of NAND1 and NAND2, there is never a time when the inputs reach a state that should not be used, as with the SR latches that must avoid certain states. So when C is low, the state remains constant and when C is high, the state tracks the D input. The final values are given in the second truth table.

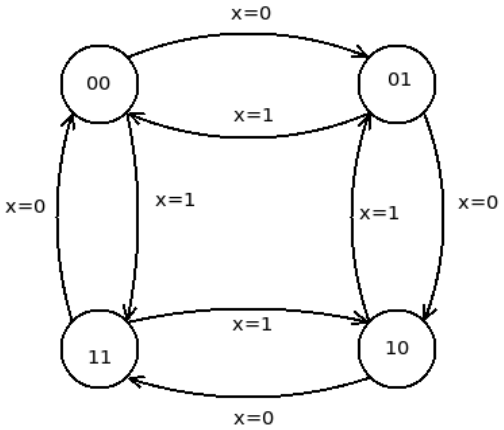
4. When the clear line is low, the value of Q will be low regardless of the state of D. When the value of Clear is high, the value of Q will be equal to the value of D at the time of the rising clock edge.

D	CLEAR	Q
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Chapter 8 review exercises

1. Because switches suffer from bounce, the circuit could interpret the bounces as clock pulses as well. This would mean that the circuit might be clocked more than once for a given flip of the switch.

2. 

3. 

4. Two flip-flops are needed to represent all four possible states.

5.

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	0	1	0	1
x 1	1	0	1	0

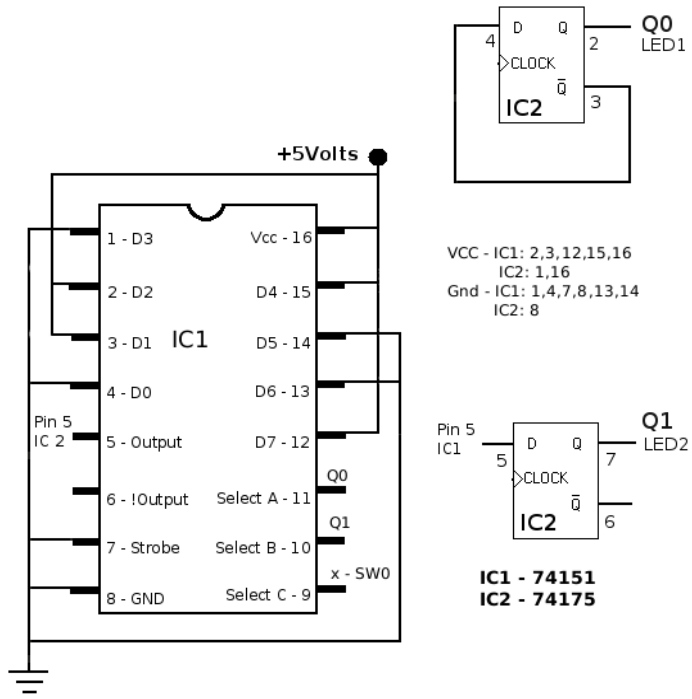
$Q_{1N}(x,Q_1,Q_0)$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	1	0	0	1
x 1	1	0	0	1

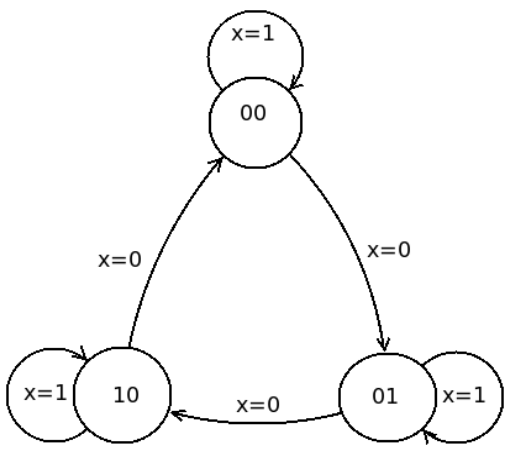
$Q_{0N}(x,Q_1,Q_0) = Q_0'$

The minimal expression for $Q1_N$ is xQ1'Q0' + x'Q1'Q0 + xQ1Q0 + x'Q1Q0 which is not very minimal. For this reason, the design that follows uses a multiplexer to implement the input for the second flip-flop. The first flip-flop requires a value that can be taken directly off of the flip-flop itself Q0'.

Remember to be careful when using the mux, and insure that the Select C line is the most significant bit for the logical expression.



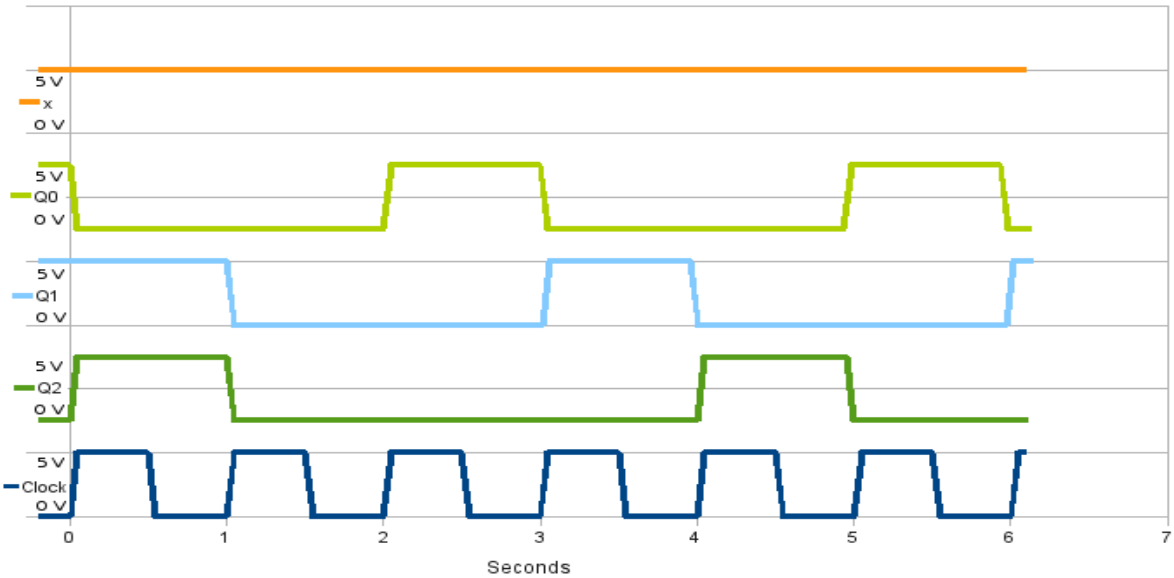
6.



7. The state machine has 3 states so it requires 2 flip-flops. $2^1 < 3 <= 2^2$

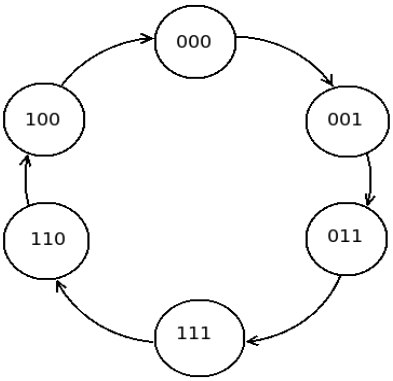
   The state 11 is not used. The next chapter will discuss the design of systems with unused states.

Chapter 9 review exercises

1. A state machine that has 7 states will require 3 flip-flops. $2^2 < 7 <= 2^3$
   a. With no external inputs, only the existing states provide input to determine the next state, so the K-maps will be a 4x2 rectangle.
   b. With one external input, there will be 4 total inputs, so the K-maps will be 4x4 squares.
   c. With JK flip-flops, as both the J and K are inputs to the flip-flop, there would now be 6 next state input values to determine.
2. A state machine that has 14 states will require 4 flip-flops. $2^3 < 14 <= 2^4$
   a. With no external inputs, only the existing states provide input to determine the next state, so the K-maps will be a 4x4 rectangle.
   b. With one external input, there will be 5 total inputs. The K-maps will be 8x4 rectangles, which are often unwieldy. In this case, alternate minimization techniques should be explored.
   c. Using a JK flip-flop with 14 states that requires 4 flip-flops, there would be a total of 8 state inputs. One J input and one K input for each of the flip-flops.
3. 

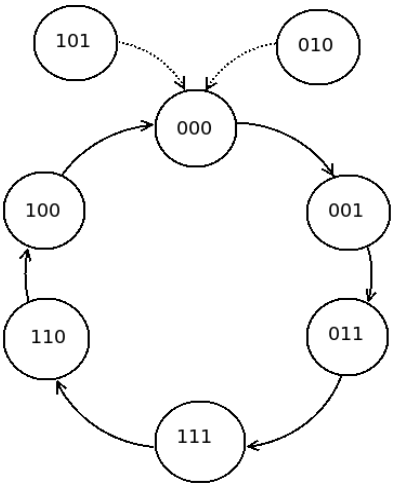
4. A state machine traverses the states listed in this order 000 → 001 → 011 → 111 → 110 → 100 → 000. There is no external input.

a.



b. Of the 8 possible states, 101 and 100 are not represented.

c.



d.

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
Q_2' 0	0	0	1	0
Q_2 1	0	0	1	1

$Q_{2N}(Q_2,Q_1,Q_0) = Q_1Q_0 + Q_2Q_1$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
Q_2' 0	0	1	1	0
Q_2 1	0	0	1	0

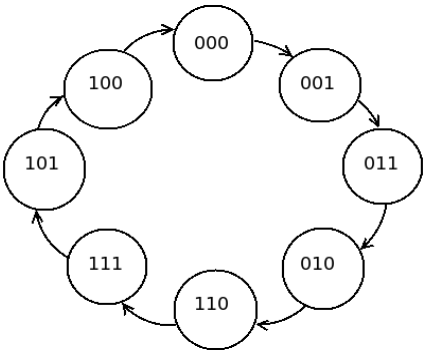
$Q_{1N}(Q_2,Q_1,Q_0) = Q_1Q_0 + Q_2'Q_0$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
Q_2' 0	1	1	1	0
Q_2 1	0	0	0	0

$Q_{0N}(Q_2,Q_1,Q_0) = Q_2'Q_1' + Q_2'Q_0$

5. The two bit sequence 00 → 01 → 11 → 10 → 00 is a Gray code. Gray codes only have one bit change for each transition.

a.



b.

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
Q_2' 0	0	0	0	1
Q_2 1	0	1	1	1

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
Q_2' 0	0	1	1	1
Q_2 1	0	0	0	1

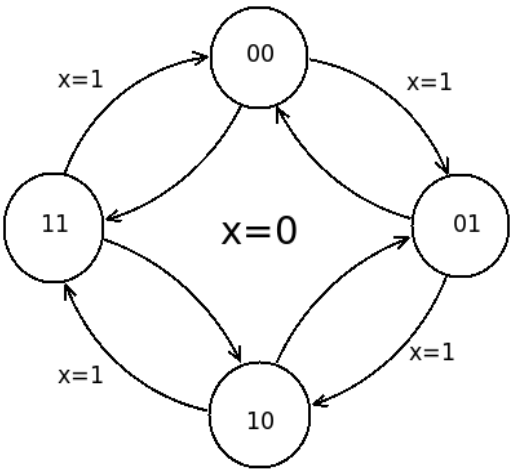
$Q_{2N}(Q_2,Q_1,Q_0) = Q_1Q_0' + Q_2Q_0$ $\quad$ $Q_{1N}(Q_2,Q_1,Q_0) = Q_1Q_0' + Q_2'Q_0$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
Q_2' 0	1	1	0	0
Q_2 1	0	0	1	1

$Q_{0N}(Q_2,Q_1,Q_0) = Q_2'Q_1' + Q_2Q_1$

6. A two bit counter is to be built that will count forward, 00 → 01 → 10 → 11 → 00, when a logical input is set high and counts in reverse order when it is low.

a.



b.

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	1	0	1	0
x 1	0	1	0	1

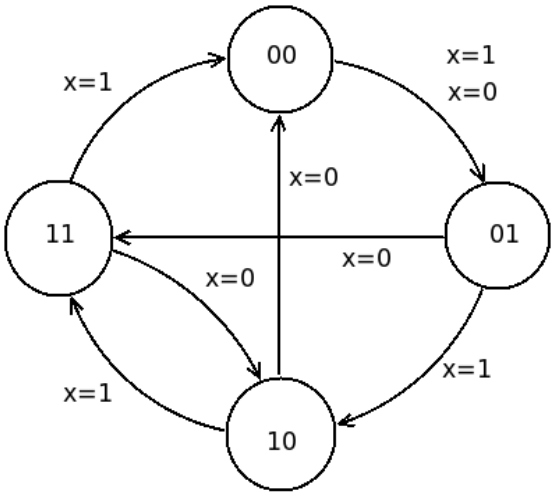
$Q1_N(x,Q_1,Q_0) = x'Q_0'Q_1'+xQ_1'Q_0+x'Q_1Q_0+x'Q_1Q_0'$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	1	0	0	1
x 1	1	0	0	1

$Q_{0N}(x,Q_1,Q_0) = Q_0'$

7. A two bit counter is to be built that will count forward, 00 → 01 → 10 → 11 → 00, when a logical input is set high and as a Gray code when it is low (00 → 01 → 11 → 10 → 00).

a.



b.

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	0	1	1	0
x 1	0	1	0	1

$Q_{1N}(x,Q_1,Q_0) = x'Q_0 + Q_1'Q_0 + xQ_1Q_0$

	$Q_1'Q_0'$ 00	$Q_1'Q_0$ 01	Q_1Q_0 11	Q_1Q_0' 10
x' 0	1	1	0	0
x 1	1	0	0	1

$Q_{0N}(x,Q_1,Q_0) = x'Q_1' + xQ_0'$