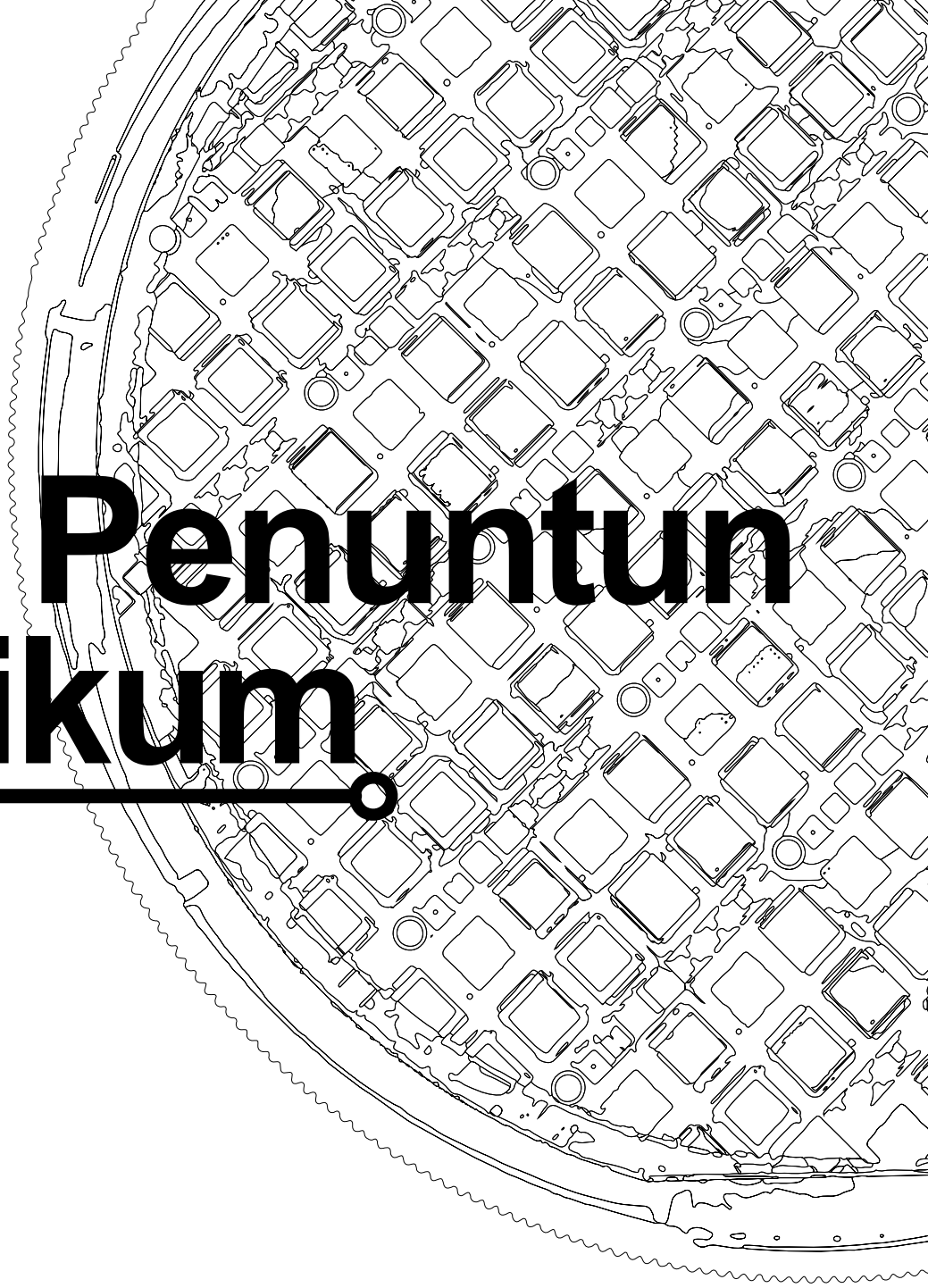
UNIVERSITAS
INDONESIA

Veritas, Probitas, Iustitia



Buku Penuntun Praktikum

Elektronika 2

Laboratorium Elektronika
Departemen Fisika
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Indonesia
2018

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa, akhirnya penyusunan Buku Penuntun Praktikum Elektronika Digital edisi 2018 dapat diselesaikan. Buku penuntun ini merupakan, acuan yang akan digunakan oleh praktikan yang akan melakukan Praktikum Elektronika II dan merupakan lanjutan dari Praktikum Elektronika I sebelumnya.

Pada edisi ini, setiap modul mengalami penyempurnaan dari modul sebelumnya dan telah disesuaikan dengan Mata Kuliah Elektronika dan perkembangan dunia elektronika. Penambahan juga dilakukan seperti pada modul 6 – 9 yang menggunakan perangkat *ZYBO™ FPGA Board* dengan menggunakan *VHDL* sebagai bahasa pemrogramannya. Kami berharap, praktikan tidak hanya terasah kemampuannya pada sisi *hardware* saja namun juga pada bagian *back-end* (*software),* serta alur pemikiran konstruktifnya.

Akhirnya, kami mengucapkan terima kasih kepada bapak Dr. rer. nat. Agus Salam selaku Ketua Departemen Fisika yang telah banyak men-*support* baik moril maupun materil hingga penyusunan buku ini dapat terlaksana dengan baik. Buku Penuntun Praktikum ini jauh dari kata sempurna, maka saran dan kritik yang membangun selalu kami nantikan demi penyempurnaan dan perkembangan kita semua.

Buku ini kami persembahkan secara special kepada Departemen Fisika UI, semoga dapat bermanfaat. Amin.

Depok, 1 Maret 2018

Sastra Kusuma Wijaya, Ph.D
Dian Wulan Hastuti, S.Si
Affan Hifzhi, S.Si
Rizki Arif

DAFTAR ISI

TATA TERTIB PRAKTIKUM ELEKTRONIKA
LABORATORIUM ELEKTRONIKA, DEPARTEMEN FISIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS INDONESIA

1. Praktikan harus hadir **maksimal 10 menit sebelum praktikum dimulai**, bagi praktikan yang terlambat tidak dapat mengikuti praktikum pada hari tersebut, dan percobaan pada hari tersebut dinyatakan gagal.
2. Pada saat berada di laboratorium, praktikum harus tenang, tertib, sopan, dan bertanggungjawab. Tas, jaket, buku, dan perlengkapan lainnya yang tidak diperlukan untuk praktikum dititipkan di loker.
3. Praktikan dapat mengikuti praktikum apabila memenuhi syarat-syarat sebagai berikut:
    a. **Membawa Kartu Praktikum**
    b. **Membawa Kotak Komponen** yang telah dipinjamkan sebelumnya (Jaminan Kotak Komponen Rp 50.000,-)
    c. **Membawa Laporan Praktikum** berupa:
        i. **Laporan Pendahuluan modul hari-H**
        ii. **Laporan Akhir modul sebelumnya, beserta lampiran data pengamatannya**
    d. **Lulus Tes Pendahuluan (minimum 50% dari nilai total)**
    e. Apabila tidak memenuhi syarat (a), praktikan wajib melaporkan ke co-PJ dan dikenakan **denda Rp 15.000,-**
    f. Apabila tidak memenuhi syarat (b), praktikan wajib melaporkan ke co-PJ dan dikenakan **denda Rp 30.000,-**
    g. Apabila tidak memenuhi syarat (c) dan (d) maka **praktikan tidak dapat mengikuti praktikum pada hari tersebut dan percobaan pada hari tersebut dinyatakan gagal**
4. Jika ada perlengkapan praktikum yang hilang, praktikan wajib melaporkan kepada co-PJ sebelum praktikum dimulai
5. Bagi praktikum yang berhalangan hadir, dapat memberikan **surat keterangan resmi** yang akan diserahkan kepada co-PJ atau Kepala Laboratorium
6. Praktikan harus **memperoleh data melalui praktikum yang dilakukan oleh kelompoknya sendiri**. Apabila ditemukan menggunakan data dari kelompok lain, praktikan akan dianggap gagal untuk modul tersebut
7. **Praktikan yang gagal diwajibkan untuk membayar denda susulan sebesar**
    a. **Rp 50.000,- untuk pertemuan atau modul pertama**
    b. **Rp 75.000,- untuk pertemuan atau modul kedua**
8. **Ketidakhadiran dengan alasan apapun, termasuk gagal, izin, sakit, dan alpha, hanya diizinkan maksimal dua kali. Apabila melebihi dua kali, praktikan yang bersangkutan tidak lulus praktikum.**
9. Selama praktikum, praktikan harus menjaga kebersihan, ketertiban, dan kenyamanan lingkungan laboratorium. Praktikan juga wajib menjaga keselamatan dirinya. Selama berada di laboratorium, praktikan **dilarang mengenakan sandal dan/atau baju kaos, merokok, makan, atau mengganggu kelompok lain**.
10. Selama praktikum, **praktikan dilarang meninggalkan ruangan laboratorium tanpa seizin Asisten Laboratorium.**
11. Praktikan harus mengembalikan meja praktikum kembali ke kondisi awal setelah praktikum selesai. Sisa-sisa kabel, komponen yang terbakar, kertas, dan benda-benda lain yang sudah tidak terpakai dapat dibuang pada tempat yang telah disediakan.
12. Setelah praktikum selesai, salinan data wajib diserahkan kepada Asisten Laboratorium pada hari itu juga
13. **Praktikan harus mengganti komponen-komponen yang hilang atau rusak.** Penggantian dapat diambil dari uang jaminan, namun praktikan juga dapat menambahkan atau mengganti alat atau komponen yang sama.

14. **Praktikan harus meminta tanda tangan Asisten Laboratorium pada Kartu Praktikum dan salinan lampiran data pengamatannya.**
15. **Praktikan dapat diberikan peringatan atau dikelarkan apabila melanggar tata tertib ini**

Sistem Penilaian Praktikum terdiri dari:

- **Laporan Pendahuluan**
  - Sistematika Penulisan dan Bahasa
  - Teori Dasar
  - Tugas Pendahuluan
  - Simulasi
- **Penilaian Kerja**
  - Penggunaan Alat Ukur
  - Prosedur Praktikum
  - Perakitan Rangkaian
  - Pengambilan Data
  - Kerja Sama Tim
  - Kerapihan Meja Kerja
- **Laporan Akhir**
  - Sistematika Penulisan dan Bahasa
  - Data Pengamatan
  - Analisis
  - Kesimpulan
  - Tugas Akhir

dengan **Komponen Penilaian**:

- **Praktikum** **50%**
  - Tes Pendahuluan 30%
  - Lap. Pendahuluan 20%
  - Kerja 25%
  - Lap. Akhir 25%
- **Proyek Alat** **25%**
  - Presentasi 40%
  - Paper 30%
  - Alat 30%
- **UAS** **25%**
- **Total** **100%**

Depok, 20 Februari 2018
Ketua Laboratorium Elektronika

Sastra Kusuma Wijaya, Ph.D

MODULE 1
DIGITAL INTEGRATED CIRCUITS: AND GATE, OR GATE, THE INVERTER, THE NOR GATE, THE NAND GATE

OBJECTIVES

1. To become familiar with the characteristics and symbols of and AND gate and an OR Gate.
2. To determine experimentally the truth table of a combined AND gate and OR gate.
3. To determine experimentally the truth table for a NOR gate.
4. To use NOR logic to construct a logic inverter.
5. To use NOR logic to construct a NAND gate and determine a truth table for this gate.

BASIC INFORMATIONS

In this preceding experiments you worked with linear ICs. In the remaining experiments you will study digital ICs. Digital ICs are *logic circuits*, the building blocks of digital computers and calculators. The basic digital circuits are rather simple and will serve as an introduction to digital ICs.

Logic Circuits

In digital electronics, a gate is a logic circuit with one output and one or more inputs; an output signal occurs for certain combinations of input signals. In this experiment we examine the AND and OR Gate.

Logic circuits can be in one of two states such as on or off, high or low, magnetized or unmagnetized, and so on. A toggle switch is a simple example of a two-state device.

AND Gate

Figure 1.1 shows a diode circuit with a switch input and a load resistor of 100 kΩ. The supply voltage is +5 V. when the switch is in the ground position, the diode is forward-biased and approximately 0.7 V appears across the diode. Therefore, the output voltage is low when the input is low.

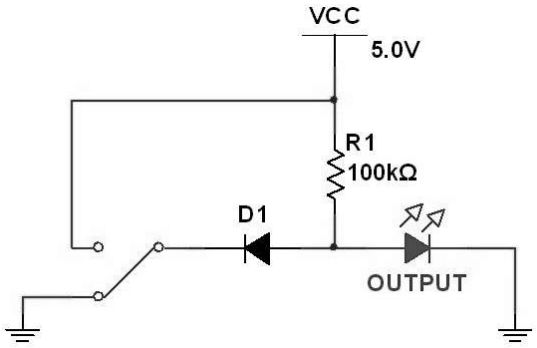


Figure 1.1. A forward-biased diode acts like a closed switch.

On the other hand, when the switch is at +5 V, the net voltage across the diode-resistor combination is 0. As a result, the diode is non-conducting. Since there is no current though the load resistor, the output is pulled up to the supply voltage. In other words, the output is HIGH (+5 V) when the input is HIGH.

Now look at the two-input AND gate of figure 1.2(a). When both switches are in the ground position, both diodes are conducting and the output is low. If S_1 is switched to +5V and S_2 is left in the ground position, then the output is still low because D_2 still conducts. Conversely, if S_1 is in the ground position and S_2 is at +5V, diode D_1 is conducting and the output is still low.

The only way to get a high output with and AND gate is to have all input high. If S_1 and S_2 are both at +5V, both diodes are non-conducting. In this case, the output is pulled up to the supply voltage because there is no current through the load resistor. By adding more diodes and switches, we can get 3-input AND gates, 4-input AND gates, and so on. Regardless of how many inputs and AND gate has, the operation is the same because it is an all-or-nothing gate. That is, all inputs must be high to get a high output. If any input is low, the output is low.

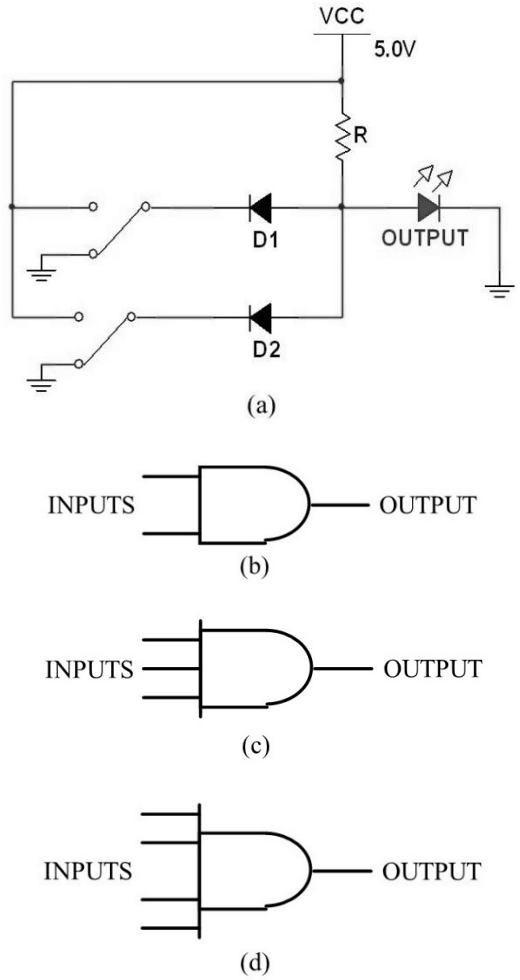


Figure 1.2. AND gate. **(a)** Diode circuit; **(b)** 2-input; **(c)** 3-input; **(d)** 4-input.

Transistors, MOSFETs, and other devices can also be used in the construction of AND gates. Figure 1.2(b) shows the schematic symbol for a 2-input AND gate of any design. Figure 1.2(c) shows the symbol for a 3-input AND gate, while figure 1.2(d) is the 4-input AND gate. For these AND gates the action can be summarized like this: All inputs must be high to get a high output.

Truth Table for Two-Input AND Gate

The action of logic circuit is usually summarized in the form of *truth tables*. These are tables that show the output for all combinations of the input signals. Table 1.1 shows the truth table for a 2-input AND gate.

Binary means 'two'. Computers use the binary number system. Rather than having digits 0 to 9, a binary number system has only digits 0 and 1. This is better suited to digital electronics where the signals are low or high, switches are open or closed, lights are off or on, and so on. In our experiments, we will use positive logic; this means binary 0 represents the low state and binary 1 represents the high state. With this in mind, table 1.2 is the truth table of a 2-input AND gate as it is usually shown. This gives the same information as table 1.1, expect it uses a binary code where 0 is low and 1 is high.

Table 1.1. Two-input AND Gate

Inputs		Output
A	**B**	
Low	Low	Low
Low	High	Low
High	Low	Low
High	High	High

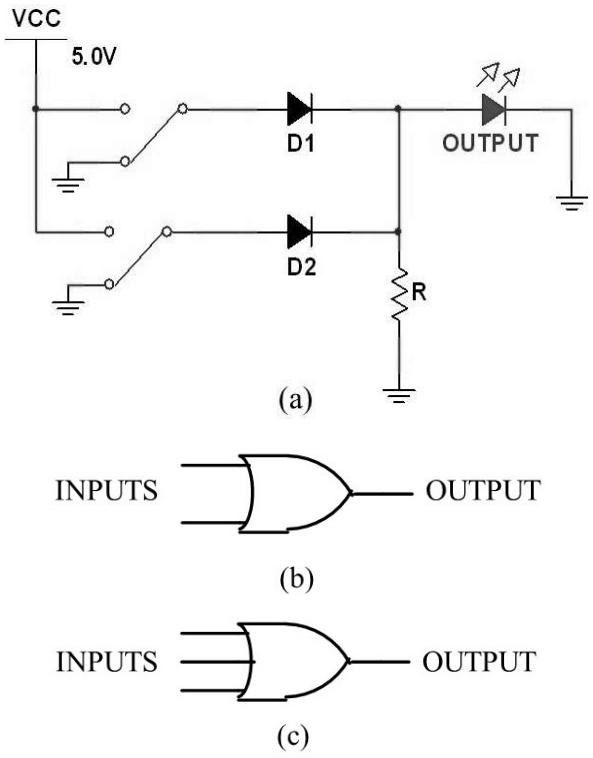
Table 1.2. Two-input AND Gate

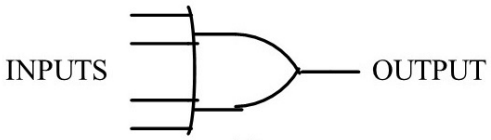
Inputs		Output
A	**B**	
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate and Truth Table

Figure 1.3(a) shows 2-input OR gate. When both switches are in the ground position, the diodes are non-conducting, and the output is low. If either switch is set to +5V, then its diode conducts and the output is approximately +4.3V. In fact, both switches can be at +5V and the output will be around +4.3V (the diodes are in parallel).

Therefore, if either input is high or if both are high, the output is high. Table 1.3 summarizes the operation of a 2-input OR gate in terms of binary 0s and 1s. As you see, if both inputs are low, the output is low. If either input is high, the output is high. If both inputs are high, the output is high.





(d)

Figure 1.3. OR Gate. **(a)** Diode circuit; **(b)** 2-input; (**c)** 3-input; **(d)** 4-input

Unlike the AND gate where all inputs must be high to get a high output, the OR gate has a high output if any input is high. Figure 1.3(b) shows the symbol for a two-input OR gate. By adding more diodes to the gate, we can produce 3-input OR gates, 4-input OR gates, and so on. Figures 1.3(c) and (d) show the schematic symbols for 3- and 4-input OR gates of any design.

Table 1.3. Two-input OR gate

Inputs		Output
A	**B**	
0	0	0
0	1	1
1	0	1
1	1	1

Combined AND-OR Gates

Combinations of AND and OR gates may be used to perform complex logic operation in computers. Figure 1.4 is an example of combining AND and OR gates. Figure 1.4 is an example of combining AND and OR gates. To analyze this circuit, consider what happens for all possible inputs starting with all low, one low, and so on. For instance, if all inputs are low, the AND gate has a low output; therefore, both inputs to the OR gate are low and the final output is low. This is the first entry sown in table 1.4.

Next, consider A low, B low, and C high. The OR gate has a high input; therefore, its final output is high. This is the second entry in table 1.4. By analyzing the remaining input combinations, you can get the other entries shown in the truth table. (you should analyze the remaining entries.)

IC Gates

Nowadays, most logic circuits are available as ICs. Transistor-transistor Logic (TTL) became commercially available in 1964. Since then, it has become the most popular family of digital ICs. In this experiment you will work with TTL gates.

An IC 7408, one of the many available ICs in the TTL family. As you see, this dual in-line package contains 4 AND gates. For this reason, it is called *quad two-input* AND gate. Notice that pin14 is the supply pin. For TTL devices to work properly, the supply voltage must be between +4.75 and +5.25 V. This is why +5V is the nominal supply voltage specified for all TTL devices. Notice also pin 7, the common ground for the chip. The other pins are for inputs and outputs.

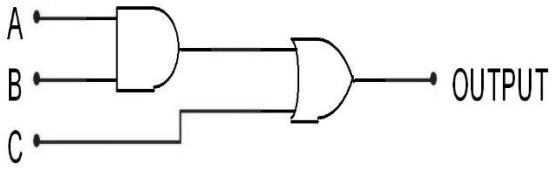


Figure 1.4. AND-OR Circuit

The four AND gates are independent of each other. In other words, they can be connected to each other or to other TTL devices such as the quad two-input OR gate (IC 7432). Again, notice pin 14 connects to the supply voltage and pin 7 to ground.

Table 1.4. AND-OR Circuit

Inputs			Output
A	**B**	**C**	
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	1

Boolean Equations

Boolean algebra is a special algebra used with logic circuits. In Boolean algebra, the variables can have only one of two values: 0 or 1. Another thing that is different about Boolean algebra is the meaning of the plus and times signs. In Boolean algebra, the '+' sign stands for the OR operation. For instance, if the inputs to an OR gate are A and B, the output Y is given by

$$Y = A + B$$

Read this equation as Y equals A OR B. Similarly, the * sign is used for the AND operation. Therefore, the output of a 2-input AND gate is written as

$$Y = A * B$$

or simply as

$$Y = AB$$

Read this as Y equals A AND B.

These expressions can be combined to describe any logic circuit. For instance, the AND gate in figure 1.4 can be expressed in Boolean algebra as AB. This output supplies one input to the OR gate whose output (and the final output of the circuit) is

$$Y = AB + C$$

NOT Logic

A NOT circuit is simply an *inverter*, as in figure 1.5(a) -an amplifier, biased to cut off- whose output is 180° out of phase with its input. When 0 V (a logic low) or no input is applied, the transistor is cut off and the output is at V_{CC}; that is, it is high. When +5 V ($+V_{CC}$ or a logic high) is applied to the base, the transistor saturates driving the collector voltage to 0.1 V, a logic low. The schematic symbol for a NOT or INVERTER circuit is shown in figure 1.5(b).

The Boolean expression for the characteristics of an inverter is given by

$$Y = \overline{A}$$

The *bar* over the A represents NOT. Thus, if the letter A represents a high level (1). $\overline{A}$ Represents low, and if A = 0, $\overline{A} = 1$.

The 7404 IC is a TTL gate with six inverters. As with the 7408 and 7432, pin 14 is the supply and pin 7 is the ground.

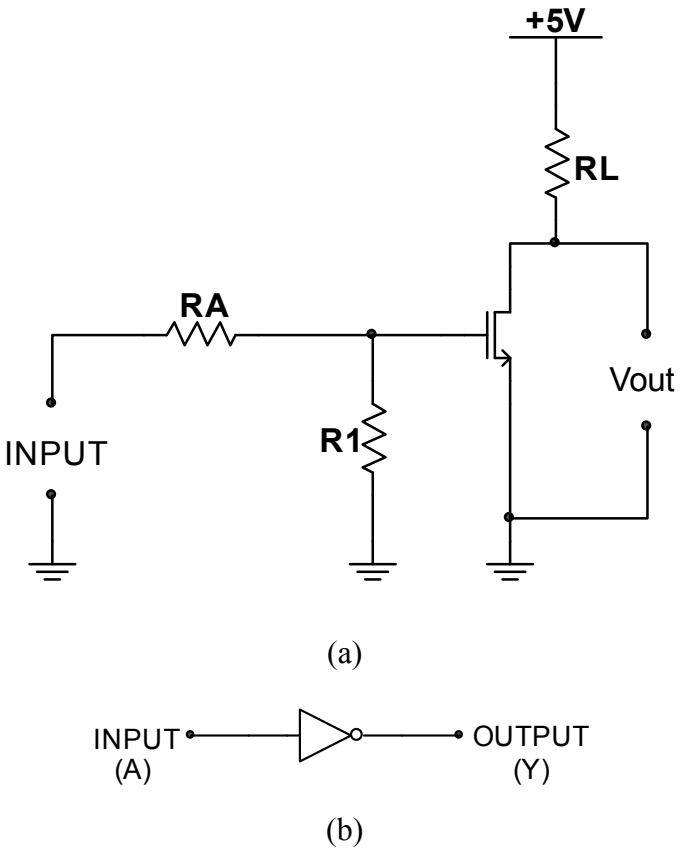


Figure 1.5. (a) NOT or Inverter Circuit; **(b)** Logic Symbol

NOR and NAND Gate

The three building-block circuits, AND, OR, and NOT, serve as the basis for other logic circuits. The NOR gate combines NOT and OR logic. What characterizes a NOR circuit is that a low input is produced when a high signal is applied to input A, *nor* to input B, *nor* to input *N*, *nor* to any combination of inputs. A high output is produced when all the inputs are low. Thus the output states for the NOT-OR or NOR are the inverse of the OR gate.

Figure 1.6 is a schematic symbol for a NOR gate with two inputs. The truth table of a 2-input NOR gate is shown in table 2.1, and the Boolean expression for a NOR gate is given by

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

A circuit which combines the NOT and AND functions is called a NAND gate. A 2-input NAND gate is show in figure 1.7, and its truth table is shown in table 1.6. The output is like that which would be produced by a NOT AND Circuit; hence the term 'NAND'. The NAND gate is therefore an AND gate with its output inverted. The Boolean expression for NAND gate is

$$\overline{A \bullet B} = C$$

TTL Logic Chips

Present state of the art employs integrated-circuit (IC) TTL logic in the manufacture of NOT, NOR, and NAND gates. ICs are nicknamed 'chips' because the actual electronics are manufactured on small-size substrates that appear as chips from a larger block of material. In this experiment you will use the 7427, a TTL positive-logic IC. This device is a triple 3-input NOR gate.

Figure 1.8 is a top view of the 7427 showing the inputs and outputs of each of the three gates. Also shown are the connection for $+V_{CC}$, terminal 14, and the connection for the ground in terminal 7. The 7427 operates with a supply +5V.

De Morgan's Theorem

It is desirable to connect gates together in as few a number as possible to create a desired output result given a fixed set of input conditions. Alternatively, it may be necessary to utilize one type of gate to produce several other logic functions. Purchasing one IC type in bulk quantity has the advantage of reducing the cost of these chips.

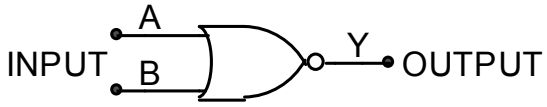


Figure 1.6. Two-input NOR-gate Logic Symbol

Two theories are used to facilitate the objectives. The firs, Boolean Algebra, utilizes rules based on logic gate operations. The later, De Morgan's Theorem, is examined here.

Table 1.5. Two Input NOR-Gate

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

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

De Morgan states, simply, that the inverse of Boolean relationship is expressed as a new relationship that is the opposite in value and function of the original. That is, the state of the input is inverted (A to $\overline{A}$) and the function is inverted (OR to AND and AND to OR). To apply this concept, consider the Boolean expression for a NOR gate.

To 'demorganize' that expression, first invert each input and the function, so that the Boolean expression becomes

$$Y = \overline{A} \bullet \overline{B}$$

De Morgan's theorem states that these two expressions are identical; that

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

The truth tables for these two produce results the same as those of table 1.5. Take one set of inputs, say, A = 0 and B = 0, and apply them to both expressions. The original NOR expression says A + B inverted. 0 + 0 results in a zero. Inverting this produces a final result of 1.

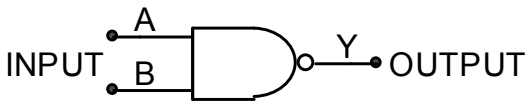


Figure 1.7. Two-input NAND-gate Logic Symbol

Now the demorganized expression, an inverted is AND with B inverted. In this example, a 0 inverted is a 1, and 1 AND 1 produces a result of 1. Note that both expressions produced the same result for the same input condition.

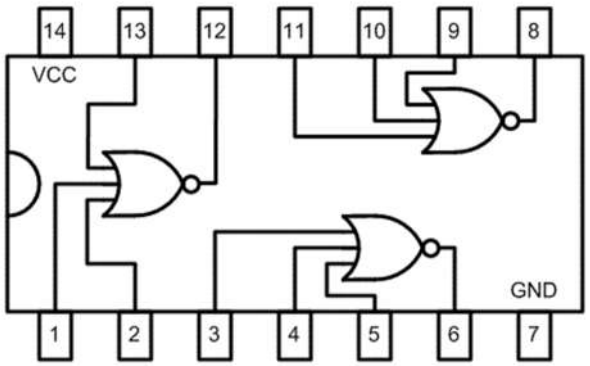
Table 1.6. Two-input AND-Gate

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

$$\overline{A \bullet B} = C$$

SUMMARY

1. Digital electronics deals with voltages that are in one of two states, either high or low.
2. Digital circuits are called *logic circuits* because certain combinations of inputs determine the output.
3. In positive logic, a binary 0 represents low voltage and a binary 1 is high voltage.
4. The simplest logic circuits are 2-input OR gates and 2-input AND gates
5. All inputs must be high to get a high output an AND gate
6. An OR gate has a high output if any input is high.
7. A *truth table* is a concise summary of all input output combinations.
8. TTL is the most popular family of digital ICs.
9. A NOT circuit is a logic inverter, converting a binary 1 into a 0 or a 0 into a 1
10. A NOR gate is an OR circuit whose output is inverted. It is a NOT OR gate
11. A NAND gate is an AND circuit whose output is inverted.
12. The truth table of NAND gate is that of an AND gate, with the output inverted.



POSITIVE LOGIC: $\overline{A + B + C} = Y$

Figure 1.8. Top View and Block Diagram of a 7427

SELF-TEST

1. Are digital circuits the same as linear circuits?
2. In a 3-input AND gate all inputs must be _____ to get a _____ output.
3. In a 4-input OR gate at least _____ input must be high to get a _____ output.
4. With positive logic, a binary 0 represents the _____ state and a binary 1 the _____ state.
5. _____ is the most popular family of digital ICs, two examples being the 7408 and the 7432. The

first is a _____ 2-input AND gate and the second is a quad two-input OR gate.
6. The nominal supply voltage for TTL is _____.
7. If *each* of the inputs of a three-input NAND gate is high, the output is _____.
8. The expression 1 + 0 represents a _____ gate, one of whose inputs is _____, the other _____
9. A binary 1 is changed into a binary 0 by a circuit called a(n) _____ or _____ circuit.
10. A circuit whose logic is the inverse of AND logic is called a(n)_____ gate
11. A circuit whose logic is the inverse of OR logic is called a(n)_____ gate.
12. What is the De Morgan alternate expression for a NAND gate?

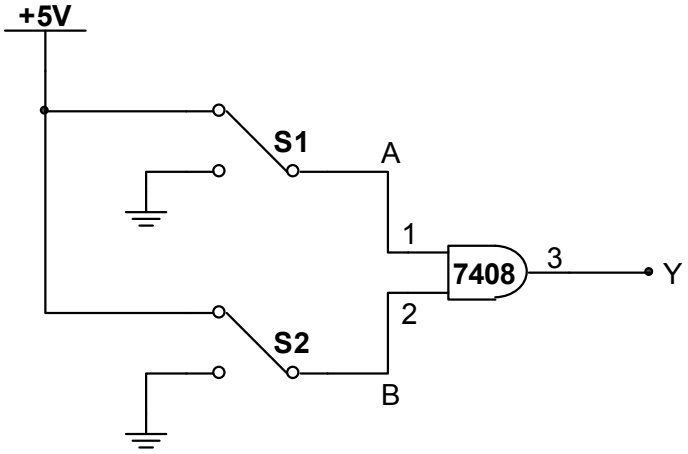
----------PROCEDURE----------

MATERIALS REQUIRED

- Power Supply DC
- Digital Multimeter
- ICs: 7408, 7432, 7427, 7404, 7400
- Resistors and switches
- Logic Breadboard; Three SPDT Switches

AND Gate

1. Connect the circuit of figure 1.9 (remember to connect pin 14 to + 5V and pin 7 to ground)
2. Set the switches as needed to get the different input combinations shown in table in figure 1.9. Record the state of the output as a 0 or 1 for each input possibility

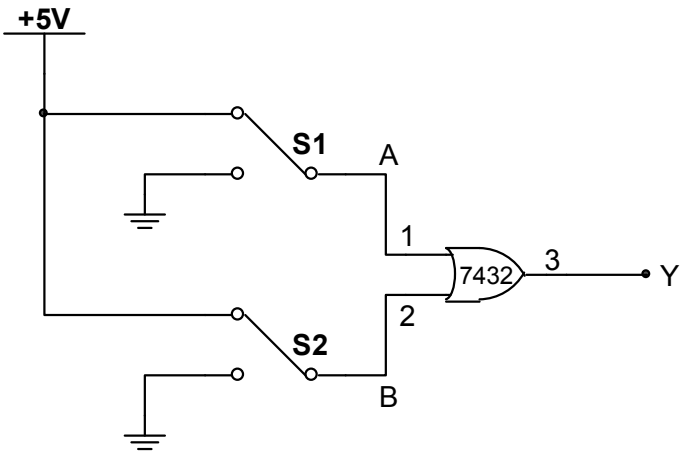


Inputs		Y
A	B	
0	0	
0	1	
1	0	
1	1	

Figure 1.9. AND Gate Experiment

OR Gate

1. Connect the circuit of figure 1.10.
2. Measure the output voltage for each input combination of table shown in figure 1.10. Record



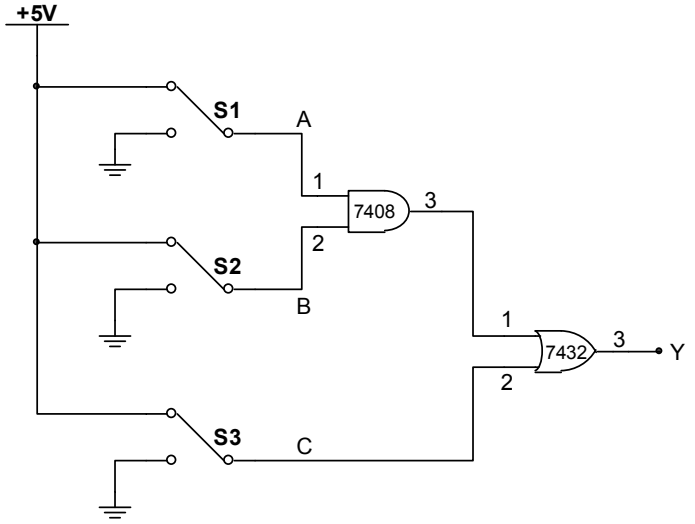
Inputs		Y
A	B	
0	0	
0	1	
1	0	
1	1	

Figure 1.10. OR Gate Experiment

Combined AND-OR Gate

1. Connect the circuit of figure 1.11.
2. Set the switches for each input shown in figure 1.11. Record the output states as 0s and 1s.
3. Design a 3-input circuit with any combination of gates to get a high output only when all inputs are high. Draw the circuit!
4. Verify the circuit experimentally. Record your results in a truth table. What is the Boolean expression of this circuit?
5. Design a 4-input OR gate using any combination of gates. Draw the circuit.

6. Verify the circuit experimentally and record the results in a truth table. What is the Boolean expression of this circuit?



Inputs			*Y*
A	*B*	*C*	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

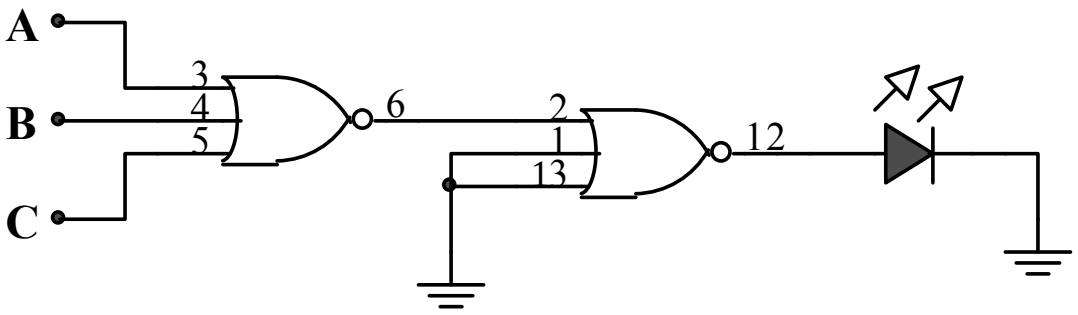
Figure 1.11. Combined AND-OR Experiment

NOR Gate Logic

1. Connect the circuit shown in figure 1.12 and complete the truth table. Take the picture of each steps.
2. Connect the circuit shown in figure 1.13 and complete the truth table. Take the picture of each steps.
3. Connect the circuit shown in figure 1.14 and complete the truth table. What is the Boolean expression for the circuit?

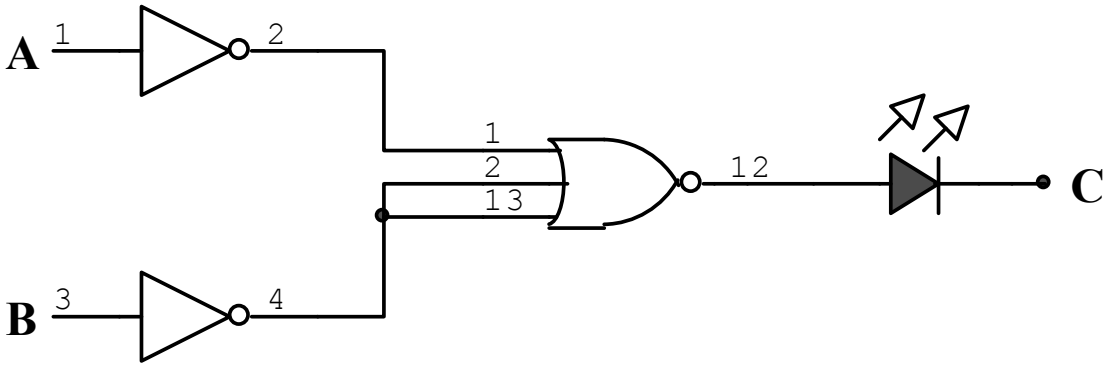
NAND Gate Logic

1. Connect the circuit shown in figure 1.12 and complete the truth table (replace the 7427 with 7410). Take the picture of each steps.
2. Connect the circuit shown in figure 1.13 and complete the truth table (replace the 7427 with 7410). Take the picture of each steps.
3. Connect the circuit shown in figure 1.14 and complete the truth table (replace the 7427 with 7410). What is the Boolean expression for the circuit?



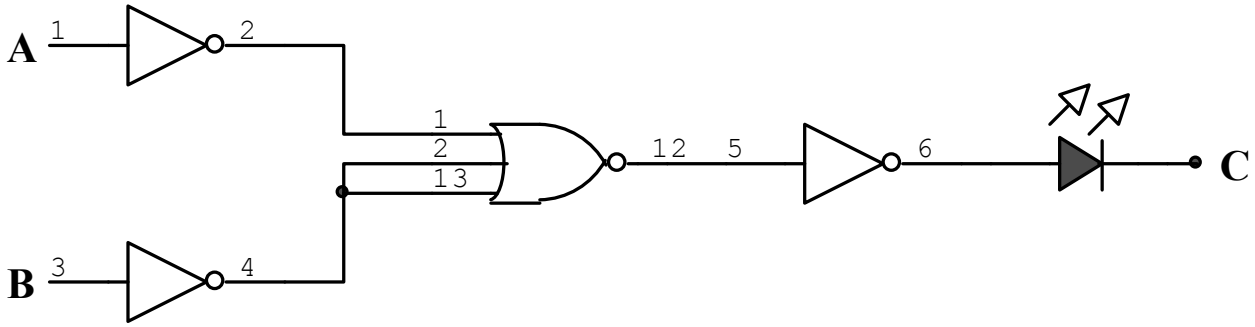
A	*B*	*C*	**OUTPUT**
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Figure 1.12. Experimental Circuit 4 and Truth Table 4



A	B	C
0	0	
0	1	
1	0	
1	1	

Figure 1.13. Experimental Circuit 5 and Truth Table 5



A	B	C
0	0	
0	1	
1	0	
1	1	

Figure 1.14. Experimental Circuit 6 and Truth Table 6

QUESTIONS

1. Using the logic family type in the experiment, identify the voltage level for the two logic states of a gate's output.
2. For your logic family, can unused inputs remain floating (open)? Explain your answer using the logic family characteristics as supporting information.
3. How many gates can the output of a single gate in your logic family drive? Contrast this to other logic families (TTL, CMOS, etc.)
4. What are the characteristics of a NOR gate?
5. What are the characteristics of a NAND gate?

MODULE 2

DIGITAL ICS: BINARY ADDITION AND THE FULL ADDER; DECODER AND ENCODER

OBJECTIVES

1. To learn the rules of binary addition.
2. To convert a decimal into a binary number, and a binary into a decimal number.
3. To explore the uniqueness of an exclusive-OR gate.
4. To construct a full adder using IC logic blocks.
5. Study of 8 to 3 lines encoder.
6. Study of 3 to 8 lines decoder.

BASIC INFORMATIONS

Binary Number

The binary system of arithmetic uses only two symbols (0 and 1) to represent all quantities. This system finds wide use in computers because the 0 and 1 are easily represented by the 2-state digital circuits.

Counting is started in the binary system in the same way as in the decimal system with 0 for zero and 1 for one. But at 2 in the binary system there are no more symbols. Therefore, the same move must be taken at two in the binary system that is taken at 10 in the decimal system: It is necessary to place a 1 in the position to the left and start again with a 0 in the original position. Table 2.1 is a list of numbers shown in both decimal and binary form.

The order of binary number is not designated unit, tens, hundreds, thousands, and so forth, as in the decimal system. Instead, the order is 1, 2, 4, 8, 16, 32, and so on, reading from right to left with the position farthest to the right being 1. Table 2.2 shows more decimal quantities and their equivalents in binary form. Note how the positions are numbered right to left.

Table 2.1. Decimal and Binary Numbers

Decimal	*Binary*	*Decimal*	*Binary*
0	0	6	110
1	1	7	111
2	10	8	1000
3	11	9	1001
4	100	10	1010
5	101	11	1011

These values are found by raising the base radix (2) by an exponential value equivalent to its position in the number. The smallest binary digit called the *least significant bit* (LSB) is binary digit position 0. It has a numerical weight of $2^0 = 1$. The weight of the next digit is $2^1 = 2$, then $2^2 = 4$, and so forth. Notice that each position weight is twice that of the preceding digit.

Converting binary values to decimal is achieved by multiplying the position weight of each digit by the value (1 or 0) in the position. These products are added to produce the final decimal equivalent of the original binary number. For example, let us convert 110101 to its decimal value. There are six binary digits with the LSB in rightmost place. The weights of these digits (bits) are LSB = 1 and then 2, 4, 8, and 16 and finally, 32. Thus, $110101_2 = 53_{10}$. The subscript denotes the base value of the number system used for each number (2 for binary and 10 for decimal).

Table 2.2. Decimal Numbers and Their Binary Equivalents

	Binary								
Decimal	256	128	64	32	16	8	4	2	1
34				1	0	0	0	1	0
15						1	1	1	1
225		1	1	1	0	0	0	0	1
75			1	0	0	1	0	1	1

The method used to convert a decimal number to its binary equivalent may be called *divide and remainder*. Divide the original decimal value by 2, the binary bas value. The result is a quotient and a remainder. The remainder becomes the binary number starting with the LSB. Divide the quotient again by 2. The remainder is the next binary bit. The quotient result is again divided by the base value with the remainder becoming the third binary digit. This is repeated until the quotient becomes 0.

Addition of binary quantities is very simple and is based on the following three rules:

1. 0 + 0 = 0
2. 0 + 1 = 1
3. 1 + 1 = 0 with a 1 carry to the left

Table 2.3 is an example of binary addition using the rules stated.

The factors to be added are 75 and 225. Starting at the right, we have 1 + 1 = 0 with a 1 carry (rule 3).

The next position to the left is added: 0 + 1 = 1. 0 with 1 carried to the third position. The third position consists of 0 + 0 = 0 + 1(carry) = 1. This procedure given in binary form as 100101100, which is equal to 256 + 32 + 8 + 4 = 300. This sum is exactly what we would expect to get by adding the decimal quantities 225 and 75.

Binary quantities can also be subtracted, multiplied, and divided, using rules similar to those for addition.

Table 2.3. Adding Binary Numbers

	Binary Value								
Carry:	1	1					1	1	
225 =	0	1	1	1	0	0	0	0	1
+75 =	+0	0	1	0	0	1	0	1	1
300 =	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**

Exclusive-OR Gate

Figure 2.1(a) is a schematic diagram for a special circuit called an *exclusive-OR*. The Boolean expression for this circuit is $Y = A\overline{B} + \overline{A}B$. Table 2.4 is the truth table for this circuit. Output Y will be high if A is low and B is high or the reverse is true. Output Y is low whenever the two inputs are both low or both high.

Examine table 2.4 carefully and note that the output is in one state when the inputs agree and in another state when they disagree. This respect allows the exclusive OR to be used for comparing binary bit values.

Table 2.4. Exclusive-OR

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

The Boolean operator for an exclusive-OR is an OR operator (+) enclosed in a circle: $\oplus$. As such, another Boolean expression for the circuit in figure 2.1 is

$$Y = A \oplus B$$

You will find many uses for this circuit which packaged in its own IC (7486). The schematic symbol for this gate is shown in figure 2.1(b).

Binary Half Adder and Truth Table

The simplest binary adder is called a *half adder* and is capable of combining two binary numbers and providing an output and a carry when necessary. The first step in understanding the operation of a half adder is to investigate the input combinations and the resulting outputs based on the rules of binary addition. Table 2.5 is a truth table showing these combinations. The table shows that a binary 1 on one input with a 0 in the other (rule 2) results in a binary 1 sum and binary 0 carry. A binary 1 on both inputs results in a binary 0 sum and a binary 1 carry (rule 3). A binary 0 on both inputs results in a binary 0 sum and binary 0 carry (rule 1).

Table 2.5. Truth Table for Half Adder

Input		**Sum**	**Carry**
A	**B**		
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Consider the sum and carry as two separate truth table results generated by the inputs A and B. Note that the sum has generated an exclusive-OR table and the carry, an AND result. Figure 2.2 is the schematic of the circuit that produces this half adder truth table.

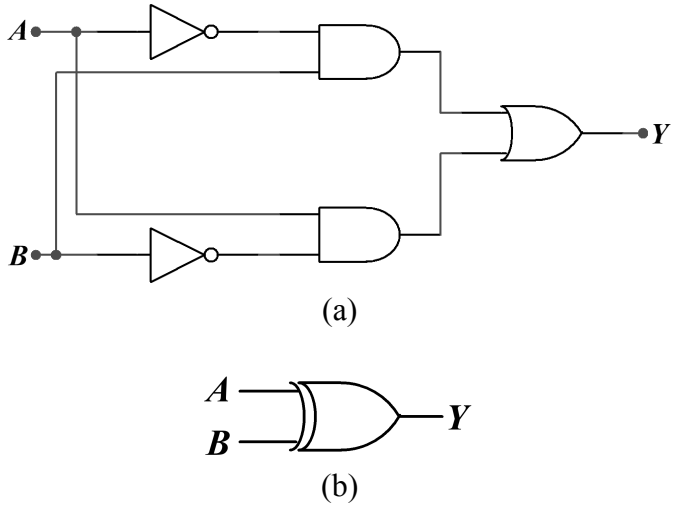


Figure 2.1. (a) Exclusive-OR; **(b)** logic schematic symbol

The half adder has only limited use because there are no provisions for a carry input from a previous adder.

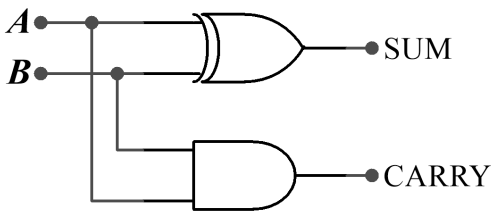


Figure 2.2. Half Adder

Binary Full Adder and Truth Table

When a carry and the two quantities to be added are considered as inputs, the input combinations increase to eight as shown in table 2.6. An adder capable of producing the required outputs for the eight input combinations is called a *full adder*. The full adder is shown in the block diagram of figure 2.3.

The full adder shown represents a single position in a binary-adder system. Because many such adders are combined in a large computer, each full adder is represented as a block in the computer logic diagram. An example of a five-position binary adder is shown in figure 2.4. The actual number of positions in such an adder depends on the size of the computer and the type of calculations the computer is designed for.

Table 2.6. Truth Table for Full Adder

Inputs			Outputs	
A	**B**	**C**	**Sum**	**Carry**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

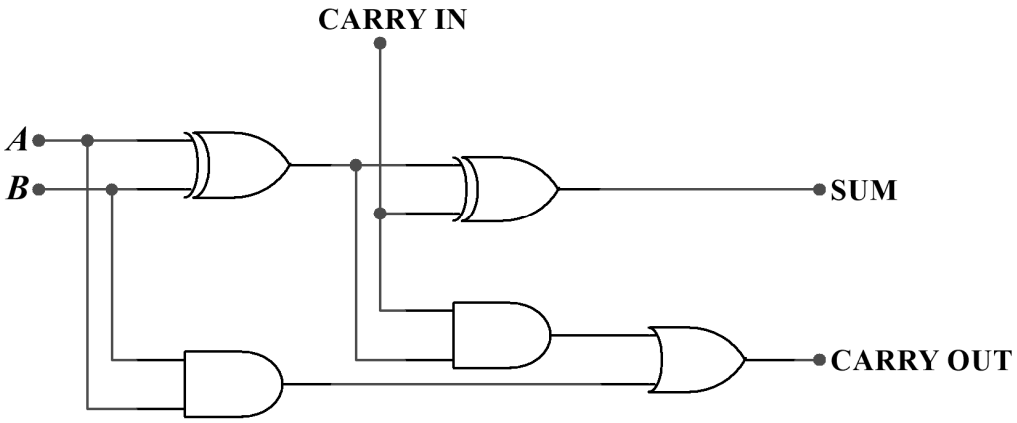


Figure 2.3. Full Adder

SUMMARY

1. The binary number system for digital computers uses only two symbols, 1 and 0. These have the same meaning as 1 and 0 in the decimal number system you are so familiar.
2. In the decimal or base-10 system the value of each digit in a number is some power of 10 and depends on its position in the number. For example, in the number 527, the 7 is in the units (10^0) column and counts for 1 x 7, or 7; the 2 is in the tens (10^1) column and counts for 2 x 10, or 20. The 5 is in the hundreds (10^2) column and counts for 5 x 10^2 or 500.
3. Numbers in the binary system are formed exactly as they are in the decimal, except the value of a column is a power of 2 rather than of 10, with the extreme right-hand column having the value 2^0 or 1. The next column on the left has the value 2^1 or 2; the next 2^2 or 4; the next 2^3 or 8; and so on. The values of the first seven binary columns, reading form right to left are:
4. To convert binary numbers to decimal, use the added weight process. To convert in the reverse direction, use the divide-and-remainder method.

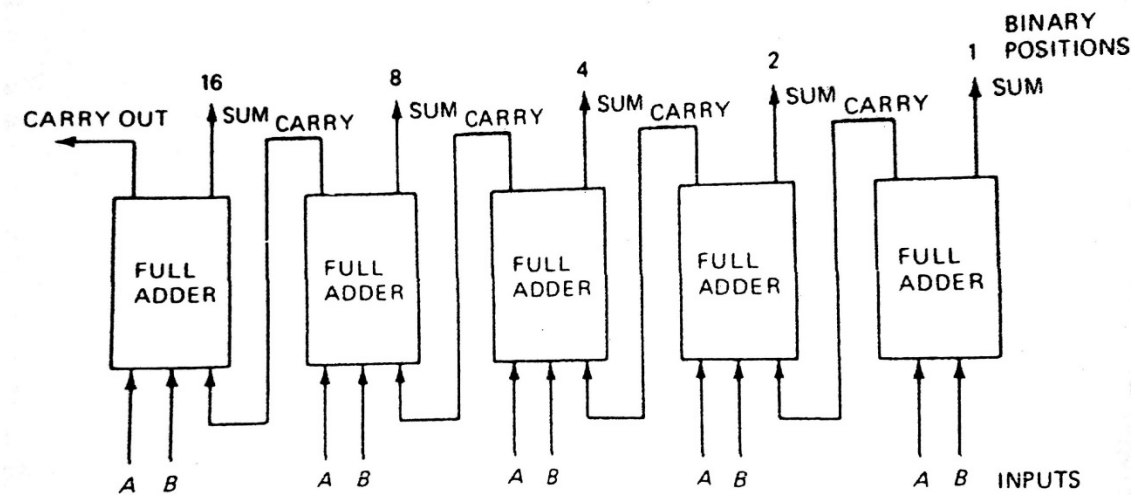


Figure 2.4. Block Diagram of Five-Position Binary Adder

5. Addition of binary numbers is based on the following rules:
   a. 0 + 0 = 0
   b. 1 + 0 = 1
   c. 1 + 1 = 0 with 1 carry to the left
6. An exclusive-OR gate is a unique circuit that produces a 0 when the two inputs are the same and a one when they are opposite of each other.
7. A *half-adder* is a binary adder (figure 2.2) which combines two binary digits and provides an output and a carry. A half adder has four possible input combinations (table 2.5).
8. A *full-adder* (figure 2.3) is a binary adder which combines three binary digits and provides an output and a carry. One of the inputs may be a carry from a previous arithmetic operation. A *full-adder* has eight possible input combinations (table 2.6).

SELF-TEST

1. A number written in binary form has one and only one equivalent decimal value. _____ (true/false)
2. The number 479 written in binary form is _____.
3. The result of adding these two binary numbers, 10011100 and 10001101 is_____.
4. The value of the number 11010011 in the decimal form is_____.
5. Give an example of application of decoder and encoder circuit!

----------PROCEDURE----------

MATERIALS REQUIRED

- Power Supply: Variable regulated low-voltage
- Digital multimeter
- Resistors
- Integrated circuits 7408, 7432, 7486, 7411, 4072
- 3 SPDT Toggle Switches, LED

Half Adder Circuit

1. Connect the *half-adder* circuit (figure 2.5)
2. Connect pin 14 of each ICs to +5 V of the supply and pin 7 to the ground.
3. Change the input as shown in table 2.7 and write down the condition of the output.

Table 2.7. Logic of Half Adder

Inputs		Outputs	
A	**B**	**Sum**	**Carry**
Low	*Low*		
Low	*High*		
High	*Low*		
High	*High*		

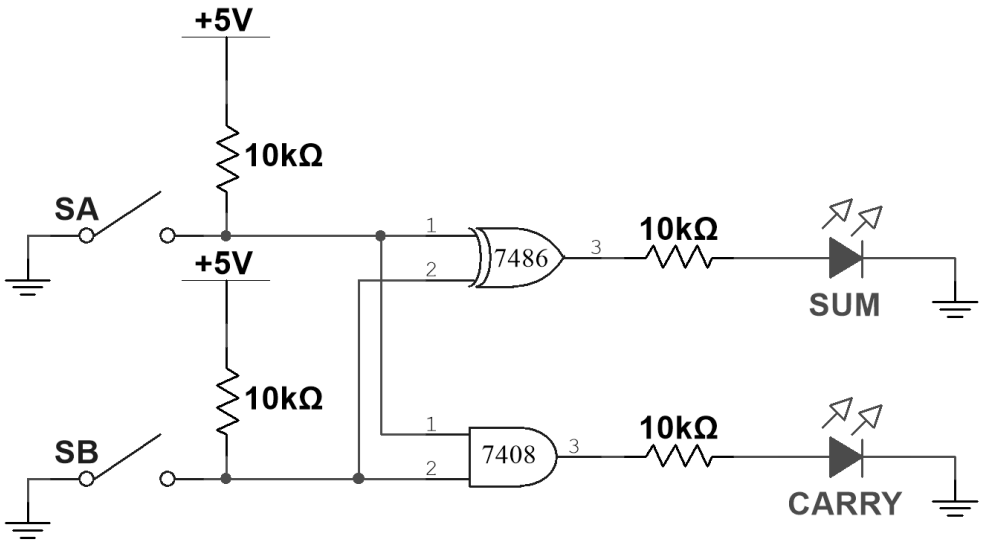


Figure 2.5. Experimental Half Adder

Full Adder Circuit

1. Connect the *full-adder* circuit (figure 2.6).
2. Connect pin 14 of each ICs to +5 V of the supply and pin 7 to the ground.
3. Change the input as shown in table and write down the condition of the output.
4. Using techniques developed for the full adder, create a truth table for a full subtractor. What is the Boolean expression for the difference and borrow results?

Table 2.8. Logic of Full Adder

Inputs			Outputs	
A	**B**	**C**	**Sum**	**Carry**
Low	*Low*	*Low*		
Low	*Low*	*High*		
Low	*High*	*Low*		
Low	*High*	*High*		
High	*Low*	*Low*		
High	*Low*	*High*		
High	*High*	*Low*		
High	*High*	*High*		

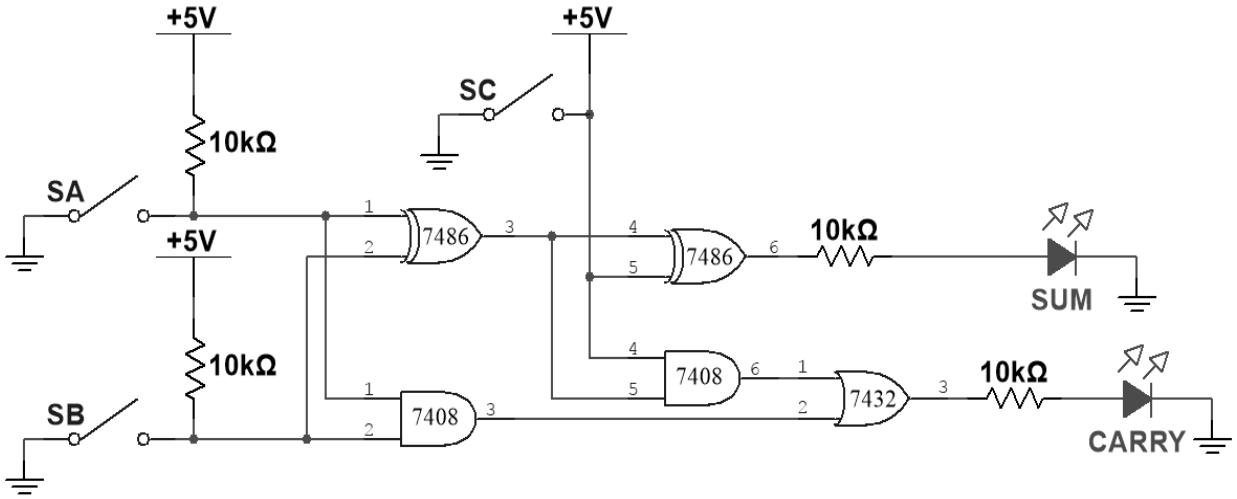


Figure 2.6. Experimental Full Adder

8 to 3 Lines Encoder Circuit

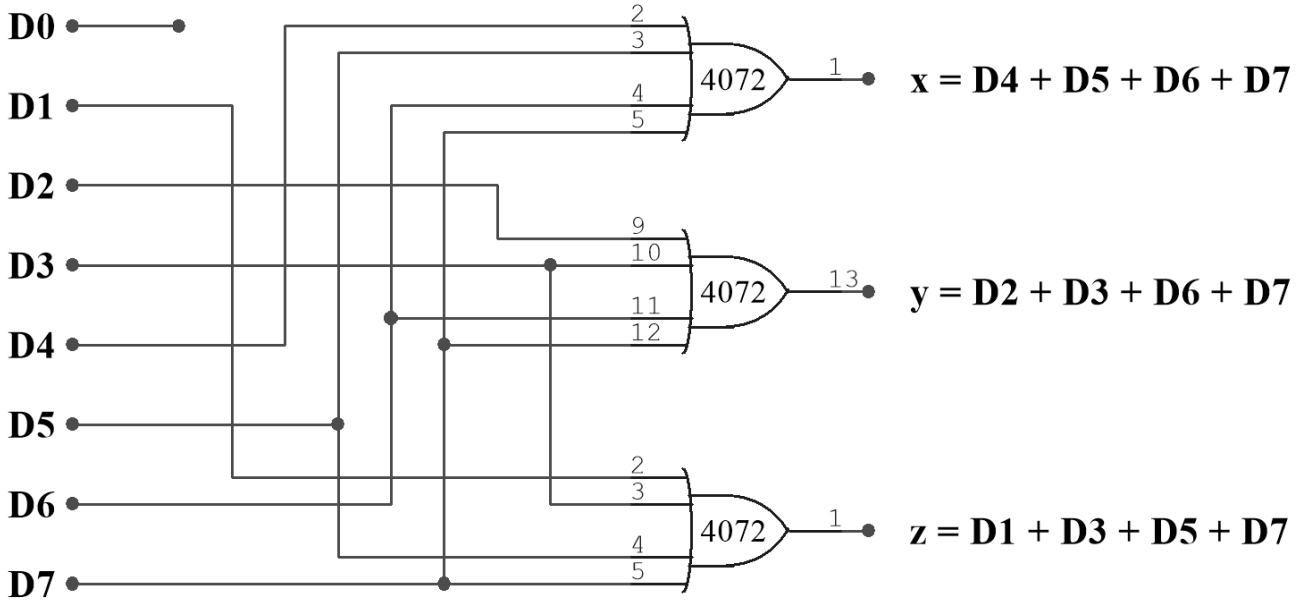


Figure 2.7. Experimental Circuit of 8 to 3 Lines Encoder

1. Make connections as shown in figure 2.7!
2. Connect pin 14 of each ICs to +5 V of the supply and pin 7 to the ground!
3. Connect input 1 or 0 to encoder circuit as shown in figure 2.7 as per truth table!
4. Switch on the instrument!
5. Observe output on 8 bits LED display!
6. Repeat step number 3 to 5 for other input combinations!
7. Verify the truth table!

Table 1.9. Truth Table for 8 to 3 Lines Encoder

D0	D1	D2	D3	D4	D5	D6	D7	x	y	z
1	0	0	0	0	0	0	0	**0**	**0**	**0**
0	1	0	0	0	0	0	0	**0**	**0**	**1**
0	0	1	0	0	0	0	0	**0**	**1**	**0**
0	0	0	1	0	0	0	0	**0**	**1**	**1**
0	0	0	0	1	0	0	0	**1**	**0**	**0**
0	0	0	0	0	1	0	0	**1**	**0**	**1**
0	0	0	0	0	0	1	0	**1**	**1**	**0**
0	0	0	0	0	0	0	1	**1**	**1**	**1**

3 to 8 Lines Decoder Circuit

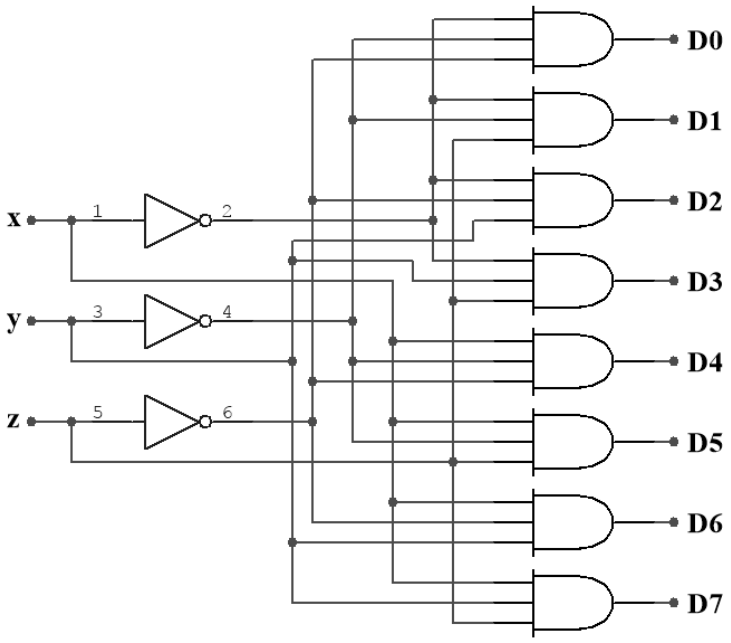


Figure 2.8. Experimental 3 to 8 Lines Decoder

1. Make connections as shown in figure 2.8!
2. Connect pin 14 of each ICs to +5 V of the supply and pin 7 to the ground!
3. Connect input 1 or 0 to decoder circuit as shown in figure 2.8 as per truth table!
4. Switch on the instrument!
5. Observe output on 8 bits LED display!
6. Repeat step number 3 to 5 for other input combinations!
7. Verify the truth table!

Table 1.10. Truth Table of 3 to 8 Lines Decoder

x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	0	1	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
0	1	0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0	1	1	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
1	0	0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
1	0	1	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
1	1	0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

QUESTIONS

1. Why is the binary number system preferred to the decimal system for use in computers?
2. What is the main difference of half-adder and full-adder?
3. Write the quantity 8999 in binary form!
4. Convert the binary quantity 10011111 to its decimal equivalent!
5. Explain 3 example of application of decoder and encoder circuit!

MODULE 3

DIGITAL ICs: FLIP-FLOPS

OBJECTIVES

1. To construct an RS flip-flop using NOR gates.
2. To observe the action of a D flip-flop.
3. To observe the action of a T flip-flop.
4. To observe the action of a JK flip-flop.

BASIC INFORMATION

RS Flip-Flop

Figure 3.1 shows the schematic symbol for a set-reset latch or RS flip-flop. A high voltage ($+V_{CC}$) applied to the set *S* input with a low (0 V) to the reset *R* input forces the output *Q* to V_{CC} (high) and $\overline{Q}$ low (0 V). A high *S* input therefore sets the output to 15 V, where it remains even though the inputs are removed.

A high reset *R* and low set *S* causes the outputs to switch or flip-flop to a high $\overline{Q}$ and a low *Q*. This is referred to as the *reset condition* of the flip-flop. The circuit remains latched in its current condition until the reverse input conditions are applied. The circuit latches in either of two states. A high *S* inputs sets *Q* to high; a high *R* input resets *Q* to low. Output *Q* remains in a given state until triggered into the opposite state.

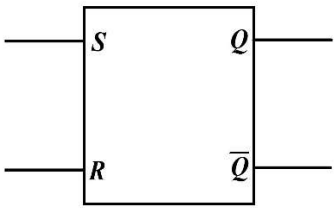


Figure 3.1. Symbol for RS Flip-flop

Table 3.1 summarizes the operation. When both control input are low, no change can occur in the output and the circuit remains latched in its last state. This condition is called the *inactive state* because nothing changes.

When *R* is low and *S* is high, the circuit sets the *Q* output to a high. On the other hand, if *R* is high and *S* is low, the *Q* output resets to low. The $\overline{Q}$ output is the inverse of the Q output.

Look at the final entry of table 3.1. *R* and *S* are high simultaneously. This is called an *invalid condition*; it is never used because it leads to paradoxical operation. It means you are trying to set and reset the flip-flop at the same time, which is a contradiction. From now on, an asterisk in a truth table indicates an invalid condition.

Table 3.1. RS Latch

R	*S*	*Q*	Comment
0	0	NC	No Change
0	1	1	Set
1	0	0	Reset
1	1	*	Invalid

NOR Latches

Figure 3.2(a) is a NOR latch, or RS flip-flop. As shown in table 3.1, a low *R* and a low *S* produce the inactive state; in this state, the circuit stores or remembers. A low *R* and a high *S* represent the set state, while a high *R* and a low *S* give the reset state. Finally, a high *R* and a high *S* produce an invalid condition, where the output is uncertain; therefore, we must avoid $R = 1$ and $S = 1$ when using NOR latch.

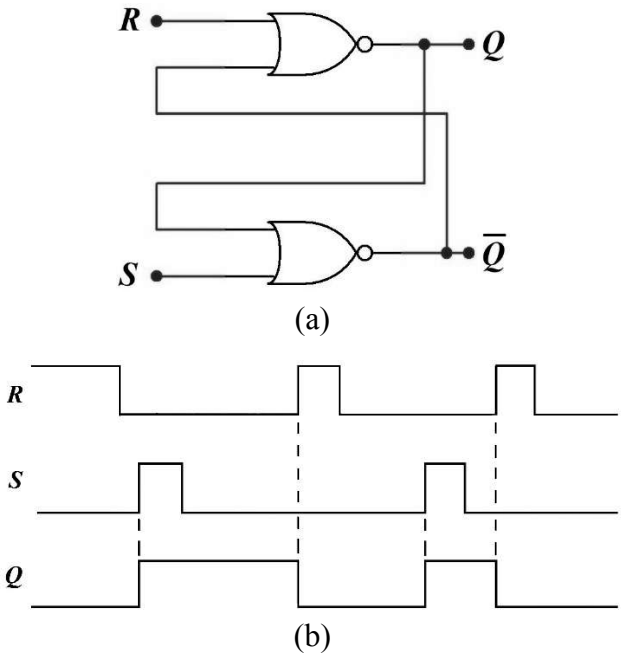


Figure 3.2. NOR-Latch Timing Diagram

NAND Latches

Figure 3.3 shows an RS latch built with cross-coupled NAND gates. Because of the NAND-gate inversion, the

inactive and invalid conditions are reversed as shown in table 3.2. Therefore, whenever you use a NAND latch, you must avoid having both inputs low at the same time.

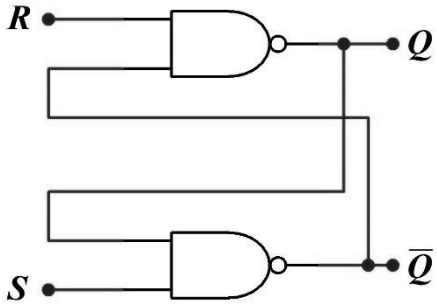


Figure 3.3. NAND Latch

Table 3.2. NAND Latch

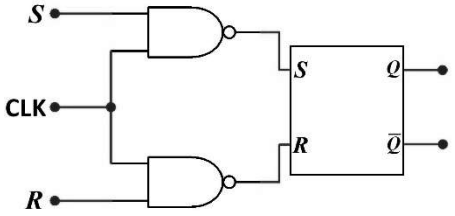
R	*S*	*Q*	Comment
0	0	*	Invalid
0	1	1	Set
1	0	0	Reset
1	1	NC	No Change

Clocking

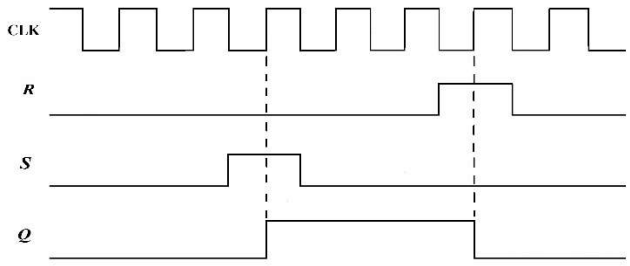
Computers use thousands of flip-flops. To coordinate the overall action, a square-wave signal called the *clock* is sent to each flip-flop. This signal prevents the flip-flops from changing states until the right time.

Figure 3.4(a) shows a clocked RS flip-flop. The idea is simple. When the clock is low, the AND gates are disabled, and the *S* and *R* signals cannot reach the flip-flop. But when the clock goes high, the *S* and *R* signals can drive the flip-flop, which then sets, resets, or does nothing depending on the values of *S* and *R*. The point is the clock controls the timing of the flip-flop action.

Figure 3.4(b) shows the timing diagram. *Q* goes high when *S* is high and *CLK* goes high. *Q* returns to the low state when *R* is high and *CLK* goes high. Using a common clock signal to drive many flip-flops allows us to synchronize the operation of the different sections of a computer.



(a)



(b)

Figure 3.4. (a) Clocked RS Flip-Flop; **(b)** Timing Diagram

Table 3.3 summarizes the operation of the clocked *RS* flip-flop. When the clock is low, the output is latched in its last state. When the clock is high, the circuit will set if *S* is high or reset if *R* is high. *CLK*, *R*, and *S* all high simultaneously is an invalid condition, which is never used deliberately.

Table 3.3. Clocked NAND Latch

R	*S*	*CLK*	*Q*
0	0	0	NC
0	1	0	NC
1	0	0	NC
1	1	0	NC
0	0	1	NC
0	1	1	1
1	0	1	0
1	1	1	*

D Latches

A data or *D* flip-flop is specifically designed to store the data state inputted to it and to hold that information until the data is changed and the flip-flop is clocked.

Figure 3.5 shows one way to build a *D* latch. Because of the inverter, data bit *D* drives the *S* input and the complement $\overline{D}$ drives the *R* input. Therefore, a high *D* sets the latch, and a low *D* resets it. Table 3.4 summarizes the operation of the *D* latch. Especially important, there is no invalid condition in this truth table. The inverter guarantees that *S* and *R* are always in opposite sites; therefore, it is impossible to set up an invalid condition.

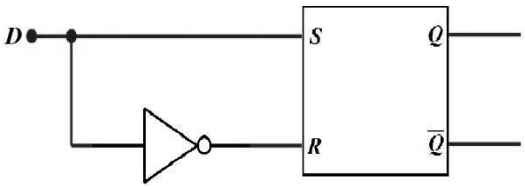


Figure 3.5. D Latch

Table 3.4. Unclocked D Latch

D	Q
0	0
1	1

Usually, a D flip-flop is clocked as shown in figure 3.6. When *CLK* is low, the AND gates are disabled and the RS latch remains inactive. When *CLK* is high, *D* and $\overline{D}$ can pass through the AND gates and set or reset latch. Table 3.5 summarizes the operation. *X* represents a "don't care" condition; it stands for either 0 or 1. While *CLK* is low, the output cannot change, no matter what *D* is. When *CLK* is high, however, the output equals the input.

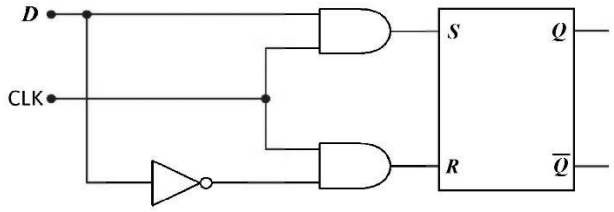


Figure 3.6. Clocked D Latch

Table 3.5. Clocked D Latch

CLK	*D*	*Q*
0	X	NC
1	0	0
1	1	1

Edge-triggered D Flip-Flops

In figure 3.7(a), the time constant of the input *RC* circuit is designed to be much smaller than the clock pulse width. Because of this, the capacitor can charge fully when *CLK* goes high; this exponential charging produces a narrow positive voltage spike across the resistor. Later, the trailing edge of the clock pulse results in a narrow negative spike.

The narrow positive spike enables the AND gates for an instant; the narrow negative spike does nothing. The effect is to activate the input gates during the positive spike, equivalent to sampling the value of *D* complement hit the latch inputs, forcing *Q* to set or reset.

This kind of operation is called *edge triggering* because the flip-flop responds only when the clock is changing states. The triggering of figure 3.7(a) occurs on the positive-going edge of the clock; this is why it is referred to as *positive-edge triggering.*

Figure 3.7(b) is the timing diagram. The crucial idea is this: The output can change only on the rising edge of the clock. Put another way, data is stored only on the positive-going edge. The truth table for the edge-triggered *D* flip-flop except that the information under *CLK* is changed from 0 to STEADY STATE and 1 to ⌐ ,indicating a positive going transition.

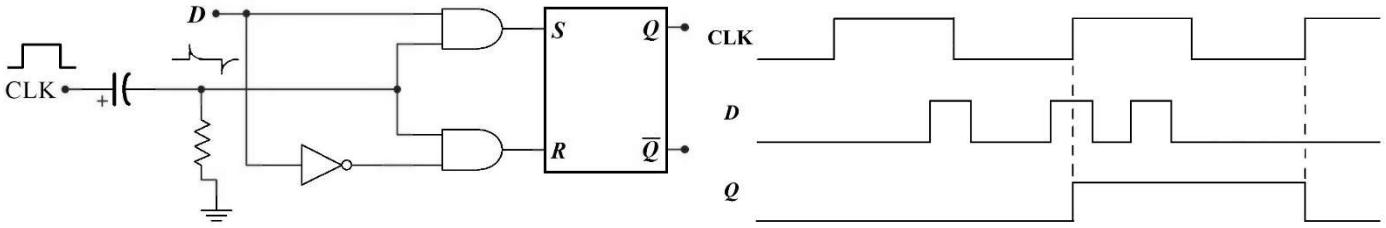


Figure 3.7. (a) Edge-triggered Flip-flip; **(b)** Timing Diagram

PRESET and CLEAR

When power is first applied, flip-flops come up in random states. To get some computers started, an operator has to push a master reset button. This is a *CLEAR* (reset) signal to all flip-fops. Also, it is necessary in some computers to *PRESET* (synonymous with set) certain flip-flops before a computer run.

Figure 3.8 shows how to include both functions in a *D* flip-flop. The edge triggering is the same as previously described. In addition, the OR gates allow us to slip in a high PRESET sets the latch: a high CLEAR resets it.

PRESET is sometimes called *direct set*, and RESET is sometimes called *direct reset*. The word 'direct' means unlocked. For instance, a clear signal may come from a push button, the output will rest when the operator pushes the CLEAR button.

Logic Symbol

Figure 3.9(a) is the logic symbol of a *positive-edge* triggered D flip-flop. The *CLK* input has a small triangle, a reminder of the edge triggering. When you see this symbol, remember what is means: The *D* input is sampled and stored on the rising edge of the clock. Also included are the PRESET and CLEAR. This means

a low PRESET will set the flip-flop; a low CLEAR will reset it. As a reminder of the phase reversal, inversion bubbles are shown on the PRESET and CLEAR inputs.

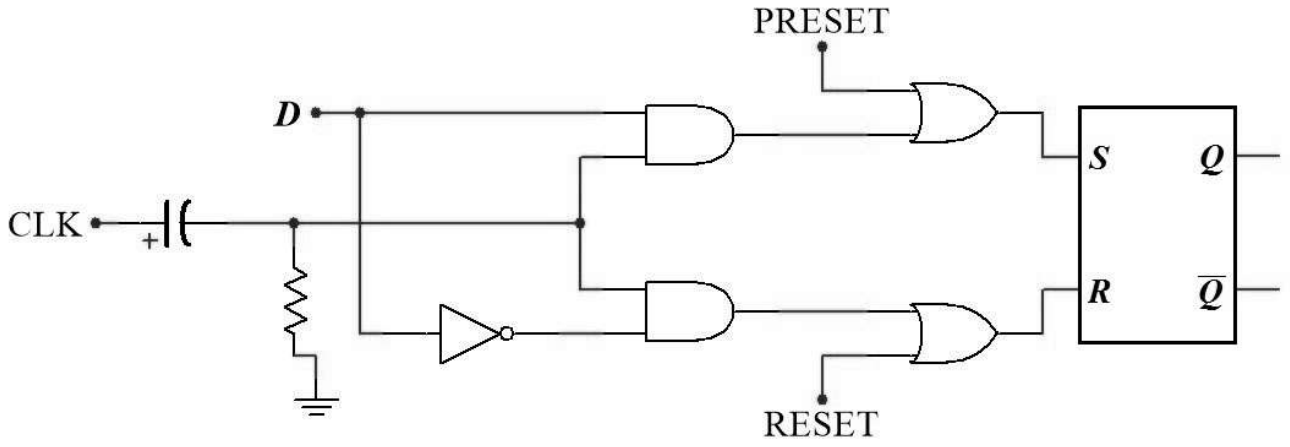


Figure 3.8. Edge-triggered D Flip-flop with PRESET and RESET

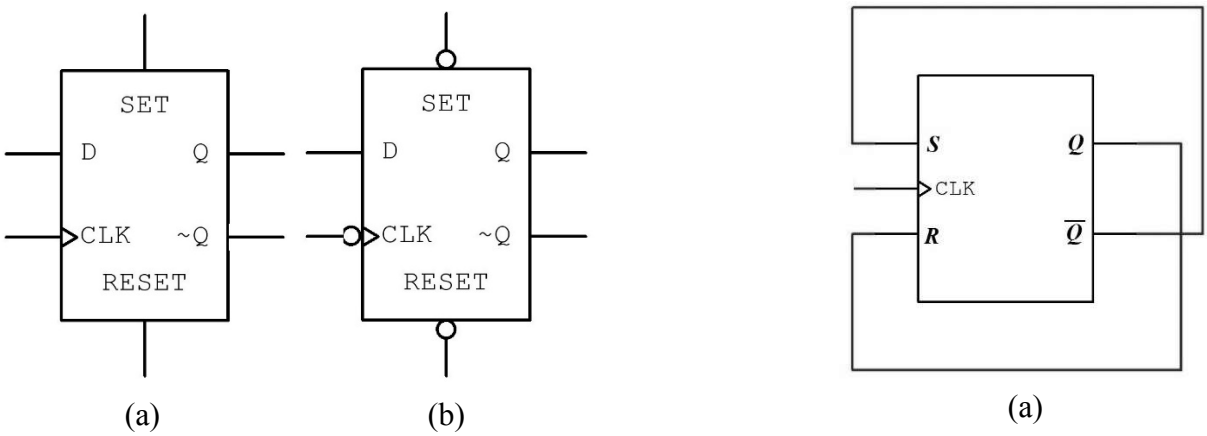


Figure 3.9. Symbols for edge-triggered D Flip-flop. **(a)** Active-high PRESET and CLEAR; **(b)** Active-low PRESET and CLEAR

Toggle Flip-Flop

Figure 3.10(a) shows a toggle flip-flop. The outputs of this flip-flop switch or toggle with every positive transition of the input clock. Because of the cross-coupling between the output and the inputs, the opposite input condition is supplied after each change of the output. Thus, the flip-flop will toggle to the opposite state when the next clock edge is applied to the *CLK* input.

Figure 3.10(b) is a timing diagram for the toggle flip-flop. Note that the output frequency at *Q* is one-half the frequency of the *CLK* input. Because of this, a toggle flip-flop is also known as a divide-by-2- flip-flop.

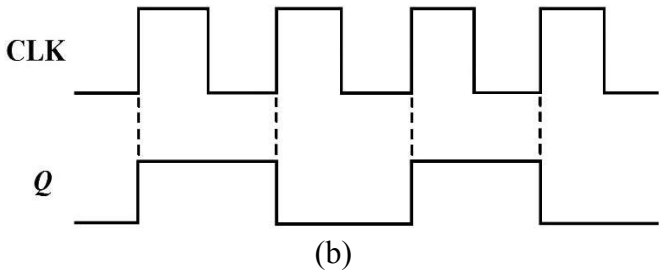


Figure 3.10. (a) Toggle Flip-flop; **(b)** Timing Diagram

Edge-triggered JK Flip-Flops

Figure 3.11(a) shows one way to build a JK flip-flop. As before, an *RC* circuit with a short time constant converts the rectangular *CLK* pulse to narrow spikes. The *J* and *K* inputs are control inputs; they determine what the circuit will do on the positive clock edge. When J and K are low, both inputs are disabled and the circuit is inactive.

When *J* is low and *K* is high, the flip-flop is reset. On the other hand, when *J* is high and *K* is low, the flip-flop is driven into the set state on the next positive *CLK*

edge. The final possibility is both *J* and *K* are high. It means the flip-flop will *toggle* on the next positive clock edge. Figure 3.11(b) is a visual summary of the action. When *J* is high and *K* is low, the rising clock edge set *Q* to high. When *J* is low and *K* is high, the rising clock edge resets *Q* to low. Finally, if both *J* and *K* are high, the output toggles once each rising clock edge.

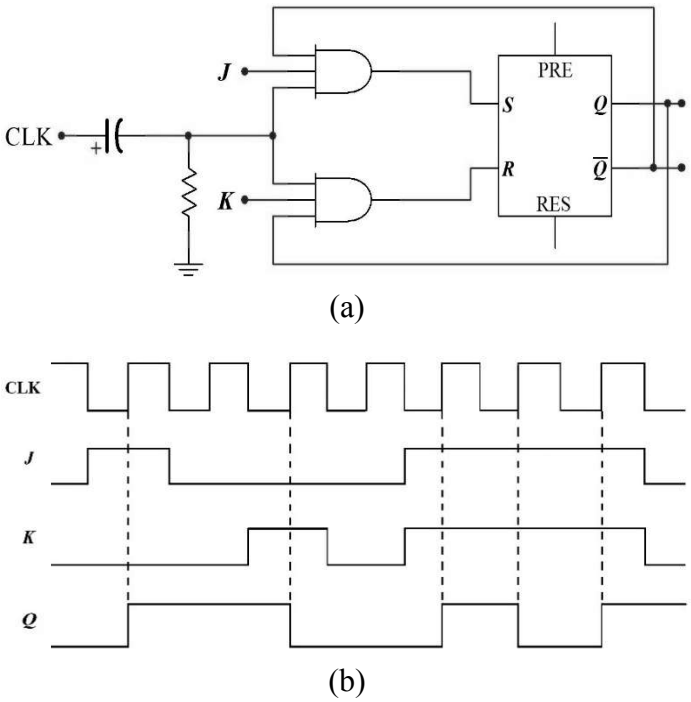


Figure 3.11. **(a)** Edge Triggered JK Flip-flop; **(b)** Timing Diagram

Table 3.6 summarizes the action. The circuit is inactive when the clock is low, high, or on its negative edge. Likewise, the circuit is inactive when *J* and *K* are both low. Output changes occur only on the rising edge of the clock as indicated by the last three entries of the table. The output either resets, sets, or toggle.

Table 3.6. Positive-edge-triggered JK Flip-flop

CLK	***J***	***K***	***Q***
0	*X*	*X*	NC
1	*X*	*X*	NC
↓	*X*	*X*	NC
X	0	0	NC
↑	0	1	0
↑	1	0	1
↑	1	1	Toggle

A variety of JK flip-flops are available in IC form. Figure 3.12(a) is the symbol for one type. It uses positive-edge triggering, and responds to high PRESET and CLEAR. Figure3.12(b) is a *positive-edge-triggered* JK flip-flop that responds to low preset and clear signal. If the IC design includes an internal inverter on the clock input, we get negative-edge triggering which is preferred in some applications. As a reminder of this negative-edge triggering, figure 3.12(c) has a bubble at the clock input; it also has active-low PRESET and CLEAR.

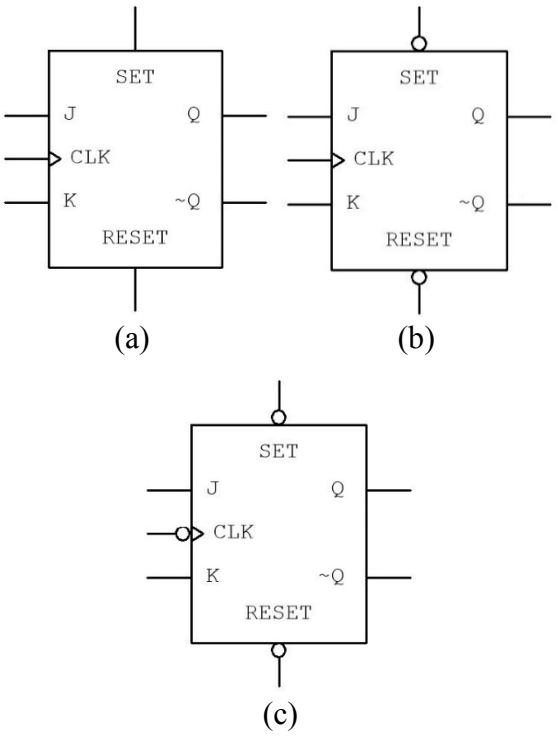


Figure 3.12. Symbols for JK Flip-flop. **(a)** Positive Edge-triggering with Active-high PRESET and CLEAR; **(b)** Positive Edge-triggering with Active-low PRESET and CLEAR; **(c)** Negative Edge-triggering with Active-low PRESET and CLEAR

SUMMARY

1. A *flip-flop* can remain in its last state until an external trigger forces it into the other state. Because of this, it is a memory element.
2. In the inactive state, a flip-flop stores or remembers because it remains in its last state.
3. An *invalid condition* when both *R* and *S* are high in an RS flip-flop. This undesirable state is forbidden because it represents a contradiction.
4. One way to build an RS flip-flop is with cross-coupled NOR gates. Alternatively, NAND gates can be used.
5. Usually, a signal called the *clock* determines when a flip-flop can change states.

6. By including an inverter, we can convert an RS flip-flop into a D flip-flop. The big advantage of the D flip-flop is the lack of an invalid condition.
7. A *positive-edge-triggered* D flip-flop stores the data bit only on the rising edge of the clock.
8. PRESET and CLEAR allow a direct set or a direct reset of a flip-flop, regardless of what the clock is doing.
9. A toggle flip-flop changes state with each clock cycle and is known as a *divide-by-2* flip-flop.
10. Depending on the values of *J* and *K*, a JK flip-flop will either do nothing, set, reset, or toggle.

SELF-TEST

1. RS flip-flops can be built with cross-coupled _____ or _____ gates.
2. A square-wave signal called the _____ can synchronize the operation of many flip-flops.
3. A flip-flop that responds only on the rising _____ of the clock is called a _____.
4. The output of a toggle flip-flop is _____ the frequency of the clock input.
5. For a JK flip-flop to toggle, J must be _____ and K must be_____.

----------PROCEDURE----------

MATERIALS REQUIRED

- AC and DC power supply +5 V
- Signal generator, oscilloscope
- ICs: 7402, 7404, 7474, 7476
- Resistors, LED, switches

RS Latch

1. Connect the NOR latch of figure 3.13!
2. Set the *R* and *S* switches to the input combinations of table 3.7! Follow the order shown; record the *Q* and $\overline{Q}$ outputs for each input!

Table 3.7. RS Latch

R	S	Q	$\overline{Q}$
0	0		
0	1		
1	0		
1	1		

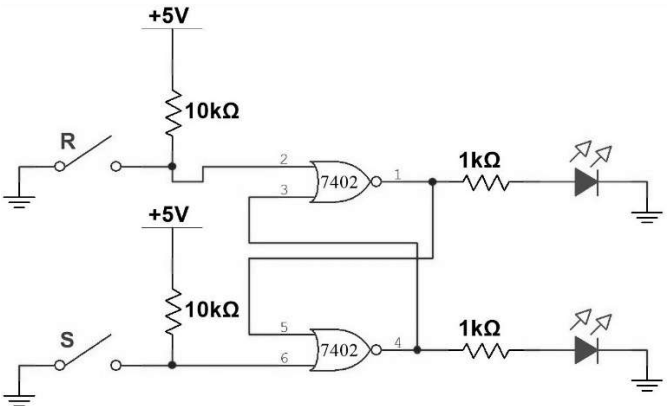


Figure 3.13. Experimental RS Latch

D Latch

1. Connect the clocked *D* latch of figure 3.14!
2. Connect a square wave generator to the *CLK* input! Set the generator for 5 V at 1 KHz!
3. Set the *D* switch to the low input! Measure and record *Q* and $\overline{Q}$ in table 3.8!
4. Repeat the preceding step for the *D* switch at the high input!

5. Remove the square-wave generator and set this input high! Observe that switching the *D* input does not cause the output to switch!

Table 3.8. D Latch

D	CLK	Q
0	↑	
1	↑	

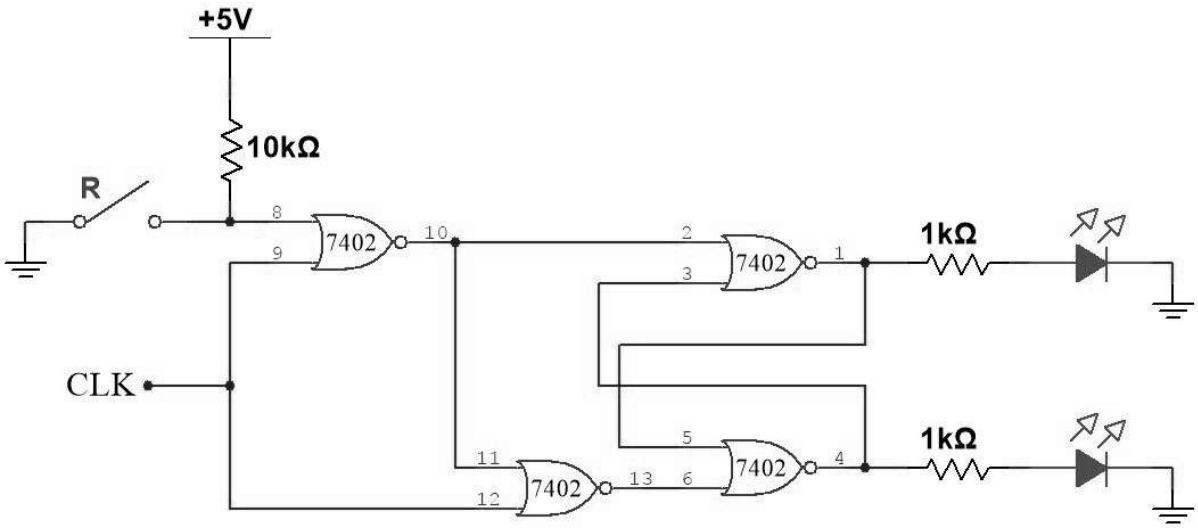


Figure 3.14. Clocked D Flip-flop

Edge-triggered D Flip-Flop

1. Connect the circuit of figure 3.15!
2. Close S_1 and ground the CLK input. Open S_2 and close S_3! Note that the flip-flop is in the reset state. Open S_3, and the *Q* output should remain low (green LED on).
3. Close S_2 (preset), and the output *Q* should go to the set condition (red LED on). Open S_2, and the flip-flop remains set.
4. Close S_1 (low input)! Remove the ground to *CLK* and replace it with the square-wave generator set as in step 1 *D* Latch! Record the *Q* output in table 3.9!
5. Open S_1 (high input)! Record the *Q* output in table 3.9!

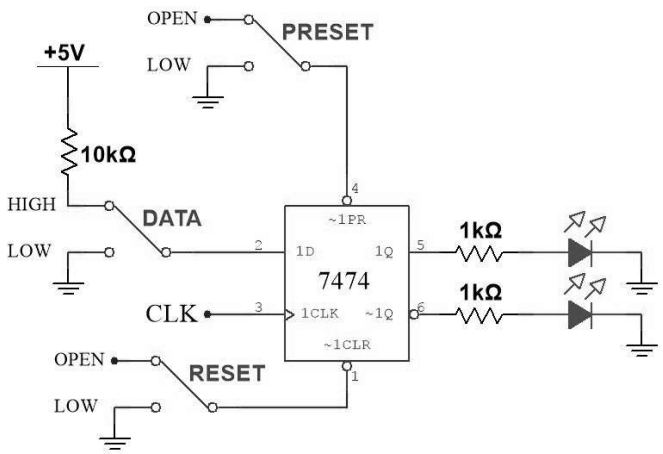


Figure 3.15. Experimental Edge-triggered D Flip-flop

Table 3.9. Edge-triggered D Flip-flop

D	CLK	Q
0	↑	
1	↑	

JK Flip-Flop

1. Connect the circuit of figure 3.16! Set *J* and *K* inputs low. Connect the square-wave generator to the *CLK* input and set it as in step 1 D Latch experiment!

2. Close S_2 and open S_4! Note how these presets the Q output. Open S_2 and close S_4, placing the J and K inputs into the reset condition!
3. Open S_2 and S_4! Q should not change. If this is what happens, write 'NC' in table 3.10!
4. Set up the other J and K inputs listed in table 3.10! Record the Q outputs (record 'toggle' for the last entry if it is working correctly)!
5. Leave both J and K high! Measure and calculate the frequency of the Q output and record the value here

$f =$ _____

Table 3.10. JK Flip-flop

J	K	CLK	Q
0	0	↑	
0	1	↑	
1	0	↑	
1	1	↑	

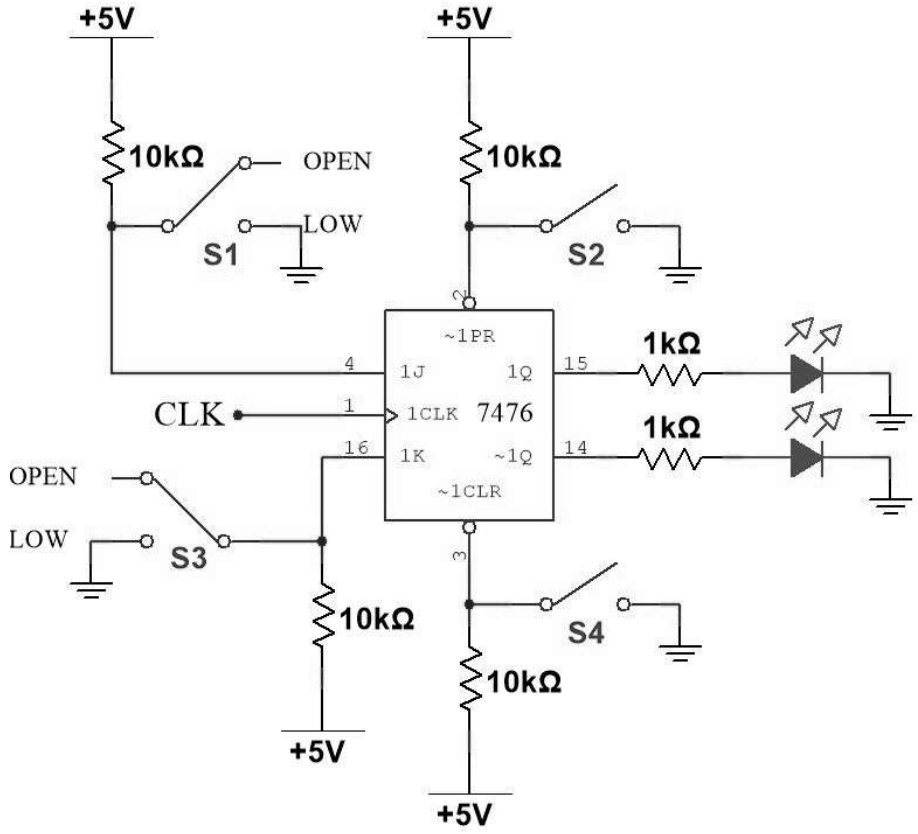


Figure 3.16. Experimental Edge-triggered JK Flip-flop

QUESTIONS

1. Describe what the Q output did when you change the R and S switches!
2. Is the D flip-flop positive- or negative-edge triggered?
3. Describe what a D flip-flop of figure 3.15 did!
4. Describe what a D latch does!
5. Are the PRESET and CLEAR active-low or active-high?
6. Explain the difference between the *CLK* and Q output frequencies!

MODULE 4

DIGITAL ICs: COUNTERS

OBJECTIVES

1. To examine Binary counting circuits.
2. To observe Ring and Johnson counter operations.

BASIC INFORMATION

A *counter* is a circuit that produces a set of unique output combinations in relation to the number of applied input pulses. The number of unique outputs of a counter is known as its *modulus, modulo,* or *mod number.*

Binary Counters

Figure 4.1 shows a binary or ripple up counter. JK flip-flops are used because of their versatility. The *J* and *K* inputs are all connected together to an ENABLE input. When ENABLE is OFF (low), all the JK inputs are held low, placing the flip-flops into a 'no change' condition. Once the ENABLE is set ON (high), the JK inputs are forced high and each flip-flop is set into a 'toggle' condition. Each flip-flop will change state when a positive to negative transition is experienced on its *CLK* input.

Applying a square-wave signal to the clock input of FF_0 and setting ENABLE high allow FF_0 to continuously toggle each time a clock pulse is applied.

Flip-flop FF_1 is clocked from the *Q* output of FF_0, so its output has a frequency that is one-half that of FF_0 or one-fourth the frequency of the input *CLK*. In turn, FF_2 is driven by FF_1 and its output is on-eighth that of the *CLK* and the FF_3 output is one-sixteenth summary of the binary counter outputs. Figure 4.2 is a timing diagram summary of the binary counter outputs. Note that the output of each flip-flop toggles when a low transition is applied to its clock input.

Flip-flop FF_0 is the least-significant bit in the counting sequence, followed in numerical weight by FF_1, FF_2, and finally FF_3. Assigning 1s to high outputs and 0s to low-level outputs, one can construct a truth table (figure 4.3). Note that the outputs change sequentially as the number of clock cycles (count) increases. There are 16 different output combinations or states produced by this counter, which returns to all 0s on the sixteenth count. This makes this a *mod 16 binary ripple up counter*.

Counting Down

Switching the clock inputs of each flip-flop to the $\overline{Q}$ outputs causes the counting sequence to start at 1111, on the first clock after releasing the reset input, down to 0000. The flip-flops still toggle on a positive-to-negative transition, but this change now comes when the *Q* output of the previous flip-flop (which is the inverse of its $\overline{Q}$) goes from a low to a high.

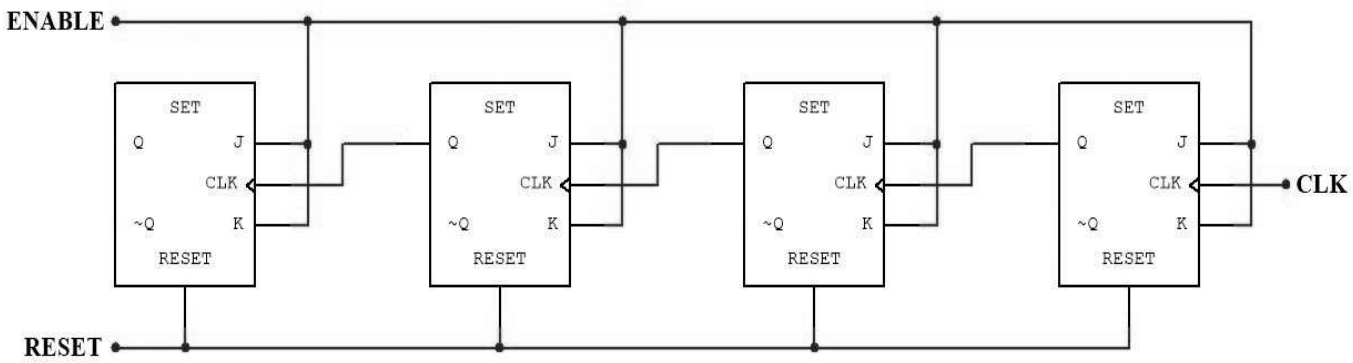


Figure 4.1. Binary Ripple Up Counter

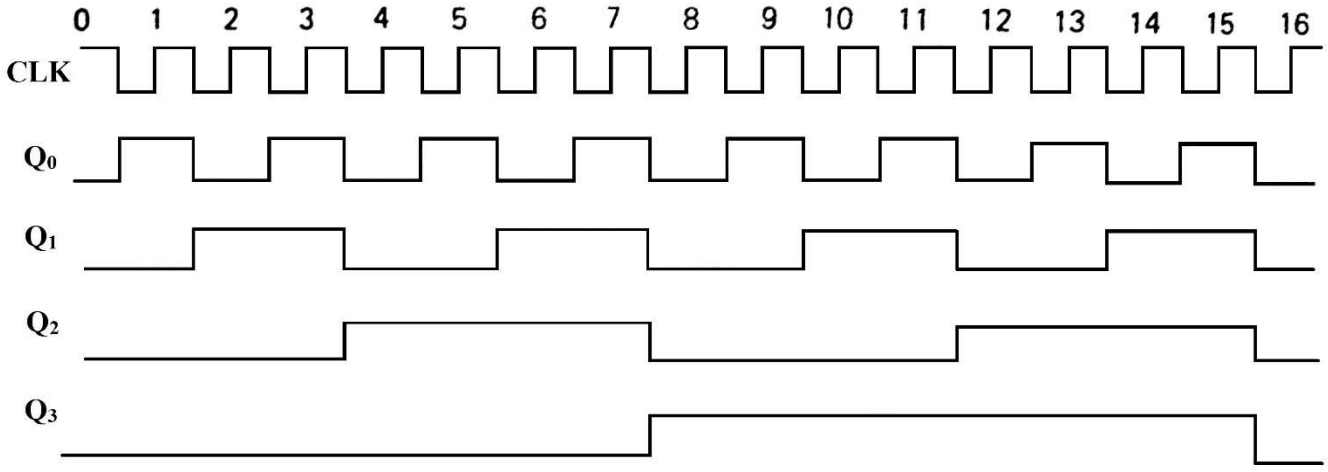


Figure 4.2. Binary Up Counter Timing Diagram

Ring counters

Figure 4.4 illustrates a ring counter. It is constructed by connecting the Q and $\overline{Q}$ outputs from one flip-flop to the *J* and *K* inputs of the next flip-flop. To complete the ring, the outputs of the final flip-flop are wired to the inputs of the first flip-flop. This counter has the characteristic that one and only one flip-flop is set (Q=1) at any time. To start the counter, FF_0 is set and the rest are reset (note the application of the start input to the FF_0 preset input and the FF_1 to FF_3 reset input). Since the outputs of each flip-flop are connected to the inputs of the following unit, FF_1 is receiving set inputs while the others are receiving reset inputs. The effect is that on the first clock cycle FF_1 goes set while the rest are reset. The set condition has shifted from FF_0 to FF_1. As clock cycles arrive, this set condition continues to shift around the ring. The process is illustrated in the timing diagram of figure 4.5. A truth table is not required for this circuit because of the exclusive nature of the single set flip-flop for each clock cycle (count).

COUNT	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	1	0	0	0

Figure 4.3. Binary Up Counter Truth Table

As can be seen from the timing diagram, there are four unique output states for this counter, rendering it a mod 4 ring counter.

Johnson Counter

The modulo number of a ring counter can be doubled by switching the Q and $\overline{Q}$ outputs of the last flip-flop so that the $\overline{Q}$ output now feed the FF_0 *J* input and Q the *K* input. This is the configuration for the Johnson Counter (figure 4.6) which is started at 0000. Note that this condition supplies reset inputs to FF_1 send set input condition to FF_0. On the first clock input, FF_0 sets while the rest remain reset. Now set inputs are applied to FF_0 and FF_1. The next clock input sets FF_0 and FF_1 while resetting FF_2 and FF_3. The process continues until all four flip-flops are set as illustrated in figure 4.7. Once all four are set, the outputs of FF_3 then send reset inputs to FF_0. On the succeeding clocks, the process is reversed until the counter once again has all reset outputs. Note from the timing diagram that there are now eight different output conditions causing this to be a mod 8 Johnson Counter.

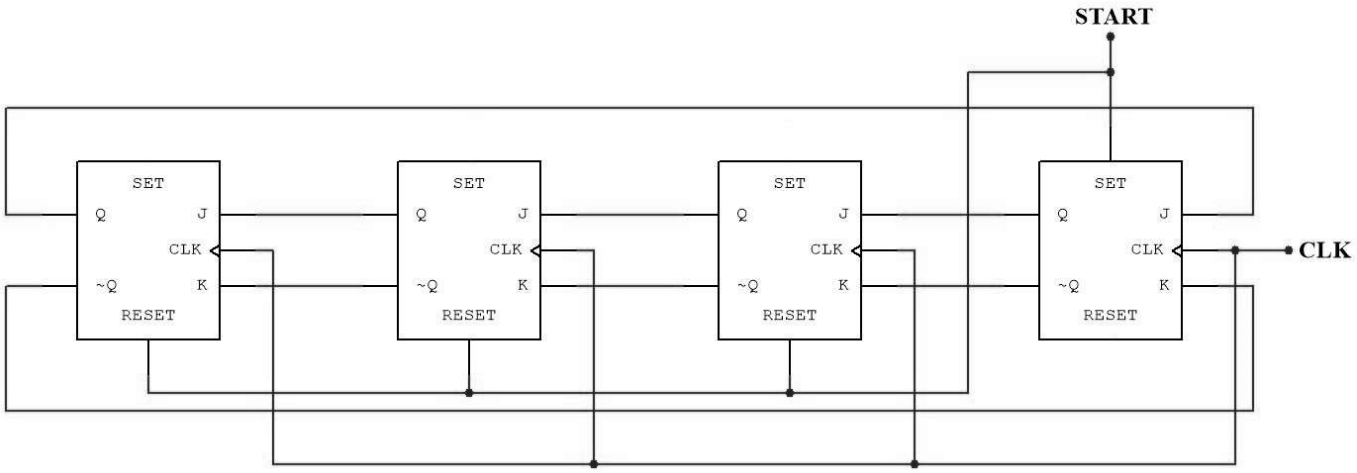


Figure 4.4. Ring Counter

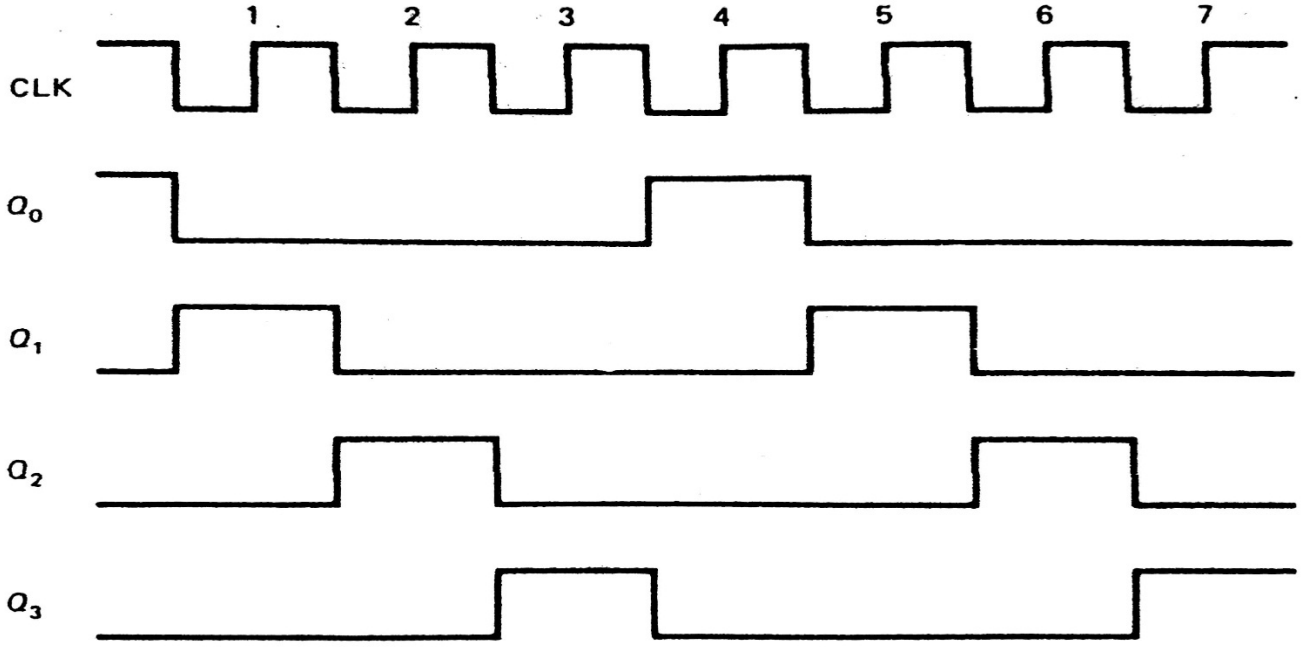


Figure 4.5. Ring Counter Timing Diagram

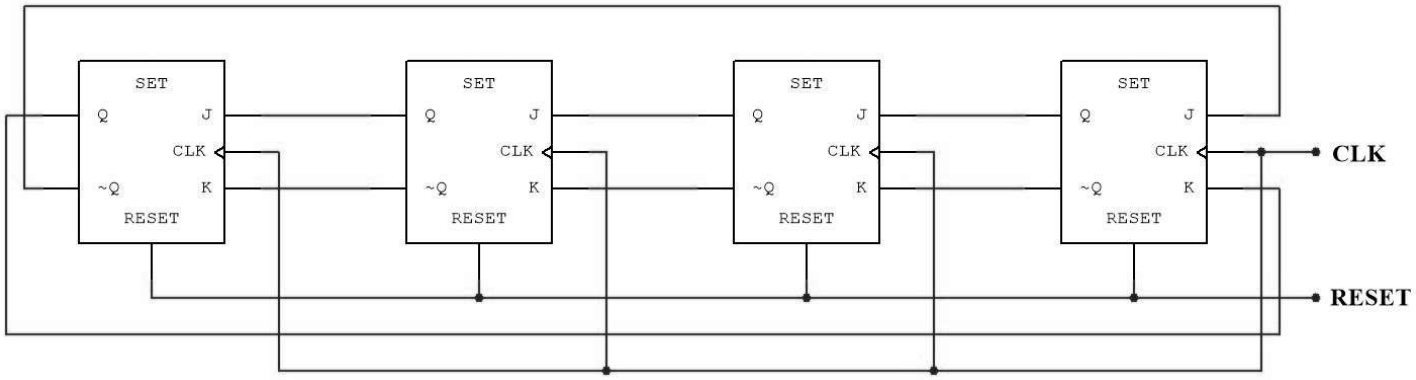


Figure 4.6. Johnson Counter

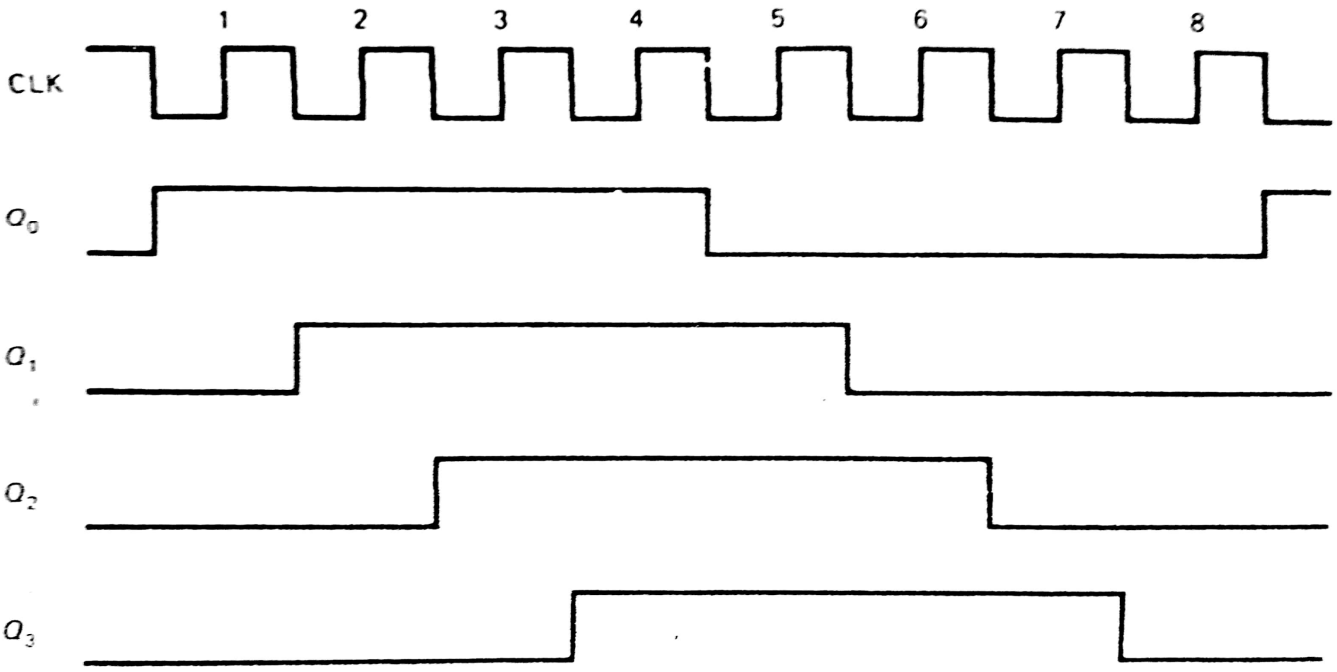


Figure 4.7. Johnson Counter Timing Diagram

SUMMARY

1. Counters are digital circuits that produce different output states for each applied clock cycle.
2. An up counter yields an increasing binary count representation as its outputs, while a down counter produces a decreasing count sequence.
3. Only one flip-flop is set in a ring counter. This set condition is the shifted through the ring on succeeding clock cycles.
4. A Johnson counter doubles the modulo number of a ring counter without the addition of any circuitry.

SELF-TEST

1. Which flip-flops are set in figure 4.1 after seven clock cycles are applied following the release of the RESET signal?
2. What is the highest binary number that could be represented by the counter in figure 4.1 if the *CLK* input to FF_3 were open?
3. Which flip-flop of the ring counter in figure 4.4 is set on the 19^{th} clock cycle after START is released?
4. What is the state of the Johnson Counter outputs in figure 4.6 following nineteen clock cycles after the release of the reset signal?

----------PROCEDURE----------

MATERIALS REQUIRED

- Power Supply: +5 V
- Multimeter, Signal Generator, Oscilloscope
- Resistors
- ICs: 7476
- Switches, LEDs

Binary Up Counter

1. Construct the binary counter of figure 4.8!
2. Close S_1 and S_2! Apply a 5 Hz 5 V square wave to the *CLK* input! What is the state of the counter at this time?
3. Open S_2! Explain why the output does not change!
4. Open S_1! What is the count sequence of this counter? Does it count up or down?
5. Close S_1 and S_2! Move the clock inputs of each flip-flop except FF_0 from the Q outputs to the $\overline{Q}$ outputs. Be careful to leave the 1 kΩ resistors and LEDs connected to the Q outputs!
6. Open S_1 and S_2! Is the counter counting up or down? What is the modulo number for this counter?
3. Close S_1! What is the state of the outputs of the counter?
4. Open S_1! Draw a timing diagram for six clock cycles! What is the modulo number for this counter? What would happen to the operation of the circuit if the reset of FF_0 were connected to the start input instead of the preset? Perform this test and verify your answer!
5. Close S_1! Switch the Q and $\overline{Q}$ connections of FF_3! Keep the 1 kΩ resistor and LED connected to the Q output! Remove the +5 V to the reset of FF_0! Move the start input to the preset of FF_0 to its reset input and connect the preset to +5 V+! What is the state of the outputs of the counter?
6. Open S_1! Draw the timing diagram for 10 clock cycles! What is the name and modulo number of this counter?

Ring Counter

1. Construct the ring counter of figure 4.9!
2. Apply the 5 Hz 5 V square wave to the *CLK* input!

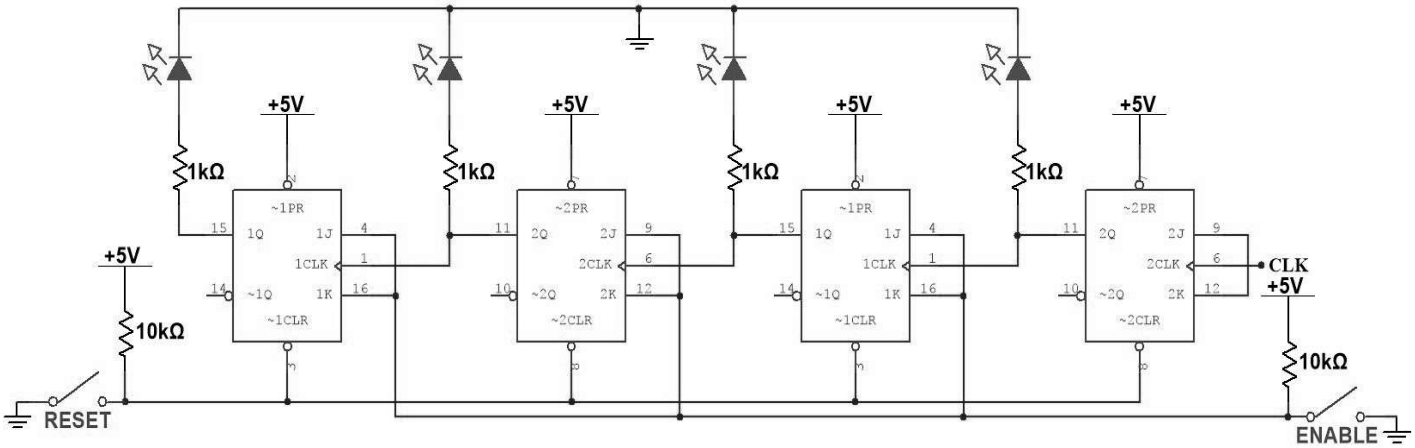


Figure 4.8 Experimental Binary Up Counter

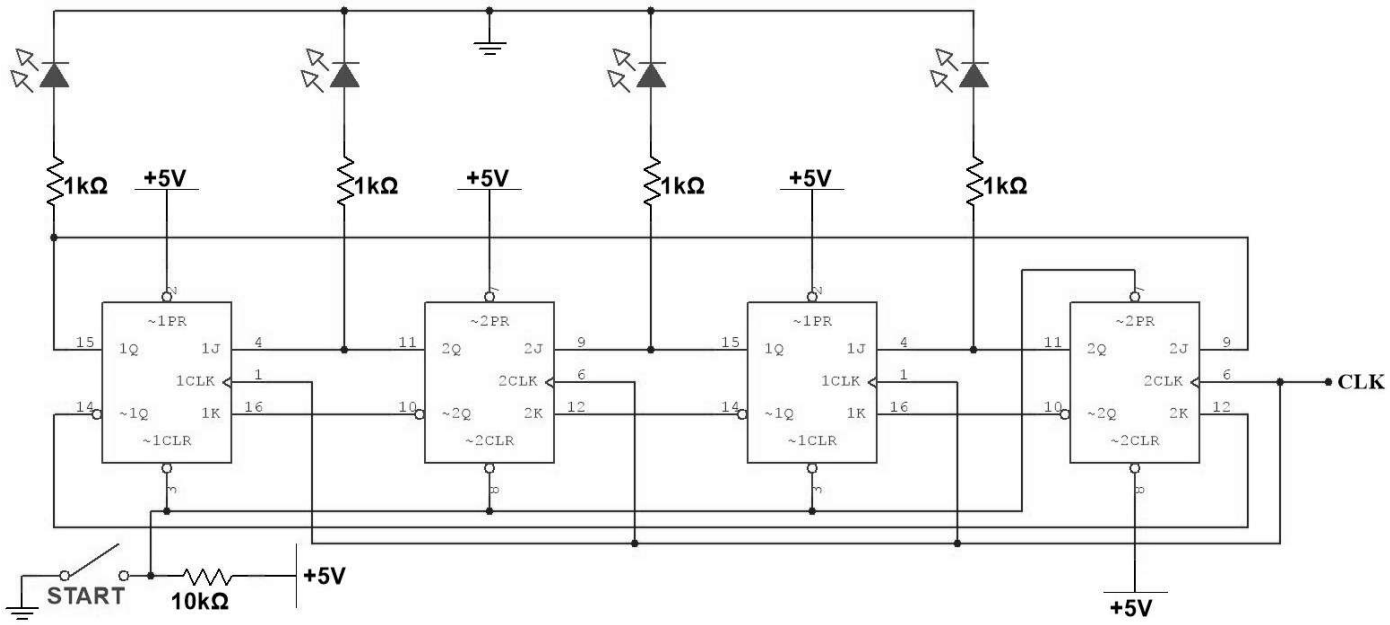


Figure 4.9. Experimental Ring Counter

QUESTIONS

1. Answer the questions listed in the procedure of every step!
2. Explain the output differences between a binary or ripple up counter and a ring counter!
3. How does a Johnson counter differ form a ring counter?
4. List an advantage for each type of counter used in this experiment!

MODULE 5

THE 555 TIMER

OBJECTIVES

1. To measure the frequency and duty cycle of an *astable* 555 timer.
2. To measure the pulse width out of a *monostable* 555 timer.
3. To examine the signal out of a voltage-controlled oscillator.
4. To build a *sawtooth* generator using a 555 timer.

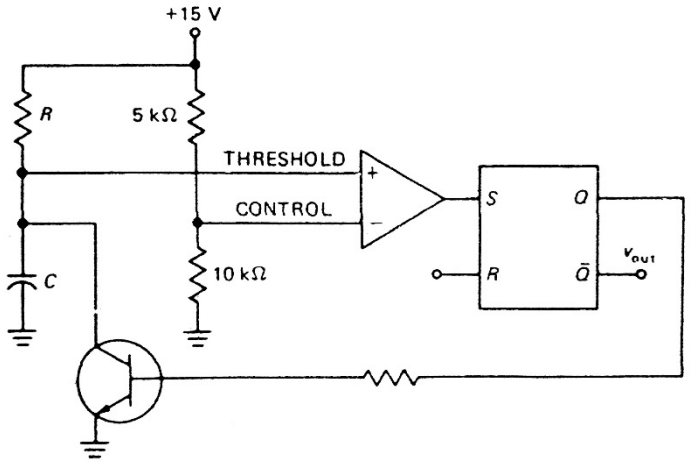
BASIC INFORMATION

Basic Timing Concept

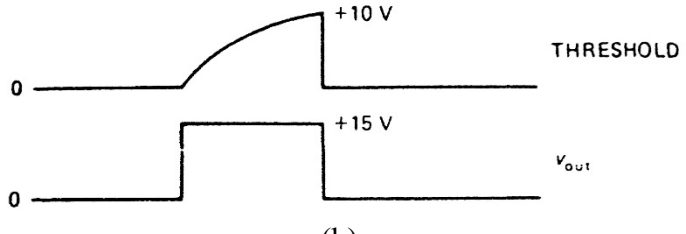
Figure 5.1 (a) illustrates some basic ideas needed in our later discussion of the 555 timer. Assume output *Q* is high. This saturates transistor and clamps the capacitor voltage at ground. In other words, the capacitor is short-circuited and cannot charge.

The non-inverting input voltage of the op amp is called the *threshold voltage,* and the inverting input voltage is referred to as the *control voltage.* With the RS flip-flop set, the saturated transistor holds the threshold voltage divider.

Suppose we apply a high voltage to the *R* input. This resets the RS flip-flop. Output *Q* goes to 0 and this cuts off the transistor. Capacitor *C* is now free to charge. As the capacitor charges, the threshold voltage increases.



(a)



(b)

Figure 5.1. Basic Timing Concept

Eventually, the threshold voltage becomes slightly greater than the control voltage (+10 V). The output of the op-amp then goes high, forcing the RS flip-flop to set. The high *Q* output saturates the transistor and this quickly discharges the capacitor.

Notice the two waveforms in figure 5.1(b). An exponential rise is across the capacitor, and a positive going pulse appears at the $\overline{Q}$ output.

555 Block Diagram

The NE555 timer introduced by Signetics is an 8-pin IC that can be connected to external components for either *astable* or *monostable* operation. Figure 5.2 shows a simplified block diagram. Notice the upper op-amp has a threshold input (pin 6) and a control input (pin 5). In most applications, the control input is not used, so that the control voltage equals $+2V_{CC}/3$ developed by the three 5 kΩ voltage divider. As before, whenever the threshold voltage exceeds the control voltage, the high output from the op-amp will set the flip-flop.

The collector of the *discharge* transistor goes to pin 7. When this pin is connected to and external timing capacitor, a high *Q* output from the flip-flop will saturate the transistor and discharge the capacitor. When *Q* is low, the transistor opens and the capacitor can charge as previously described.

The complementary signal out of the flip-flop goes to pin 3, the output. When the external reset (pin 4) is grounded, it *inhibits* the device (prevents it from working). This ON-OFF feature is useful sometimes. In most applications, however, the external reset is not used and pin 4 is tied directly to the supply voltage.

Notice the lower op-amp. Its inverting input is called the *trigger* (pin 2). Because of the voltage divider, the

non-inverting input has a fixed voltage of $+V_{CC}/3$. When the trigger input voltage is slightly less than $+V_{CC}/3$, the op-amp output goes high and resets the flip-flop.

Finally, pin 1 is the chip ground, while pin 8 is the supply pin. The 555 timer will work with any supply voltage between 4.5 and 16 V.

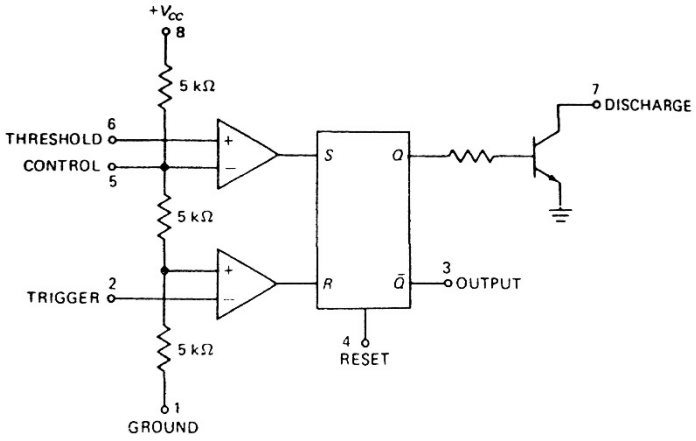


Figure 5.2. Block Diagram of 555 Timer

Monostable Operation

Figure 5.3(a) shows the 555 timer connected for *monostable* (one-shot) operation. It produces a single fixed pulse out each time a trigger pulse is applied to pin 2 (figure 5.3(b)). When the trigger input is slightly less than $+V_{CC}/3$, the lower op-amp has a high output and resets the flip-flop. This cuts off the transistor, allowing the capacitor to charge.

When the threshold voltage is slightly greater than $+2V_{CC}/3$, the upper op-amp has a high output, which sets the flip-flop. As soon as Q goes high, it turns on the transistor; this quickly discharges the capacitor.

The trigger input is a narrow pulse with a quiescent value of $+V_{CC}$. The pulse must drop below $+V_{CC}/3$ to reset the flip-flop sets; this saturates the transistor and discharges the capacitor. As a result, we get one rectangular output pulse.

The capacitor C has to charge through resistance R. The larger the RC time constant, the longer it takes for the capacitor voltage to reach $+2V_{CC}/3$. In other words, the RC time constant controls the width of the output pulse. Solving the exponential equation for capacitor voltage gives this formula for the pulse width

$$W = 1.1(RC)$$

For instance, if $R = 22$ kΩ and $C = 0.068$ μF, then the output of the *monostable* 555 timer is

$$W = 1.1\ x\ 22(10^3)\ x\ 0.068(10^{-6}) = 1.65\ ms$$

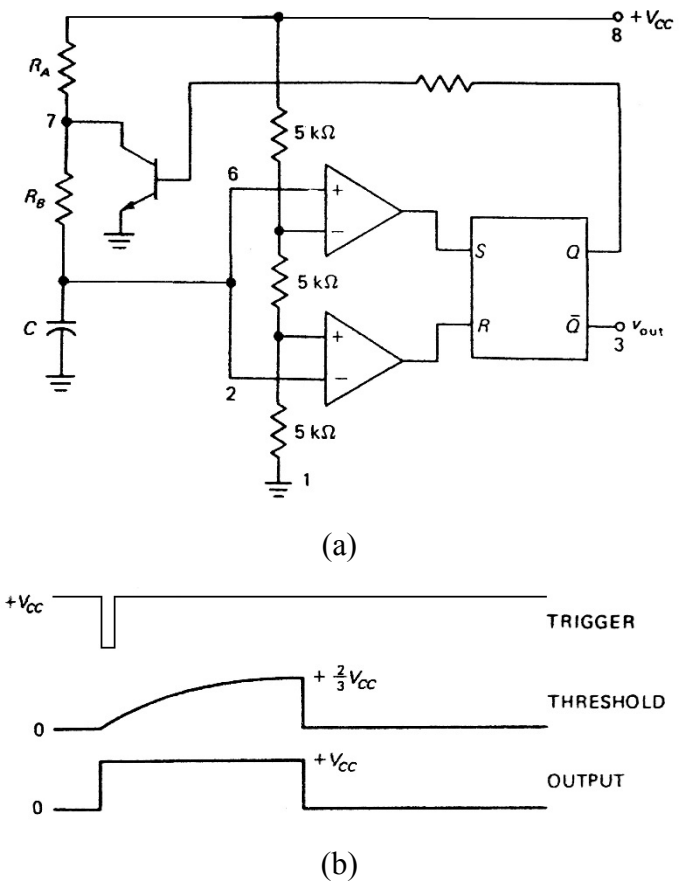


Figure 5.3. (a) *Monostable* Operation; **(b)** Waveforms

Normally, a schematic diagram does not show the op-amps, flip-flop, and other components inside the 555 timer. Rather, you will see a schematic diagram like figure 5.4 for the *monostable* 555 timer. Incidentally, notice that pin 5 (control) is bypassed to ground through a small capacitor, typically 0.01 μF. This provides noise filtering for the control voltage.

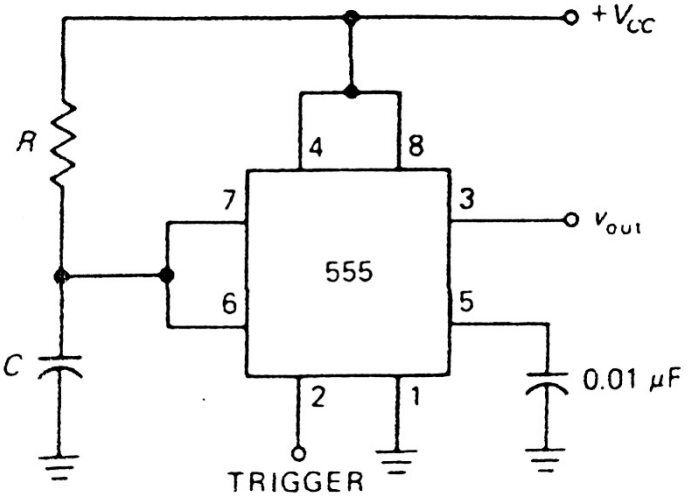


Figure 5.4. *Monostable* 555 Timer

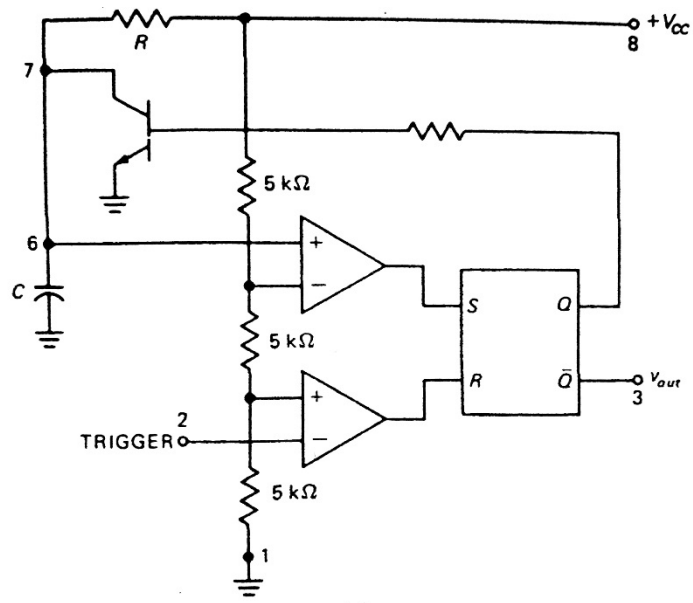
Astable Operation

Figure 5.5(a) shows the 555 timer connected for *astable* or free-running operation. The output is a

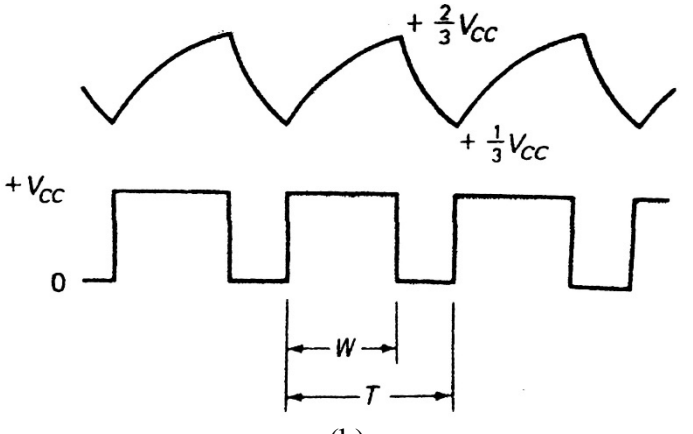
square-wave signal. When Q is low, the transistor is cut off and the capacitor is charging through a total resistance of $R_A + R_B$. Because of this, the charging time constant is $(R_A+R_B)C$. As the capacitor charges, the threshold voltage increases.

Eventually, the threshold voltage exceeds $+2V_{CC}/3$; then the upper op-amp has a high output and this sets the flip-flop. With Q high, the transistor saturates and grounds pin 7. Now the capacitor discharges through R_B. Therefore, the discharging time constant in R_BC. When the capacitor voltage drops slightly below $+V_{CC}/3$, the lower op-amp has a high output and this resets the flip-flop.

Figure 5.5(b) illustrates the waveforms. As you see, the timing capacitor has an exponentially rising and falling voltage. The output is a rectangular wave. Since the charging time constant is longer that the discharging time constant, the output is not symmetrical; the high state lasts longer than the low state.



(a)



(b)

Figure 5.5. (a) *Astable* Operation; **(b)** Waveforms

To specify how asymmetric the output is, we will use the duty cycle defined as

$$D = \frac{W}{T}\ x\ 100\%$$

Depending on resistances R_A and R_B, the duty cycle is between 50 and 100 percent.

A mathematical solution to the charging and discharging equations gives the following formulas. The output frequency is

$$f = \frac{1.44}{(R_A + 2R_B)}\ x\ 100\%$$

And the duty cycle is

$$D = \frac{(R_A + R_B)}{(R_A + 2R_B)}\ x\ 100\%$$

If R_A is much smaller than R_B, the duty cycle approaches 50 percent.

Figure 5.6 shows the *astable* 555 timer as it usually appears. Again notice how pin 4 (reset) is tied to the supply voltage and how pin 5 (control) is bypassed to ground through a 0.01 µF capacitor.

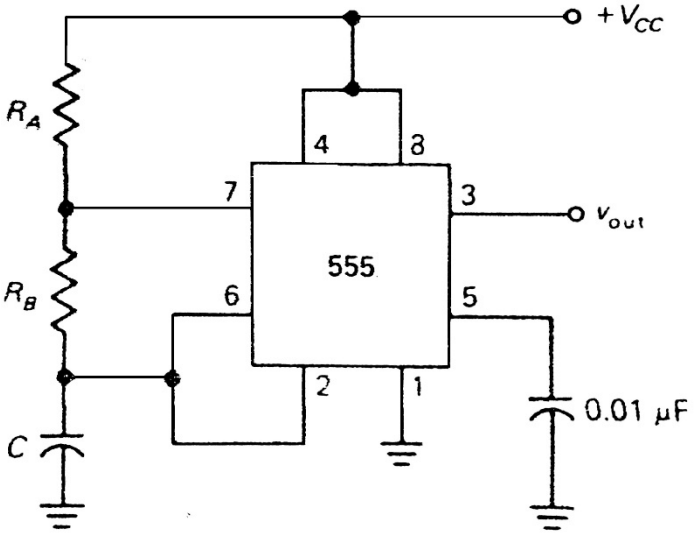


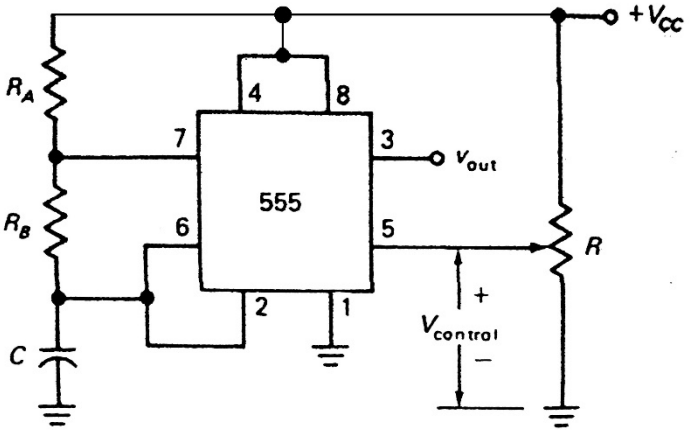
Figure 5.6. *Astable* 555 Timer

Voltage-Controlled Oscillator

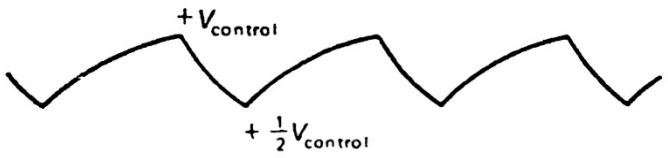
Figure 5.7(a) shows a *voltage-controlled oscillator* (VCO). Recall that pin 5 (control) connects to the inverting input of the upper op-amp. Normally, the control voltage is $+2V_{CC}/3$ because of the internal voltage divider. In figure 5.7(a), however, the voltage from an external potentiometer overrides the internal voltage. In other words, by adjusting the potentiometer, we can change the control voltage.

Figure 5.7(b) illustrates the voltage across the timing capacitor. Note that it varies between $+V_{control}/2$ and $+V_{control}$. If we increase $V_{control,}$ it takes the capacitor longer to charge and discharge therefore, the frequency decreases. As a result, we can change the frequency of the circuit by varying the control voltage.

Incidentally, the control voltage may come from a potentiometer or it may be the output of another transition circuit, op-amp, and so on.



(a)



(b)

Figure 5.7. (a) Voltage-controlled Oscillator; **(b)** Waveform

Sawtooth Generator

A constant charging current produces linear ramp of voltage across a capacitor. The PNP transistor of figure 5.8(a) produces a constant charging current equal to

$$I_C = \frac{V_{CC} \quad V_E}{R}$$

where

$$V_E = V_{BE} + \frac{R_2}{R_1 + R_2} V_{CC}$$

For instance, if V_{CC} = 15 V, R = 20 kΩ, R_1 = 5 kΩ, R_2 = 10 kΩ, and V_{BE} = 0.7 V, then

$$V_E = 0.7\,V + 10\,V = 10.7\,V$$

and

$$I_C = \frac{15\,V - 10.7\,V}{20\,\text{k}\Omega} = 0.215\,mA$$

When a trigger starts the *monostable* 555 timer of figure 5.8(a), the PNP current source forces a constant charging current into the capacitor. Therefore, the voltage across the capacitor is a linear ramp as shown in figure 5.8(b). The *slope S* of the linear ramp is defined as the rise over the run, or

$$S = \frac{V}{T}$$

where V is the peak voltage at time. For instance, if V = 10 V and T = 2 ms, then the slope S is 5 V/ms. This says the ramp rises 5 V/ms.

Since the basic capacitor equation is

$$V = \frac{Q}{C}$$

we can divide both sides by T to get

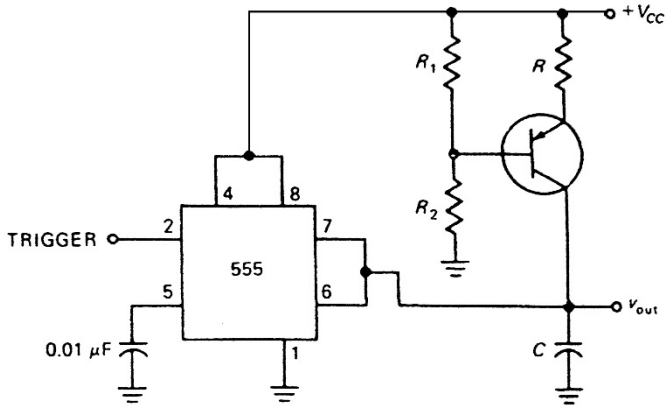
$$\frac{V}{T} = \frac{Q/T}{C}$$

When the charging current is constant, this reduces to

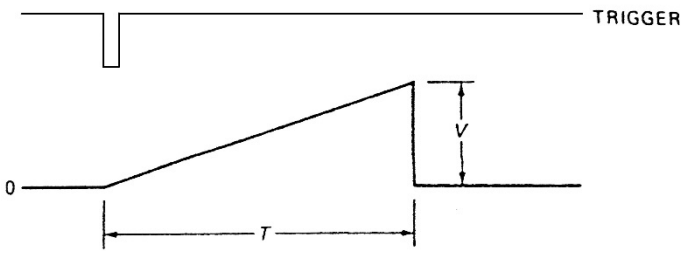
$$S = \frac{I}{C}$$

In other words, you can predict the slope of a linear ramp using the ratio of charging current to capacitance. If the charging current is 0.215 mA (found earlier) and the capacitance is 0.022 µF, the ramp will have a slope of

$$S = \frac{0.215mA}{0.022\mu F} = 9.77V/ms$$



(a)



(b)

Figure 5.8. (a) *Sawtooth* Generator; **(b)** Waveform

SUMMARY

1. A high set (S) input sets the output of an RS flip-flop to the high state. A high Reset (S) input resets the output to the low state.
2. In a 555 timer the non-inverting input of the upper op-amp is called the *threshold voltage*; the inverting input is called the *control voltage.*
3. When the threshold voltage exceeds the control voltage, the RS flip-flop is set. This saturates the discharge transistor.
4. The inverting input of the lower op-amp in a 555 is called the *trigger*.
5. When trigger voltage is less than $+V_{CC}/3$, the RS flip-flop is reset. This cuts off the *discharge* transistor.
6. The 555 timer can be connected for *astable* and *monostable* operation.
7. Normally, the control voltage of a 555 timer equals $+2V_{CC}/3$ because of the internal voltage divider. In VCO application, however, an external voltage is applied to the control pin override the voltage from the internal voltage divider.
8. By using a PNP current source, the 555 timer can produce linear amps.

SELF-TEST

1. To saturate the transistor of figure 5.2(a), the *Q* output must be _____ V.
2. In figure 3.2(a), the control voltage equals _____ V.
3. To set the RS flip-flop of figure 5.3, the threshold voltage must be slightly greater than the _____ voltage.
4. In figure 5.5, R = 68 kΩ and C = 0.050 μF. The pulse width of the output is _____ ms.
5. In figure 5.7, if R_A = 27 kΩ, R_B = 70 kΩ, and C = 0.22μF. The frequency of the output is _____ Hz and the duty cycle is _____ percent.

----------PROCEDURE----------

MATERIALS REQUIRED

- Regulated AC/DC Power Supply ~ +15 V
- Oscilloscope, signal Generator
- Resistors, Capacitors, Potentiometer, Transistors
- ICs: Op-Amp 741, NE555 Timer

Astable 555 timer

1. Calculate the frequency and duty cycle in figure 5.9 for the resistances listed in table 5.1. Record the results under $f_{calculated}$ and $D_{calculated}$!
2. Connect the circuit of figure 5.9 with R_A = 10 kΩ and R_B = 100 kΩ!
3. Measure *W* and *T*. Work out the frequency and duty factor. Record under $f_{measured}$ and $D_{measured}$ in table 5.1!
4. Look at the voltage across the capacitor (pin 6). You should see an exponentially rising and falling wave between 5 V and 10 V!
5. Repeat steps 2 through 4 for the other resistances of table 5.1!

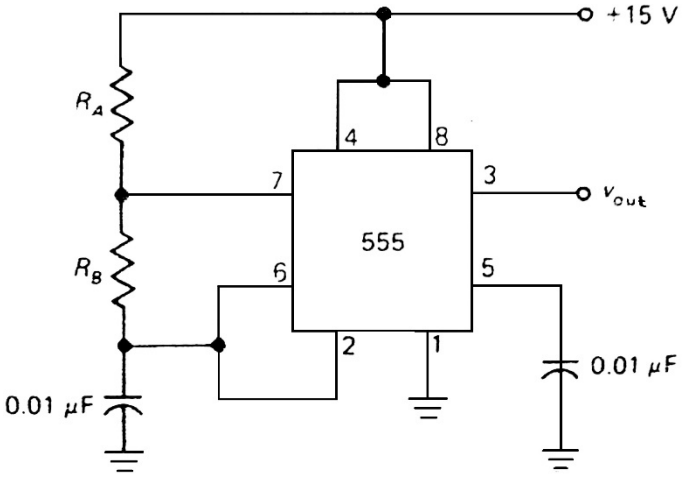


Figure 5.9. Experimental Astable 555 Timer

Table 5.1. *Astable* Operation

R_A (kΩ)	R_B (kΩ)	f_{calc}	D_{calc}	f_{meas}	D_{meas}
47	100				
100	47				
47	47				

Voltage Controlled Oscillator

1. Connect the circuit of VCO in figure 5.10!
2. Look at the output with an oscilloscope!
3. Vary the 1 kΩ potentiometer and notice what happens. Record the maximum and minimum frequencies of the output!

$f_{maximum}$ = _____

$f_{minimum}$ = _____

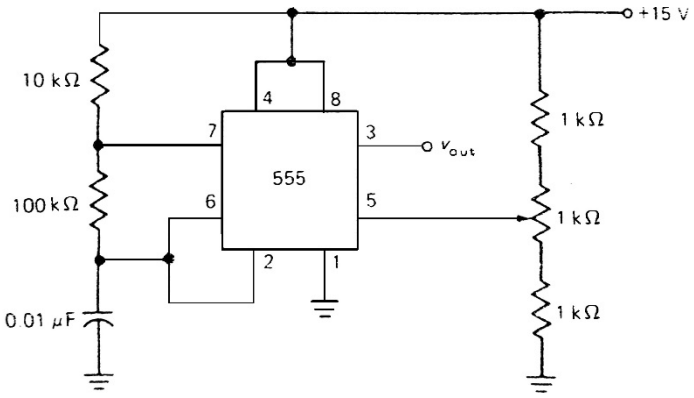


Figure 5.10. Experimental Voltage-controlled Oscillator

Monostable 555 Timer

1. Figure 5.11 shows a Schmitt trigger driving monostable 555 timer. Calculate the pulse width for each *R* listed in table 5.2! Record the results under $W_{calculated}$!
2. Connect the circuit of figure 5.11 with an R of 33 kΩ!
3. Look at the output of the Schmitt trigger (pin 6 of 741). Set the frequency of the sine-wave input to 1 KHz.!
4. Adjust the sine-wave level until you get a Schmitt trigger output with a duty cycle of approximately 90%!
5. Look at the output of the 555 timer and measure the pulse width! Record this value under $W_{measured}$ in table 5.2!
6. Repeat steps 2-5 for the remaining values of table 5.2!

Table 5.2. *Monostable* Operation

R (kΩ)	W_{calc}	W_{meas}
22		
33		
47		

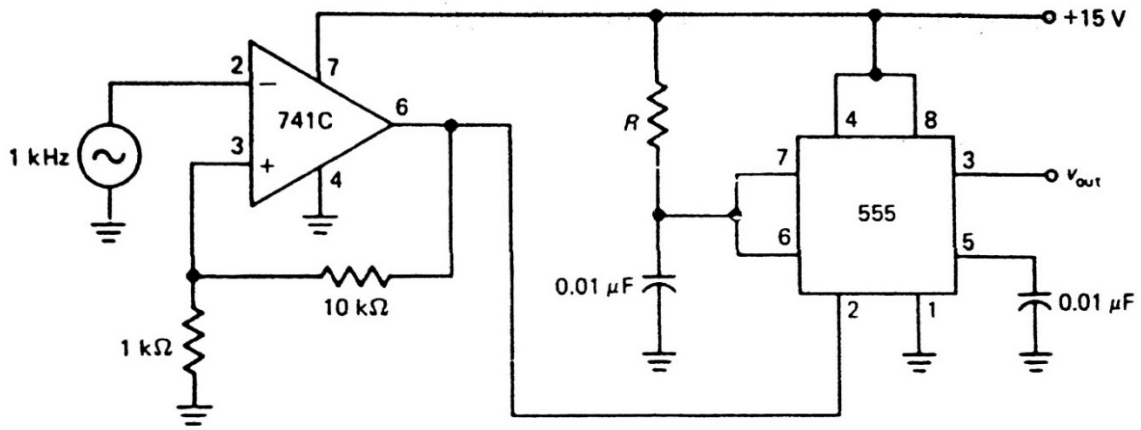


Figure 5.11. Experimental *Monostable* Operation

Sawtooth Generator

1. Calculate the charging current in figure 5.12 for each value of *R* shown in table 5.3. Record the values!
2. Calculate the slope of capacitor voltage in volts/milisecond. Record under $S_{calculaied}$ in table 5.3!
3. Connect the circuit of figure 5.12 with an *R* of 10 kΩ!
4. Set the signal generator to 1 KHz of AC frequency! Adjust the level to get a duty cycle of approximately 90 percent out of the Schmitt trigger!
5. Look at the output voltage! It should be a *sawtooth*. Measure the ramp voltage and time! Then, work out the slope in voltages/milisecond. Record the value under $S_{measured}$ in table 5.3!
6. Repeat steps 3 through 5 for the remaining values of *R* in table 5.3!

Table 5.3. *Sawtooth* Generator

R (kΩ)	I_{charge} (mA)	S_{calc} (V/ms)	S_{meas} (V/ms)
10			
33			
47			

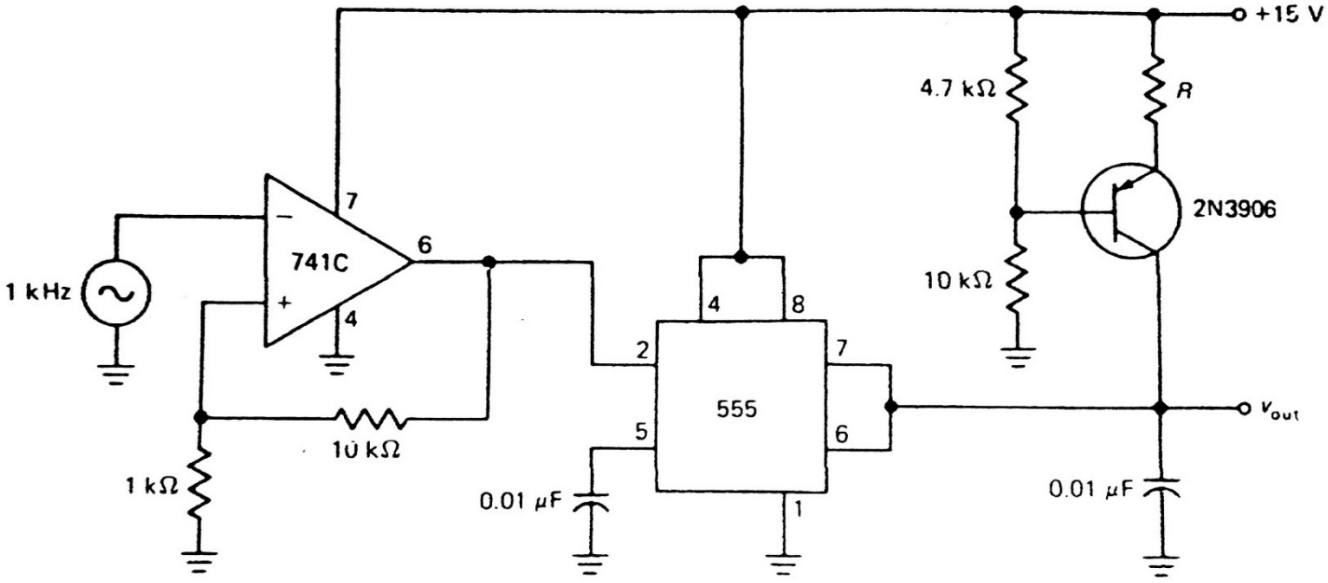


Figure 5.12. Experimental *Sawtooth* Generator

QUESTIONS

1. How does the ratio of R_A and R_B affect the duty cycle of an *astable* 555 timer?
2. What effect does *R* have on the *sawtooth*?
3. What happens to the width of the output if the timing resistor is increased?
4. What effect does increasing the timing capacitor have on the frequency out of an *astable* 555 timer?

MODULE 6
HALF ADDER, FULL ADDER, AND DECODER USING VHDL

OBJECTIVES

1. To create a new project in Vivado™ using VHDL
2. To use the provided ZYBO Master Constraint file to constrain the pin locations
3. To design and construct Half Adder, Full Adder, and Encoder logic using VHDL
4. To simulate, synthesize, and implement the design
5. To program the completed design onto the ZYBO Development Board FPGA

BASIC INFORMATIONS

Binary Number

The binary system of arithmetic uses only two symbols (0 and 1) to represent all quantities. This system finds wide use in computers because the 0 and 1 are easily represented by the 2-state digital circuits.

Counting is started in the binary system in the same way as in the decimal system with 0 for zero and 1 for one. But at 2 in the binary system there are no more symbols. Therefore, the same move must be taken at two in the binary system that is taken at 10 in the decimal system: It is necessary to place a 1 in the position to the let and start again with a 0 in the original position. Table 6.1 is a list of numbers shown in both decimal and binary form.

The order of binary number is not designated unit, tens, hundreds, thousands, and so forth, as in the decimal system. Instead, the order is 1, 2, 4, 8, 16, 32, and so on, reading from right to left with the position farthest to the right being 1. Table 6.2 shows more decimal quantities and their equivalents in binary form. Note how the positions are numbered right to left.

Table 6.1. Decimal and Binary Numbers

Decimal	*Binary*	*Decimal*	*Binary*
0	0	6	110
1	1	7	111
2	10	8	1000
3	11	9	1001
4	100	10	1010
5	101	11	1011

These values are found by raising the base radix (2) by an exponential value equivalent to its position in the number. The smallest binary digit called the *least significant bit* (LSB) is binary digit position 0. It has a numerical weight of $2^0 = 1$. The weight of the next digit is $2^1 = 2$, then $2^2 = 4$, and so forth. Notice that each position weight is twice that of the proceeding digit.

Converting binary values to decimal is achieved by multiplying the position weight of each digit by the value (1 or 0) in the position. These products are added to produce the final decimal equivalent of the original binary number. For example, let us convert 110101 to its decimal value. There are six binary digits with the LSB in rightmost place. The weights of these digits (bits) are LSB = 1 and then 2, 4, 8, and 16 and finally, 32. Thus, $110101_2 = 53_{10}$. The subscript denotes the base value of the number system used for each number (2 for binary and 10 for decimal).

Table 6.2. Decimal Numbers and Their Binary Equivalents

	Binary								
Decimal	256	128	64	32	16	8	4	2	1
34				1	0	0	0	1	0
15						1	1	1	1
225		1	1	1	0	0	0	0	1
75			1	0	0	1	0	1	1

The method used to convert a decimal number to its binary equivalent may be called *divide and remainder*. Divide the original decimal value by 2, the binary bas value. The result is a quotient and a remainder. The remainder becomes the binary number starting with the LSB. Divide the quotient again by 2. The remainder is the next binary bit. The quotient result is again divided by the base value with the remainder becoming the third binary digit. This is repeated until the quotient becomes 0.

Addition of binary quantities is very simple and is based on the following three rules:

4. 0 + 0 = 0
5. 0 + 1 = 1
6. 1 + 1 = 0 with a 1 carry to the left

Table 6.3 is and example of binary addition using the rules stated.

The factors to be added are 75 and 225. Starting at the right, we have 1 + 1 = 0 with a 1 carry (rule 3). The next position to the left is added: 0 + 1 = 1. 0 with 1 carried to the third position. The third position consists of 0 + 0 = 0 + 1(carry) = 1. This procedure given in binary form as 100101100, which is equal to 256 + 32 + 8 + 4 = 300. This sum is exactly what we would expect to get by adding the decimal quantities 225 and 75.

Binary quantities can also be subtracted, multiplied, and divided, using rules similar to those for addition.

Table 6.3. Adding Binary Numbers

	Binary Value								
Carry:	1	1					1	1	
225 =	0	1	1	1	0	0	0	0	1
+75 =	+0	0	1	0	0	1	0	1	1
300 =	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**

Exclusive-OR Gate

Figure 6.1(a) is a schematic diagram for a special circuit called an *exclusive-OR*. The Boolean expression for this circuit is $Y = A\overline{B} + \overline{A}B$. Table 6.4 is the truth table for this circuit. Output Y will be high if A is low and B is high or the reverse is true. Output Y is low whenever the two inputs are both low or both high.
Examine table 6.4 carefully and note that the output is in one state when the inputs agree and in another state when they disagree. This respect allows the exclusive OR to be used for comparing binary bit values.

Table 6.4. Exclusive-OR

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

The Boolean operator for an exclusive-OR is an OR operation (+) enclosed in a circle: $\oplus$. As such, another Boolean expression for the circuit in figure 6.1 is

$$Y = A \oplus B$$

You will find many uses for this circuit which packaged in its own IC (7486). The schematic symbol for this gate is shown in figure 6.1(b).

Binary Half Adder and Truth Table

The simplest binary adder is called a *half adder* and is capable of combining two binary numbers and providing an output and a carry when necessary. The first step in understanding the operation of a half adder is to investigate the input combinations and the resulting outputs based on the rules of binary addition. Table 2.5 is a truth table showing these combinations. The table shows that a binary 1 on one input with a 0 in the other (rule 2) results in a binary 1 sum and binary 0 carry. A binary 1 on both inputs results in a binary 0 sum and a binary 1 carry (rule 3). A binary 0 on both inputs results in a binary 0 sum and binary 0 carry (rule 1).

Table 6.5. Truth Table for Half Adder

Input		**Sum**	**Carry**
A	**B**		
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Consider the sum and carry as two separate truth table results generated by the inputs A and B. Note that the sum has generated an exclusive-OR table and the carry, an AND result. Figure 6.2 is the schematic of the circuit that produces this half adder truth table.

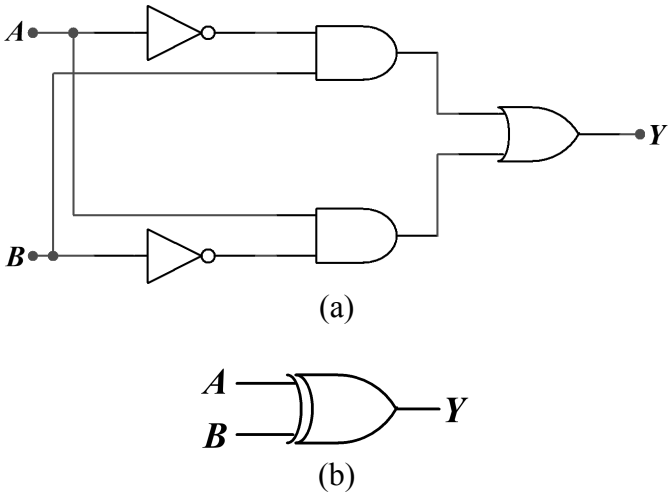


Figure 6.1. (a) Exclusive-OR; **(b)** Logic Schematic Symbol

The half adder has only limited use because there are no provisions for a carry input from a previous adder.

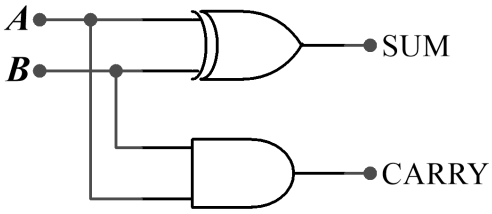


Figure 6.2. Half Adder

Binary Full Adder and Truth Table

When a carry and the two quantities to be added are considered as inputs, the input combinations increase to eight as shown in table 6.6. An adder capable of producing the required outputs for the eight input combinations is called a *full adder*. The full adder is shown in the block diagram of figure 6.3.
The full adder shown represents a single position in a binary-adder system. Because many such adders are combined in a large computer, each full adder is represented as a block in the computer logic diagram. The actual number of positions in such an adder depends on the size of the computer and the type of calculations the computer is designed for.

Table 6.6. Truth Table for Full Adder

Inputs			Outputs	
A	**B**	**C**	**Sum**	**Carry**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

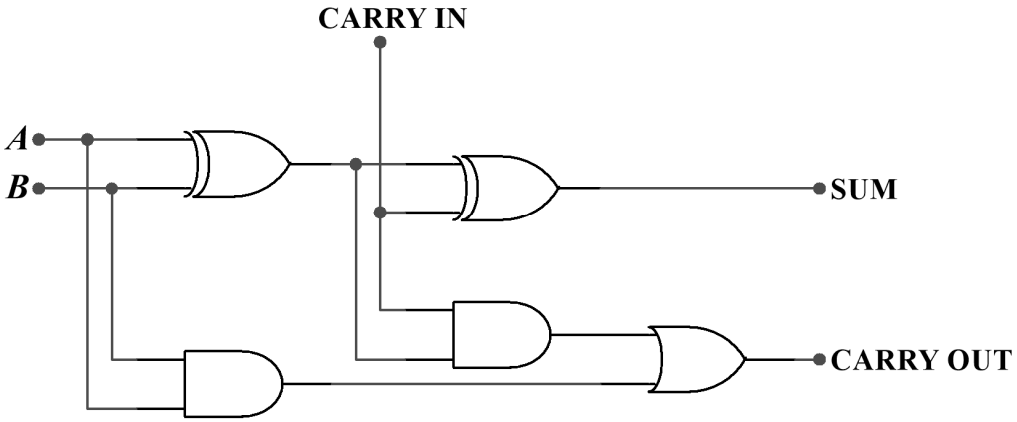


Figure 6.3. Full Adder

Binary Decoder and Truth Table

Decoder circuits are used to decode encoded information. A binary decoder, depicted in figure 6.4, is a logic circuit with n inputs and 2^n outputs. Only one output is asserted at a time, and each output corresponds to one valuation of the inputs. The decoder also has an enable input, E_n, that is used to disable the outputs; if $E_n = 0$, then none of the decoder outputs is asserted. If $E_n = 1$, the valuation of $w_{n-1} \cdots w_1 w_0$ determines which of the outputs is asserted.

An n-bit binary code in which exactly one of the bits is set to 1 at a time is referred to as *one-hot encoded*, meaning that the single bit that is set to 1 is deemed to be "hot." The outputs of a binary decoder are one-hot encoded.

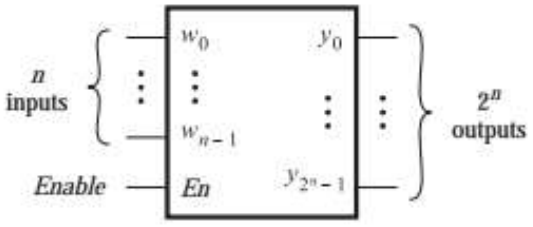
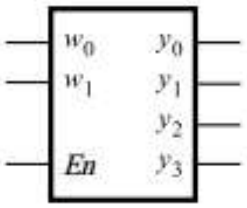
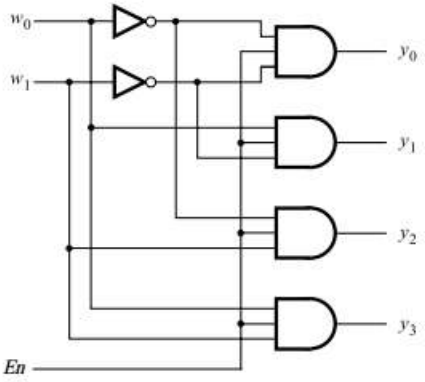


Figure 6.4. A n-to-2^n binary decoder

A 2-to-4 decoder is given in figure 6.5. The two data inputs are w_1 and w_0. They represent a two-bit number that causes the decoder to assert one of the outputs y_0, . . . , y_3.



(a)



(b)

Figure 6.5. (a) Graphical Symbol; **(b)** Logic Circuit

Although a decoder can be designed to have either active-high or active-low outputs, in figure 6.5 active-

high outputs are assumed. Setting the inputs w_1w_0 to 00, 01, 10, or 11 causes the output y_0, y_1, y_2, or y_3 to be set to 1, respectively. A graphical symbol for the decoder is given in part (a) of the figure, and a logic circuit is shown in part (b).

Table 6.7. Truth Table for 2-to-4 decoder

En	w_1	w_2	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

----------PROCEDURE----------

MATERIALS REQUIRED

- ZYBO Zynq7000
- Micro-USB Power USB Cable
- Vivado™ Software
- PC/Laptop 64bit

Half Adder

1. Create a new project, give description about the module's definition, entity, and port that will be used
2. Draw and design a half adder logic using VHDL. **Set the I/O definitions and constraints: input in the switches sw0 and sw1 to a and b, and the LEDs led0 and led1 to sum and carry**
3. Elaborate and simulate the design. Verify the schematic with the initial design's logic
4. Simulate the design. Check the behavioural simulation pattern
5. Synthesize, implement, and program the device. Record the state of the output for each input possibility

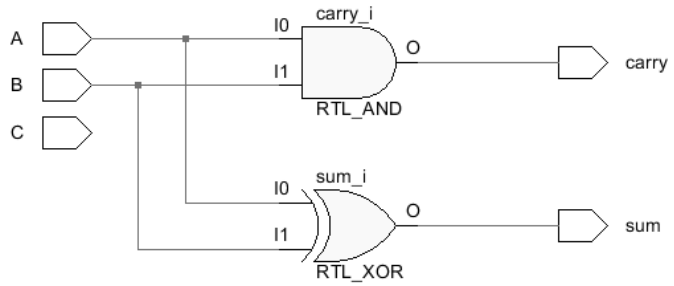


Figure 6.6. Half Adder Experiment

Table 6.8. Half Adder Value Table

Inputs		*Outputs*	
A	**B**	**Sum**	**Carry**
0	0		
0	1		
1	0		
1	1		

Full Adder

1. Create a new project, give description about the module's definition, entity, and port that will be used
2. Draw and design a full logic using VHDL. **Set the I/O definitions and constraints: input in the switches sw0, sw1, and sw2 to a, b, and c, and the LEDs led0 and led1 to sum and carry**
3. Elaborate and simulate the design. Verify the schematic with the initial design's logic
4. Simulate the design. Check the behavioural simulation pattern
5. Synthesize, implement, and program the device. Record the state of the output for each input possibility

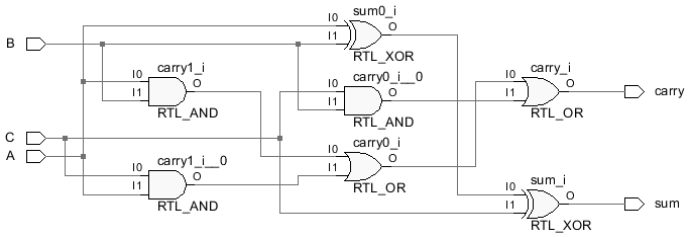


Figure 6.7. Full Adder Experiment

Table 6.9. Full Adder Value Table

Inputs			*Outputs*	
A	**B**	**C**	**Sum**	**Carry**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Decoder

1. Create a new project, give description about the module's definition, entity, and port that will be used
2. Draw and design an encoder logic using VHDL. **Set the I/O definitions and constraints: input in the switches sw0 and sw1 to a and b, and the**

LEDs led0, led1, led2, and led3 to z0, z1, z2, and z3

3. Elaborate and simulate the design. Verify the schematic with the initial design's logic
4. Simulate the design. Check the behavioural simulation pattern
5. Synthesize, implement, and program the device. Record the state of the output for each input possibility

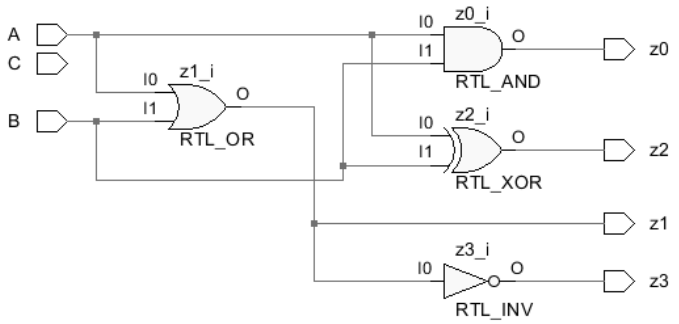


Figure 6.8. Decoder Experiment

Table 6.10. Decoder Value Table

Inputs		*Outputs*			
A	**B**	**Z0**	**Z1**	**Z2**	**Z3**
0	0				
0	1				
1	0				
1	1				

QUESTIONS

1. Use algebraic manipulation to prove that $x \oplus (x \oplus y) = y$!
2. Design a circuit that can add three unsigned four-bit numbers. Use four-bit adders and any other gates needed!

MODULE 7
BCD SEVEN SEGMENT USING VHDL

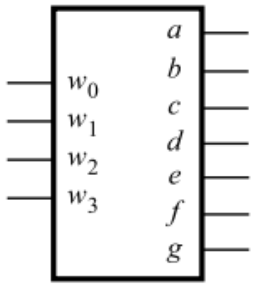
OBJECTIVES

1. To design and construct a BCD Seven Segment logic using VHDL
2. To simulate, synthesize, and implement the design
3. To program the completed design onto the ZYBO Development Board FPGA

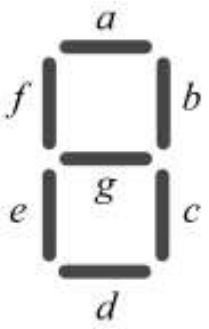
BASIC INFORMATIONS

Code Converters

The purpose of the decoder and encoder circuits is to convert from one type of input encoding to a different output encoding. For example, a 3-to-8 binary decoder converts from a binary number on the input to a one-hot encoding at the output. An 8-to-3 binary encoder performs the opposite conversion.



(a)



(b)

Figure 7.1. (a) Code Converter; **(b)** 7-segment Display

There are many other possible types of code converters. One common example is a BCD-to-7-segment decoder, which converts one binary-coded decimal (BCD) digit into information suitable for driving a digit-oriented display. As illustrated in figure 7.1 (a), the circuit converts the BCD digit into seven signals that are used to drive the segments in the display. Each segment is a small light-emitting diode (LED), which glows when driven by an electrical signal. The segments are labelled from *a* to *g* in the figure.

For each valuation of the inputs $w_3, \ldots, w_0$, the seven outputs are set to display the appropriate BCD digit. Note that the last 6 rows of a complete 16-row truth table are not shown. They represent don't-care conditions because they are not legal BCD codes and will never occur in a circuit that deals with BCD data. A circuit that implements the truth table can be derived using synthesis techniques. Finally, we should note that although the word *decoder* is traditionally used for this circuit, a more appropriate term is *code converter*. The term *decoder* is more appropriate for circuits that produce one-hot encoded outputs.

Table 7.1. Truth Table for BCD Seven Segment

w_3	w_2	w_1	w_0	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

----------PROCEDURE----------

MATERIALS REQUIRED

- ZYBO Zynq7000
- Micro-USB Power USB Cable
- Vivado™ Software
- PC/Laptop 64bit
- Seven Segment
- Resistors

BCD Seven Segment

1. Create a new project, give description about the module's definition, entity, and port that will be used
2. Draw and design a BCD Seven Segment logic using VHDL. **Set the I/O definitions and constraints: input in the switches sw0, sw1, sw2, and sw3 to Ain, Bin, Cin, and Din, and the PMODs that will be used to Aout, Bout, Cout, Dout, Eout, Fout, and Gout**
3. Elaborate and simulate the design. Verify the schematic with the initial design's logic
4. Simulate the design. Check the behavioural simulation pattern
5. **Connect the PMODs to the Seven Segment with the resistors, depending on the Seven Segment that will be used (common cathode or common anode)**
6. Synthesize, implement, and program the device. Record the state of the output for each input possibility

Table 7.2. BCD Seven Segment Value Table

Inputs				*Outputs*							
Ain	**Bin**	**Cin**	**Din**	**Aout**	**Bout**	**Cout**	**Dout**	**Eout**	**Fout**	**Gout**	**Value**
0	0	0	0								
0	0	0	1								
0	0	1	0								
0	0	1	1								
0	1	0	0								
0	1	0	1								
0	1	1	0								
0	1	1	1								
1	0	0	0								
1	0	0	1								

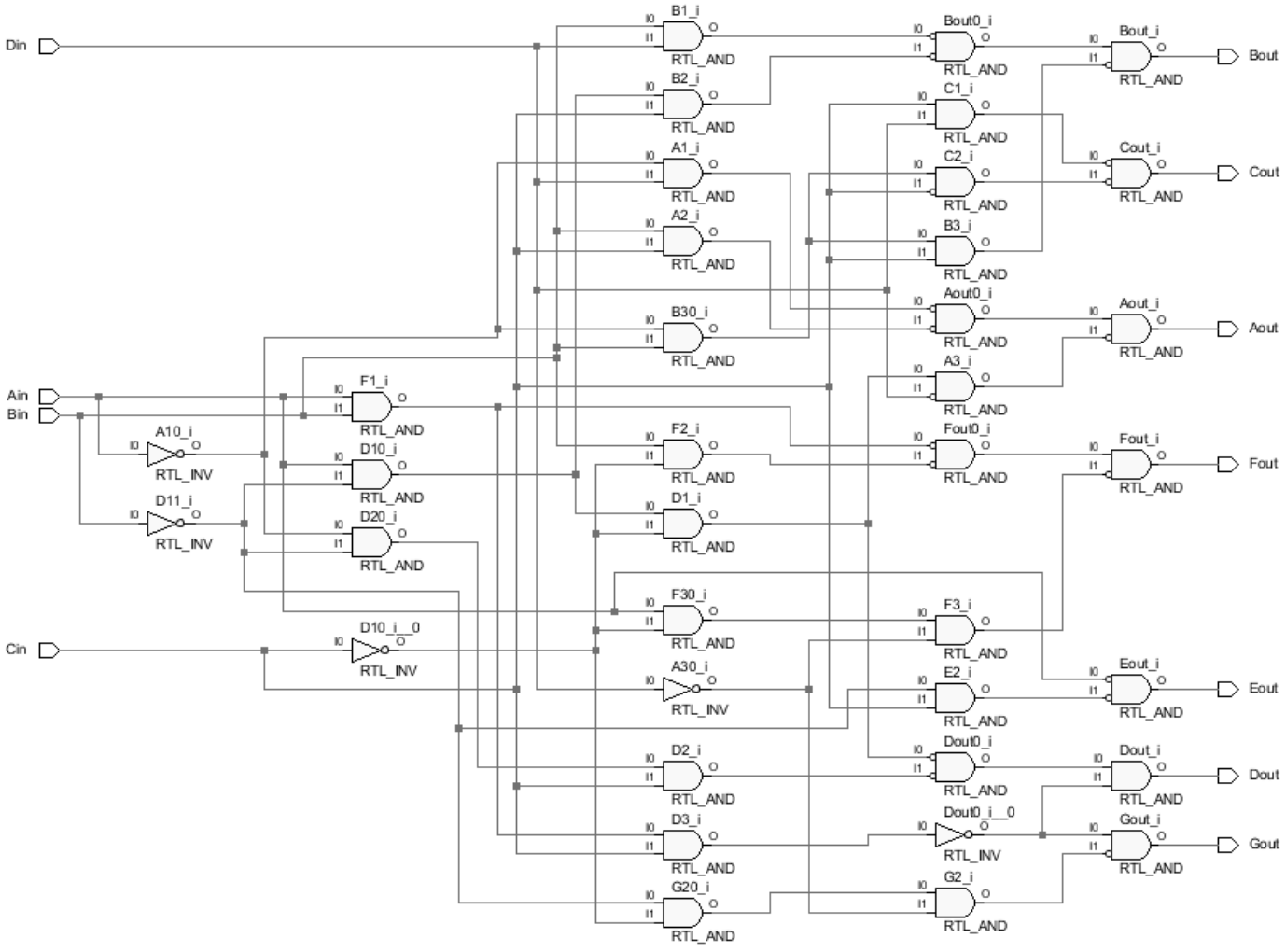


Figure 7.2. BCD Seven Segment Experiment

QUESTIONS

1. Write VHDL code for a BCD-to-7-segment code converter, using a selected signal assignment in table 7.1!
2. Derive minimal sum-of-products expressions for the outputs a, b, and c of the 7-segment display in figure 7.1!
3. Derive minimal sum-of-products expressions for the outputs d, e, f, and g of the 7-segment display in figure 7.1!

MODULE 8
SEQUENTIAL BCD COUNTER USING VHDL

OBJECTIVES

1. To design and construct a Sequential BCD Counter logic using VHDL
2. To simulate, synthesize, and implement the design
3. To program the completed design onto the ZYBO Development Board FPGA

BASIC INFORMATION

BCD Counter

Binary-coded-decimal (BCD) counters can be designed using the approach reset synchronization. A two-digit BCD counter is presented in figure 8.1. It consists of two modulo-10 counters, one for each BCD digit. Note that in a modulo-10 counter it is necessary to reset the four flip-flops after the count of 9 has been obtained. Thus the *Load* input to each stage is equal to 1 when $Q_3 = Q_0 = 1$, which causes 0s to be loaded into the flip-flops at the next positive edge of the clock signal. Whenever the count in stage 0, BCD_0, reaches 9 it is necessary to enable the second stage so that it will be incremented when the next clock pulse arrives. This is accomplished by keeping the *Enable* signal for BCD_1 low at all times except when $BCD_0 = 9$.

In practice, it has to be possible to clear the contents of the counter by activating some control signal. Two OR gates are included in the circuit for this purpose. The control input *Clear* can be used to load 0s into the counter. Observe that in this case *Clear* is active when high.

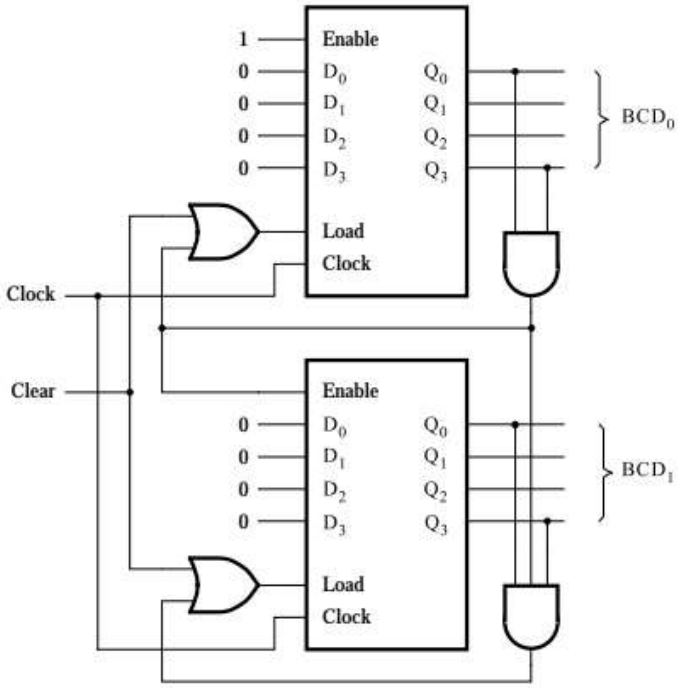


Figure 8.1. A two-digit BCD Counter

In any digital system there is usually one or more clock signals used to drive all synchronous circuitry. In the preceding counter, as well as in all counters presented in the previous figures, we have assumed that the objective is to count the number of clock pulses. Of course, these counters can be used to count the number of pulses in any signal that may be used in place of the clock signal.

----------PROCEDURE----------

MATERIALS REQUIRED

- ZYBO Zynq7000
- Micro-USB Power USB Cable
- Vivado™ Software
- PC/Laptop 64bit
- Seven Segment
- Resistors

Sequential BCD Counter

1. Create a new project, give description about the module's definition, entity, and port that will be used
2. Draw and design a Sequential BCD Counter logic using VHDL. **Set the I/O definitions and constraints: input in the switch sw0 to enable, button button0, button1, and button2, to rst, up, and down, and the LEDs led0, led1, led2, and led3 to q0, q1, q2, and q3**
3. Elaborate and simulate the design. Verify the schematic with the initial design's logic
4. Simulate the design. Check the behavioural simulation pattern
5. Synthesize, implement, and program the device. Record the state of the output for each input possibility
6. **Change the output from LEDs to Seven Segment and reprogram the VHDL, I/O**

definitions, and constraints accordingly. Rerun the procedure from step 3 to 5.

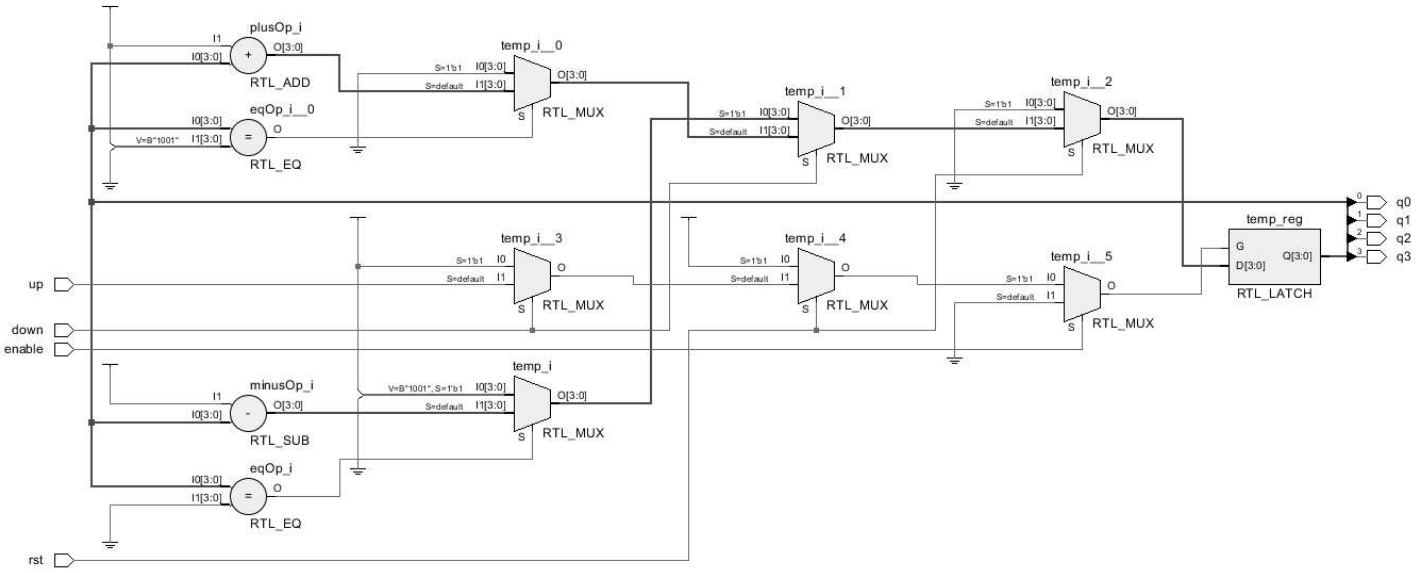


Figure 8.2. Sequential BCD Counter Experiment

QUESTIONS

1. A digital control circuit has three inputs: *Start*, *Stop* and *Clock*, as well as an output signal *Run*. The *Start* and *Stop* signals are of indeterminate duration and may span many clock cycles. When the *Start* signal goes to 1, the circuit must generate *Run* = 1. The *Run* signal must remain high until the *Stop* signal goes to 1, at which time it has to return to 0. All changes in the *Run* signal must be synchronized with the *Clock* signal
   a. Design the desired control circuit
   b. Write VHDL code that specifies the desired circuit

MODULE 9
STATE MACHINE USING VHDL

OBJECTIVES

1. To learn about the concept of state machine and it's design techniques
2. To design and construct a Traffic Light Controller logic using VHDL
3. To simulate, synthesize, and implement the design
4. To program the completed design onto the ZYBO Development Board FPGA

BASIC INFORMATION

State Machine

In this experiment we deal with a general class of circuits in which the outputs depend on the past behaviour of the circuit, as well as on the present values of inputs. They are called *sequential circuits*. In most cases a clock signal is used to control the operation of a sequential circuit; such a circuit is called a *synchronous sequential circuit*. The alternative, in which no clock signal is used, is called an *asynchronous sequential circuit*. Synchronous circuits are easier to design and are used in a vast majority of practical applications.

Synchronous sequential circuits are realized using combinational logic and one or more flip-flops. The general structure of such a circuit is shown in figure 9.1. The circuit has a set of primary inputs, W, and produces a set of outputs, Z. The values of the outputs of the flip-flops are referred to as the *state*, Q, of the circuit. Under control of the clock signal, the flip-flop outputs change their state as determined by the combinational logic that feeds the inputs of these flip-flops. Thus the circuit moves from one state to another. To ensure that only one transition from one state to another takes place during one clock cycle, the flip-flops have to be of the edge-triggered type. They can be triggered either by the positive (0 to 1 transition) or by the negative (1 to 0 transition) edge of the clock. We will use the term *active clock edge* to refer to the clock edge that causes the change in state.

The combinational logic that provides the input signals to the flip-flops derives its inputs from two sources: the primary inputs, W, and the present (current) outputs of the flip-flops, Q. Thus changes in state depend on both the present state and the values of the primary inputs.

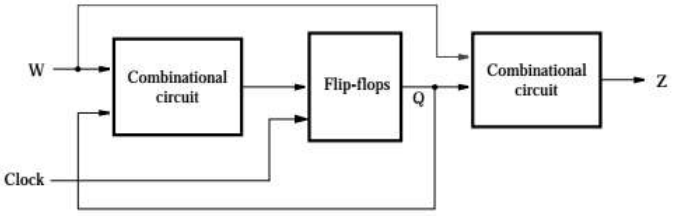


Figure 9.1. The general form of a sequential circuit

Figure 9.1 indicates that the outputs of the sequential circuit are generated by another combinational circuit, such that the outputs are a function of the present state of the flip-flops and of the primary inputs. Although the outputs always depend on the present state, they do not necessarily have to depend directly on the primary inputs. Thus the connection show in blue in the figure may or may not exist. To distinguish between these two possibilities, it is customary to say that sequential circuits whose outputs depend only on the state of the circuit are of *Moore* type, while those whose outputs depend on both the state and the primary inputs are of *Mealy* type. These names are in honour of Edward Moore and George Mealy, who investigated the behaviour of such circuits in the 1950s.

Sequential circuits are also called *finite state machines* (FSMs), which is a more formal name that is often found in technical literature. The name derives from the fact that the functional behaviour of these circuits can be represented using a finite number of states. In this chapter we will often use the term *finite state machine*, or simply *machine*, when referring to sequential circuits.

Table 9.1. Sequences of input and output signals

Clock cycle	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w	0	1	0	1	1	0	1	1	1	0	1
z	0	0	0	0	0	1	0	0	1	1	0

The first step in designing a finite state machine is to determine how many states are needed and which transitions are possible from one state to another. There is no set procedure for this task. The designer must think carefully about what the machine has to accomplish. A good way to begin is to select one particular state as a *starting* state; this is the state that the circuit should enter when power is first turned on or when a *reset* signal is applied. For our example let us assume that the starting state is called state *A*. As long as the input *w* is 0, the circuit need not do anything, and so each active clock edge should result in the circuit remaining in state *A*. When *w* becomes equal to 1, the machine should recognize this, and

move to a different state, which we will call state *B*. This transition takes place on the next active clock edge after *w* has become equal to 1. In state *B*, as in state *A*, the circuit should keep the value of output *z* at 0, because it has not yet seen *w* = 1 for two consecutive clock cycles. When in state *B*, if *w* is 0 at the next active clock edge, the circuit should move back to state *A*. However, if *w* = 1 when in state *B*, the circuit should change to a third state, called *C*, and it should then generate an output *z* = 1. The circuit should remain in state *C* as long as *w* = 1 and should continue to maintain *z* = 1. When *w* becomes 0, the machine should move back to state *A*. Since the preceding description handles all possible values of input w that the machine can encounter in its various states, we can conclude that three states are needed to implement the desired machine.

Now that we have determined in an informal way the possible transitions between states, we will describe a more formal procedure that can be used to design the corresponding sequential circuit. Behaviour of a sequential circuit can be described in several different ways. The conceptually simplest method is to use a pictorial representation in the form of a state diagram, which is a graph that depicts states of the circuit as nodes (circles) and transitions between states as directed arcs. The state diagram in figure 9.2 defines the behaviour that corresponds to our specification. States *A*, *B*, and *C* appear as nodes in the diagram. Node *A* represents the starting state, and it is also the state that the circuit will reach after an input *w* = 0 is applied. In this state the output *z* should be 0, which is indicated as *A*/*z*=0 in the node. The circuit should remain in state *A* as long as *w* = 0, which is indicated by an arc with a label *w* = 0 that originates and terminates at this node. The first occurrence of *w* = 1 (following the condition *w* = 0) is recorded by moving from state *A* to state *B*. This transition is indicated on the graph by an arc originating at *A* and terminating at *B*. The label *w* = 1 on this arc denotes the input value that causes the transition. In state *B* the output remains at 0, which is indicated as *B*/*z*=0 in the node.

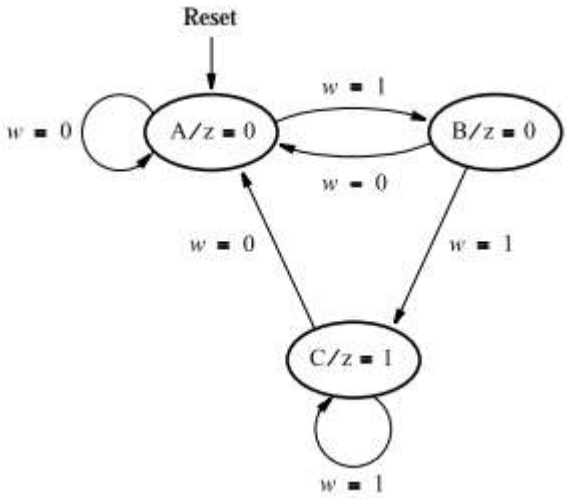


Figure 9.2. State diagram of a simple sequential circuit

When the circuit is in state *B*, it will change to state *C* if w is still equal to 1 at the next active clock edge. In state *C* the output *z* becomes equal to 1. If *w* stays at 1 during subsequent clock cycles, the circuit will remain in state *C* maintaining *z* = 1. However, if *w* becomes 0 when the circuit is either in state *B* or in state *C*, the next active clock edge will cause a transition to state *A* to take place.

In the diagram, we indicated that the *Reset* input is used to force the circuit into state *A*, which is possible regardless of what state the circuit happens to be in. We could treat *Reset* as just another input to the circuit, and show a transition from each state to the starting state A under control of the input *Reset*. This would complicate the diagram unnecessarily. States in a finite state machine are implemented using flip-flops. Since flip-flops usually

have reset capability, we can assume that the *Reset* input is used to clear all flip-flops to 0 by using this capability. We will indicate this as shown in figure 8.2 to keep the diagrams as simple as possible

----------PROCEDURE----------

MATERIALS REQUIRED

- ZYBO Zynq7000
- Micro-USB Power USB Cable
- Vivado™ Software
- PC/Laptop 64bit
- Resistors
- 2 Green LEDs, 2 Yellow LEDs, 3 Red LEDs, and 1 White LED

State Machine

1. Draw and design a state graph for the Traffic Light Controller, then convert the state graph to a State Machine Chart. State transitions will automatically occur after the specified delay time.
2. Create a new project, give description about the module's definition, entity, and port that will be used

3. Design the Traffic Light Controller logic using VHDL. **Set the I/O definitions and constraints: input in the button button0 and button1 to rst and mode, and the PMODs for all of the LEDs that will be used (Green, Yellow, Red, and White)**
4. Elaborate and simulate the design. Verify the schematic with the initial design's logic
5. Simulate the design. Check the behavioural simulation pattern
6. **Construct the circuit using the provided LEDs, resistors, and a breadboard. Connect the circuit to the PMODs using jumpers/wires, according to the constraints**
7. Synthesize, implement, and program the device. Record the state of the output for each input possibility

Figure 9.3. Traffic Light Diagram

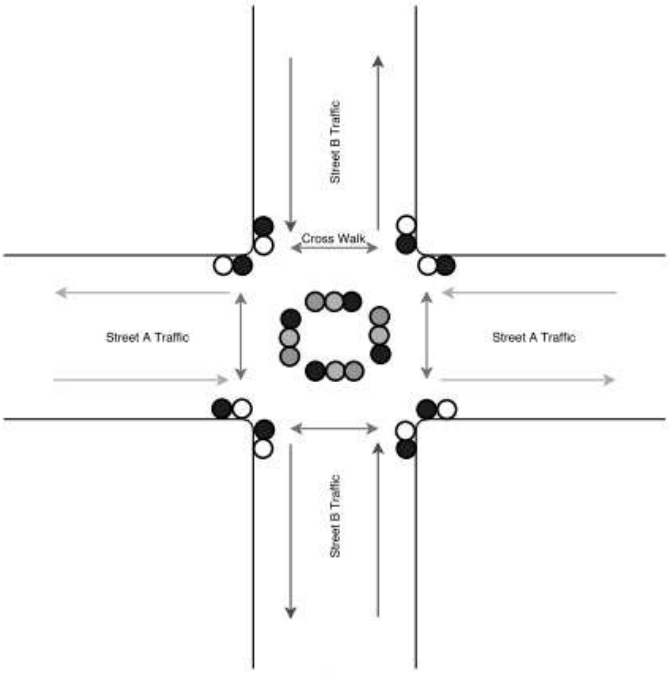


Table 9.2. Traffic Light Sequences

Street A	Street B	Street C
Green	Red	Red
Yellow	Red	Red
Red	Green	Red
Red	Yellow	Red
Red	Red	White
Red	Red	Red

Table 9.3. Traffic Light Sequences Timing

Street A	Street B	Pedestrian
Green - 4 sec	Green - 3 sec	White - 2 sec
Yellow - 2 sec	Yellow - 1 sec	Red - flashes 4 seconds at 1Hz, then solid 10 sec
Red - 10 sec	Red - 12 sec	

Notes:

- Traffic light A should consist of 3 lights: Green (Ga), Yellow (Ya), and Red (Ra)
- Traffic Light B should consist of 3 lights: Green (Gb), Yellow (Yb), and Red (Rb)
- The Pedestrian Crossing should consist of 2 lights: White (Ww), and Red (Rw)
- A *Maintenance Mode* will also be implemented. When *Maintenance Mode* is active, all three lights (R, Y, G) for each traffic light (A, B, P) should flash at 1Hz.
- When *Maintenance Mode* is switched off, all lights should reset to the starting mode of (Ga, Rb, Rw)

QUESTIONS

1. An FSM is defined by the state-assigned table in figure 9.1. Derive a circuit that realizes this FSM using D flip-flops!
2. Derive a circuit that realizes the FSM defined by the state-assigned table in figure 9.1 using JK flip-flops!

Table 9.4. State-assigned table for problems 1 and 2

Present State $y_2 y_1$		Next State $w = 0$ $Y_2 Y_1$	Next State $w = 1$ $Y_2 Y_1$	Output z
0	0	1 0	1 1	0
0	1	0 1	0 0	0
1	0	1 1	0 0	0
1	1	1 0	0 1	1

3. A sequential circuit has two inputs, w_1 and w_2, and an output, z. Its function is to compare the input sequences on the two inputs. If $w_1 = w_2$ during any four consecutive clock cycles, the circuit produces $z = 1$; otherwise, $z = 0$. For example

w_1 0110111000110
w_2 1110101000111
z 0000100001110

Derive a suitable circuit!