Lecture Notes for Digital Electronics

1 Basic Digital Concepts

By converting continuous analog signals into a finite number of discrete states, a process called digitization, then to the extent that the states are sufficiently well separated so that noise does create errors, the resulting digital signals allow the following (slightly idealized):

- storage over arbitrary periods of time - flawless retrieval

  and reproduction of the stored information - flawless

  transmission of the information

Some information is intrinsically digital, so it is natural to process and manipulate it using purely digital techniques. Examples are numbers and words.

The drawback to digitization is that a single analog signal (e.g. a voltage which is a function of time, like a stereo signal) needs many discrete states, or bits, in order to give a satisfactory reproduction. For example, it requires a minimum of 10 bits to determine a voltage at any given time to an accuracy of 0.1%. For transmission, one now requires 10 lines instead of the one original analog line.

The explosion in digital techniques and technology has been made possible by the incredible increase in the density of digital circuitry, its robust performance, its relatively low cost, and its speed. The requirement of using many bits in reproduction is no longer an issue: The more the better.

This circuitry is based upon the transistor, which can be operated as a switch with two states. Hence, the digital information is intrinsically binary. So in practice, the terms digital

and binary are used interchangeably. In the following sections we summarize some conventions for defining the binary states and for doing binary arithmetic.

1.1 Binary Logic States

ᵗlᵗhe following table attempts to make correspondences between conventions for defining binary logic states. In the case of the logic gates we will be using in the lab, the Low voltage state is roughly 0—1 Volt and the High state is roughly 2.5—5 Volts. See page 475 of the text for the exact conventions for as well as other hardware gate technologies.

Boolean Logic	Boolean Algebra	Voltage State (positive true)	Voltage State (negative true)
'[Yue (ᴵlᴵ)	1	High (H)	Low (L)
False (F)			

'l'he convention for naming these states is illustrated in Fig. 1. 'l'he "positive true" case is illustrated. 'l'he relationship between the logic state and label (in this case "switch open") at some point in the circuit can be summarized with the following:

`The labelled voltage is High (Low) when the label's stated function is True (False) .`

In the figure, the stated function is certainly true (switch open), and this does correspond to a high voltage at the labelled point. (Recall that with the switch open, Ohm's Law implies that with zero current, the voltage difference across the "pull up" resistor is zero, so that the labelled point is at +5 Volts. With a closed switch, the labelled point is connected to ground, with a 5 Volt drop across the resistor and a current of I V/R 5 mA through it.)
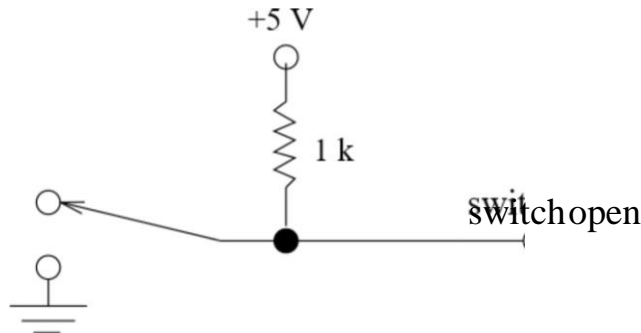

Figure 1: Illustration for labelling logic states ("positive true").

With the convention known as "negative true", the label would be changed to "switch closed" with a bar over it: switch closed. Our statement becomes:

'The labelled voltage is Low (High) when the label's stated function is True (False) . So in the figure, the stated function (switch closed) is true when the voltage is low. 'l'he bar is meant to envoke the boolean inversion operation: F, É T, T, and so forth.

1.2 Binary Arithmetic

Each digit in binary is a 0 or a 1 and is called a bit, which is an abbreviation of binary digit. There are several common conventions for representation of numbers in binary.

The most familiar is unsigned binary. An example of a 8-bit number in this case is

$$1111_2 = 0 \times 2^7\ 1 \times 2^6 + \ldots + 1 \times 2^0 = 64 + 8 + 4 + 2 + 1\ 79_{10}$$

(Generally the subscripts will be omitted, since it will be clear from the context.) To convert from base 10 to binary, one can use a decomposition like above, or use the following algorithm illustrated by 79: 79/2 39, remainder 1, then 39/2 19 r 1, and so forth. Then assemble all the remainders in reverse order.

The largest number which can be represented by n bits is $2^{71} - 1$. For example, with 4 bits the largest number is $1111_2 = 15$.

The most significant bit (MSB) is the bit representing the highest power of 2, and the LSB represents the lowest power of 2.

Arithmetic with unsigned binary is analogous to decimal. For example I-bit addition and multiplication are as follows: 0 +0 0, 0 + 1 1, 1 + 1 0, 0 x 0 0, 0 x 1 0, and $\times 1 = 1$. Note that this is different from Boolean algebra, as we shall see shortly, where 1+1= 1.

Another convention is called BCD ("binary coded decmal"). In this case each decimal digit is separately converted to binary. Therefore, since 7 01112 and 9 — 10012, then 79 = 01111001 (BCD). Note that this is *different* than our previous result. We will use BCD quite often in this course. It is quite convenient, for example, when decimal numerical displays are used.

Yet another convention is Gray code. You have a homework problem to practice this. This is less commonly used.

1.2.1 Representation of Negative Numbers

There are two commonly used conventions for representing negative numbers.

With sign magnitude, the MSB is used to flag a negative number. So for example with 4-bit numbers we would have 0011 3 and 1011 —3. This is simple to see, but is not good for doing arithmetic.

With 2's complement, negative numbers are designed so that the sum of a number and its 2's complement is zero. Using the 4-bit example again, we have 0101 5 and its 2's complement —5 = 1011. Adding (remember to carry) gives 10000 0. (The 5th bit doesn't count!) Both addition and multiplication work as you would expect using 2's complement. There are two methods for forming the 2's complement:

1. Make the transformation 0 —+ 1 and 1 —+ 0, then add 1.

2. Add some number to -2^{MSB} to get the number you want. For 4-bit numbers an example of finding the 2's complement of 5 is —5 = —8 3 + 0011 1011.

1.2.2 Hexadecimal Representation

It is very often quite useful to represent blocks of 4 bits by a single digit. Thus in base 16 there is a convention for using one digit for the numbers 0,1,2,...,15 which is called hexadecimal. It follows decimal for 0—9, then uses letters A—F.

Decimal	Binary	Hex

o	0000	1
1	0001	2
2	0010	3
3	0011	4
4	0100	5
5	0101	6
6	0110	7
7	0111	8
8	1000	9
9	1001	
10	1010	
11	1011	
12	1100	c
13	1101	
14	1110	
15		

2 Logic Gates and Combinational Logic

2.1 Gate Types and Truth Tables

Thebmsic logic gates are AND, OR, NAND, NOR, XOR, INV, and BUF. last two are not standard terms; they stand for "inverter" and "buffer" , respectively. 'l'he symbols for these gates and their corresponding Boolean expressions are given in 'l'able 8.2 of the text which, for convenience, is reproduced (in part) in Fig. 2.

Name	Expression	Symbol	Negative trt* symbol
ANO			
NAND	A		
NOR			

INVERT **A**

BUFFER **A**

AOB

Figure 2: Table 8.2 from the text.

All of the logical gate functions, as well as the Boolean relations discussed in the next section, follow from the truth tables for the AND and OR gates. We reproduce these below. We also show the XOR truth table, because it comes up quite often, although, as we shall see, it is not elemental.
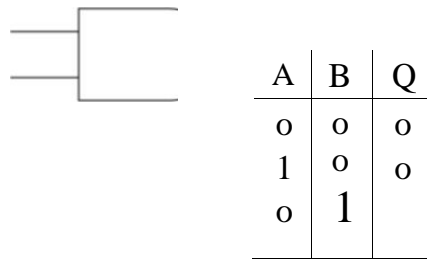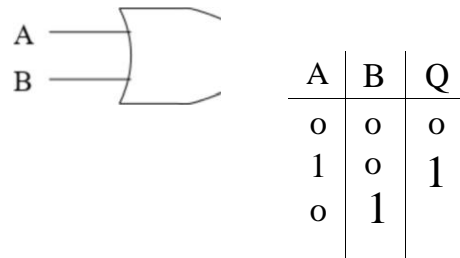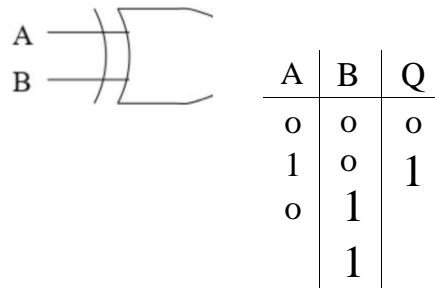
Figure 3: AND gate.

Figure 4: OR gate.

Figure 5: XOR (exclusive OR) gate.

2.2 Boolean Algebra and DeMorgan's Theorems

Boolean algebra can be used to formalize the combinations of binary logic states. 'lhe fundamental relations are given in 'lhble 8.3 of the text. In these relations, A and B are

binary quantities, that is, they can be either logical true (T or 1) or logical false (F or 0). Most of these relations are obvious. Here are a few of them:

$$AA = A; \quad A + A = A; \quad A + \overline{A} = 1;$$

Recall that the text sometimes uses an apostrophe for inversion (N). We use the standard overbar notation (Ä).

We can use algebraic expressions to complete our definitions of the basic logic gates we began above. Note that the Boolean operations of "multiplication" and "addition" are defined by the truth tables for the AND and OR gates given above in Figs. 3 and 4. Using these definitions, we can define all of the logic gates algebraically 'I'he truth tables can also be constructed from these relations, if necessary. See Fig. 2 for the gate symbols.

- AND: Q = AB (see Fig. 3)
- OR: $Q = A + B$ (see Fig. 4)

- NAND: Q= AB

- NOR: Q= $A +$

- XOR: Q= $A \oplus B$ (defined by truth table Fig. 5)

- INV: Q= A

- BUF: Q= A

2.2.1 Example: Combining Gates

Let's re-express the XOR operation in terms of standard Boolean operations. 'I'he following truth table evaluates the expression Q ÄB + AD.

				Q
O	O	O	O	O
1	O	O	1	1
O	1	1	O	1
1	1	O	O	O

We see that this truth table is identical to the one for the XOR operation. 'I'herefore, we can write

$$A \oplus B = \overline{A}B + A\overline{B} \tag{1}$$

A schematic of this expression in terms of gates is given in Fig. 6 (as well as Fig. 8.25 of the text). Recall that the open circles at the output or input of a gate represent inversion.
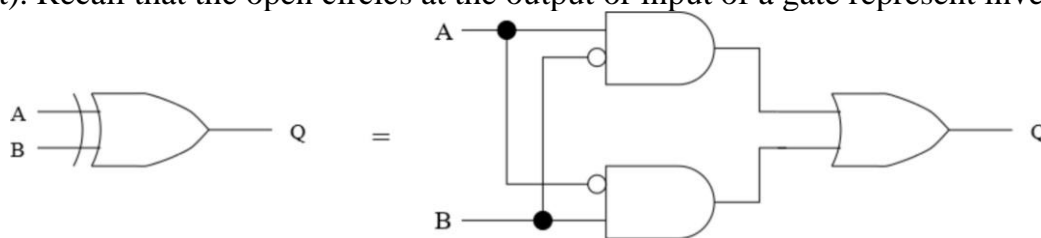
Figure 6: Realization of the XOR gate in terms of AND and OR gates.

2.2.2 Gate Interchangeablilty

In an example from the homework, we can make an INV gate from a 2-input NOR gate. Simply connect the two inputs of the NOR gate together. Algebraically, if the two original NOR gate inputs are labelled B and C, and they are combined to form A, then we have $Q = \overline{B+C} = \overline{A+A} = \ddot{A}$, which is the INV operation.

Note that an INV gate can not be made from OR or AND gates. For this reason the OR and AND gates are not universal. So for example, no combination of AND gates can be combined to substitute for a NOR gate. However, the NAND and NOR gates are universal.

2.2.3 DeMorgan

Perhaps the most interesting of the Boolean identities are the two known as DeMorgan's Ïheorems:

$$\overline{A + B} = \bar{A}\bar{B} \qquad (\text{or, } A + B = \overline{\overline{A}\overline{B}}) \quad (2)$$

$$\overline{AB} = \overline{A} + \overline{B} \qquad (\text{or, } AB = \overline{\overline{A} + \overline{B}}) \quad (3)$$

Ïhese expressions turn out to be quite useful, and we shall use them often.

An example of algebraic logic manipulation follows. It is the one mentioned at the end of Lab 1. One is to show that an XOR gate can be composed of 4 NAND gates. From the section above we know A Q) B ÄB + AB. Since AÄ 0 and 13B 0, we can add these, rearrange, and apply the two DeMorgan relations to give

$$A \oplus B = A(\overline{A} + \overline{B}) + B(\ddot{A}+B) \; A(\ddot{A}B) + {}^{B(\overline{AB})} = \big(A(AB) \; B(AB))$$

2.3 Symbolic Logic

Ïhe two DeMorgan expressions above can be envoked using gate symbols by following this prescription: Change gate shape (AND—OR) and invert all inputs and outputs.

By examining the two rightmost columns of Fig. 2, one sees that the transformation between 3rd and 4th columns for the gates involving AND/OR gates works exactly in this way. For example, the DeMorgan expression AB Ä+B is represented symbolically by the equivalence between the 3rd and 4th columns of the 2nd row ("NAND") of Fig. 2. We will go over how this works, and some more examples, in class.

2.4 Logic Minimization and Karnaugh Maps

As we found above, given a truth table, it is always possible to write down a correct logic expression simply by forming an OR of the ANDS of all input variables for which the output is true (Q 1). However, for an arbitrary truth table such a procedure could produce a very lengthy and cumbersome expression which might be needlessly inefficient to implement with gates.

Ïhere are several methods for simplification of Boolean logic expressions. Ïhe process is usually called "logic minimization", and the goal is to form a result which is efficient. Ïwo methods we will discuss are algebraic minimization and Karnaugh maps. For very complicated problems

the former method can be done using special software analysis programs. Karnaugh maps are also limited to problems with up to 4 binary inputs.

Let's start with a simple example. 'l'he table below gives an arbitrary truth table involving 2 logic inputs.

'T'al)le 1: Example of simple arbitrary truth table.

A	B	Q
o	o	1
o	1	1
1	o	
1	1	o
		1

ıl'here are two overall stategies:

1. Write down an expression directly from the truth table. Use Boolean algebra, if desired, to simplify.

2. Use Karnaugh mapping ("K-map"). 'l'his is only applicable if there are 4 inputs.

In our example above, we can use two different ways of writin down a result directly from the truth table. We can write down all TRUE terms and OR the result. 'l'his gives

$$Q = \bar{A}\bar{B} + \bar{A}B + AB$$

While correct, without further simplification this expression would involve 3 2-input AND gates, 2 inverters, and 1 3-input OR gate.

Alternatively, one can write down an expression for all of the FALSE states of the truth table. 'l'llis is simpler in this case:

$$\rightarrow Q = \overline{A\bar{B}} = \bar{A} + B$$

where the last step results from Eqn. 3. Presumably, the two expressions can be found to be equivalent with some algebra. Certainly, the 2nd is simpler, and involves only an inverter and one 2-input OR gate.

Finally, one can try a K-map solution. 'l'he first step is to write out the truth table in the form below, with the input states the headings of rows and columns of a table, and the corresponding outputs within, as shown below.

'rable 2: K-map of truth table.

1

'l'he steps/rules are as follows:

1. Form the 2-dimensional table as above. Combine 2 inputs in a "gray code" way — see 2nd example below.

2. Form groups of I's and circle them; the groups are rectangular and must have sides of length $2^n \times 2^m$, where n and m are integers 0, 1, 2,. . ..

3. 'I'he groups can overlap.

4. Write down an expression of the inputs for each group.

5. OR together these expressions. 'I'hat's it.

6. Groups can wrap across table edges.

7. As before, one can alternatively form groups of (Vs to give a solution for Q.

8. 'I'he bigger the groups one can form, the better (simpler) the result.

9. 'I'here are usually many alternative solutions, all equivalent, some better than others depending upon what one is trying to optimize.

Here is one way of doing it:

'I'he two groups we have drawn are Ä and B. So the solution (as before) is:

$$1 \quad 0 \quad Q = \bar{A} + B$$

2.4.1 K-map Example 2

Let's use this to determine which 3-bit numbers are prime. ('I'his is a homework problem.) We assume that 0, 1, 2 are not prime. We will let our input number have digits a2a1ao. Here is the truth table:

Here is the corresponding K-map and a solution.

Note that where two inputs are combined in a row or column that their progression follows gray code, that is only one bit changes at a time. 'I'he solution shown above is:

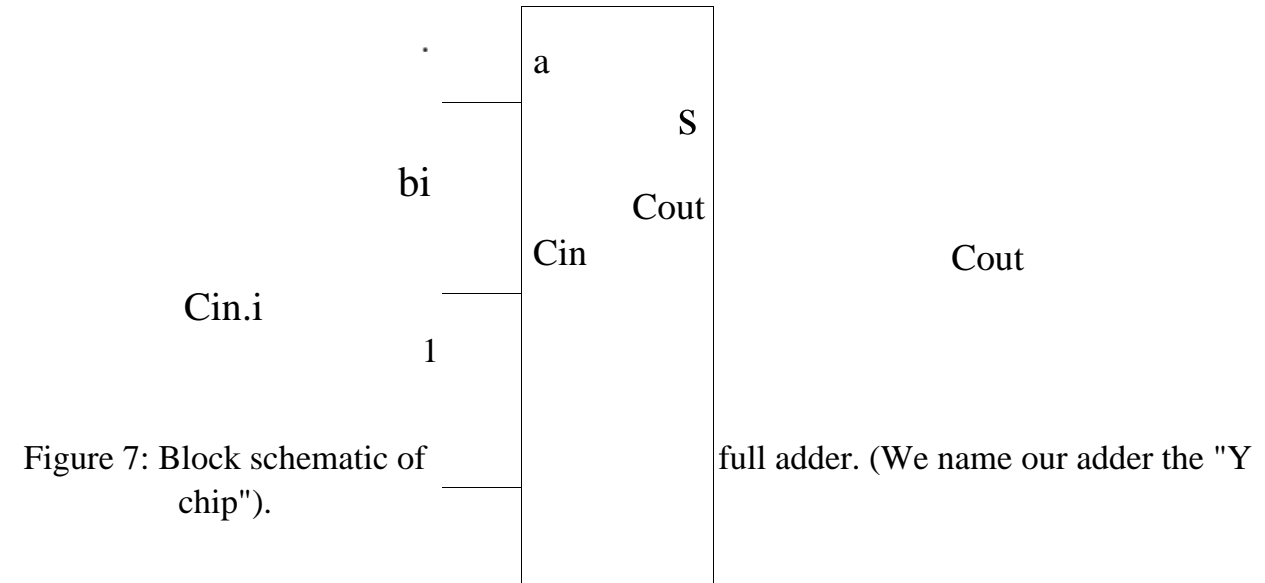Q al ao +   ao(al + 02)

'rable 3: 3-digit prime finder.

Decimal		al	ao	Q
o	o	o	o	o
1	o	o	1	
2	o	1	o	o
3	o	1	1	1
4	1	o	o	o
5	1	o	1	1
6	1	1	o	o
7	1	1	1	1

Table 4: K-map of truth table.

ao	00	01	11	10
o	o	o	1	
1	o	1	1	

2.4.2 K-map Example 3: Full Adder

In this example we will outline how to build a digital full adder. It is called "full" because it will include a "carry-in" bit and a "carry-out" bit. 'I'he carry bits will allow a succession of I-bit full adders to be used to add binary numbers of arbitrary length. (A half adder includes only one carry bit.)

a

S

bi
Cout

Cin   Cout

Cin.i

1

Figure 7: Block schematic of   full adder. (We name our adder the "Y chip").

᠎l'he scheme for the full adder is outlined in Fig. 7. Imagine that we are adding two n-bit binary numbers. Let the inputs ai and bi be the i-th bits of the two numbers. 'lhe carry in bit Cin_i represents any carry from the sum of the neighboring less significant bits at position $i-1$. 'That is, Cini 1 if bt—l 1, and is 0 otherwise. ᵗiᵗhe sum Si at position i is therefore the sum of at, , bi, and Cini. (Note that this is an arithmetic sum, not a Boolean OR.) A carry for this sum sets the carry out bit, Couti 1, which then can be applied to the sum of the i + 1 bits. 'I'he truth table is given below.

Cint				Couti
o	o	o	o	o
o	o	1	1	
o	1	o	1	
o	1	1	o	1
1	o	1	o	
1	o	o	1	1
1	1	1	o	1
1	1	o	1	1

With Cini 0, we see that the output sum Si is just given by the XOR operation, ai Q) bi. And with Cini 1, then Si ai Q) bi. Perhaps the simplest way to express this relationship is the following:

Si Cini Q) (ai bi)

'I᷈o determine a relatively simple expression for Couti, we will use a K-map:

Cilli\ai bi	00	01	11	10
o	o	o	1	
1	o	1	1	1

11

'l'his yields

$$Couti\ aibi + Cint\bullet ai + Cint\bullet bi\ aibi + Cini(ai + bi)\ which$$

in hardware would be 2 2-input OR gates and 2 2-input AND gates.

As stated above, the carry bits allow our adder to be expanded to add any number of bits. As an example, a 4-bit adder circuit is depicted in Fig. 8. 'l'he sum can be 5 bits, where the MSB is formed by the final carry out. (Sometimes this is referred to as an "overflow" bit.)
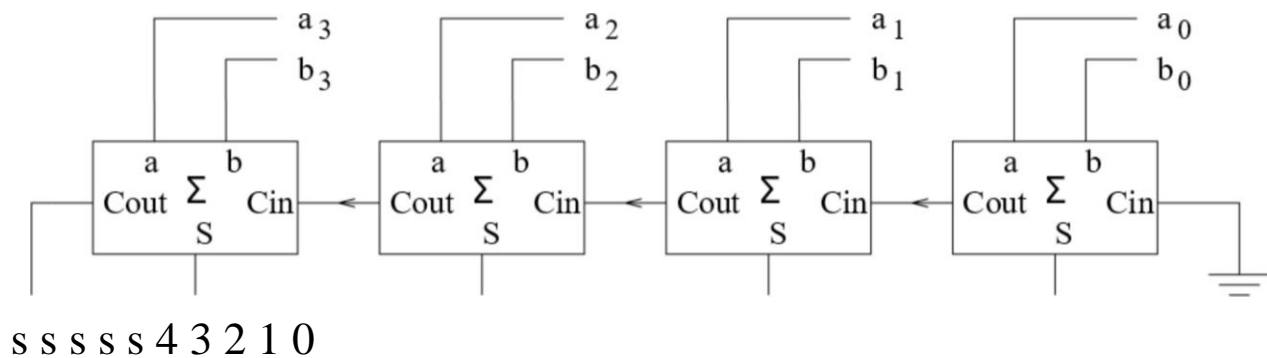

s s s s s 4 3 2 1 0

Figure 8: Expansion of I-bit full adder to make a 4-bit adder.

2.4.3 Making a Multiplier from an Adder

In class we will discuss how to use our full adder (the "E chip") to make a multiplier.

2.5 Multiplexing

A multiplexer (MUX) is a device which selects one of many inputs to a single output. 'l'he selection is done by using a,n input address. Hence, a MUX can take many data bits and put them, one at a time, on a single output data line in a particular sequence. 'l'his is an example of transforming parallel data to serial data. A demultiplexer (DEMUX) performs the inverse operation, taking one input and sending it to one of many possible outputs. Again the output line is selected using an address.

A MUX-DEMUX pair can be used to convert data to serial form for transmission, thus reducing the number of required transmission lines. 'l'he address bits are shared by the MUX and DEMUX at each end. If n data bits are to be transmitted, then after multiplexing, the number of separate lines required is $\log 2\ n + 1$, compared to n without the conversion to serial. Hence for large n the saving can be substantial. In Lab 2, you will build such a system.

Multiplexers consist of two functionally separate components, a decoder and some switches or gates. 'l'he decoder interprets the input address to select a single data bit. We use the example of a 4-bit MUX in the following section to illustrate how this works.

2.5.1 A 4-bit MUX Design

We wish to design a 4-bit multiplexer. 'l'he block diagram is given in Fig. 9. 'l'here are 4 input data bits Do 1)3, 2 input address bits Ao and Al, one serial output data bit Q, and an (optional) enable bit E which is used for expansion (discussed later). First we will design the decoder.
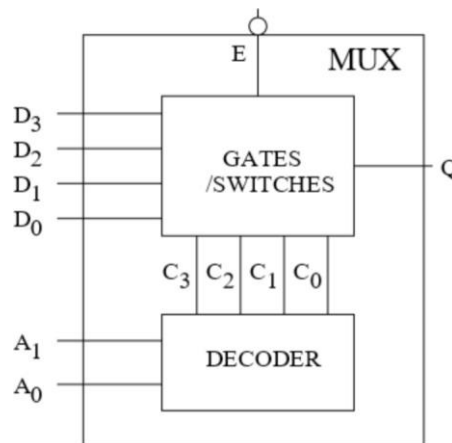

Figure 9: Block diagram of 4-bit MUX.

We need m address bits to specify 2^m data bits. So in our example, we have 2 address bits. 'l'he truth table for our decoder is straightforward:

Al		CO			C3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

'l'he implementation of the truth table with standard gates is also straightforward, as given in Fig. 10.
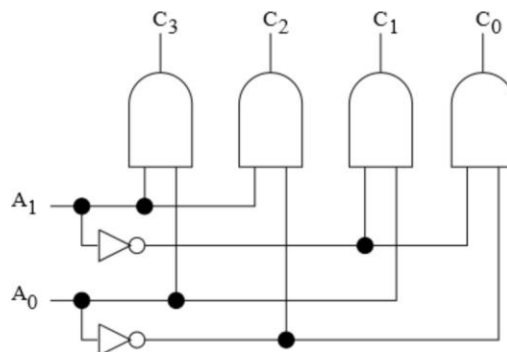

Figure 10: Decoder for the 4-bit MUX.

13

For the "gates/switches" part of the MUX, the design depends upon whether the input data lines carry digital or analog signals. We will discuss the analog possibility later. The digital case is the usual and simplest case. Here, the data routing can be accomplished simply by forming 2-input ANDS of the decoder outputs with the corresponding data input, and then forming an OR of these terms. Explicitly,

$$Q \qquad + CID\ 1 + C'2D'2 + C3D3$$

Finally, if an ENABLE line E is included, it is simply ANDed with the righthand side of this expression. T'llis can be used to switch the entire MUX IC off/on, and is useful for expansion to more bits. as we shall see.

3 Flip-Flops and Introductory Sequential Logic

We now turn to digital circuits which have states which change in time, usually according to an external clock. 'l'he flip-flop is an important element of such circuits. It has the interesting property of memory: It can be set to a state which is retained until explicitly reset.

3.1   Simple Latches

'l'he following 3 figures are equivalent representations of a simple circuit. In general these are called flip-flops. Specifically, these examples are called SR ("set-reset") flip-flops, or SR latches.
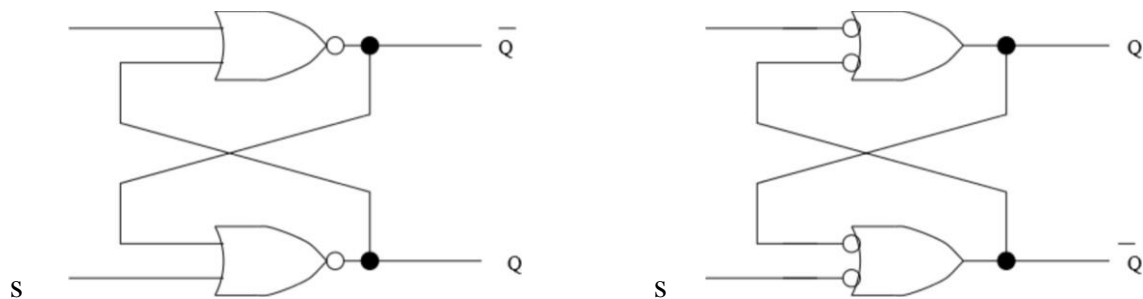
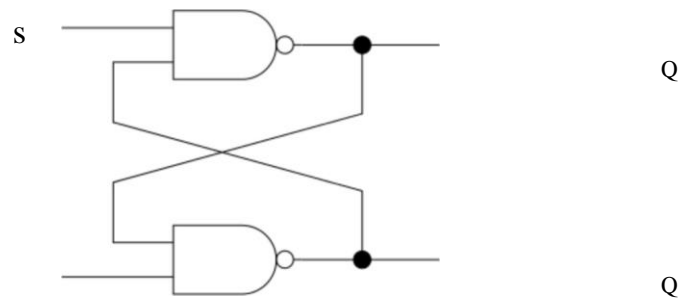Figure 11: '['wo equivalent versions of an SR flil)-flop (or "SR latch").

Figure 12: Yet another equivalent SR flip-flop, as used in Lab 3.

'l'he truth table for the SR latch is given below.

S	R		Q	Q	
1	o	o	1	1	
o	1	1	o	o	1 etains
o	1	o	1		vious
1	o	1	o	o	

15

'l'he state described by the last row is clearly problematic, since Q and $\overline{Q}$ should not be the same value. 'l'hus, the S R 1 inputs should be avoided.

l*Yom the truth table, we can develop a sequence such as the following:

1. $R = 0, S = 1 \Rightarrow Q = 1$ (set)

2. $R = 0, S = 0 \Rightarrow Q - 1$ (Q 1 state retained: "memory')

3. $R = 1, S = 0 \Rightarrow Q = 0$ (reset)

4. $R = 0, S = 0 \Rightarrow Q = 0$ (Q 0 state retained)

In alternative language, the first operation "writes" a true state into one bit of memory. It can subsequently be "read" until it is erased by the reset operation of the third line.

3.1.1 Latch Example: Debounced Switch

A useful example of the simple SR flip-flop is the debounced switch, like the ones on the lab prototyping boards. 'l¹he point is that any simple mechanical switch will bounce as it makes contact. Hence, an attempt to provide a simple transition from digital HIGH to LOW with a mechanical switch may result in an unintended series of transitions between the two states as the switch damps to its final position. So, for example, a digital counter connected to Q would count every bounce, rather than the single push of the button which was intended.

'l'he debounced configuration and corresponding truth table are given below. When the switch is moved from A to B, for example, the output Q goes LOW. A bounce would result in A B 1, which is the "retain previous" state of the flip-flop. Hence, the bounces do not appear at the output Q.
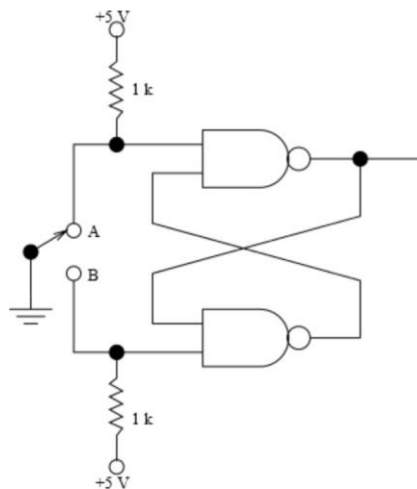

Figure 13: A debounced switch.

1	1	
0		
1	1	1 retains
		previous not
0	0	allowed

3.2 Clocked Flip-flops

We will soon get used to the idea of a clock as an essential element of digital circuitry. When we speak of a clock signal, we mean a sequence of evenly spaced digital high and low signals proceeding at a fixed frequency. That is, the clock is a continuous sequence of square wave pulses. There are a number of reasons for the importance of the clock. Clearly it is essential for doing any kind of counting or timing operation. But, its most important role is in providing synchronization to the digital circuit. Each clock pulse may represent the transition to a new digital state of a "state machine" (simple processor) we will soon encounter. Or a clock pulse may correspond to the movement of a bit of data from one location in memory to another. A digital circuit coordinates these various functions by the synchronization provided by a single clock signal which is shared throughout the circuit. A more sophisticated example of this concept is the clock of a computer, which we have come to associate with processing speed (e.g. 330 MHz for typical current generation commercial processors.

We can include a clock signal to our simple SR flip-flop, as shown in Fig. 14. The truth table, given below, follows directly from our previous SR flip-flop, except now we include a label for then clock pulse for the inputs and the output. This is because the inputs have no effect unless they coincide with a clock pulse. (Note that a specified clock pulse conventionally refers to a HIGH level.) As indicated in the truth table, the inputs SIC Rib 0 represent the flip-flop memory state. Significantly, one notes that the interval between clock pulses also corresponds to the "retain previous state" of the flip-flop. Hence the information encoded by the one bit of flip-flop memory can only be modified in synchronization with the clock.
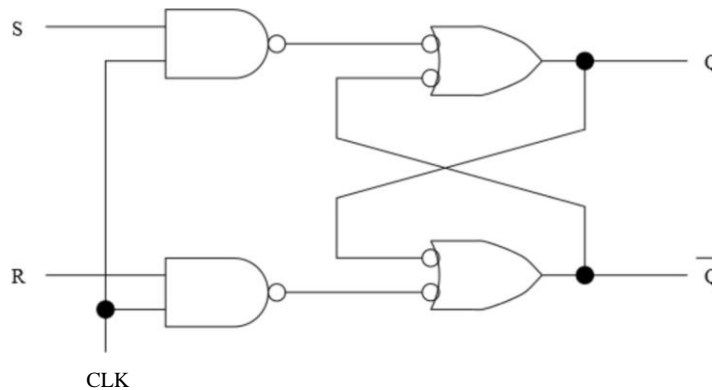

Figure 14: A clocked SR flirflop.

1	0	1
0	1	
0	0	con-I
1	1	avoid

We are now set to make a subtle transition for our next version of the clocked flip-flop. 'l'he flip-flop memory is being used to retain the state between clock pulses. In fact, the state set up by the S and R inputs can be represented by a single input we call "data", or D. 'l'his is shown in Fig. 15. Note that we have explicitly eliminated the bad S R 1 state with this configuration.

We can override this data input and clock sychronization scheme by including the "jam set" (S) and "jam reset" (R) inputs shown in Fig. 15. 'l'hese function just as before with the unclocked SR flip-flop. Note that these "jam" inputs go by various names. So sometimes the set is called "preset" and reset is called "clear" , for example.
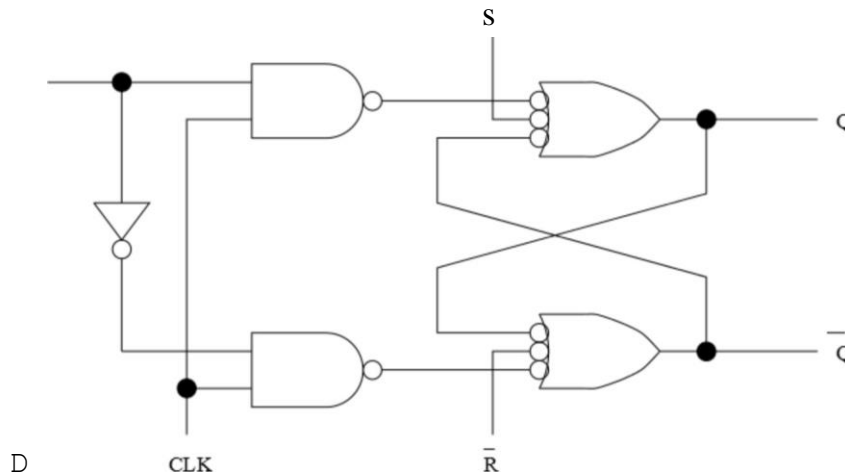

Figure 15: A "ID-type transparent" flip-flop with jam set and reset.

A typical timing diagram for this flirflop is given in Fig. 16. Note that the jam reset signal R overrides any action of the data or clock inputs.
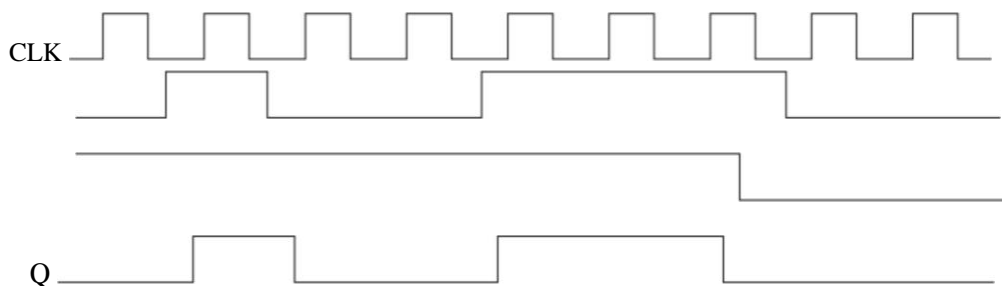

Figure 16: Example of timing diagram for the transparent D flip-flop. (It is assumed that S is held HIGH throughout.)

3.'2.1 Edge Triggered Flip-Flops

We need to make one final modification to our clocked flip-flop. Note that in the timing diagram of Fig. 16 that there is quite a bit of apparent ambiguity regarding exactly when the D input gets latched into Q. If a transition in D occurs sometime during a clock HIGH, for example, what will occur? 'l'he answer will depend upon the characteristics of the particular electronics being used. 'l'his lack of clarity is often unacceptable. As a point of terminology, the clocked flip-flop of Fig. 15 is called a transparent D-type flip-flop or latch. (An example in   is the 7475 IC.)

'l'he solution to this is the edge-triggered flip-flop. We will discuss how this works for one example in class. It is also discussed some in the text. 'rriggering on a clock rising or falling edge is similar in all respects to what we have discussed, except that it requires 2—3 coupled SR-type flip-flops, rather than just one clocked SR flip-flop. 'l'he most common type is the positive-edge triggered D-type flip-flop. 'l'his latches the D input upon the clock transition from LOW to HIGH. An example of this in is the 7474 IC. It is also common to employ a negative-edge triggered D-type flip-flop, which latches the D input upon the clock transition from HIGH to LOW.

'l'he symbols used for these three D-type flip-flops are depicted in Fig. 17. Note that the small triangle at the clock input depicts positive-edge triggering, and with an inversion symbol represents negative-edge triggered. 'l'he JK type of flip-flop is a slightlier fancier version of the D-type which we will discuss briefly later. Not shown in the figure are the jam set and reset inputs, which are typically included in the flip-flop IC packages. In timing diagrams, the clocks for edge-triggered devices are indicated by arrows, as shown in Fig. 18.
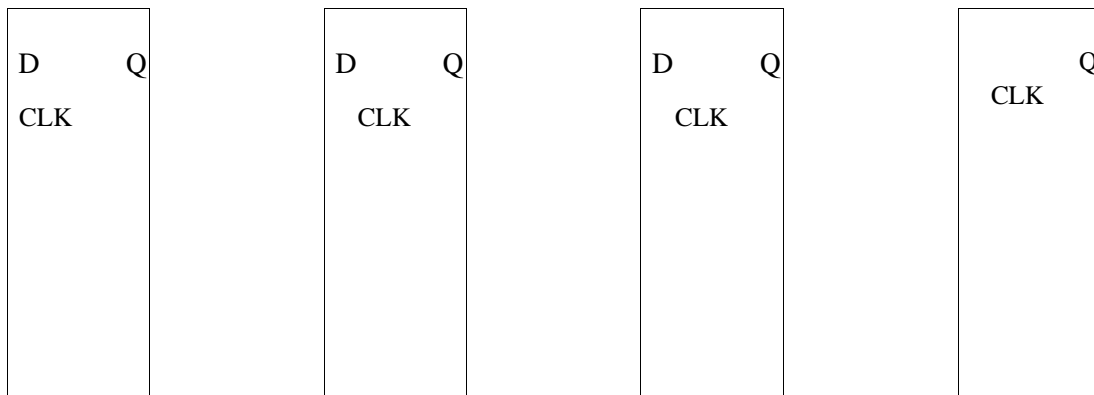
Figure 17: Symbols for D-type and JK flip-flops. Left to right: transparent D-type, positiveedge triggered D-type, negative-edge triggered D-type, and positive-edge triggered J K-type.

SUL SISL

Figure 18: Clocks in timing diagrams for positive-edge triggered (left) and negative-edge triggered (right) devices.

For edge-triggered devices, the ambiguity regarding latch timing is reduced significantly. But at high clock frequency it will become an issue again. Typically, the requirements are as follows:

- 'l'he data input must be held for a time tsetup before the clock edge. Typically, tsetup $\approx$20 ns or less.

- For some ICs, the data must be held for a short time thold after the clock edge. Typically thold 3 ns, but is zero for most newer ICs.

- 'l'he output Q appears after a short propagation delay tprop of the signal through the gates of the IC. Typically, tprop ns.

1*Yom these considerations we see that for clocks of frequency much less than 1/ (Ions) = 100 MHz, these issues will be unimportant, and we can effectively consider the transitions to occur instantaneously in our timing diagrams.

20

4 Counters, Registers, and State Machines

We can now apply what we know about basic flip-flops circuit elements to develop new functions: counters and registers. In doing so, we will introduce the "state machine", a clocked sequential "processor". We will examine this latter topic in more detail in a few weeks.

4.1 Divide by Two Counter

The edge-triggered D-type flip-flops which we introduced in the previous Section are quite useful and versatile building blocks of sequential logic. A simple application is the divide-by-2 counter shown in Fig. 19, along with the corresponding timing diagram.
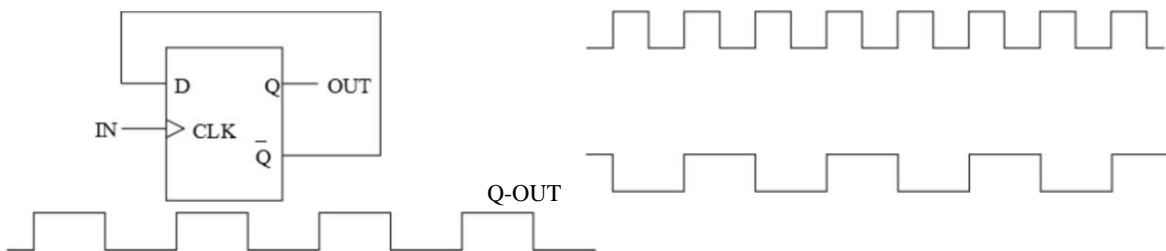

Figure 19: Positive edge-triggered D-type flip-flop connected as divide-by-2 counter.

4.1.1 Using the JK Flip-flop

In Lab 4 you will build an asynchronous (ripple) counter using a sequence of cascaded JK flip-flops, rather than the D-type which is used in our discussion below. For reference, the JK truth table is given in Fig. 20. Note that there is no fundamental advantage to using the JK instead of the D-type, only that the JK, with the additional J K 1 state, makes the divide-by-2 function slightly simpler to implement.

1	o	1
o	1	con-
o	o	I
1	1	

Figure 20: 'l'he JK Flip-flop.

4.2 Asynchronous Counter

Flip-flops can be connected in series, as shown in Fig. 21. 'l'he resulting outputs are given in Fig. 22. (Note that labels in these two figures correspond when A 2^0 B 2^1 C 2^2 ,and D 2^3 . Hence, this is a 4-bit counter, with maximum count 2^4 — 15. It is clearly possible to expand such a counter to an indefinite number of bits.

While asynchronous counters are easy to assemble, they have serious drawbacks for some applications. In particular, the input must propogate through the entire chain of flip-flops before the correct result is achieved. Eventually, at high input rate, the two ends of the chain, representing the LSB and NTSB, can be processing different input pulses entirely (Perhaps in lab you can see this effect on the oscilloscope with a very high input frequency.) 'l'he solution to this is the synchronous counter, which we will discuss below as an example of a state machine.
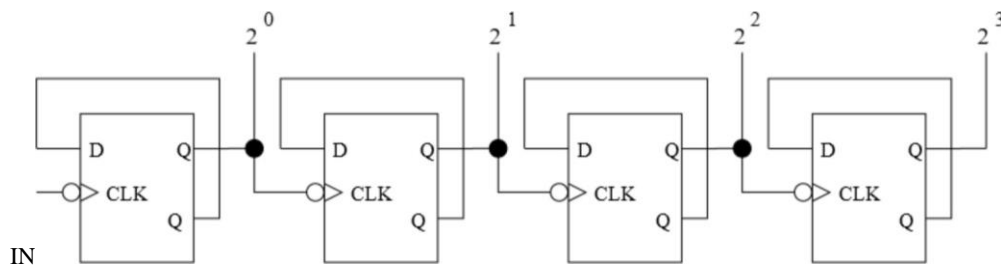
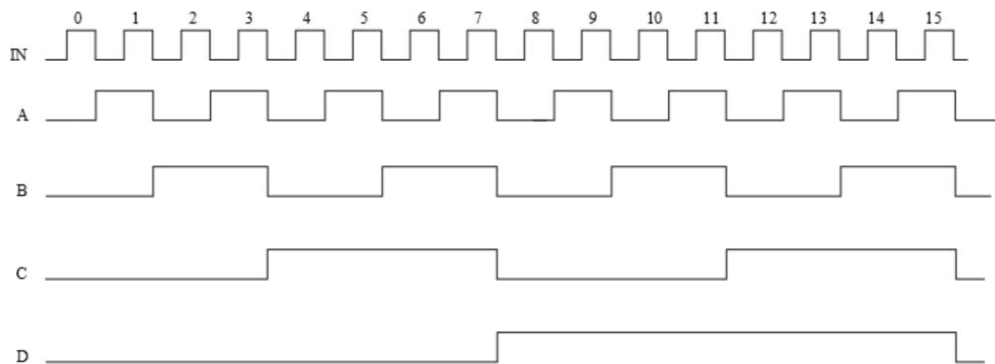Figure 21: Asynchronous ("ripple") counter made from cascaded D-type flip-flops.

Figure 22: Waveforms generated by the ripple counter.

4.3 Registers
4.3.1 Basic Register

'l'he figure below represents a 4-bit memory. We can think of it as 4 individual D-type flip-flops. 'l'he important point about a data register of this type is that all of the inputs are latched into memory synchronously by a single clock cycle.
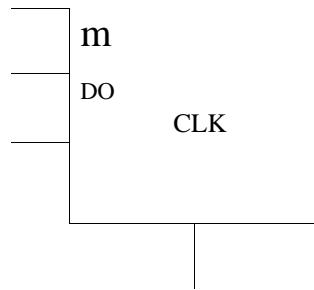
(23

22

m

DO

CLK

Figure 23: 4-bit data register.

4.3.2 Shift Registers

The figure below is an example of a 4-bit shift register. These configurations are quite useful, particularly for transforming serial data to parallel, and parallel to serial. In the circuit below, a pulse appearing at "serial in" would be shifted from the output of one flipflop to the next on each clock cycle. Hence a serial bit pattern at the input (4 bits long in this example) would appear as 4 parallel bits in the outputs Qo—Q3 after 4 clock cycles. This represents the serial-to-parallel case.

CLK

Q0   Q1   Q2   Q3

SERIAL
IN   D   Q   D   Q   D   Q   D   Q
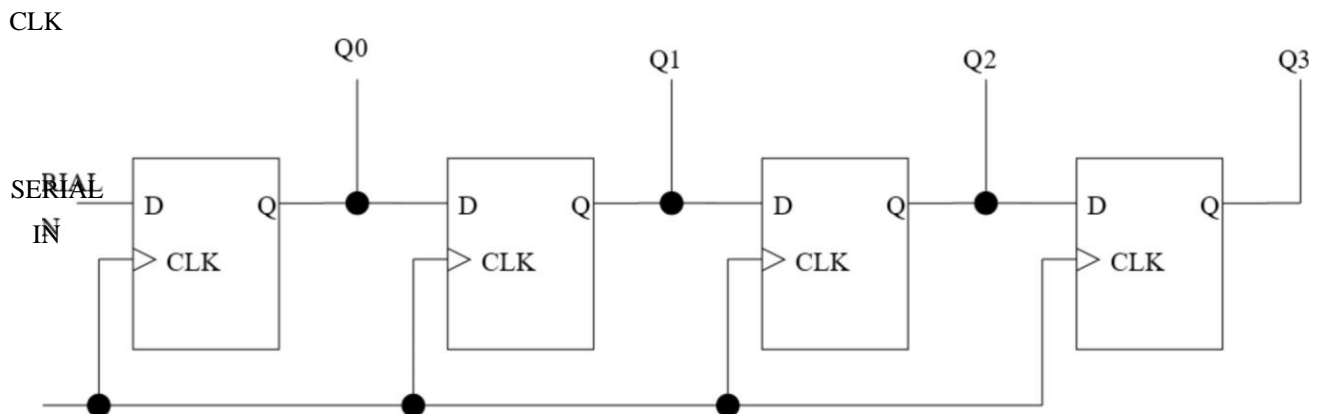   CLK   CLK   CLK   CLK

Figure 24: 4-bit shift register.

We will discuss several examples of shift registers a few lectures hence.

5 Analog/ Digital Conversion

In this section we discuss the important topic of analog to digital conversion (often written A/D), and digital to analog conversion (D/A). On one hand, most electrical measurements are intrinsically analog. take advantage of the great capabilities available for digital data storage, processing, and computation, on the other hand, requires the conversion of analog to digital. Hence, analog to digital (A/D) conversion techniques have become extremely important. A great deal of technical effort has gone into producing A/D converters (ADCs) which are fast, accurate, and cheap. D/A converters (DACs) are also very important. For example, video monitors convert digital information generated by computers to analog signals which are used to direct the electron beam at a specified portion of the monitor screen. DACs are conceptually simpler than ADCs, although it is diffcult in practice to

23

build a precise DAC. We will discuss D/A conversion before A/D. But first we go over some underlying ideas.

5.1 A/D Resolution

First of all we should keep in mind that there are several different schemes for encoding analog information as bits, depending upon what is required by a particular application. One extreme is that of encoding the complete analog signal in as much detail as possible. For example, a musical instrument produces an analog signal which is readily converted to an analog electrical signal using a microphone. If this is to be recorded digitally, one naturally would choose to digitize enough information so that when the recording is played back, the resulting audio is not perceived to be significantly different from the original. In this case the analog signal is a voltage which varies with time, V (t).

At any time to, V (to) can be sampled and converted to digital. 'i'he analog signal must be sampled for a finite time, called the sampling time, At. One may guess that it is necessary to sample the analog signal continuously, with no gaps between consecutive samples. 'l'his turns out to be overkill. 'l'he Nyquist Theorem states that if the maximum frequency of inerest in the analog input is fmax, then perfect reproduction only requires that the sampling frequency fs,unp be slightly greater than twice fmax 'l'hat is,

$$f_{\text{sam}} > 2\text{fmax}$$

For example, for audio signals the maximum frequency of interest is usually 20 kHz. In this case the input analog must be sampled at a little over 40 kHz. In fact, 44 kHz is typically used.

Alternatively, it might not be of interest to represent the entire analog input digitally. Perhaps only one feature of the analog signal is useful. One example is "peak sensing," where one samples and digitizes the input only at the instant where an instrument's output achieves a maximum analog output. Or one may average ("integrate") an input signal over some predefined time, retaining only the average value to be digitized.

For any of these sampling schemes, there remains the issue of how many bits are to be used to describe the sampled signal V (to). 'l'his is the question of A/D resolution. We need a standard definition of resolution. Let's say, for example, that we choose to digitize the input using 12 bits. 'l'llis means that we will try to match our analog input to 1 of $2^{12} = 4096$ possible levels. 'l'his is generally done by ascribing a number from 0 to 4095. So, assuming our ADC works correctly, the digital estimate of the analog input can, at worst, be wrong by the range of the LSD. On average, the error is half of this. 'l'his defines the resolution. Therefore, for our 12-bit example, the resolution is 1/(2 • 4096), or a little worse than 0.01%.

5.2 D/A Conversion

'l'he basic element of a DAC is the simplest analog divider: the resistor. First, we need to review the two important properties of a,n operational amplifier ("op-amp") connected in the inverting configuration. 'l'his is shown in Fig. 25. 'l'he two important properties are

1. 'l'he   input is effectively at ground. ("virtual ground")

2. 'l'he voltage gain is G V out /Vin —L/ RI. An equivalent statement is that for a current at the — input of Iin Vin/R1, the output voltage is Vout GVin$= -R_2$

—VinR2/R1. Sometimes this is written in the form Vout glin, where g is the transconductance, and g —R2 in this case.
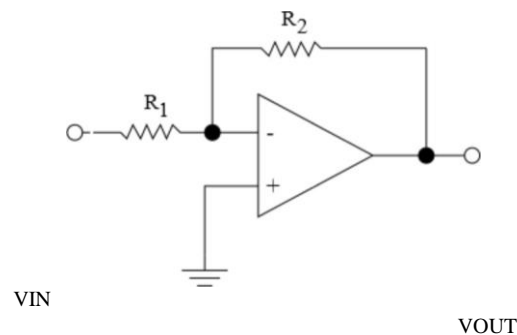

Figure 25: Inverting op-amp configuration.

ꞌlꞌhe basic idea of most DACs is then made clear by the 4-bit example illustrated in Fig. 26. ꞌlꞌhe input 4-bit digital signal defines the position of the switches labelled ao—ag. A HIGH input bit would correspond to a switch connected to 1.0 V, whereas a LOW connects to ground. ꞌlꞌhe configuration in the figure represents a binary input of 1010, or 1010. Since the virtual ground keeps the op-amp input at ground, then for a switch connected to ground, there can be no current flow. However, for switches connected to 1.0 V, the current presented to the op-amp will be 1.0 V divided by the resistance of that leg. All legs with HIGH switches then contribute some current. With the binary progression of resistance values shown in the figure, the desired result is obtained. So for the example shown, the total current to the op-amp is I 1.0/ R + 1.0/(4R) 5/(4R). ꞌiꞌhe output voltage is

$$V_{out} -RI\ 5/4\ 1.25V$$

When all input bits are HIGH (1111 = 1510), we find Vout 15/8 V. A simple check of our scheme shows that

$$(5/4)/(15/8) = 2/3 = 10/15 = 1010/1111$$

as expected.

5.2.1 The R-2R Ladder

ꞌlꞌhis represents a rather minor point, although it is an interesting idea. ꞌlꞌhe "R-2R ladder" is of practical interest because it uses only two resistor values. Since it is difficult to accurately fabricate resistors of arbitrary resistance, this is beneficial. ꞌlꞌhe two resistances of the R-2R
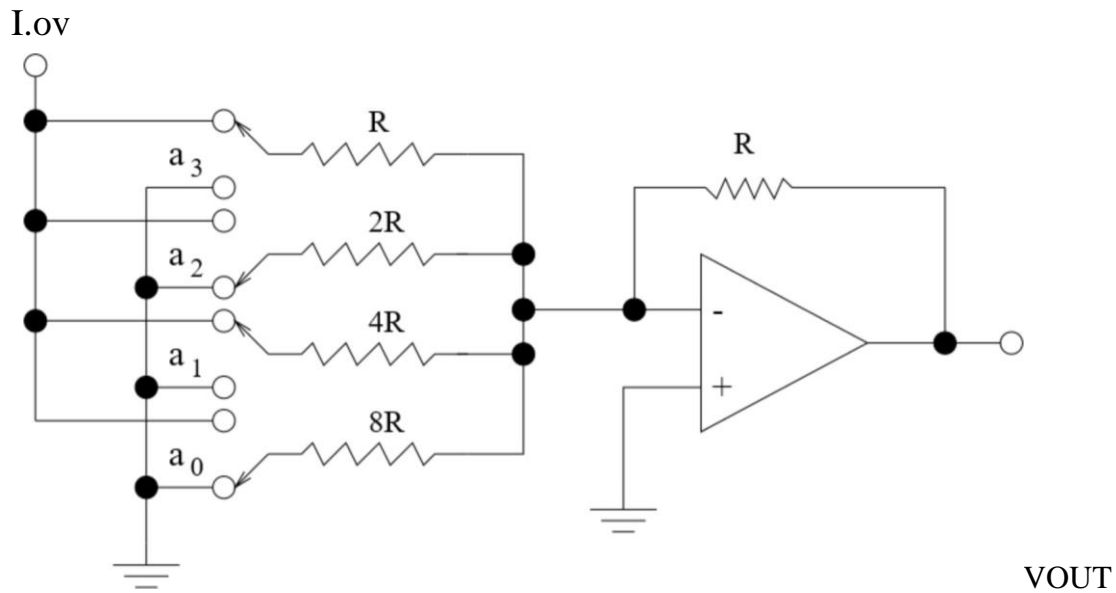
I.ov

Figure 26: Example 4-bit DAC scheme.

are to be contrasted with the scheme represented by the circuit of Fig. 26, which employs as many resistance values as there are bits. 'I'he idea behind the R-2R ladder hinges on noticing the pattern of equivalences represented by Fig. 27, which can be used to replicate an arbitrarily long ladder, and hence handle in arbitrary number of bits.
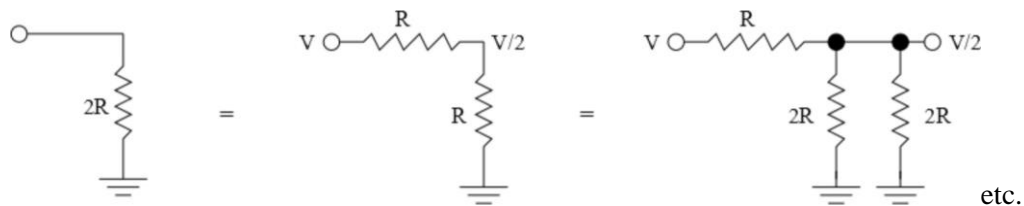

Figure 27: Principle of the R-2R ladder. 'I'he rightmost 2R resistor can be indefinitely relicated with this equivalent circuit.

5.3 A/D Conversion

ADCs fall into 3 general types of technique:

(1) parallel encoding (flash):   fast; limited accuracy

(2) successive approx. (feedback): med. fast; good accuracy

(3) single or double slope: slow; best potential accuracy

All of these techniques use a device known as a comparator. 'I'his was discussed in 431/531 and in the text Chapters 4 and 9. Here, we will not discuss how comparators work, but we do need to know what they do. 'I'here are many makes of comparators. We will use the model LM311 in lab. Figure 28 shows a comparator schematically. Internally, the comparator can be thought of as a fast, very high-gain differential amplifier ("A") with

inputs "+" and "−."We can put a "threshold voltage" at the input. Call it Vth. 'l'he circuit input Vin is connected to the "+" input. When Vin > Vth, the comparator amplifies this difference until the output reaches its largest possible value, which is determined by the connection through the pull-up resistor. In the configuration shown here, as well as in Lab 5, the 1 kQ pull-up resistor is connected to +5 V. (Note that while +5 V is convenient for many digital circuits, it is possible to use other values, such as +12 V.) When Vin < Vth, the output swings the other way. 'I'llis level is usually determined by a connection to one of the comparator pins. Here, it is ground.
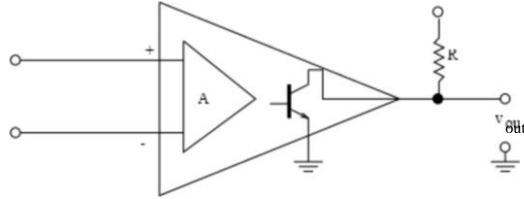

Figure 28: Comparator.

Hence, the comparator represents a one-bit ADC. When the analog input exceeds the pre-defined threshold, the output goes to digital HIGH, and when the input is less that the threshold, the output goes to digital LOW.

5.3.1 Flash ADCs

In this scheme, the input is fanned out in parallel to several comparators with monotonically increasing thresholds. 'I'he pattern of comparator outputs is then analyzed by some combinational logic (i.e. gates) to determine the output. 'I'llis technique is called flash (or parallel) encoding. We exemplify the flash ADC scheme with the 2-bit ADC shown in Fig. 29. With n 2 bits, we need to define 2^{71} 4 possible states. 'l'hese states represent 4 separate intervals. 'I'he analog input will fall into one of these intervals, and we will encode this assignment with the 2 bits. Defining the boundaries of 2^{7b} intervals requires 2^{7b} — 1 comparators, with the threshold of each comparator set to the appropriate boundary voltage.
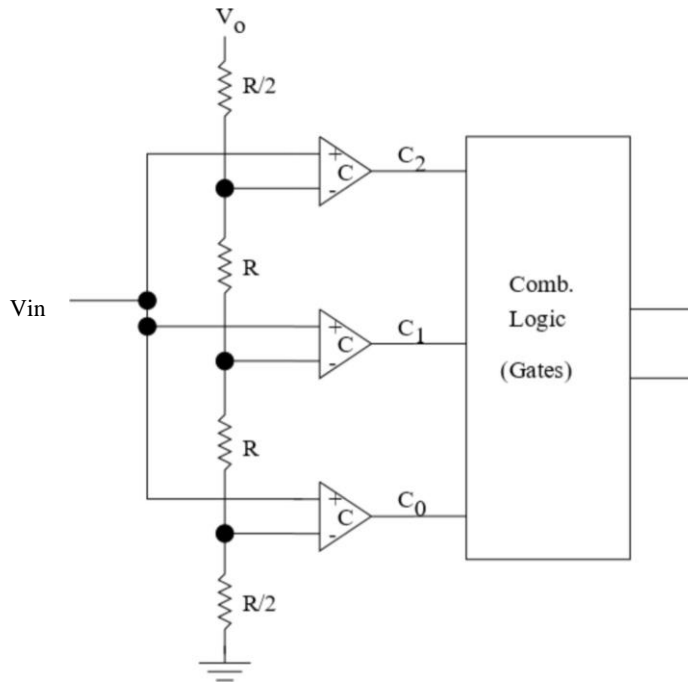
Figure 29: Schematic of a 2-bit flash ADC.

Let's go through a concrete example. Assume that our FADC circuit is designed to handle analog voltage input signals in the range —0.5 to 3.5 V. Thus, we have a 4-volt total input range, with each interval spanning 1.0 V. Therefore, each state will have a maximum error, or resolution, of half the interval, or 0.5 V. (This is 4.0/(2 • Y), as we said previously in our definition of resolution.) So an input which is in the range 2.5—3.5 V will give a HIGH output only to comparator output C2, and our digital estimation will correspond to 3.0 V. Hence, the threshold for the upper comparator (its input) should be set at 2.5 V. Similarly for the remaining comparators we work out the values which are given in the table below, where Vest is the digital estimate which corresponds to each state.

Vin range	Comparator	Threshold	vest		QIQ()
2.5-3.5 v		2.5 v	3.0 v	111	11
1.5-2.5 v		1.5 v	2.0 v	110	10
0.5-1.5 v		0.5 v	1.0 v	100	01
-030.5 v			0.0 v	000	oo

Using Ohm's and Kirchoff's Laws, we arrive at the resistance ratios shown in Fig. 29 in order to achieve the desired comparator thresholds. All that remains is to determine the gate logic to convert the pattern of comparator outputs to a 2-bit digital output. Generalizing from the above, we see that we have agreement with our previous statements: For n-bit ADC, we require $2^{7L} — 1$ comparators, and the resolution is $AL/2^{71+1}$ where AV is the full range of analog input.

5.3.2 Successive Approximation ADCs

This technique is illustrated by Fig. 30, which is also the one given for Lab 5. It uses a digital feedback loop which iterates once on successive clock cycles. The function of the

28

successive approximation register, or SAR, is to make a digital estimate of the analog input based on the I-bit output of the comparator. The current SAR estimate is then converted back to analog by the DAC and compared with the input. The cycle repeats until the "best" estimate is achieved. When that occurs, this present best estimate is latched into the output register (written into memory). By far the most common algorithm employed by SARs is the binary search algorithm. This is the one used by the SAR in Lab 5, and is illustrated in the example in the next secion.
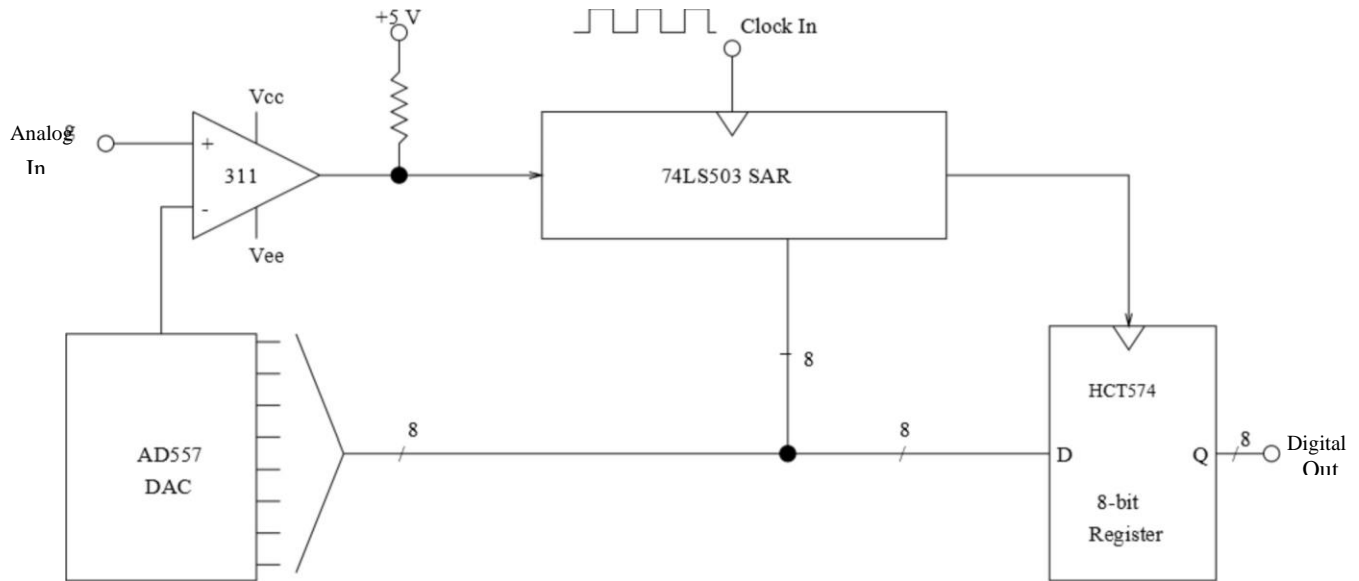

Figure 30: Scheme for 8-bit successive approximation, or feedback, ADC.

5.3.3 Binary Search Example

In this example we will see the binary search algorithm in action. The binary search algorithm can be summarized with the following words: Go to the midpoint of the remaining nonexcluded range. In our example, we assume an 8-bit ADC with an expected input voltage range of 0 to 10 V. So, naturally we choose the digital output to be 00000000_2 0 when

the input is 0 V, and $11111111_2 = 255$ when the input is 10 V. Hence, the LSB represents a voltage step AV 10/255 39.22 m V.

Let the input voltage be some arbitrary value, 7.09 V. Now let's see how the algorithm works. Translating the words for the algorithm, written above, to what the SAR actually does is straightforward. The SAR always outputs one of two results, depending upon whether the output from the comparator was TRUE or FALSE. More precisely, the comparator will issue a HIGH if the current estimate is too small compared to the actual input, or a LOW if it is too big. The SAR then does the following:

1. If estimate too small, add 1 to MSB—(n + 1) ; or
2. If estimate too big, subtract 1 from MSB—(n + 1) .

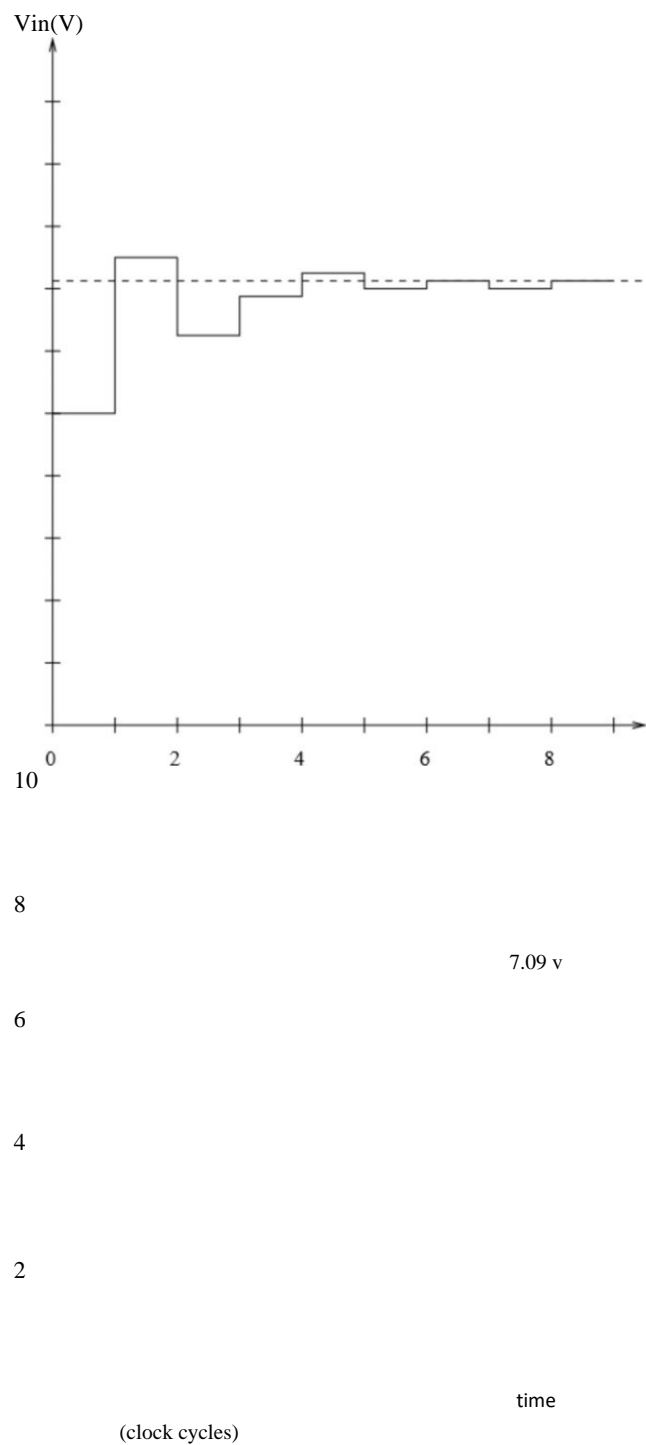where n is the current clock cycle (see table).

Figure 31: Binary search algorithm in action. ᵗiᵗhe analog input is 7.09 V. 'l'he digital estimate for each clock cycle is represented by the solid line, and corresponds to the value of Vest in the table below.

Clock Cycle, n	SAR Bits	SAR Bit Sum	vest (V)	comp. decision

		127	4.98	too
		127 64 =	7.49	small
		191 -32	6.24	too big
		159 + 16	6.86	too
		175 8 =	7.18	small
		183 ¯ 4 _	7.02	too
		179 2	7.10	small
1	01111111	181 1	7.06	too big
2	10111111	191		too
3	10011111	159		small
4	10101111	175		too big
5	10110111	183		too
6	10110011	179		small
7	10110101	181		too big
	10110100	180		too
				small

The binary search algorithm is guaranteed to find the best possible estimate in a number of clock cycles equal to the number of bits. In the example above, the best estimate was actually determined on the seventh clock cycle (n 6). But since the input value was between the digital estimates 180 and 181, there was no way for the ADC to determine which estimate was closer to the actual input value (without adding one more bit). Since the input can fall anywhere within 180 and 181 with equal likelihood, there should be no bias introduced with this method due to systematically choosing a digital estimate which is too small or too big. 'l'his is the desired outcome.

'l'he binary-search algorithm is fast and efficient, and also has the advantage that it completes its estimation in a well determined number of clock cycles. Hence, the final digital result can always be latched after n clock cycles, where n is the number of bits. (Many ADCs actually wait one additional clock cycle in order to guarantee that bits have settled, are latched properly, and are reset for the next input.

5.3.4 Single/Dua1 Slope ADCs

'l'hese techniques are slower than flash or successive approximation, but in principle can be quite accurate. 'l'he improved accuracy is for two reasons, because time, which is robustly measured using digital techniques, is used as the measured quantity, and because there is some immunity to noise pickup, especially for the dual slope case.

'l'he single slope technique is illustrated in Fig. 32, which is taken from Figure 9.54 of the text. 'l'he device near the input and the capacitor is an FET' transistor which is used as a switch. When the input to the FET' gate, which comes from the Q output of the D-type flip-flop, is LOW, then the FET' is switched off, and it draws no current. However, when Q goes HIGH, the FET' pulls the + input of the comparator to ground, and holds it there. 'l'he box marked "osc" represents a typical digital clock. 'l'he arrow within the circle connected to +Vcc is the symbol for a "current source" , which means that its output is a constant current, regardless of the impedance at its output (within reasonable bounds).

Figure 9.54. Sil

Figure 32: single-slope text.

Scheme for ADC, from
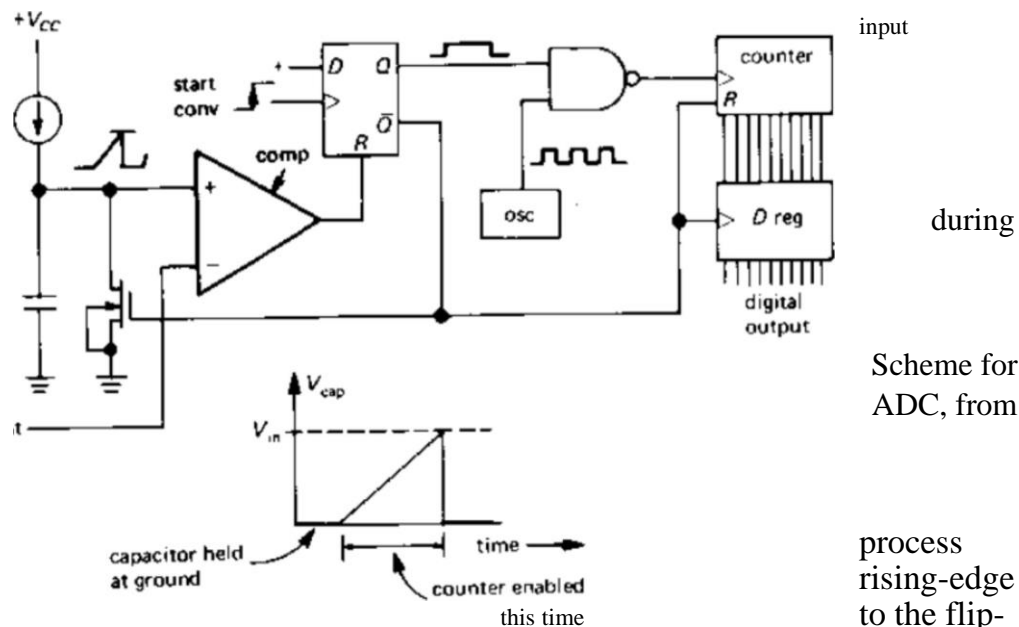
'l'he process begins when a rising-edge signal is sent to the flip-flop, for example from a debounced switch. Since the D input is HIGH, then Q goes HIGH. Hence the counter, no longer being held at reset by the flip-flop, begins counting. At the same time the FET' is switched off and a signal is sent to the — input of the comparator. Now we must analyze the nature of this signal.

'l'he voltage across a capacitor Vcap, is related to its stored charge by V Q/C, where C is the capacitance. Differentiating gives d Vc/dt I [C. Now, because of the current source, the right-hand side of this equation is a constant. Finally, since one side of the capacitor is at ground, then the comparator + input is just V . Hence, we can integrate our expression over a time interval At to give:

$$V+ vcap — (1 /C)At$$

Since I //C is a known constant, this equation allows one to convert the 1/+ input to a time At to be measured by the counter. 'I'his linear relation between 1/+ (= Vcap) and At is illustrated in the figure. 'l'he counter stops (is reset) and its final count stored in the register when V+ becomes equal to Vim, thus changing the state of the comparator. 'l'his also resets the flip-flop, thus returning the circuit to its initial state.

'l'he dual-slope ADCs work similarly, but with a two-step process. First, a capacitor is charged for a fixed time T with a current source whose current is proportional to Vin, I aVin, where a is the constant of proportionality. Hence, is proportional to T'. Vcap aVinT/C. 'i'he capacitor is then discharged at constant current I/ and the time At to do so is measured. 'I'herefore,

$$At [C/f'] [OT/C] V_{iu} = \beta$$

where 13 = 07/1/ is a known constant.

,l'his technique has two advantages compared with single-slope. First, we see from the equation above that the result is independent of C. 'I'his is good, as precise capacitance values are difficult to fabricate. Second, the integration of the input voltage in the charge-up step allows 60 Hz pickup noise (or other periodic noise) to be averaged to zero.

6 Counters, Registers, and State Machines Il

'l'he general scheme for a state machine is given in Fig. 33. It has n bits of memory, k inputs, and m outputs. It consists of a synchronous data register (lower box) which stores the machine's present state. A set of separate flip-flops can be used for this, as long as they are clocked synchronously. 'l'he logic in the upper box acts upon the current state, plus any inputs, to produce the machine's next state, as well as any outputs. Upon each pulse of the clock input CLK, the machine is moved from the present state to the next state. We will introduce this topic using counters as examples, then moving to more general applications. We will see, in fact, that the state machine prepresents a simple processor. 'l'he inputs can be generalized to be the processor program and the logic might be replaced by a random-access memory (RAM).
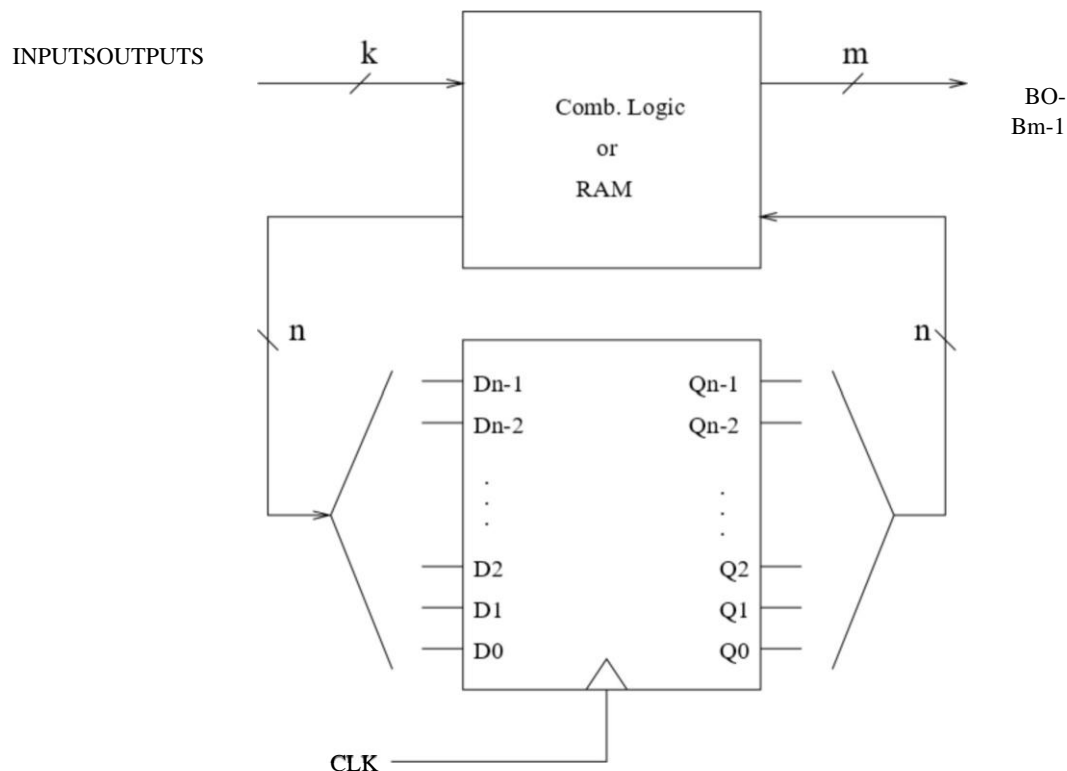

Figure 33: General scheme for state machine.

'l'he strategy for applying this scheme to a given problem consists of the following:

1. Identify the number of required states, C. 'l'he number of bits of memory (e.g. number of flip-flops) required to specify the m states is at minimum n log2(m).

2. Make a state diagram which shows all states, inputs, and outputs.

3. Make a truth table for the logic section. 'l'he table will have n + k inputs and n + m outputs.

4. Implement the truth table using our combinational logic techniques.

6.1 State Machine Introduction: Synchronous Counters

Counters implemented as state machines are always synchronous, that is the entire circuit is in phase with the clock. Recall that our previous "ripple" counters were asynchronous — logic was initiated at different times throughout the circuit. Synchronous systems are essential whenever a sequential system requires more than a very modest speed or complexity.

6.1.1 Example: Up/down 2-bit Synchronous Counter

A 2-bit counter requires 4 states, with each state corresponding to one of the 4 possible 2-bit numbers. Hence, 2 bits of memory are required. We will use 2 flip-flops (D-type) to implement this. 'l'he state diagram is given in Fig. 34. Each circle represents one of the states, and the arrows represent a clock pulse which offers a transition to another state (or possibly to remain at the present state). 'l'he 4 states are specified by the 2 bits of memory: A 00, B 01, C 10, D 11. Note that we are free to label the states as we choose, as long as they are uniquely specified. However, in this case it is easiest to choose labels which correspond to our desired outputs, that is the 2-bit binary sequence 00, 01, 10, and 11. Hence, these labels are equivalent to our desired outputs, call them QIQO, which are available at each state. (Note that the lettered labels A—D are superfluous; they could be omitted.)

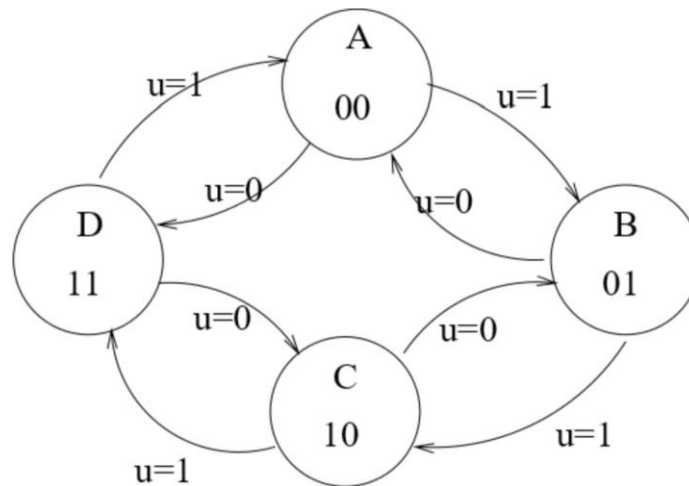
Figure 34: State diagram for 2-bit up/down synchronous counter.

Our processor has one input bit u, which programs the up-counting (u 1) or downcounting (u 0) functions. In this case, the state machine outputs are the two bits of the present state, QIQo, so we do not reproduce them in our truth table. 'l'he truth table for the logic is below.

Present State QIQO	Next State DIDO

	A	oo	B	01
1	B	01	c	10
		10	D	11
1	D	11	A	oo
O	A	oo	D	11
o	D	11	c	10
o	c	10	B	01
		01	A	oo

We can now envoke the logic as usual. We have 2 "outputs", Do and DI, which are to be evaluated separately from the truth table, or using a K-map, we see that

$$D_1 = \overline{u \oplus (Q_0 \oplus Q_1)} \; ;$$

6.1.2 Example: Divide-by-Three Synchronous Counter

Our state machine is supposed to count input pulses (input at the CLK) and set an output bit HIGH on every 3rd input pulse. Note that this could represent either a 2-bit (total) counter, or more generally the 2 least-significant bits of a many-bit counter.

We require 3 states, therefore we need 2 bits of memory (2 D-type flip-flops, for example).

These 2 flip-flops can describe 4 states, so we will have one "unused" state. A state diagram is shown in Fig. 35, with one way of labelling the states and output bit (called p) given.
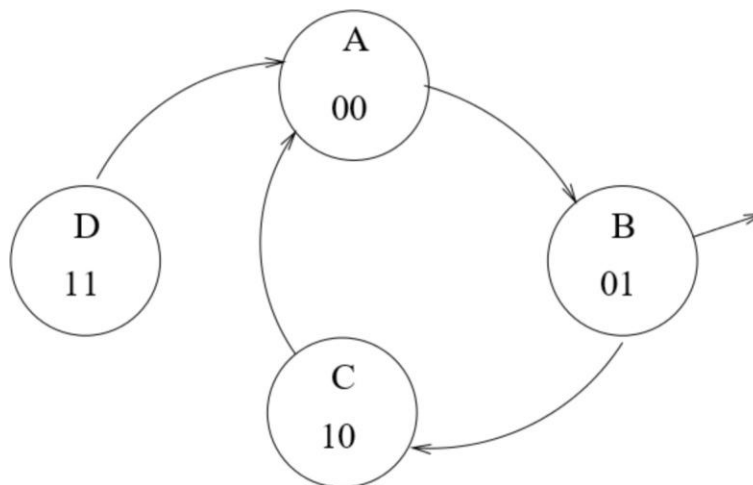

Figure 35: State diagram for a divide-by-3 synchronous counter.

The truth table for the combinational logic is below. It is important that the "extra state" D 11 be given an exit path, otherwise your processor may end up there upon power-up and remain stuck. (This effect has probably come to your attention with the "frozen" computer, which may require a reboot.) Also, note that we could have taken the output p from any of the states A, B, or C.

Present State	Next State	Output
QIQO	DIDO	

A	OO	B	01	0
	01	c	10	1
	10	A	oo	0
D	11	A	oo	0

What are the logic expressions for our 3 "outputs" of this truth table (DI, 1)2, and p) ? How would this be implemented with D-type flip-flops and logic gates? With JK flip-flops replacing the D-type?

7   Memories and Processors

7.1   Memory Terminology

We will not discuss the topic of data storage technologies per se. We are mostly interested here in the question of how data storage can be effectively organized. 'l'he important common element of the memories we will study is that they are random access memories, or RAM. 'l'his means that each bit of information can be individually stored or retrieved — with a valid input address. This is to be contrasted with sequential memories in which bits must be stored or retrieved in a particular sequence, for example with data storage on magnetic tape. Unfortunately the term RAM has come to have a more specific meaning: A memory for which bits can both be easily stored or retrieved ("written to" or "read from"). Here is a rundown on some terms:

- RAM. In general, refers to random access memory. All of the devices we are considering to be "memories" (RAM, ROM, etc.) are random access. 'i'he term RAM has also come to mean memory which can be both easily written to and read from. 'l'here are two main technologies used for RAM:

    1.)   Static RAM. 'rhese essentially are arrays of flip-flops. 'rhey can be fabricated in ICs as large arrays of tint flip-flops.) "SRAM" is intrisically somewhat faster than dynamic RAM.

    2.)   Dynamic RAM. Uses capacitor arrays. Charge put on a capacitor will produce a HIGH bit if its voltage V Q/C exceeds the threshold for the logic standard in use. Since the charge will "leak" off through the resistance of the connections in times of order 1 msec, the stored information must be continuously refreshed (hence the term "dynamic"). Dynamic RAM can be fabricated with more bits per unit area in an IC than static RAM. Hence, it is usually the technology of choice for most large-scale IC memories.

- ROM. Read-only memory. Information cannot be easily stored. 'l'he idea is that bits are initially defined and are never changed thereafter. As an example, it is generally prudent for the instructions used to initialize a computer upon initial power-up to be stored in ROM. 'l'he following terms refer to versions of ROM for which the stored bits can be over-written, but not easily.

- PROM. Programmable ROM. Bits can be set on a programming bench by burning "fusible links," or equivalent. 'l'his technology is also used for programmable array logic (PALs), which we will briefly discuss in class.

- EPROM. ROM which can be erased using ultraviolet light.

- EEPROM. ROM which can be erased electronically.

A few other points of terminology:

- As you know, a bit is a binary digit. It represents the smallest element of information.

- A byte is 8 bits.

- A "K' of memory is 2^{10} 1024 bits (sometimes written KB). And a megabit (MB) is 1K x 1K bits.

- RAM is organized into many data "words" of some prescribed length. For example, a RAM which has 8K 8192 memory locations, with each location storing a data word of "width" 16 bits, would be referred to as a RAM of size 8K >< 16. The total storage capacity of this memory would therefore be 128KB, or simply a "128K" memory. (With modern very large scale integration (VLSI) technology, a typical RAM IC might be 16 MB.

- Besides the memory "size," the other important specification for memory is the access time. T'llis is the time delay between when a valid request for stored data is sent to a memory and when the corresponding bit of data appears at the output. A typical access time, depending upon the technology of the memory, might be 10 ns.

7.2 Memory Configuration

As stated above, the term "memory" refers to a particular way of organizing information —by random access which is distinct from the less specific term "data storage." Figure 36 shows how an 8-bit RAM (8 x 1) is organized. (This is a very small memory, but illustrates the concepts.) Our RAM consists of three main components: an 8-bit multiplexer, an 8-bit demultiplexer, and 8 bits of storage. T'he storage shown consists of edge-triggered D-type flip-flops. Hence, this is evidently a "static RAM." (There is no fundamental reason for using edge-triggered flip-flops. 'l'hey could just as easily be level-triggered, like the simple "clocked" S-R flip-flop of Fig. 14.)
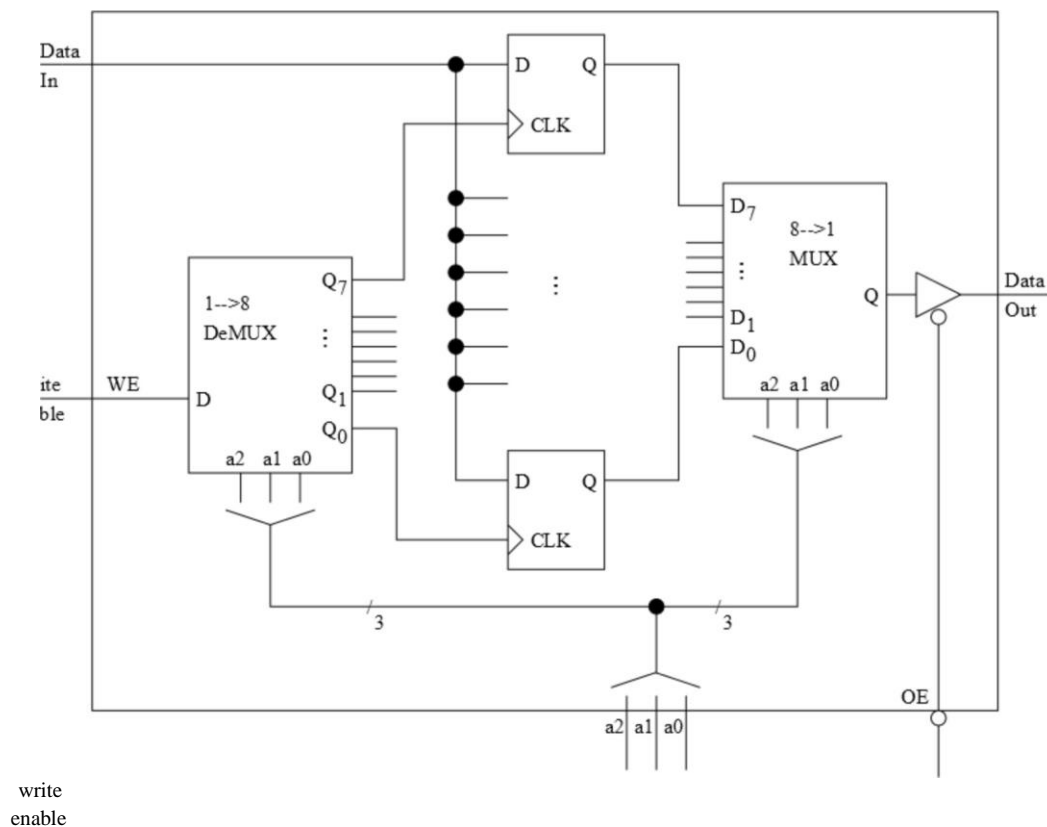
Figure 36: An 8 x 1 bit RAM.

Our example RANI has 6 external connections which are inputs (data in, write enable (WE), 3-state enable (OK), and 3 address bits (A a2a1ao), and has one output connection (data out), giving 7 external connections total, plus 2 for power/ground. write information to the RAM, one would supply a valid address, for example A 101. The data bit to be written to location 101 is to appear at the data input as either a logic HIGH or LOW signal. And to enable the writing into this bit, the WE signal must be asserted. This then appears at the Q5 output of the demultiplexer, and is passed on to the appropriate flip-flop, which stores the input data bit and passes it on to the Q5 multiplexer input.

To read data from our RAM, one asserts an address, so that the selected bit is sent to the MUX output and then the 3-state buffer. The purpose of the 3-state buffer is to ensure that no digital outputs are directly connected together, for example if our RAM output were connected to a data "bus," which in turn was connected to several other devices. Recall that the 3-state devices have outputs which are effectively disconnected if there is no enable signal. So if the output data connection of our RAM is connected to a data bus, then the OE signal must be coordinated with any other outputs also connected to the data bus. When it is OK to read data from the RAM (all other output devices are disconnected from the bus), the OE signal is asserted and the MUX output will appear at the RAM output.

One could of course also store the 8 bits of data directly to an 8-bit data register, rather than using the RANI configuration outlined above. In this case, the number of external connections is 17 (8 data in, 8 data out, and 1 clock), compared with the 7 of our RANI. For a more realistic case where the number of bits of memory n is much larger than our example, we generalize the above to arrive at $4 + \log_2 (n)$ external connections for the RAM, compared with $1 + 2n$ for the standalone register. Obviously for large n, the register is impractical, whereas the RAM remains reasonable. Actually, it is even somewhat better than this for the RAM case, since the number of external connections does not grow with the width of the stored data words. Hence, a RAM of size 1K >< 16 16 KB requires only 14 connections. This is to be compared with 32,001 connections for the register. Note that the RAM can only supply one bit at a time to the output. This may seem like a handicap, but is actually well matched to standard microprocessors.

7.3 A State Machine with Memory

For reference, our usual state machine configuration is shown again in Fig. 37. Now we consider the use of a memory with a state machine, as depicted in Fig. 38. A random access memory is used in place of the usual combinational logic. (A ROM has been specified, to emphasize that we are not changing the memory — once it is defined initially, it is only read from. The memory is used to conveniently encode the connection between present and next states.

To start with, let's assume a state machine with no external inputs or outputs. Then the state machine's present state (PS) becomes an address which is input to the ROM. The data word stored in the ROM at that address then corresponds to the next state (NS). This correspondence had been initially programmed into the ROM, just as the specific combinational logic in our old state machine had to be pre-determined. So if the PS as defined by the Q bits at the data register are, for example, 1001, then the ROM data word at address 1001 will be the NS which is then passed back to the register. When there are also external inputs, as there will be for most anything of interest, these are combined with the PS bits to form a longer address for the ROM. Similarly, any external outputs are combined with
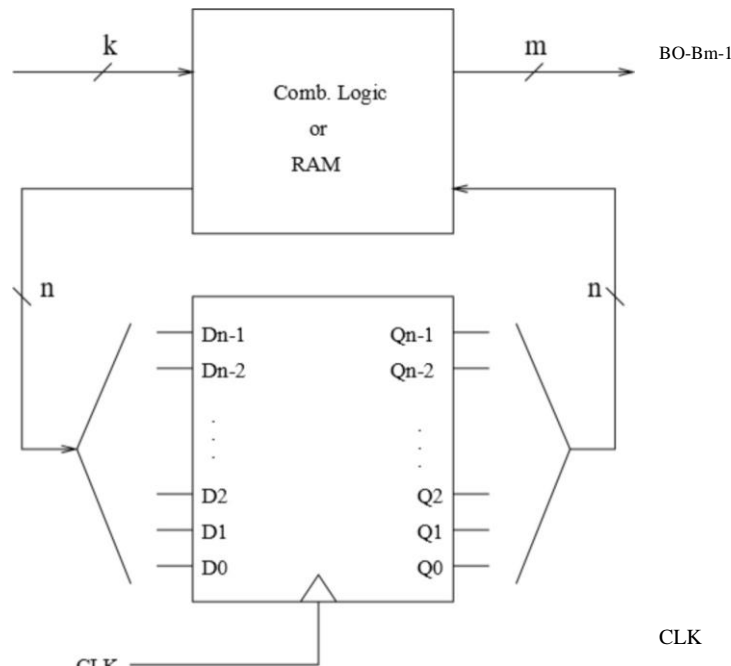
Figure 37: The standard state machine configuration.


the NS bits in the data word.

This should become clear with an example.

7.3.1 Example: Divide by 2 or 3 Counter

We will use a state machine with ROM, as in Fig. 38, to design a counter which either divides by 2 or by 3, depending upon the value of an external input bit p. This state machine will require 3 states, therefore we will need to describe 4 states, using 2 bits. We can label the states A 00, B 01, C 10, and D 11. Let p 0 be the divide by 2 case, and p 1 the divide by 3. The output bit r 1 when the present state is B. Otherwise r = 0. State D is normally unused. The truth table is below. The student should draw the corresponding state diagram.

	Present State QIQO		Next State DIDO		
o		A	B	01	1
o	01		A	oo	
o	10		A	00	
1	11		A	oo	
1		A	B	01	1
1	01	B	c	10	
1	10	c	D	11	
1	11		A	oo	

This ROM requires 3 address bits (2 for PS and 1 for input bit p), which corresponds to 8 locations in memory. Each location has a data word which has length 3 bits (2 for NS and
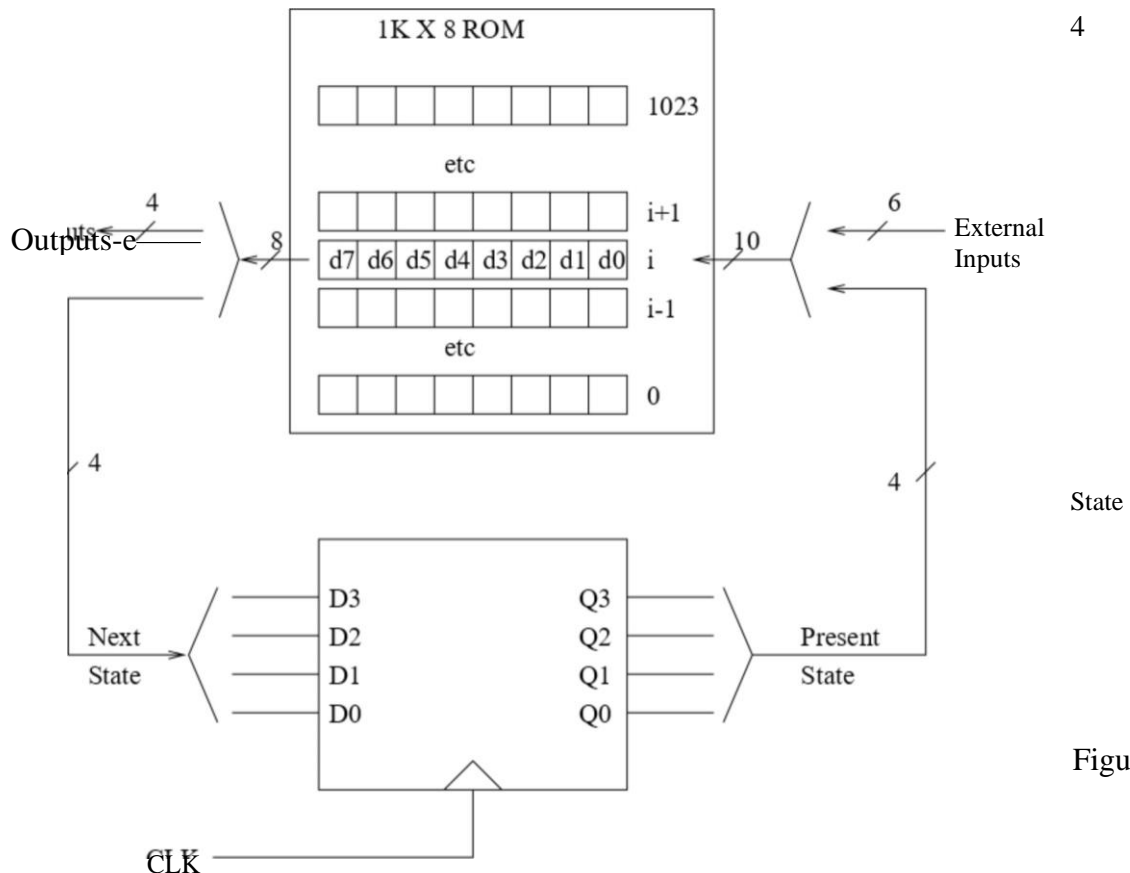
Figure 38: Troward a microprocessor: Replacing the combinational logic with a memory.

1 for the output bit r). Therefore, the size of this memory is 8 x 3, or 24 total bits. A very small ROM indeed. The programming of the ROM is very straightforward and can be read directly from the truth table above. We just need to set an encoding convention. Let the addresses be encoded as pQIQo and the data words as DI Dor. For example, let's look at the 5th row of the truth table. The address would be 100 and the data word at this address would be 010. The remaining bits of the ROM would be programmed in the same way. So one would initially "burn in" these bit patterns into the ROM and put it into the circuit. That's all there is to it. Of course if one were careful not to overwrite the memory, or if an evolving logical pattern were required, then a RAM could be used instead of the ROM.

7.3.2 Generalization to Microprocessors

A state machine with zero input bits can perform a counter-like function, but not more: its next state is limited to be a function only of the present state. A single input bit can be used to "program" the state machine to behave in one of two possible ways for each present state, as we discussed, for example, with the up/down counter of Section 4.4.1, or the example in the preceeding section. On the other hand, with n inputs, the machine can perform 2^{7b} different operations. So, for example, with n 8 the machine can perform one of 256 different operations on each clock cycle. This tremendous potential and flexibility. The input bits can themselves be sequenced — stored externally in a specific sequence which is then applied step by step to the state machine inputs on successive clock cycles. Such a stored sequence of operations is a program and the 256 operations represent the programming operations. In Fig. 38 we have essentially configured a simple microprocessor. The inputs

and outputs would need to be connected to buses (via 3-state buffers where appropriate), which in turn are also connected to memories which store the program and any output or input data. 'I'he buses would also be connected to various input/output devices, mass storage devices, etc.