Introduction to Finite State Machines

Introduction to Finite State Machines

Basically stated, a Finite State Machine (FSM) is a special case of a sequential machine, which is just a computational machine with memory. In FSM terminology, the "state" of the machine is reflected in the contents of the memory and is used to determine the output of the machine. In this, finite state machines and other sequential machines differ from simple combinational circuits in which the output depends only on the input at the time with no dependence on history or any information stored in memory. Simply put, FSM and all sequential machines have memory and combinational machines do not.

The inadequacy of pure combinational logic can be illustrated by considering a simple device – the soft drink machines in every campus building. Currently, the price of a soft drink is $1.00. A machine controlled by only combinational logic would have 2 options:

1) If a dollar bill or dollar coin is inserted, it would return a drink.
2) If a smaller coin is inserted, return the coin and indicate that it is not big enough.

Clearly, the behavior of the "combinational logic soft drink machine" is not acceptable. One expects the machine to have a memory to store the amount of money to be applied to the next purchase. What we want is for the soft drink machine to be controlled by **sequential logic**, specifically by a **FSM (Finite State Machine)**. In this example, the SDM (Soft Drink Machine) is initialized to a state called 0 – there has been no money deposited for a drink. When one places a quarter into the machine, it enters a state called 25 – there has been $0.25 deposited. In state 25, the machine waits for a money deposit in excess of $0.65 total before it will dispense a drink and possibly change. If the SDM accepts nickels, dimes, quarters, and dollar coins, it is easy to show that the number of states is finite: 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, and 100 – a total of 21 states.

Finite state machines can be seen in many other areas of life. A washing machine is easily modeled as a finite state machine, with the states being Fill, Wash, Rinse, Fill Again, and Spin. Admittedly, more sophisticated washers have more states, but the reader should get the idea. Traffic lights are also modeled as FSM, with the three traditional states being Red, Yellow, and Green. Again, realistic traffic lights have more states; but the number remains rather smalls, possibly being in the range of 10 to 15.

Finite state machines are studied in most courses in computer architecture. We note in passing that all stored program computers are theoretically finite state machines, although it is not profitable to view them as such. Consider a computer with 64 KB of memory – an extremely small value. This corresponds to 512K bits = 524, 288 bits. The memory alone of such a computer is modeled by a FSM with $2^{524288} \approx (10^{0.30103})^{524288} \approx 10^{157826.42}$ $\approx 2.6 \bullet 10^{157826}$ states – that is the number 26 followed by 157,285 zeroes. We might as well call that an infinite number. (NOTE: Every CS major should memorize a small set of numbers, including: log(2) = 0.30103, log(3) = 0.47712, the first ten powers of 2, and the fact that one year is approximately $3.16 \bullet 10^{7}$ seconds.)

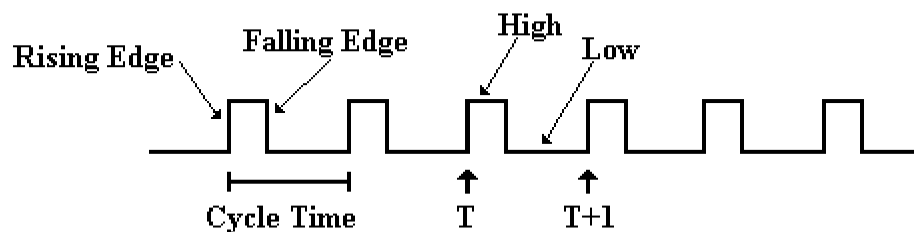
Copyright © 2005 by Edward Bosworth

The Clock

The most fundamental characteristic of synchronous sequential circuits is a system clock. This is an electronic circuit that produces a repetitive train of logic 1 and logic 0 at a regular rate, called the **clock frequency**. Most computer systems have a number of clocks, usually operating at related frequencies; for example – 2 GHz, 1GHz, 500MHz, and 125MHz.

Synchronous sequential circuits are sequential circuits that use a **clock input** to order events. Asynchronous sequential circuits do not use a common clock and, as hinted at above, are much harder to design and test. As we shall focus only on synchronous circuits, we immediately launch a discussion of the clock.

The following figure illustrates some of the terms commonly used for a clock.



We shall study **clocked flip-flops**, which are flip-flops that accept input only at fixed phases of the clock. There are three basic types of clocked flip-flops: those that accept input on the rising edge of the clock, those that accept input on the falling edge, and those that accept input when the clock is high. In our studies, we deal with **rising-edge flip-flops**.

The **clock input** is very important to the concept of a sequential circuit. At each "tick" of the clock the output of a sequential circuit is determined by its input and by its state. We now provide a common definition of a "**clock tick**" – it occurs at the **rising edge** of each pulse. We use T to represent the time at a clock tick and (T + 1) to denote the time at the next clock tick – the difference between the two is the **clock cycle time**. The inverse of the clock cycle time is the **clock frequency**. As an example, we consider a clock with a cycle time of 500 picoseconds = $500 \bullet 10^{-12}$ seconds = $0.5 \bullet 10^{-9}$ seconds. The clock frequency is thus 1.0 / ($0.5 \bullet 10^{-9}$ seconds) = $2.0 \bullet 10^{9}$ per second = 2.0 gigahertz.

By Q(T) we denote the state of a flip-flop at time T – this is basically its memory. We watch the state of the flip-flop change from Q(T) to Q(T + 1) as the clock ticks. The constraint on the design is that the state of the flip-flop changes after the input, thus we have a typical sequence as follows:

1) At time T, we have input (denoted by X) and state Q(T).
2) As a result of the input X and state Q(T), a new state is computed, This becomes available to the input only at time (T + 1) and so is called Q(T + 1).

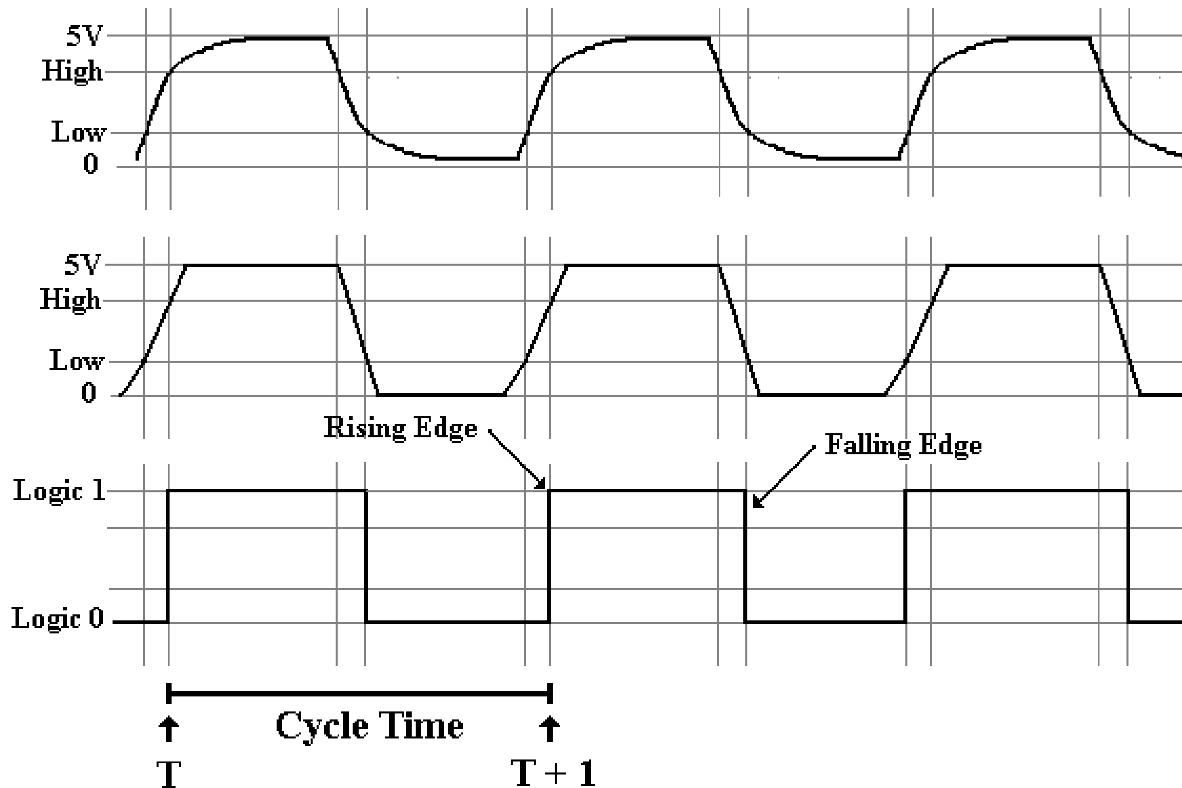
The fact that the new state, computed as a result of X and Q(T), is not available to the input of the flip-flop until the next time step greatly facilitates the design and analysis of the sequential circuits. We do not have endless feedback loops to worry about.

Copyright © 2005 by Edward Bosworth

Diversion: What the Clock Signals Really Look Like

The figure above represents the clock as a well-behaved square wave. This is far from the actual truth, as can be seen by examining the clock pulses with sufficient resolution. The following figure presents three views of the clock pulse train produced by a typical clock: a realistic physical view and two notations for approximating the clock.

In reality, the clock pulse is not square, but rises and falls exponentially. For those with mathematical interest, the clock falls in a function of the form $e^{-\alpha x}$ and rises with the form of the function $1 - e^{-\beta x}$, where $\alpha \approx \beta$. Use of this precise form does not gain us anything and leads to significant difficulties, so that unless we are troubleshooting at a very low level, we approximate the clock by either a trapezoidal wave or a square wave.



The trapezoidal wave form is used when it is important to emphasize the fact that the clock does take some time to rise and fall. One sees this form of clock representation often when examining timing diagrams for system buses. The square wave is a further abstraction of the real electrical form of the wave; fortunately it is quite often an adequate representation. The square wave representation remains at logic 0 until the real electrical clock crosses the threshold for logic high (about 2.5 volts) at which time the square wave jumps to logic 1. The square wave remains at logic 1 until the real electrical clock signal crosses the threshold for logic low (about 0.8 volts) at which time the square wave goes to logic 0.

In this course, we shall never have to worry ourselves with the actual electrical representation of a clock and seldom shall worry about the trapezoidal representation.

Copyright © 2005 by Edward Bosworth

In closing this introduction to the topic, we mention that the finite state machine approach should be considered a theoretical model that can be applied to many physical machines. As is often the case, there are cases (such as the above-mentioned computer) in which the model can be theoretically, but not usefully, applied. We advocate the use of FSM models when we find that these models help us understand some feature being studied. We do not advocate creating such models just to show it can be done.

Terminology Used in Discussing FSM

There are three terms used in these notes to describe a finite state machine. These terms are "**state diagram**", "**state table**", and "**transition table**". A finite state machine with N states will usually have its states numbered from 0 through (N – 1) inclusive. In the state diagram and state table we watch the FSM as in makes transitions from one state to another. We consider transitions between the states as being caused by external input or by the "ticking" of an internal clock. As an example of the latter circuit, consider a digital clock, which is driven by an input that presents one pulse per second. Each pulse causes the clock to change its state, which represents the time as displayed by the clock.

There are many ways to classify finite state machines; one being whether or not the FSM accepts input. If the FSM accepts input, we shall treat it as being sampled by the transition due to the internal clock. Some FSM, such as counters, do not accept input and only serve to count clock pulses. As noted above, electronic clocks can be viewed as a counter.

Modulo-N Counters

Strictly speaking, counters cannot be viewed as finite state machines, since there are an infinite number of integers. We avoid this difficulty by discussing modulo counters. Recalling that modulo-N arithmetic refers to the remainder from dividing a non-negative integer by N, we state the fact that the set of modulo-N integers is {0, 1, 2, … (N – 1)}, a finite set representable by a FSM. In class-work we normally set $N = 2^K$ (for $K > 0$) and so might have a modulo-4 counter cycling through the states 0, 1, 2, and 3. We now present the state diagram for a modulo-4 counter, more precisely called a modulo-4 up-counter as it only counts in the "up" direction. We discuss up-down counters later.

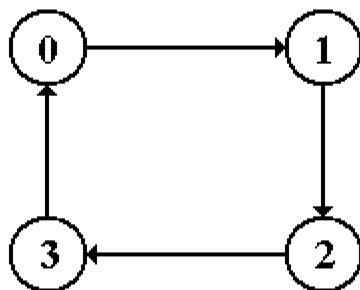


Figure: State Diagram for a Modulo-4 Counter

Many students will recognize the above figure as a directed cyclic simple graph. The fact that the graph is simple (that is – no edges from a vertex to itself) is just an artifact of this design and not true in general for all state diagrams. The directed edges in the graph represent transitions in the FSM, showing movement from one state to the next.

Copyright © 2005 by Edward Bosworth

State tables are merely a different presentation of the data found in a state diagram. State tables make an explicit presentation of the "**Present State**" (often denoted "**PS**") and the "**Next State**" (often denoted "**NS**") of the FSM. The state table of a modulo-4 counter is shown in the next figure. Recalling that $3 + 1 \equiv 0$ (modulo 4), there are no surprises here.

PS	NS
0	1
1	2
2	3
3	0

Figure: State Table for a Modulo-4 Counter

At this point, let's stop and reflect what the state diagram and state table are telling us. The FSM makes a transition from the Present State to the Next State at the occurrence of some event such as the arrival of a clock pulse. If the present state is 1, the next state is 2, etc.

The **transition table** is the binary equivalent of the state table and represents the same information in binary form. In order to create a transition table we must assign a binary number to each of the states; here the assignments are obvious: 0 = 00, 1 = 01, 2 = 10, and 3 = 11. Note that each state is represented by a 2-bit binary number. In general, the number of bits required to represent N states is obtained by solving $2^{B-1} < N \leq 2^B$, for B = bit count.

Here is the simplest presentation of the state table for the modulo-4 counter.

PS	NS
0 0	0 1
0 1	1 0
1 0	1 1
1 1	0 0

Figure: Transition Table for a Modulo-4 Counter

More often, transition tables are presented with additional information to simplify their use. In the following version, we show both the decimal and binary labeling of the states as well as the explicit representation of the binary numbers as Y_1Y_0.

PS		NS
	Y_1 Y_0	Y_1 Y_0
0	0 0	0 1
1	0 1	1 0
2	1 0	1 1
3	1 1	0 0

Figure: Transition Table with Extra Information

Note: The use of the term "transition table" may be unique to this instructor.

Copyright © 2005 by Edward Bosworth

Comparing the state diagram, state table, and transition table for the modulo-4 counter, we see that each contains a slightly different presentation of the same material. Each of the representations is suited for a particular purpose; the transition table being well suited to the design of an implementation of the FSM using flip-flops.

Output of a Modulo-4 Counter

In general, the output of a modulo-N counter is the binary number used to represent the states 0 through (N – 1); for the modulo-4 counter the output is the binary number denoted Y_1Y_0. The following figure shows an implementation of the modulo-4 counter using JK flip-flops. Note that the only input is labeled "clock", which represents a sequence of pulses to be counted, and that the output is copied directly from the two binary memory devices. Students who have taken a course in digital design will recognize the circuit; those who have not taken such a course should not worry about this additional information.

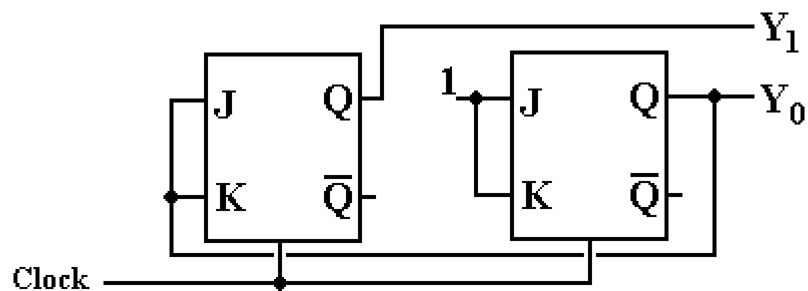


Figure: A Two Flip-Flop Implementation of a Modulo-4 Counter

Sequence Detectors

We now present sequence detectors as examples of finite state machines with input. The input to such a machine, denoted by the variable "X" is most of this author's work, is to be viewed as a sequence of binary numbers presented one bit at a time. Informally, one might say that the input X is sampled at every "tick" of the internal clock and a single bit produced.

A sequence detector is an example of a finite state machine that is different in two ways from the modulo-N counter just presented. We mention these now and explain in full just below.

1) It has input, usually denoted as X.
2) It has output associated with the transition between states.

When discussing sequence detectors, we must consider the number of bits in the sequence to be detected; normally we use the variable "N" to represent this count. It is easy to show that a finite state machine to detect an N-bit sequence should have N states. The reasoning for this last statement is based on two facts.

1) A FSM with fewer states can be built to detect the desired sequence, but it is always possible to construct several other sequences that it will mistakenly identify.
2) A FMS with more states can be built to detect the desired sequence, but it will not use all of its states.

Copyright © 2005 by Edward Bosworth

The sequence detector accepts a string of input denoted as X and has an output denoted as Z. The output Z goes to 1 upon the arrival of the last bit of the desired sequence. For example, consider a FSM designed to detect the sequence 11011. The input and output are

```
X = 11011
Z = 00001
```

We now present the state diagram of a simple 11011 sequence detector, comment on that design, and then explain how it was obtained from the specification of the problem. The more formal specification of the problem might be obvious, but we state it anyway.

Problem: The FSM is to output 1 if and only if the last five bits received, one bit at a time, were 11011.

Here is the complete state diagram for a 11011 sequence detector. Technically, it is a sequence detector that does not allow overlap. We explain the term "overlap" later.

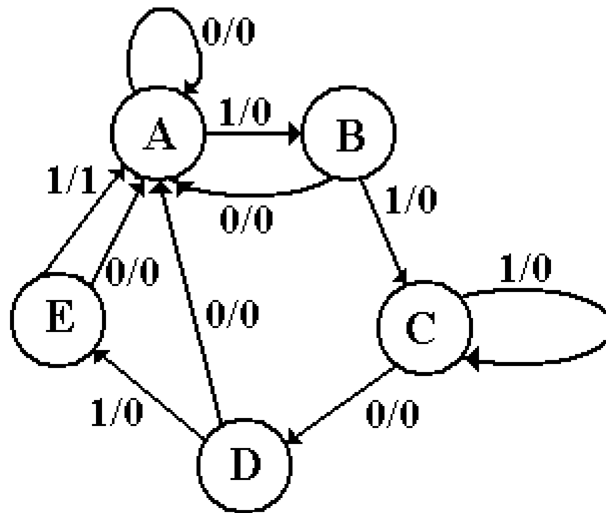


Figure: 11011 Sequence Detector With No Overlap

The reader will note a few things about the figure.

1) It is a graph on five nodes, labeled for some reason as A, B, C, D, and E.
2) It is a directed cyclic graph that is not simple. Specifically, vertices A and C have edges that begin and end on the same vertex.
3) There are two distinct edges from vertex E to vertex A.
4) The edges are labeled in the form "X/Z", so that the edge from vertex A to vertex B is labeled as "1/0", while one of the edges from vertex E to vertex A is labeled "1/1" and the other "0/0".
5) The artwork could use improvement.

This is a state diagram for a FSM in which the output is associated with the input. The label "1/0" means that at the given present state, a "1" input causes the transition indicated by the edge and that associated with that transition a "0" is output by the FSM. The "1/1" label on the edge from E to A indicates that if the FSM is in state E and gets a "1" as input, it outputs a "1" and moves to state A.

Copyright © 2005 by Edward Bosworth

Let's now build the state diagram of the state table beginning with the simplest version and continuing through a specific definition of each state to a complete version of the diagram. We begin with the fact that the FSM will have five states; these are labeled A, B, C, D, & E. By definition A is the "start state", indicating that the FSM is in this state before any bits of the sequence have arrived. State E is the last state; when the FSM is in this state it will output a 1 upon receipt of the final bit of the desired sequence.

We begin with the first step in a process designed to simplify the design of the FSM.

1) **Draw the state diagram showing only the transitions for the desired sequence.**

Here is the state diagram, omitting inputs that break the sequence.

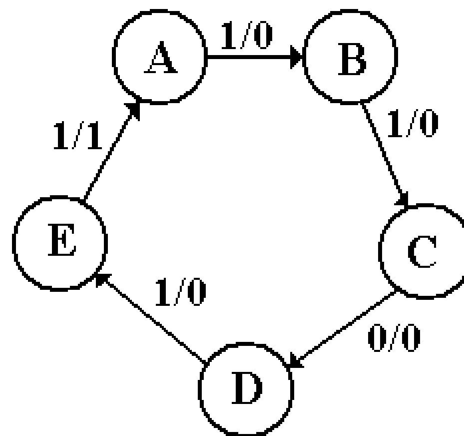


Figure: FSM Showing Only the Desired Sequence

Note that the transitions are labeled as "1/0", "1/0", "0/0", "1/0", and "1/1". If the FSM starts at state A and received the input sequence 11011, it will output 00001 and return to sate A, ready to check for another sequence. We now define the states in reverse order.

E	if the FSM is in state E, then receipt of a 1 will cause it to output a 1 and return to state A.
D	if the FSM is in state D, then receipt of the two bits 11 will cause it to output a 01 and return to state A.
C	if the FSM is in state C, then the receipt of bits 011 (read left to right) will cause the FSM to output 001 (read left to right) and return to state A.
B	if the FSM is in state B, then receipt of bits 1011 will cause it to output 0001 and return to state A.
A	if the FSM is in the start state, it requires the entire sequence 11011 to cause it to return to state A and output a 1 on the last transition.

One of the easier ways to describe the states in such a finite state machine is by use of the totally inappropriate verb forms "has" and "wants". In this context, we use the verbs to mean the following.

"Has"	What part of the sequence has been received to get to this state from state A.
"Wants"	What remaining bits are required to complete the sequence.

Copyright © 2005 by Edward Bosworth

The following table presents this definition of each of the states.

State	Has	Wants
A	Nothing	11011
B	1	1011
C	11	011
D	110	11
E	1101	1

With these definitions, we can now handle the inputs that break the sequence. The main goal in handling unexpected inputs is to determine the largest suffix of the unexpected sequence that can be applied toward the desired sequence. The **suffix** of a sequence is just the last one or more bits of the sequence, so that the sequence 11011 has the following suffixes.

Suffix of length 1 1
Suffix of length 2 11
Suffix of length 3 011
Suffix of length 4 1011
Suffix of length 5 11011.

Of course, every sequence has a suffix of length 0, that being the empty string that is hard to write on paper as it has no characters. We now examine each state.

A State A is the initial state. It is waiting on a 1. If it gets a 0, the machine remains in state A and continues to remain there while 0's are input.
B If state B gets a 0, the last two bits input were "10". This does not begin the sequence, so the machine goes back to state A and waits on the next 1.
C If state C gets a 1, the last three bits input were "111". It can use the last two of these 1's to be the first two 1's of the sequence 11011, so the machine stays in state C awaiting a 0. We might have something like 1111011, etc.
D If state D gets a 0, the last four bits input were 1100. These 4 bits are not part of the sequence and no suffix of these four bits will start the sequence, so we start over.
E If state E gets a 0, the last five bits input were 11010. These five bits are not part of the sequence and no suffix of these bits will start the sequence, so we start over.

With the explanation above, we generate the state diagram for the sequence detector as shown in the original figure. The FSM so described correctly identifies the desired sequence 11011 and correctly rejects any sequence not ending in the desired pattern.

Before presenting the state table, we note that the FSM will "accept" a longer string that ends in the desired bit pattern, so that it will accept "11001100110011011011", giving a 1 output on the input of the last 1. This fact illustrates an often confusing result from the theory of computation called the **pumping lemma**, basically that the desired pattern can be filled with a lot of interior "garbage" and still be recognized if it starts and ends properly.

Copyright © 2005 by Edward Bosworth

For a FSM with output associated with the input, we must expand the state table to include the output. As the FSM has input that determines the transition to the next state, the table must include columns for each possible variant of the input, here only X = 0 or X = 1.

Present State	Next State / Output	
	X = 0	X = 1
A	A / 0	B / 0
B	A / 0	C / 0
C	D / 0	C / 0
D	A / 0	E / 0
E	A / 0	C / 1

Figure: State Table for 11011 Sequence Detector without Overlap

The **transition table** for this FSM is a bit trickier to design than that for the mod-4 counter. Note that we need five states; solving $2^{B-1} < 5 \leq 2^B$ for B gives B = 3 by inspection; thus we need three bit integers to specify the state. But 3 bits will specify eight different items, so we have some flexibility; it can be shown that we have $\binom{8}{5} = \frac{8 \bullet 7 \bullet 6}{3 \bullet 2} = 8 \bullet 7 = 56$ different options for assigning binary numbers to the states. Considerations that arise from the desire to implement such a finite state machine using flip-flops dictate that state A be assigned the binary number 000, so that we can initialize the circuit by clearing all of the flip-flops. This still leaves us choosing from 7 numbers for 4 states; this can be done in $\binom{7}{4} = 35$ different ways. The simplest assignment is A = 000, B = 001, C = 010, D = 011, and E = 100.

There is a variant of the problem for which a different assignment of binary numbers to the states will yield a simpler solution. Consider the case in which overlap is allowed. In this case, state E on receiving a 1 outputs a 1 and returns to state C, because the last "11" in the sequence "11011" can be used as the first "11" in the next one. For such a circuit, the assignment A = 000, B = 001, C = 011, D = 100, and E = 101 remains understandable and allows for a significantly simpler circuit. Rather than prove the point, we show a variant of the transition table with these assignments and "wave our hands".

Present State	Next State / Output	
	X = 0	X = 1
A = 000	000 / 0	001 / 0
B = 001	000 / 0	011 / 0
C = 011	100 / 0	011 / 0
D = 100	000 / 0	101 / 0
E = 101	000 / 0	011 / 1

The simplification is due to the fact that all of the next states for X = 0 are even numbers (ending in binary 0) and all next states for X = 1 are odd numbers (ending in binary 1).

Copyright © 2005 by Edward Bosworth

Modulo-N Up-Down Counters

This chapter has presented finite state machines of two types.

1) Counters with no input (other than the clock) and no computed output.
2) Sequence detectors with input and output associated with the state transitions.

We now present another FSM with the goal of showing that it is possible for counters to have input other than the clock. This example is designed to reinforce the point that the only real difference between the two circuits is that one has output associated with the transitions. The FSM we now present is a modulo-4 up-down counter, with an input X that is used to select the counting direction. For X = 0, the FSM counts up and for X = 1, the FSM counts down. We use a state diagram to describe the counter.

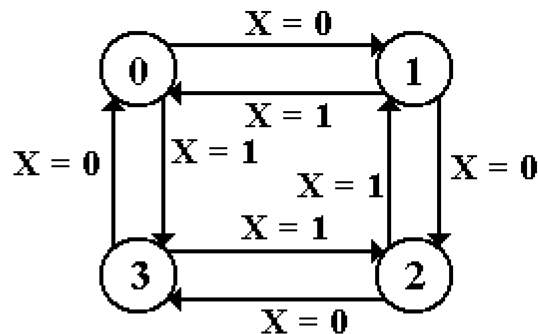


Figure: State Diagram for a Modulo-4 Up-Down Counter

As expected, the state table has two "Next State" columns, one for X = 0 and one for X = 1.

Present State	Next State	
	X = 0	X = 1
0	1	3
1	2	0
2	3	1
3	0	2

Figure: State Table for a Modulo-4 Up-Down Counter

Hint: In reading the state table and transition table for a down counter, it helps to read the table from the bottom up: PS = 3, NS = 2; PS = 2, NS = 1; etc.

The transition table for this FSM contains no surprises.

Present State		Next State	
		X = 0	X = 1
	Y_1 Y_0	Y_1 Y_0	Y_1 Y_0
0	0 0	0 1	1 1
1	0 1	1 0	0 0
2	1 0	1 1	0 1
3	1 1	0 0	1 0

Copyright © 2005 by Edward Bosworth