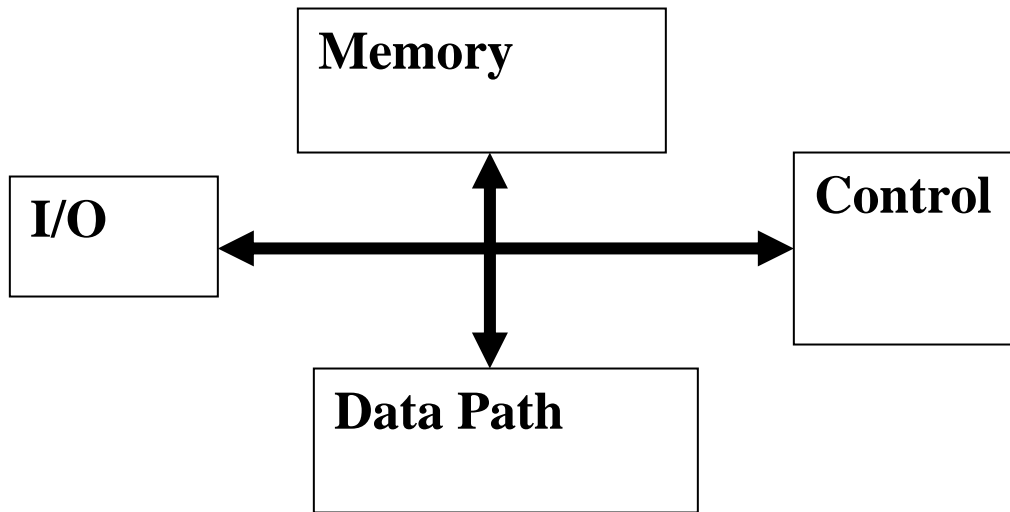


# Sequential Circuits

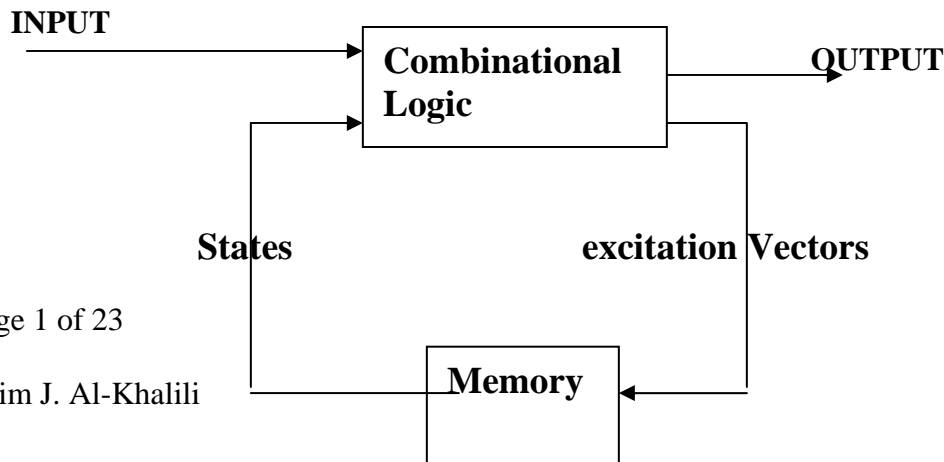
In the last few lectures we covered many data path operators. These operators have to be controlled for proper operation. In reference to the figure below, the controller is responsible for initializing and transfer of all data between, memory, data path and the I/O.



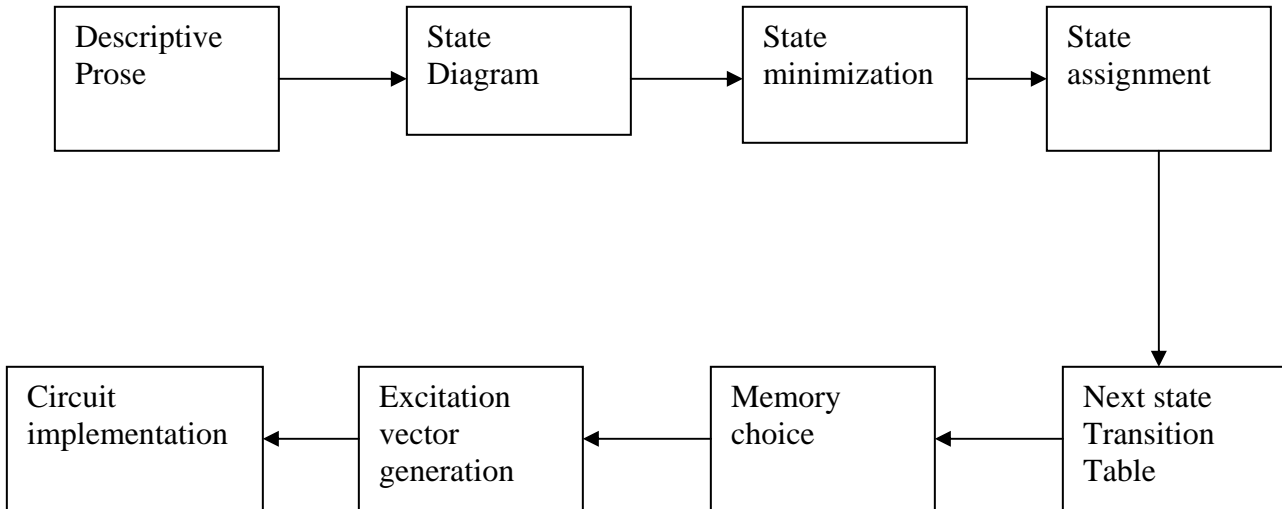
The Controller is usually implemented in Finite State Machine (FSM), Algorithmic State Machines (ASM) or microcoded.

## Finite State Machines

These circuits are synchronous and controlled by a master clock. The machine moves from one state to the next on the active edge of the clock. During each state, the machine uses the present state information and the current inputs to arrive at the next state. A general diagram of the FSM is shown below:



The starting point for designing the finite state machines is to translate the specification or the prose into a form that is an engineering description of the same. Usually state diagram is used to do this translation. Once this is done the rest of the design methodology can be automated. The design procedure then follows the path as shown in the diagram below:

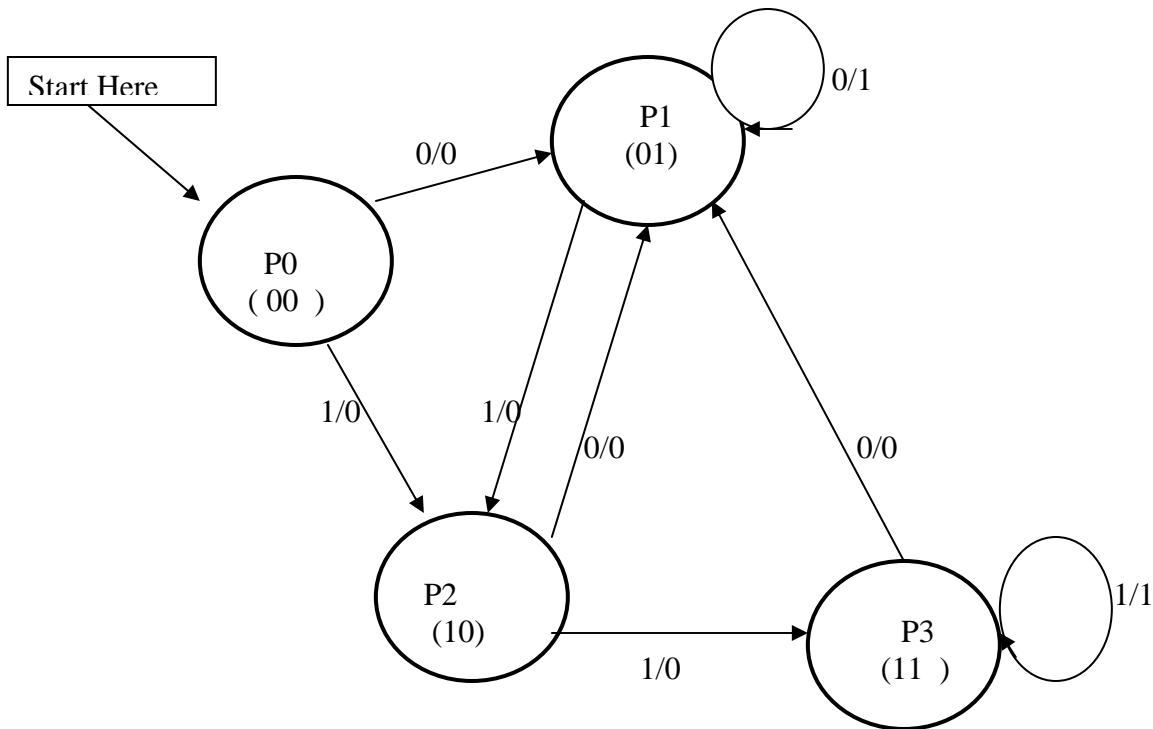


In the following examples we will show this procedure

## Example

Design a sequence detector that detects a sequence of 2 zeros or 3 ones on an incoming serial data line. Assume an asynchronous reset that initializes the machine. Let the input be  $x$ , and the output be  $z$ .

We have the following state diagram



State Assignment

State	y1	y0
P0	0	0
P1	0	1
P2	1	0
P3	1	1

Arbitrary  
Assignment

Present State	Next states and output			
	$y_1 y_0$		output, Z	
	X=0	X=1	X=0	X=1
0 0	01	10	0	0
0 1	01	10	1	0
1 1	01	11	0	1
1 0	01	11	0	0

From the table we can read Z immediately. This is  $Z = x'y_1'yo + x y_1 yo$

To determine the next states and the excitation table we draw the K-maps corresponding to each next state

$y_1, y_0$ \ x	0	1
	00	
01		1
11		1
10		1

$$y_1^+ = x$$

$y_1, y_0$ \ x	0	1
	00	1
01	1	
11	1	1
10	1	1

$$y_0^+ = x' + y_1$$

Using D flip flops then the circuit can be drawn following the next state and the output equations derived.

$D = y_1^+$                       standard D-Flip Flop next state equation

$$y_1^+ = x$$

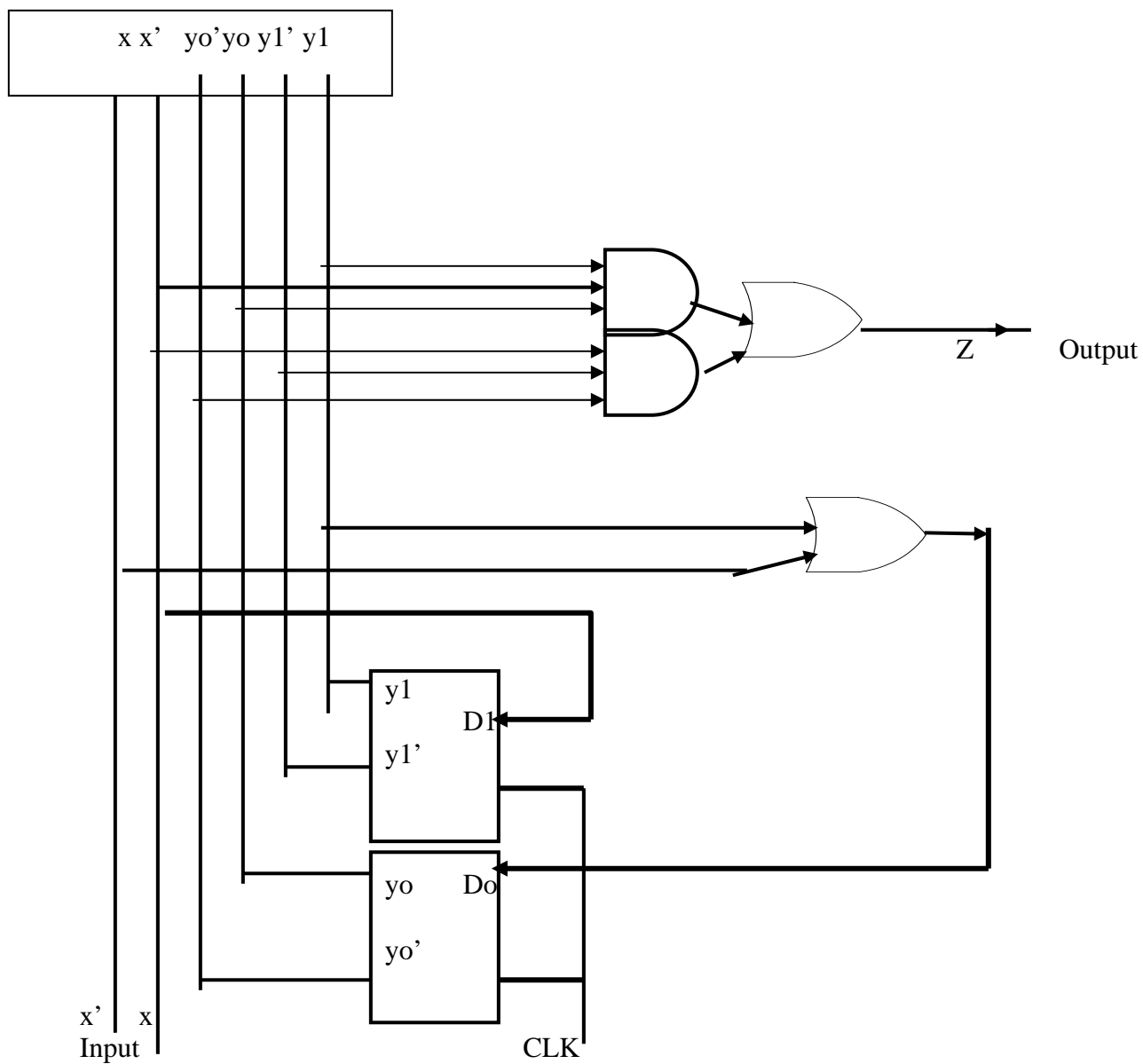
$$y_0^+ = x' + y_1$$

$$Z = x'y_1'yo + x y_1 yo$$

$$y1^+ = x$$

$$y0^+ = x' + y1$$

$$Z = x'y1'yo + x y1 yo$$

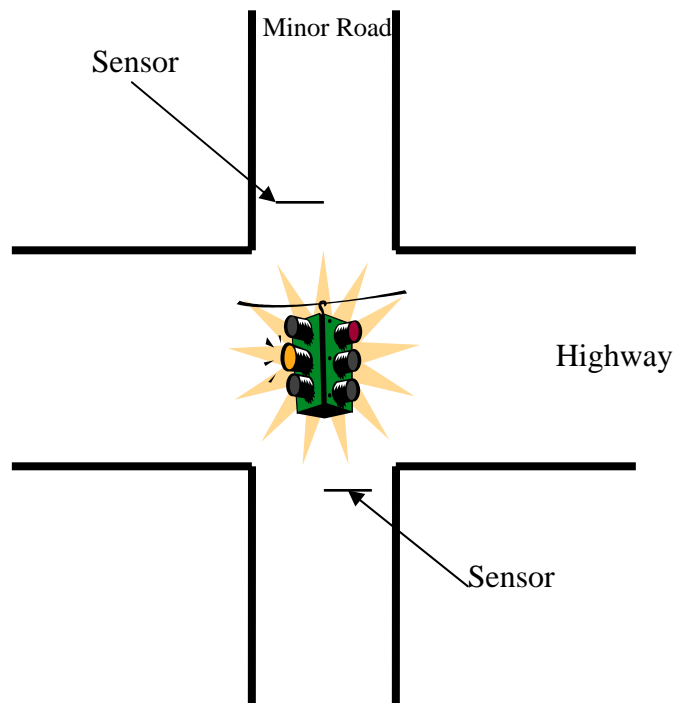


## Example 2

When a minor road crosses a highway, a traffic controller is installed to control the flow of traffic. Normally the highway is given the right of way and when there is a demand on the minor road then the highway is interrupted to give access to the minor road. You are asked to design the controller that meets the following specification.

The highway should be given the right of way, if any of the sensors on the minor road do not detect presence of a car or if the sensor does detect a car but an amount of time equal to or greater than  $\text{Timer long} = 30$  seconds has not elapsed since the last change.

If there was a car on the minor road and amount of time greater than  $\text{Timer long}$  has elapsed, then the traffic light should cycle through amber for  $\text{Timer short} = 3$  seconds and change to Red, while minor road changes to Green. The minor road now should have access of the road while there are cars but never more than  $\text{Timer long}$ . The minor road then should cycle back to red through a  $\text{Timer short} = 3$  second. While the highway cycles back to green



Initially we identify the system variables. These are

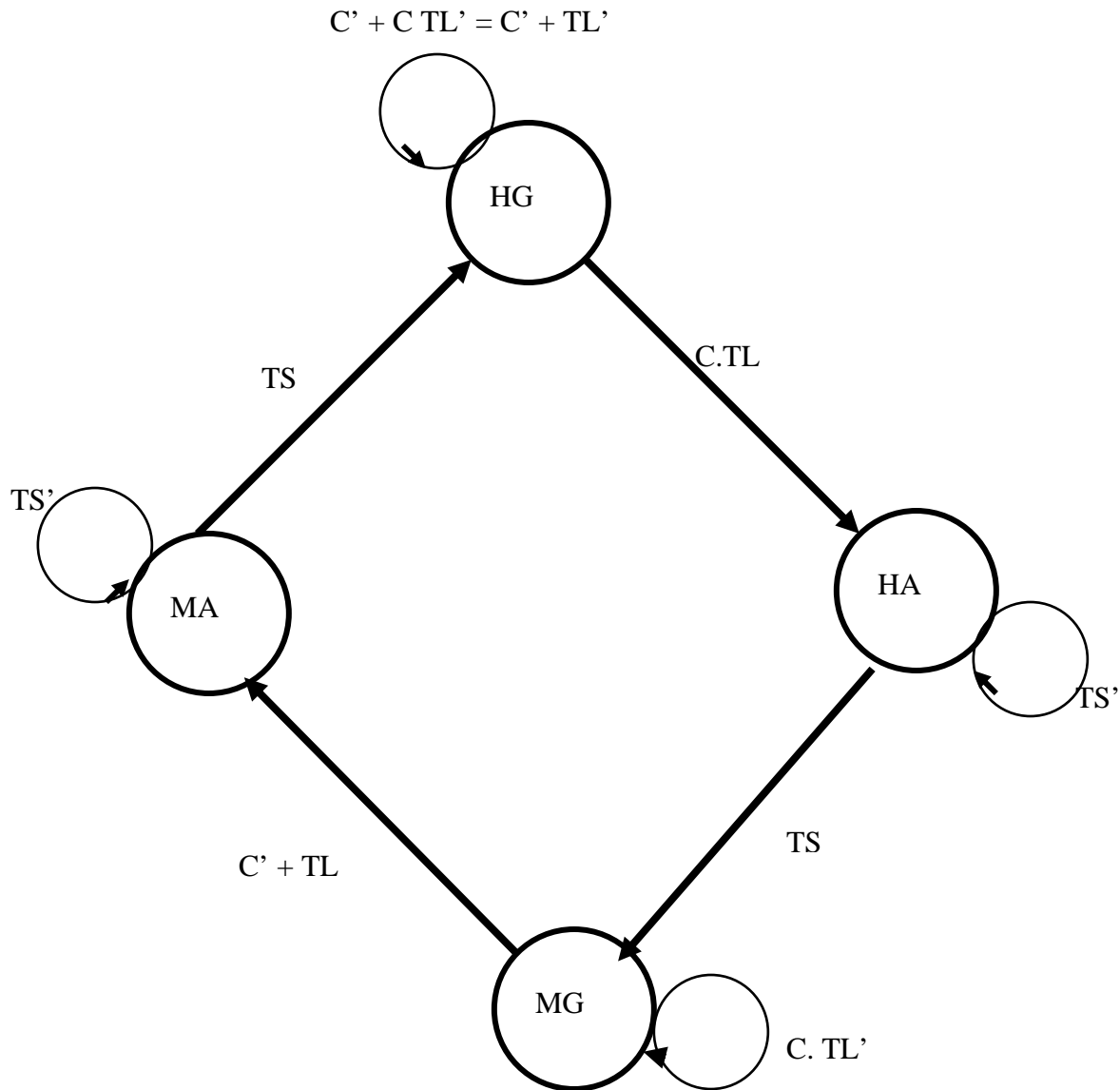
**In put signals**     R: reset, Asynchronous,  
                          C: car on minor road,  
                          TL: timer long,  
                          TS: timer short.

**Output signals**    HG= Highway traffic light green  
                          HA= Highway light amber  
                          MG= Minor road Traffic light green  
                          MA=Minor road traffic light amber

Next step is to draw the state diagram of the controller

**State Diagram**

The State diagram of the controller is shown below.



### State Assignment.

We make the following state assignment

State	y1 y0
HG	0 0
HA	0 1
MG	1 0
MA	1 1

Next step would be to draw the state diagram assuming that R is asynchronous. Since we have 3 inputs it is more convenient to code them

C	TL	TS	Code
<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>
<b>0</b>	<b>1</b>	<b>0</b>	<b>2</b>
<b>0</b>	<b>1</b>	<b>1</b>	<b>3</b>
<b>1</b>	<b>0</b>	<b>0</b>	<b>4</b>
<b>1</b>	<b>0</b>	<b>1</b>	<b>5</b>
<b>1</b>	<b>1</b>	<b>0</b>	<b>6</b>
<b>1</b>	<b>1</b>	<b>1</b>	<b>7</b>



C	TL	TS	Code
---	----	----	------

## Transition Table

Present State	Next state								
	y1 y0		y1 <sup>+</sup> y0 <sup>+</sup>						
	0	1	2	3	4	5	6	7	
0 0	00	00	00	00	00	00	01	01	
0 1	01	10	01	10	01	10	01	10	
1 1	11	00	11	00	11	00	11	00	
1 0	11	11	11	11	10	10	11	11	

Next we determine the next state equations using K-Maps

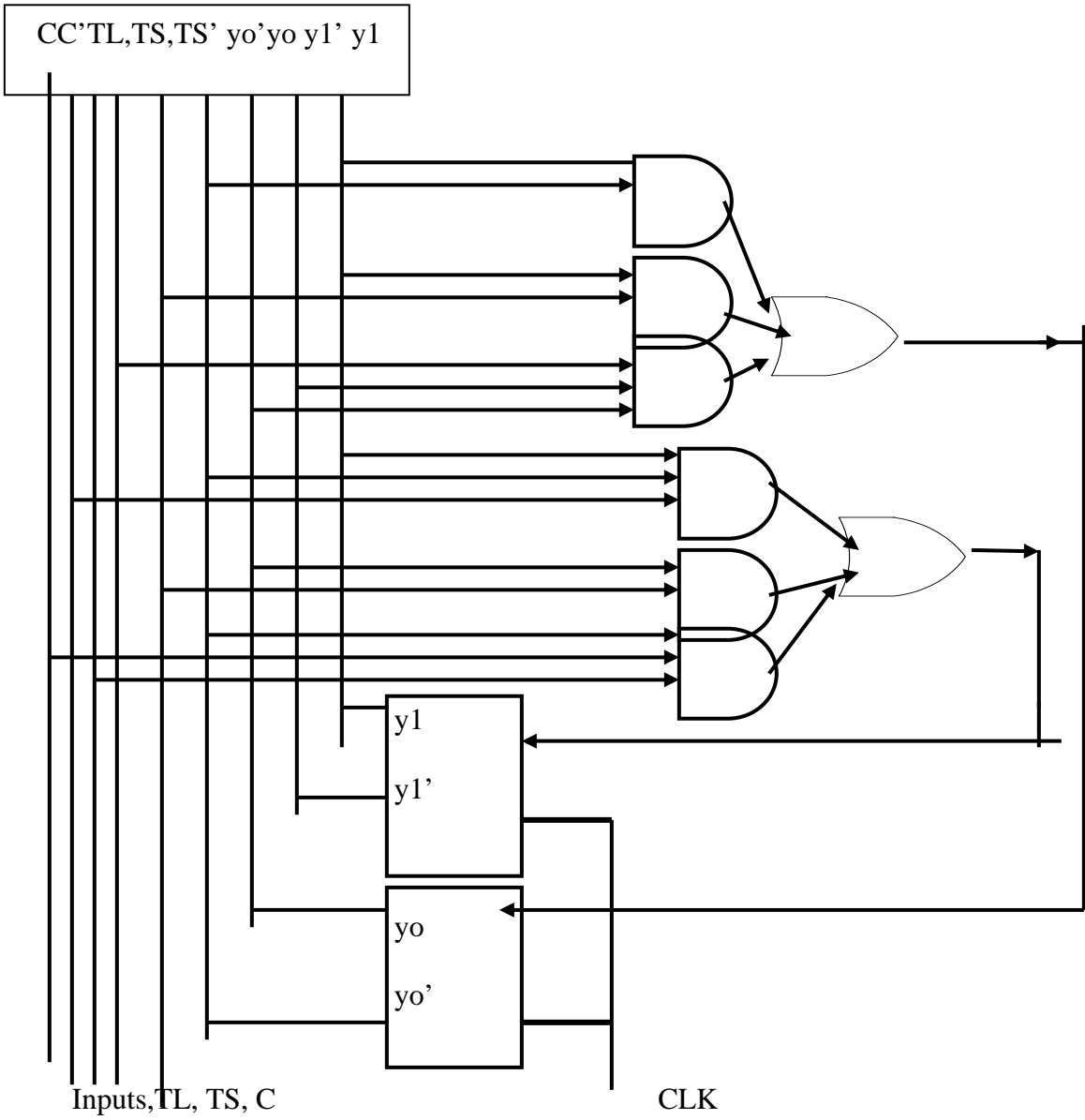
C, TL, TS	0	1	3	2	4	5	7	6
y1,yo								
00								
01		1	1			1	1	
11	1			1	1			1
10	1	1	1	1	1	1	1	1

$$y1^+ = y1yo' + y1 TS' + y1'yo TS$$

TL, TS	0	1	3	2	4	5	7	6
y1,yo								
00							1	1
01	1			1	1			1
11	1			1	1			1
10	1	1	1	1			1	1

$$yo^+ = y1 yo' C' + yo TS' + yo' C TL$$

The next step is to draw the circuit diagram.



### Example 3

Design a Toll Booth Controller that controls the signal and the barrier of a toll booth on a highway.

The Booth and Controller are shown in the figure below and have the following components:

A sensor on the driveway that shows presence of a car, ie signal  $S=1$ ,  $S=0$  otherwise.

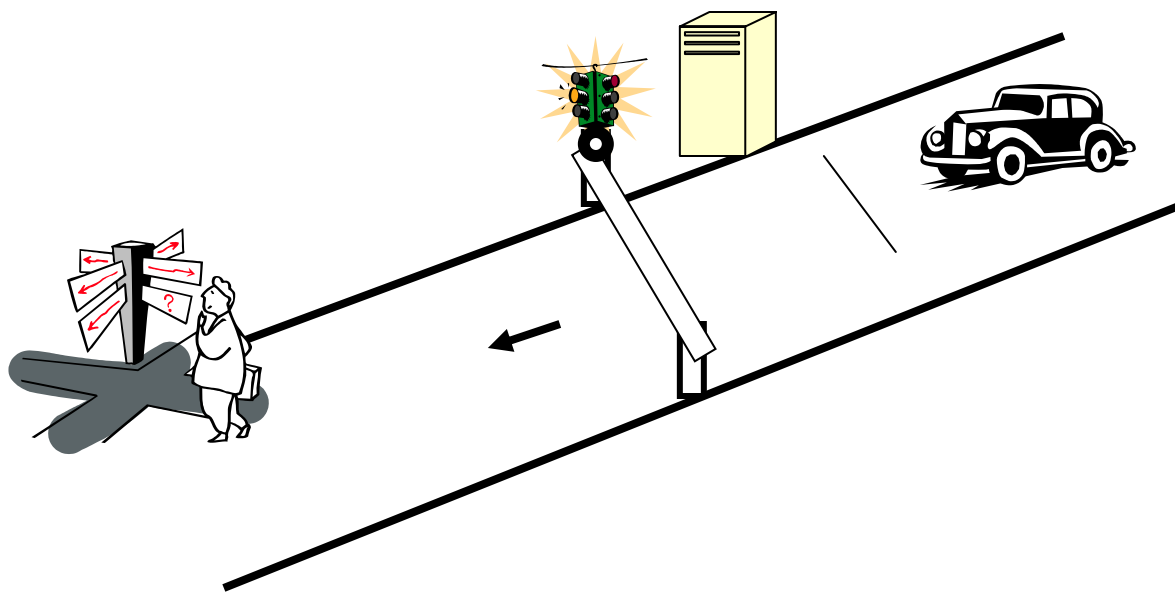
A coin machine that receives the exact coin, when coin is inserted, signal  $C=1$ , otherwise  $C=0$ .

$T=1$  traffic light is green and the barrier open.

$T=0$  traffic light is red and the barrier is closed.

At normal times the tollbooth is idle. Traffic signal is red and the barrier is closed. When a car enters the driveway of the booth, then the presence of the car is detected with  $S=1$  from the sensor.

The controller then waits for the right coin. When the coin is inserted,  $C=1$ , then the traffic light turns green  $T=1$  and the barrier is raised. When the car passes all signals are reset and the barrier is lowered. Assume that there is room for one car only at the booth.



Initially we identify the system variables. These are

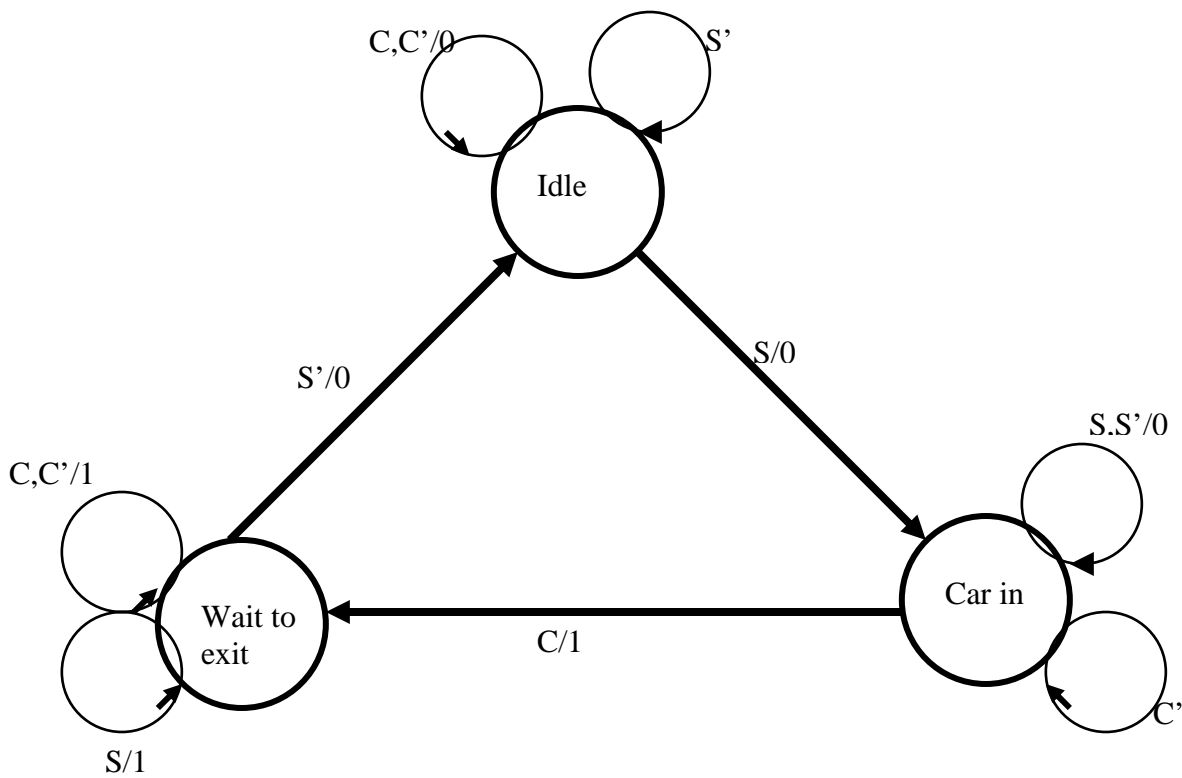
**In put signals**      $R$ = reset, Asynchronous,  $S$  car in booth,  $C$  correct coin inserted.

**Output signals**      $T$ = Traffic light Green

Next step is to draw the state diagram of the controller

# State Diagram

The State diagram of the controller is shown below. The don't care states are removed to make the design simpler.



## State Assignment.

We make the following state assignment

State	y1	y0
Idle state	0	0
Car in state	0	1
Wait to exit	1	1

Next step would be to draw the state diagram assuming that R is asynchronous

## Transition Table

Present State y1 y0	Next state			
	y1 <sup>+</sup> y0 <sup>+</sup> / T			
	S' C'	S' C	SC	SC'
0 0	00/0	00/0	01/0	01/0
0 1	01/0	11/1	11/1	01/0
1 1	00/0	00/0	11/1	11/1

Next we determine the next state equations using K-Maps

y1 y0 \ SC	00	01	11	10
00				X
01		1		X
11		1	1	X
10			1	X

$$y1^+ = Sy1 + Cy1'y0$$

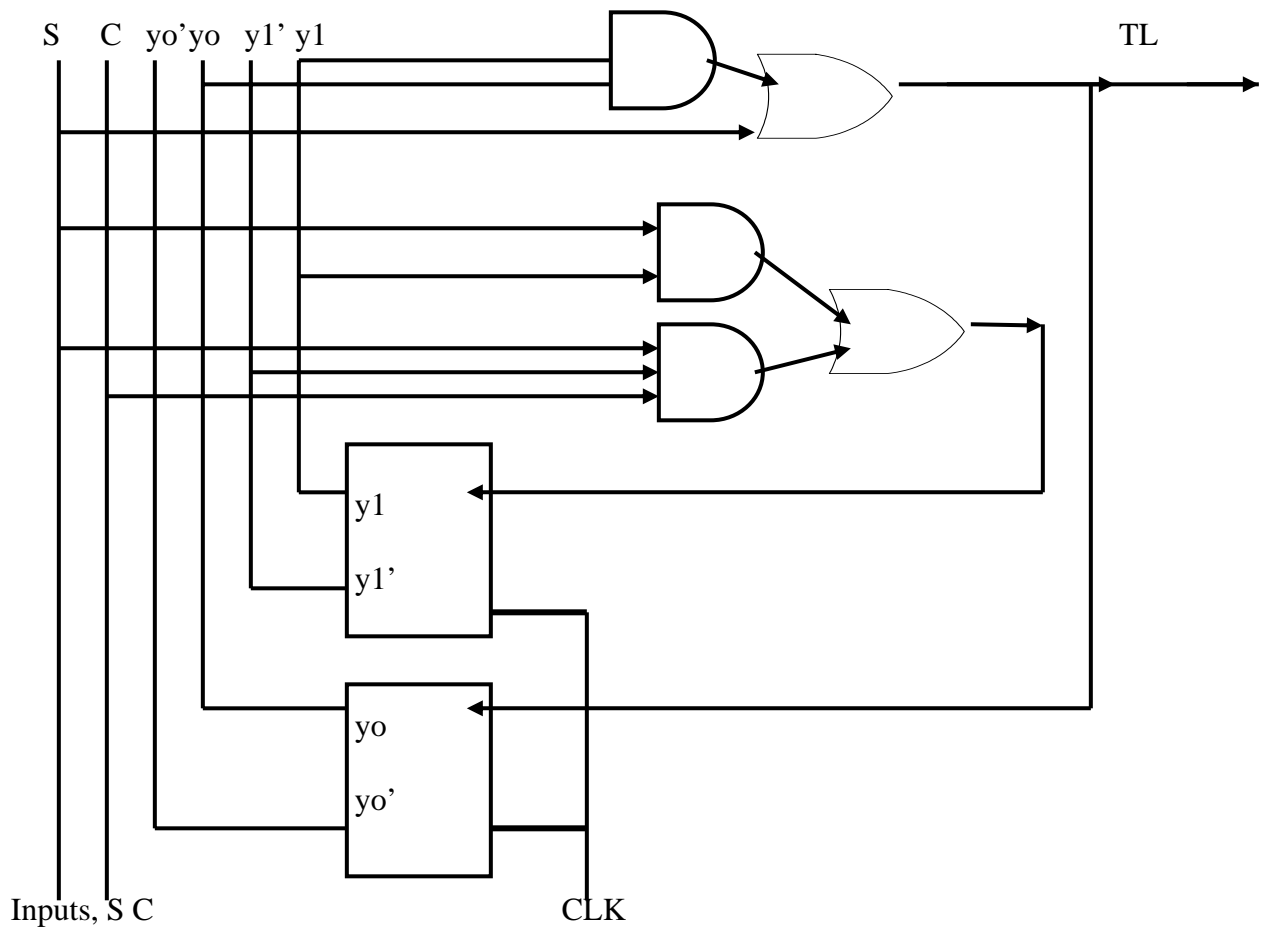
$$y0^+ = S + y1 y0$$

y1 y0 \ SC	00	01	11	10
00		1		X
01		1		X
11	1	1	1	X
10	1	1	1	X

y1 y0 \ SC	00	01	11	10
00				X
01		1		X
11		1	1	X
10			1	X

$$T = y1^+$$

The next step is to draw the circuit diagram.



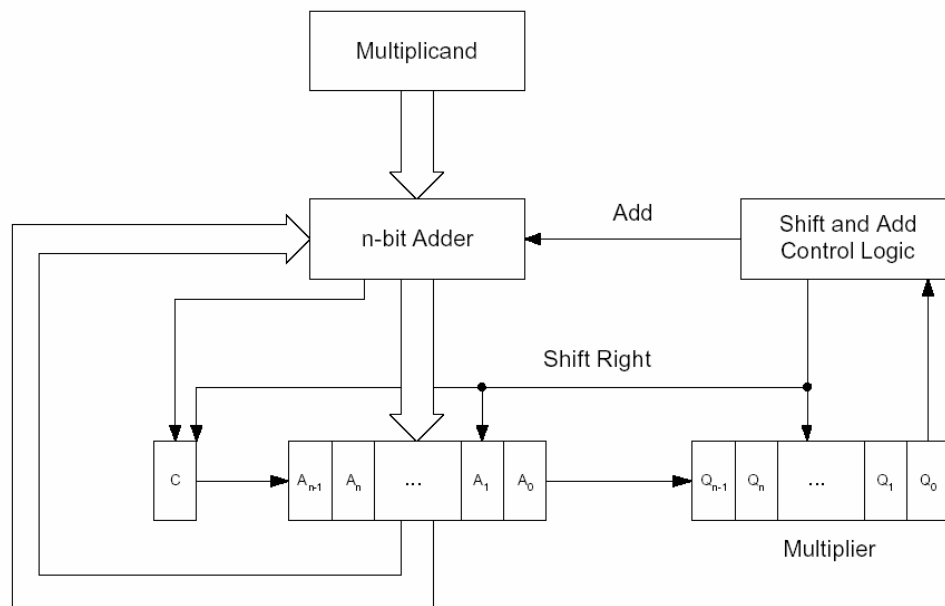
$$T = y_1^+$$

$$y_1^+ = S y_1 + C y_1' y_0$$

$$y_0^+ = S + y_1 y_0$$

# Controller for a Shift and Add Multiplier

The requirement is to design an 8-by-8 bit multiplier based on the shift and add method. The overall architecture is shown in the figure below (Fig.0-1). The circuit shall accept as inputs an 8-bit multiplier and 8-bit multiplicand as well as a Start signal. The multiplier shall then calculate the result using the shift and add method and provide the 16-bit result along with a Stop signal.



**Figure 0-1: Add/Shift Multiplier Block Diagram**

# Design Specifications

The block diagram shown in Fig.0- 2 details the breakdown of VHDL modules. In the here we are interested only in the control circuitry “ The Controller” that controls the overall flow of data which is implemented as a FSM.

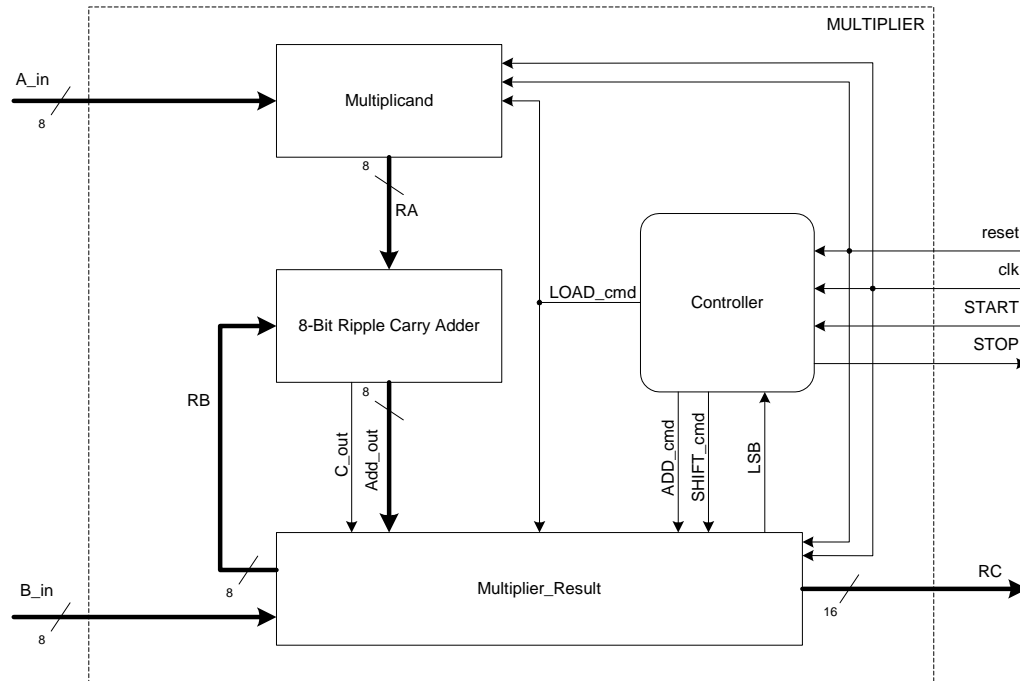


Figure 0-2: Multiplier Design Block Diagram

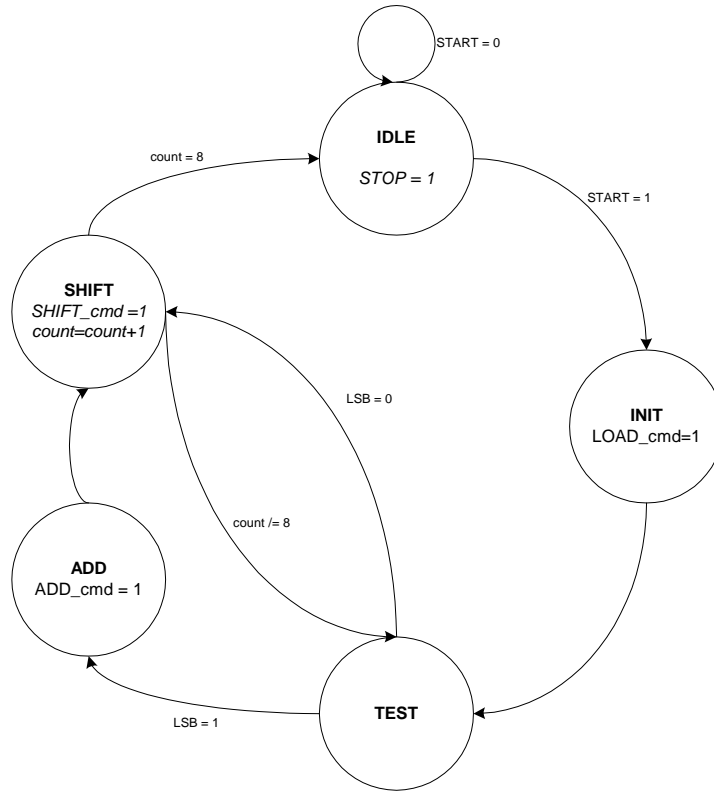
## Controller Design

The Controller is the control unit of the multiplier. It receives a START signal and consequently commands all other modules until the result is obtained and it outputs a STOP signal.

## Design

The design is implemented as a finite state machine with states and transition logic as shown in Fig. 0-3. The Start signal transitions the state machine out of the idle state and into the initialize state whereby it commands the multiplicand and multiplier to be loaded into registers. Once loaded, the state machine goes through a series of test and shift, or test, add and shift operations depending on the status of the LSB bit of the multiplier. Upon reaching the maximum count for the multiplication cycle, the state machine goes back to the idle state and outputs a Stop signal.

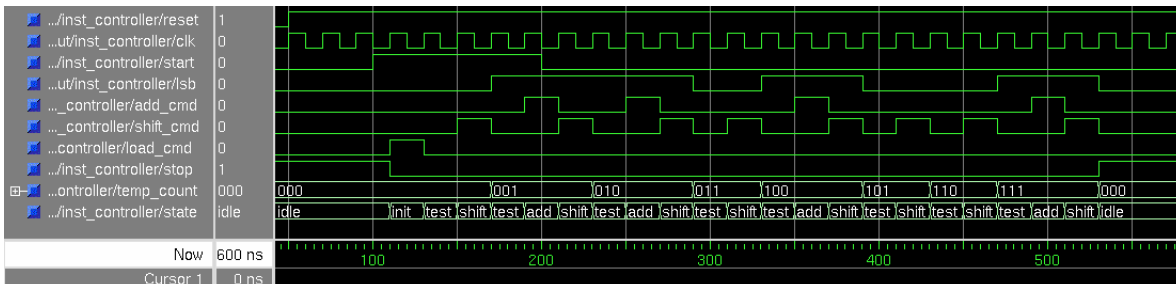




**Figure 0-3: Controller FSM Diagram**

### Simulation & Timing

The controller is synchronous to the clock and transitions through the various states occur on the rising clock edge. As can be seen from the timing diagram in Fig.0-4, the Start signal transitions the state machine out of the idle state only when sampled by the rising clock edge. Upon entering the initialize state, the LOAD\_cmd is generated. During each test state, the LSB is sampled. If the LSB was high, the add state is entered and the controller generates the ADD\_cmd. If the LSB was low, or once the add state is exited, the shift state is entered and the controller generates the SHIFT\_cmd. Upon reaching the maximum count for the multiplication cycle, the state machine goes back to the idle state and outputs a stop signal.



**Figure 0-4: Controller Simulation Timing Diagram**

# VHDL: Controller

```
-----  
--  
-- Library Name : DSD  
-- Unit Name : Controller  
--  
-----  
-----  
-- Date : Mon Oct 27 12:36:47 2003  
--  
-- Author : Giovanni D'Aliesio  
--  
-- Description: Controller is a finite state machine  
-- that performs the following in each  
-- state:  
-- IDLE > samples the START signal  
-- INIT > commands the registers to be  
-- loaded  
-- TEST > samples the LSB  
-- ADD > indicates the Add result to be stored  
-- SHIFT > commands the register to be shifted  
--  
-----  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
  
entity Controller is  
port (reset : in std_logic ;  
      clk : in std_logic ;  
      START : in std_logic ;  
      LSB : in std_logic ;  
      ADD_cmd : out std_logic ;  
      SHIFT_cmd : out std_logic ;  
      LOAD_cmd : out std_logic ;  
      STOP : out std_logic);  
end;  
  
-----  
architecture rtl of Controller is  
  
signal temp_count : std_logic_vector(2 downto 0);  
  
-- declare states  
type state_typ is (IDLE, INIT, TEST, ADD, SHIFT);  
signal state : state_typ;  
  
begin  
  
process (clk, reset)  
begin  
if reset='0' then  
state <= IDLE;  
temp_count <= "000";  
  
elsif (clk'event and clk='1') then  
  
case state is  
  
when IDLE =>  
if START = '1' then  
state <= INIT;
```

```

        else
            state <= IDLE;
        end if;

    when INIT =>
        state <= TEST;

    when TEST =>
        if LSB = '0' then
            state <= SHIFT;
        else
            state <= ADD;
        end if;

    when ADD =>
        state <= SHIFT;

    when SHIFT =>
        if temp_count = "111" then -- verify if finished
            temp_count <= "000"; -- re-initialize counter
            state <= IDLE; -- ready for next multiply
        else
            temp_count <= temp_count + 1; -- increment counter
            state <= TEST;
        end if;

    end case;

end if;

end process;

STOP <= '1' when state = IDLE else '0';
ADD_cmd <= '1' when state = ADD else '0';
SHIFT_cmd <= '1' when state = SHIFT else '0';
LOAD_cmd <= '1' when state = INIT else '0';

end rtl;

```

## Algorithmic State Machines

The Algorithmic State machines, ASM, are a more powerful method of representation of the state diagram for sequential circuits. This is because both behavior and control are visible on the diagram that gives the designer more flexibility in implementing the control structure.

The ASM representation is very similar to the flow diagram. However in state flow diagrams the concept of time is missing. In ASM representation however, the sequence of events are represented in the timely manner.

The components of the ASM are:

- 1) The state box
- 2) The decision box
- 3) The conditional box.

The conditional box is the new element that is not found in the flow diagram. The state box is represented by a rectangle containing the register transfer level operation or the output operation. The decision box is a diamond shaped box containing the decision to be taken. Usually with one input and two outputs. The conditional box is the oval shaped box containing register transfer level operations or output operation. The input of the conditional box must come from one or more decision boxes.

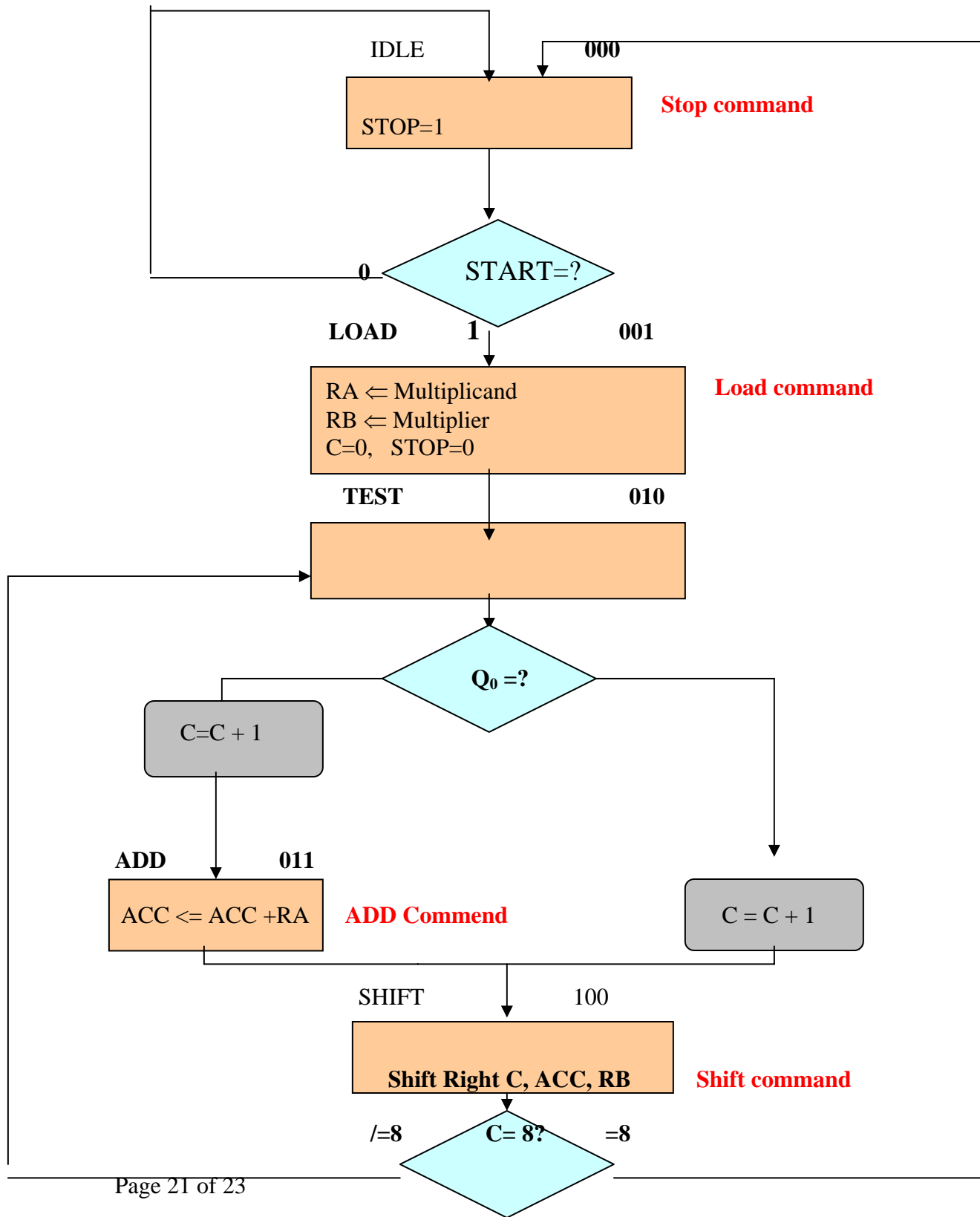
Information regarding the state, register transfers and the outputs are written on the diagram.

### Example:

Represent the control circuitry required for the shift and add multiplier discussed in the previous example:

The starting state is the IDLE state where the multiplier is waiting for the START command. Notice the information written on top of the box and within. The name of the state and its binary state assignment is written on top of the box while inside the box is the command signal STOP is written. This state is responsible for generating and maintaining this signal. After the START signal is inserted the control moves to the LOAD state. In this state the register transfer operation of loading the multiplicand and the multiplier takes place as well as resetting the STOP signal and resetting the counter C. The state Test, plus the decision box and the conditional box on right are named Algorithmic State Block. This is because all operations within this block are evaluated within the current cycle. After testing the LSB of the multiplier we either go to state ADD or SHIFT while updating the counter. In state SHIFT we check for the counter for the number of shifts performed and accordingly move to the START or to the TEST state.

This implementation is not the only method. It is possible to have several implementations depending on the designer.



### Circuit implementation:

The following table can be deduced from the diagram

Present state	Present state	Inputs	Next State	Notes
	S <sub>2</sub> S <sub>1</sub> S <sub>0</sub>	S Q <sub>0</sub> C	S <sub>2</sub> S <sub>1</sub> S <sub>0</sub>	
IDLE	0 0 0	0 X X	0 0 0	
IDLE	0 0 0	1 X X	0 0 1	
LOAD	0 0 1	X X X	0 1 0	
TEST	0 1 0	X 0 X	1 0 0	
TEST	0 1 0	X 1 X	0 1 1	
ADD	0 1 1	X X X	1 0 0	
SHIFT	1 0 0	X X 0	0 1 0	
SHIFT	1 0 0	X X 1	0 0 0	

The ASM machine uses decision boxes to arrive at its output and as such is easier to implement the algorithm using multiplexers. For this implementation we have 5 states so we need 3 Flip Flops to store the states. Each FF will receive its input from an 8 to 1 MUX. The Muxes will make the decision regarding the next state.

