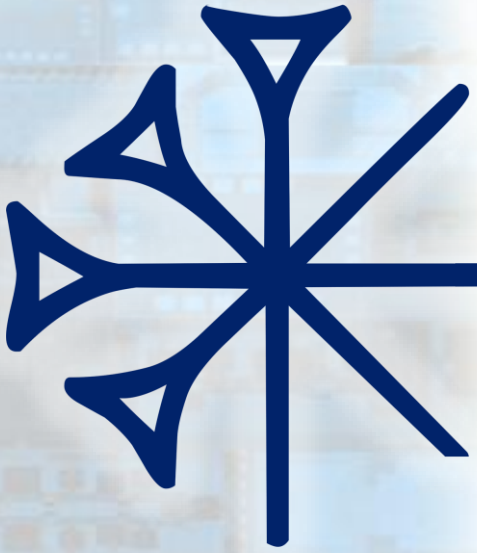
CS 303
Logic & Digital System Design

Ömer Ceylan

Sabancı Üniversitesi



Chapter 8

Register Transfer Level & Design with ASM

Sabancı Üniversitesi

Overview

- Register transfer level (RTL) to model complex digital systems
  - a level of abstraction used in describing operations of synchronous digital circuits
  - a circuit's behavior is defined in terms of the transfer of data between hardware registers, and the operations performed on data.
- Goal: Algorithmic expression of a digital system
- Algorithmic State Machine (ASM) charts
  - to model complex digital systems
  - to map a complex design into hardware

Register Transfer Level

- Designing a complex digital system using state tables becomes difficult as the number of states increases
- Remedy
  - partition the system into modular subsystems
  - each module is designed separately
  - modules are connected to each other
- A digital module is best defined by
  1. A set of registers
  2. Operations on the binary information stored in them

Register Transfer Level

- Register operations
  - move, copy, shift, count, clear, load, add, subtract
- A digital system is said to be represented at the register transfer level (RTL) when it is specified by the following three components

1. Set of registers
2. Operations performed on the data stored in the registers
3. The control supervises the sequence of operations in the system

The Control

- Control logic
  - Initiates the sequence of operations
  - generates timing (control) signals that sequence the operations in a prescribed manner
    - The outputs of control logic are binary signals that initiate the various operations in registers
  - Certain conditions that depend on the previous operations may determine the sequence of future operations
- Transfer statement
  - $R_2 \leftarrow R_1$ (“replacement” operation)
- Conditional transfer statement
  - $\mathbf{if}(T_1 = 1)\ \mathbf{then}\ R_2 \leftarrow R_1$
  - T_1 is checked and a control signal set to a proper value

Register Transfer Operations

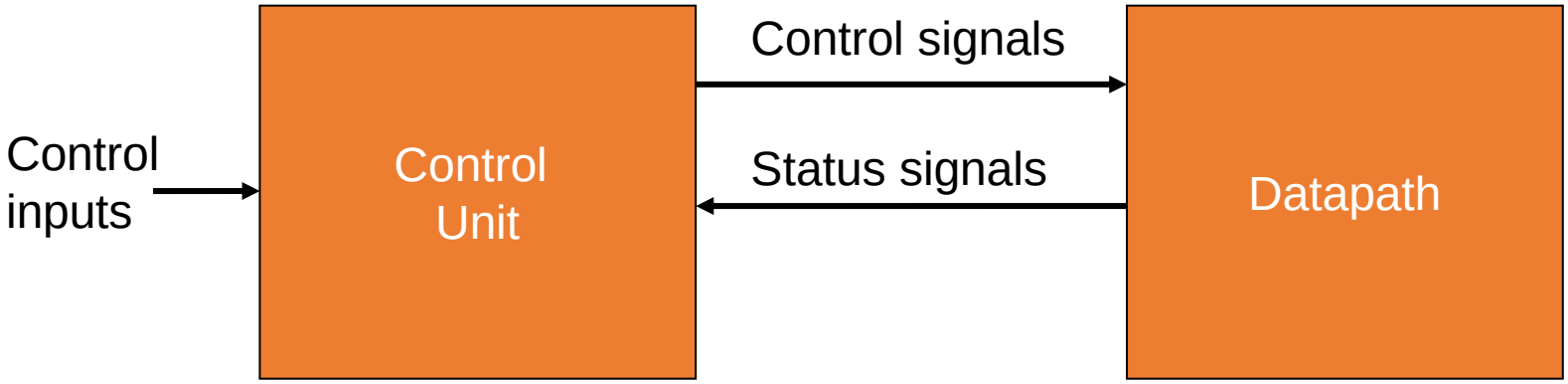
- In register transfer operations, clock is not explicitly shown
  - but, transfer is assumed to happen at the clock edge.
  - previous example
    - **if (T_1 = 1) then $R_2 \leftarrow R_1$**
    - T_1 may become 1 before the clock edge, but actual transfer happens exactly at the clock edge.
- Examples:
  - **if (T_3 = 1) then ($R_2 \leftarrow R_1$, $R_1 \leftarrow R_2$)**
  - $R_1 \leftarrow R_1 + R_2$
  - $R_3 \leftarrow R_3 + 1$
  - $R_4 \leftarrow shr\ R_4$
  - $R_5 \leftarrow 0$

Types of Register Operations

- Four categories

1. transfer operations
2. arithmetic operations
3. logic operations
4. shift operations

Datapath and Control



- One obvious distinction
  1. design of datapath (data processing path)
  2. design of control circuit

Hardware Algorithm & ASM Chart

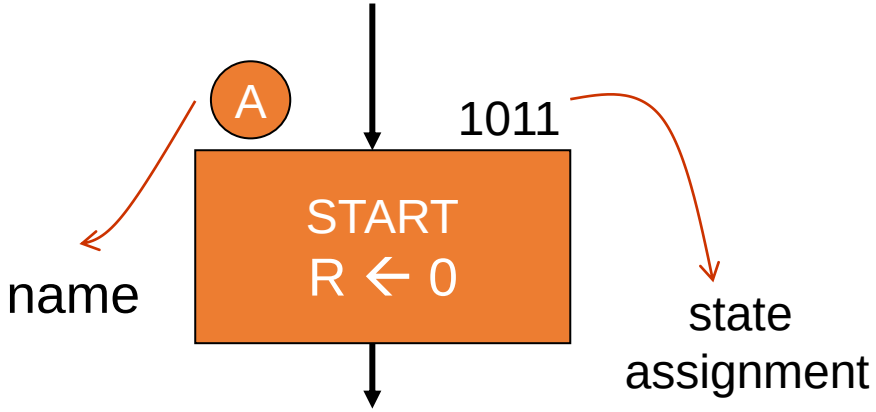
- (Digital) hardware algorithm
  - is a procedure for implementing a digital circuit with given pieces of hardware equipments
  - specifies the control sequence and datapath tasks
- Algorithmic state machine (ASM) chart is a special type of flowchart used to define digital hardware algorithms.
  - state machine is another term for a sequential circuit
- ASM chart describes
  - the sequence of events,
  - events that occur at state transitions.

ASM Chart

- Three basic elements

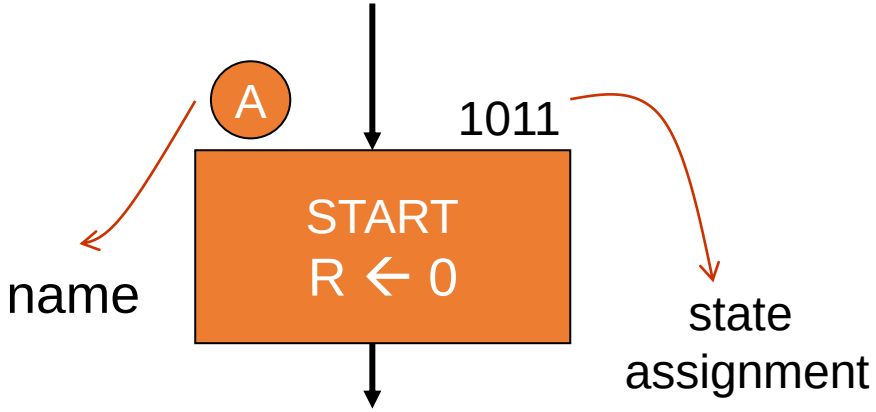
1. State box
2. Decision box
3. Conditional box

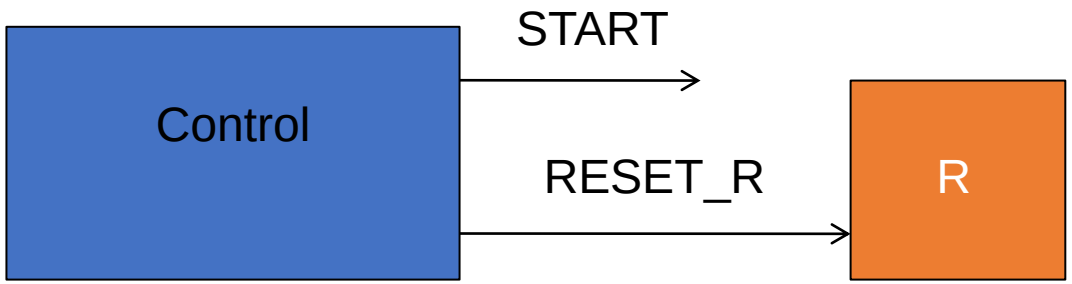
State Box



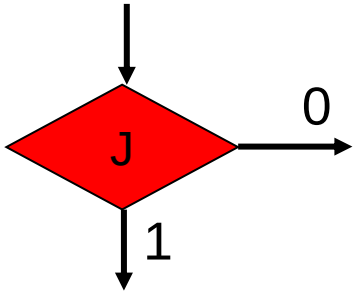
- The output signals (e.g., **START**) in the box take the specified values in the current state
  - if not specified they are 0 in other states .
- The notation **R ← 0** means that the register is cleared to 0 when the system transits from **A** to the next state

State Box


A
1011
START
R ← 0
name
state
assignment

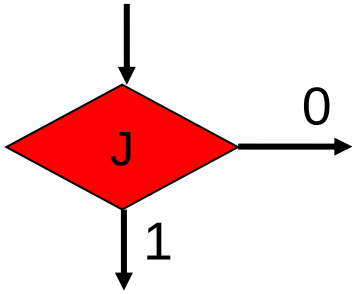

START
Control
RESET_R
R

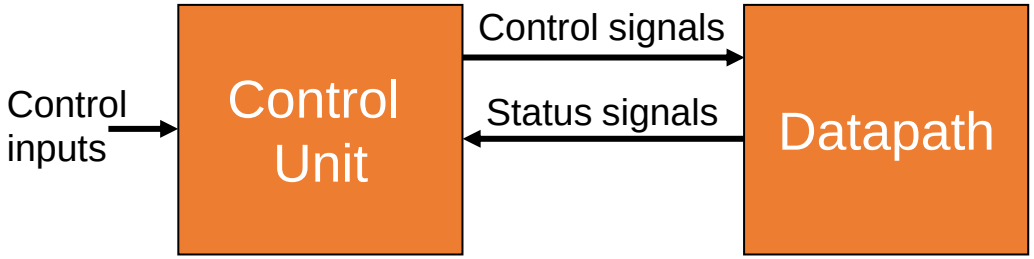
Decision Box

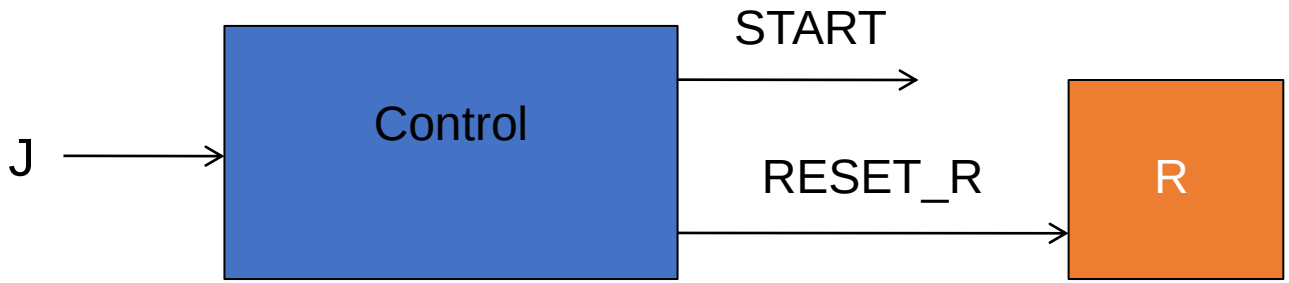


- Decision box has two or more branches going out.
- Decision is made based on the value of one or more input signals (e.g., signal J)
- Decision box must follow and be associated with a state box.
- Thus, the decision is made in the same clock cycle as the other actions of the state.

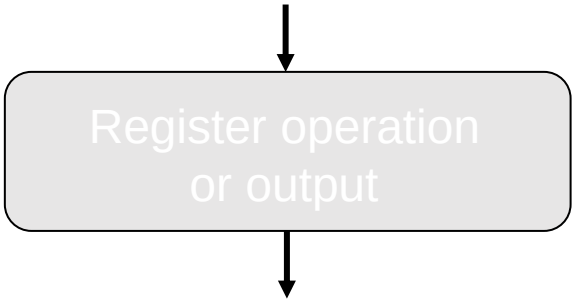
Decision Box





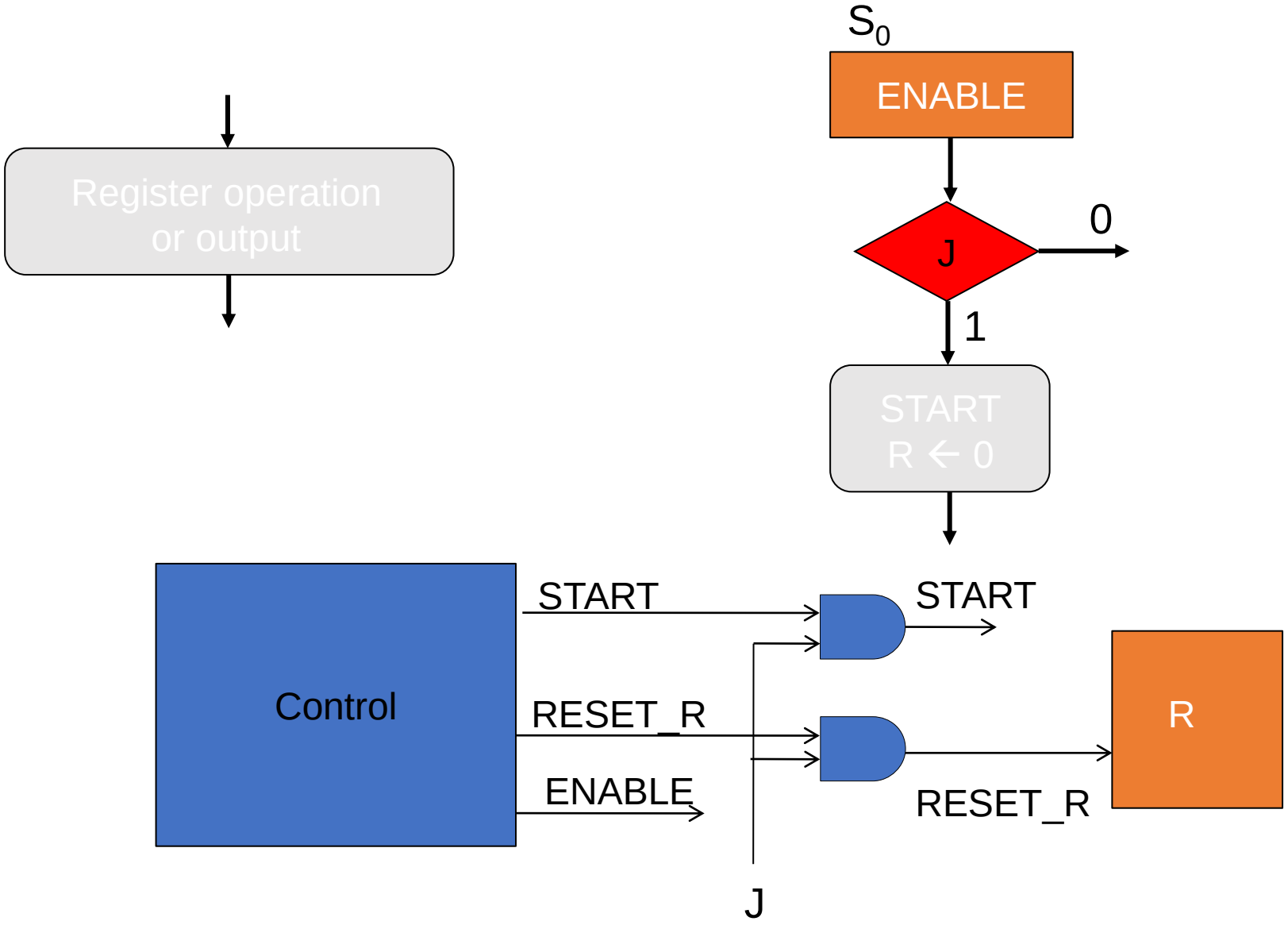


Conditional Box

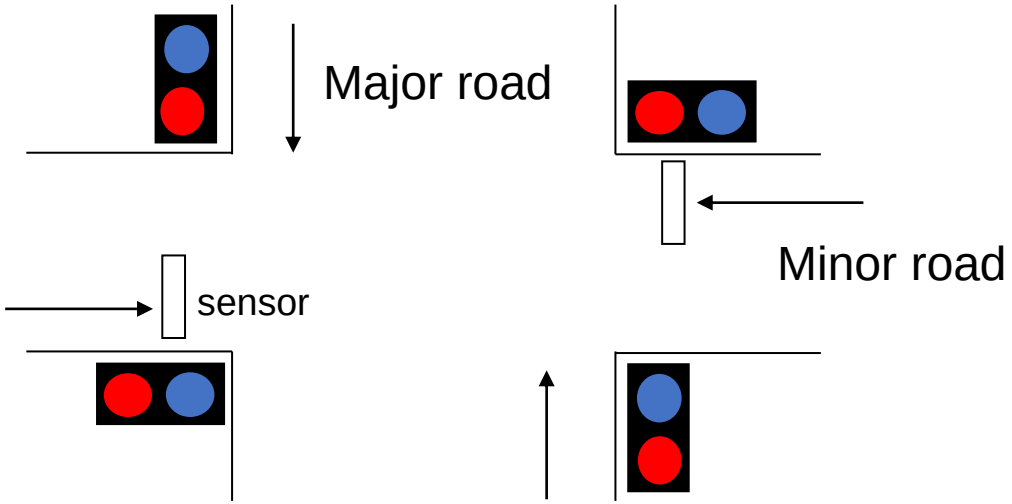


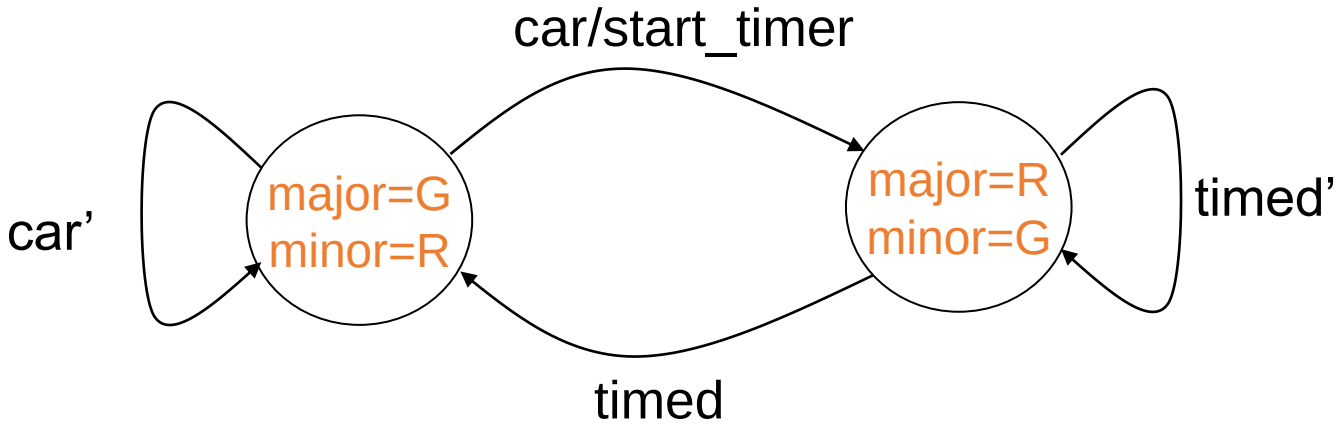
- A conditional box must follow a decision box.
- A conditional box is attached to a state box through one or more decision boxes.
- Therefore, the output signals in the conditional output box are asserted in the same clock cycle as those in the state box to which it is attached.
- The output signals can change during the current state as a result of changes on the inputs.
- The conditional output signals are sometimes referred as Mealy outputs since they depend on the input signals as well.

Conditional Box

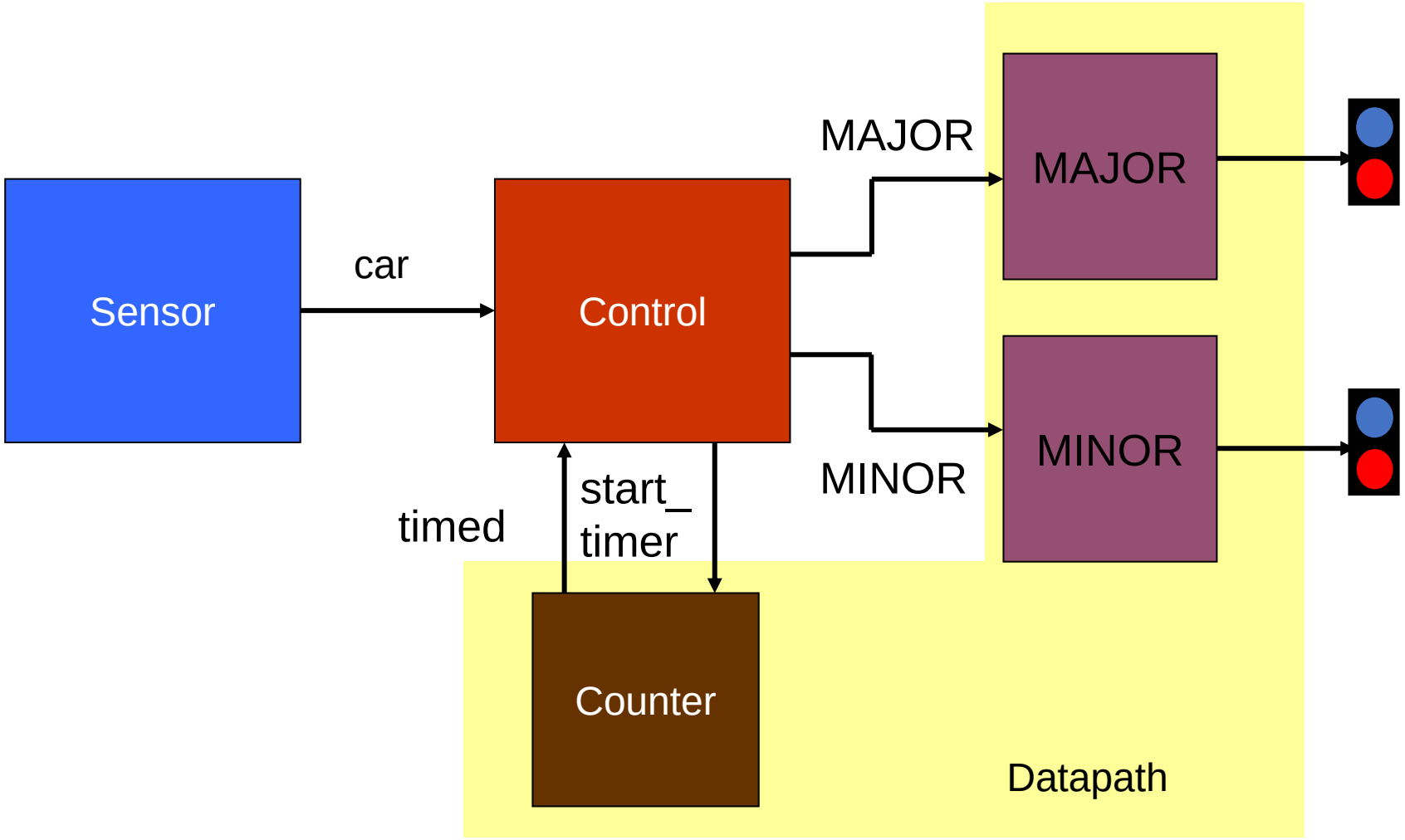


Example: Traffic Control

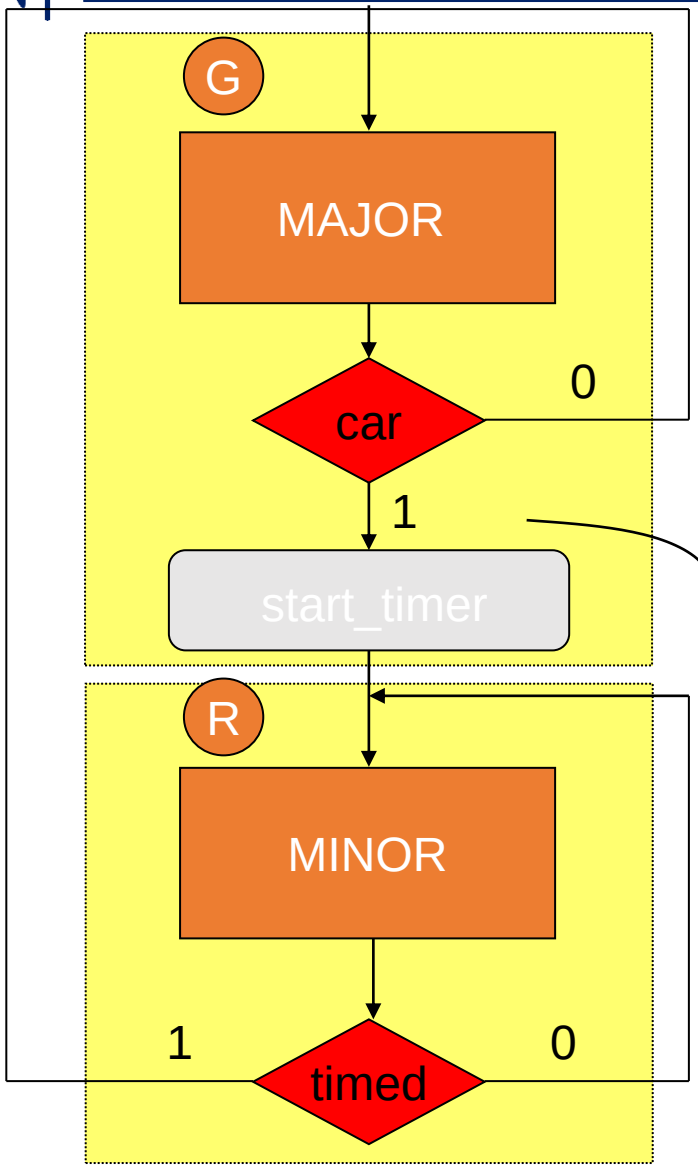




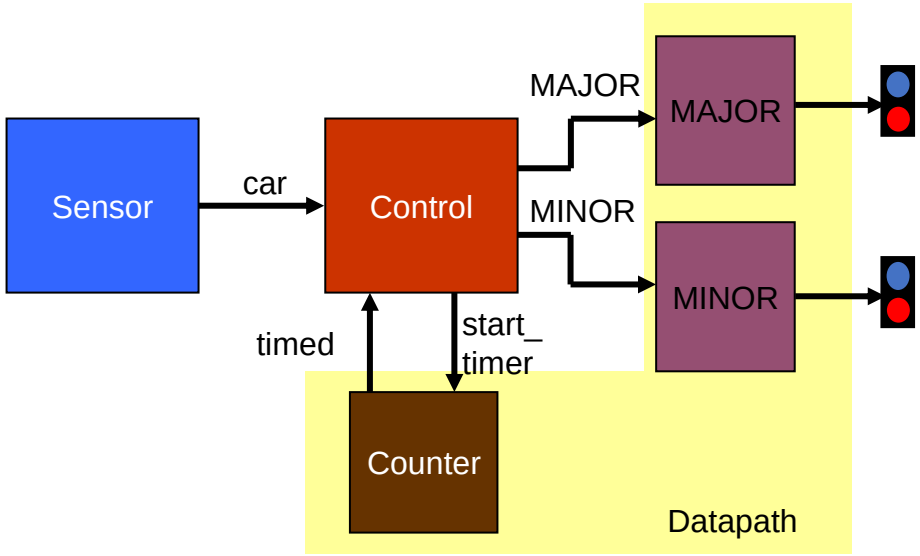
Datapath & Control



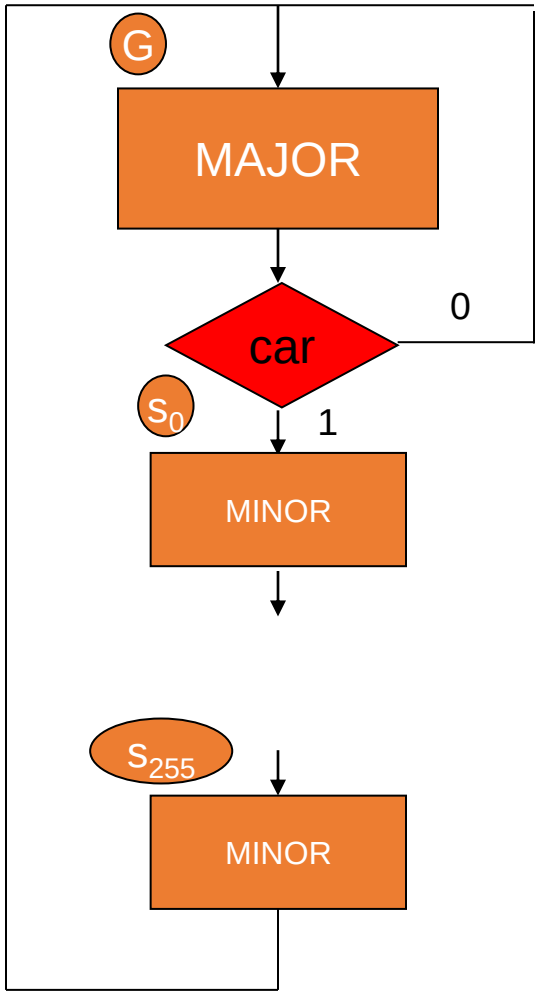
Example: ASM Chart



ASM Block

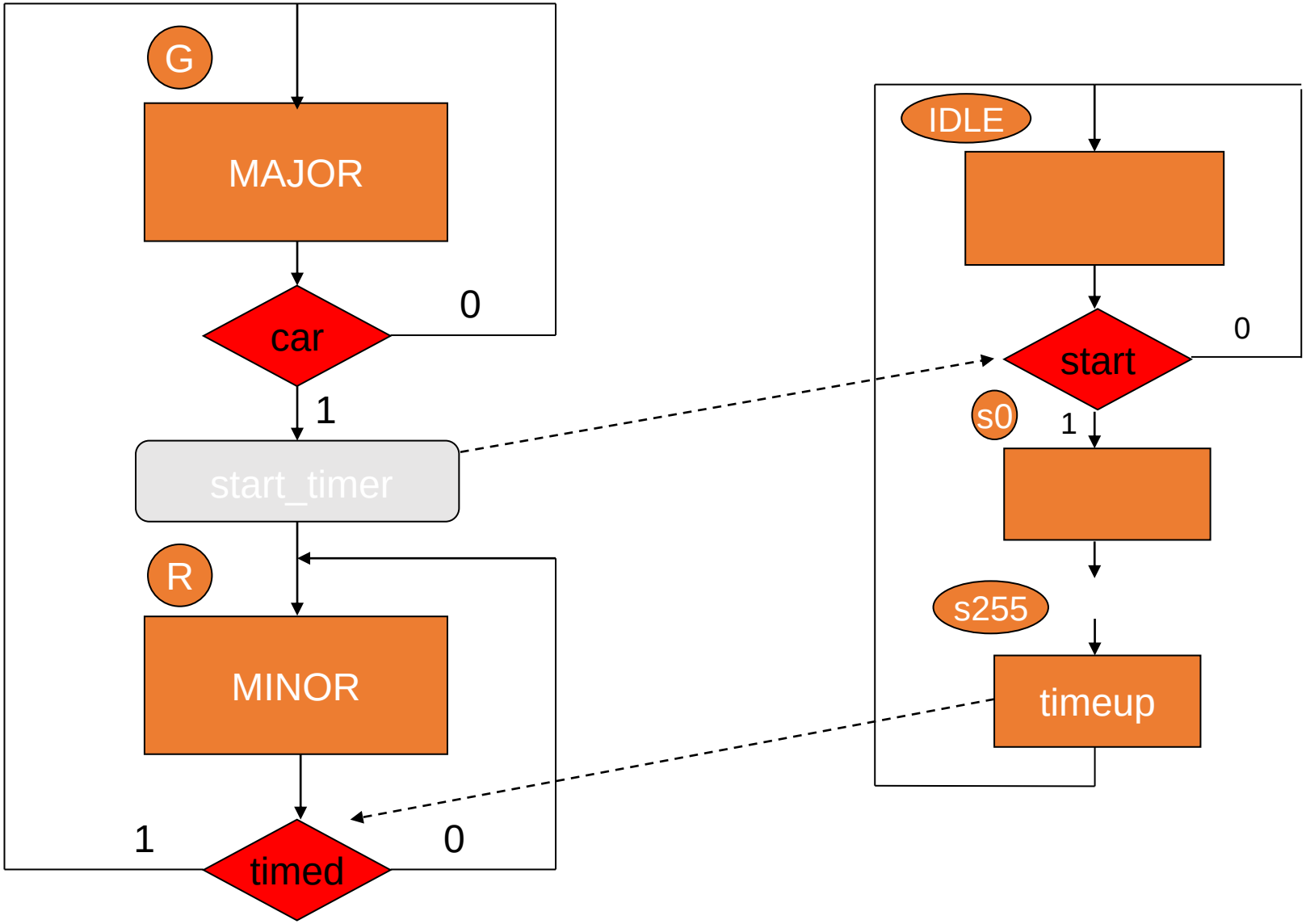


Traffic Controller with a Timer

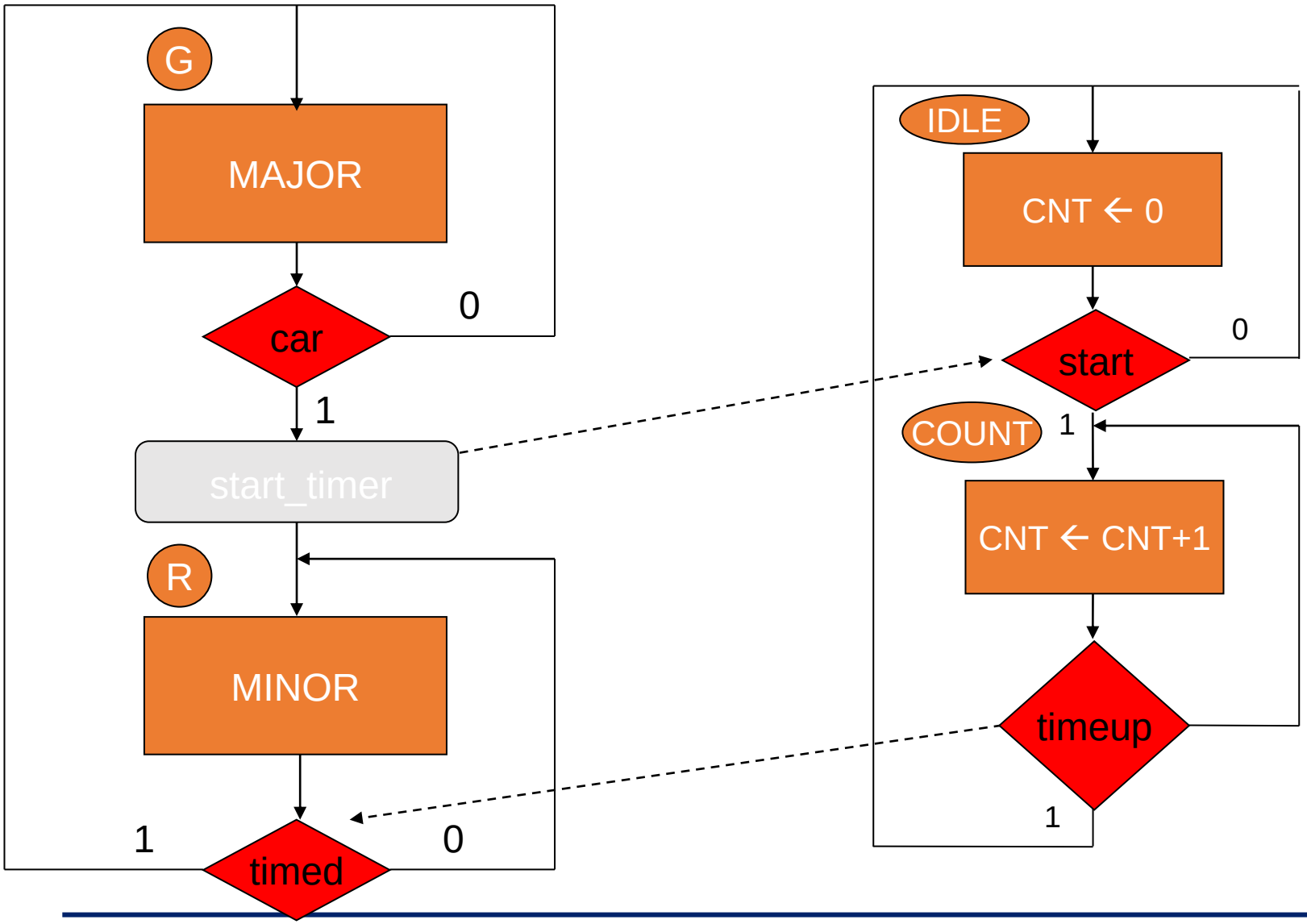


- There is an abundance of states.
- states from s_0 to s_{255} map to a simple counter
- we can just separate the traffic light controller from the timer.

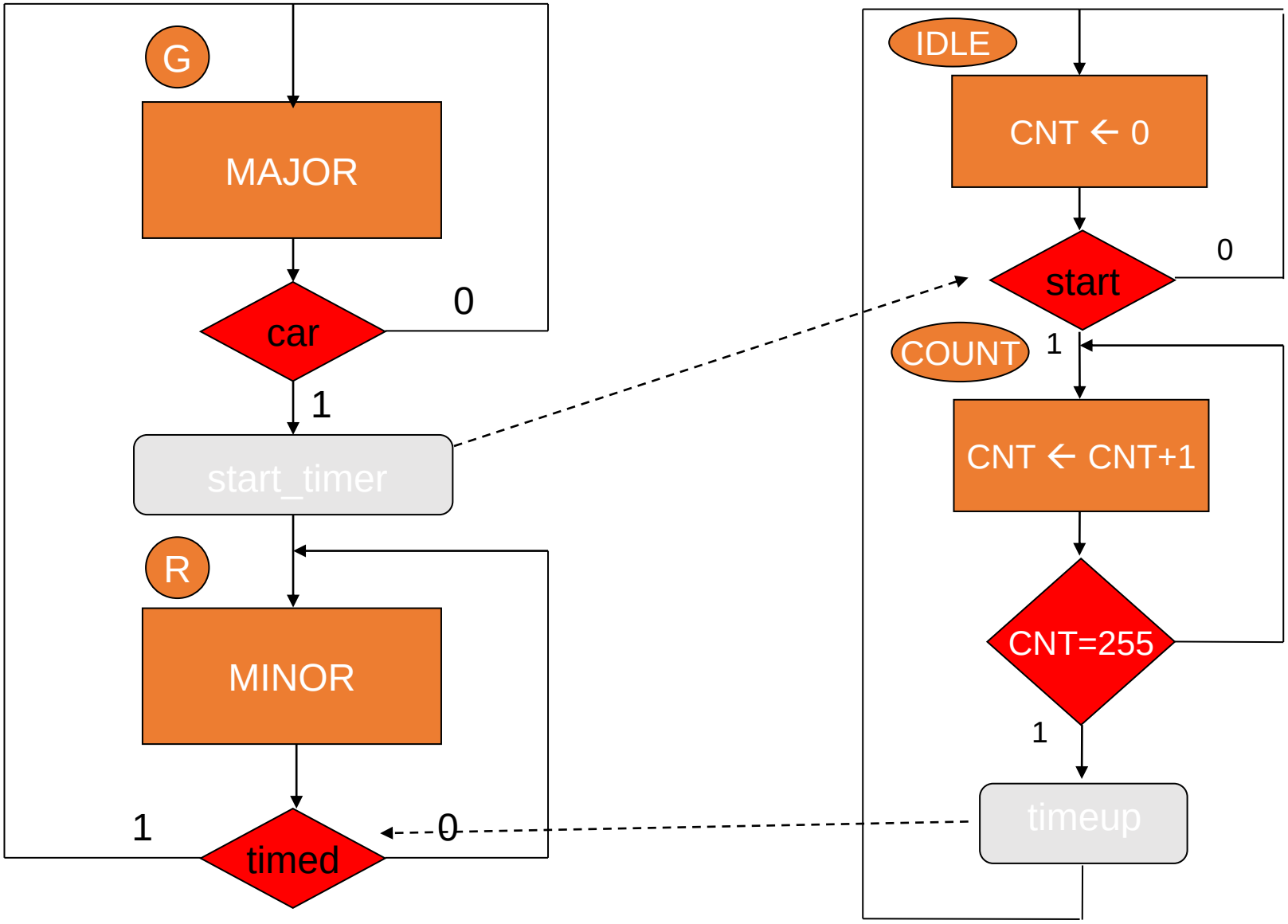
Linked ASM Charts



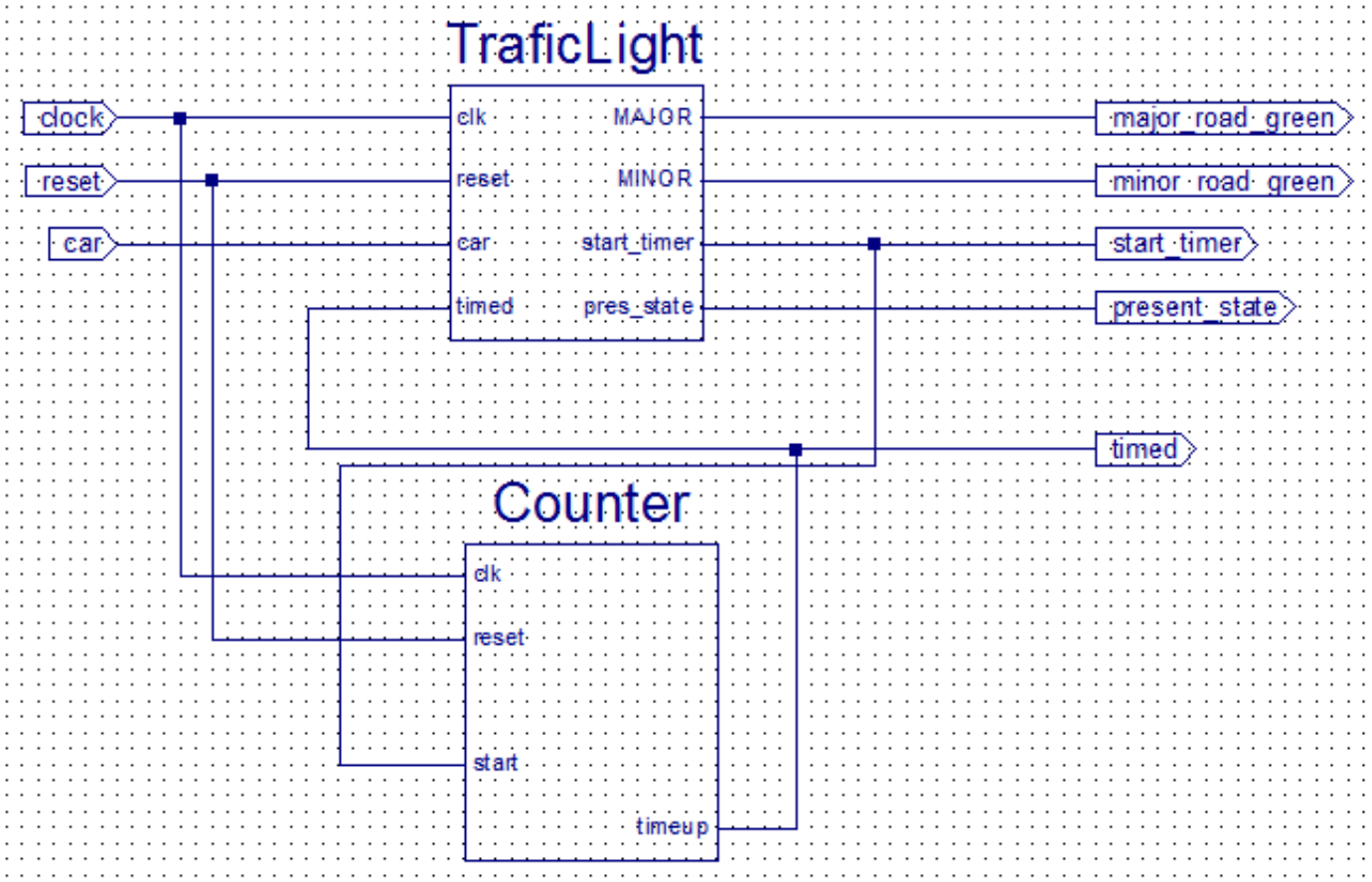
Linked ASM Charts



Linked ASM Charts



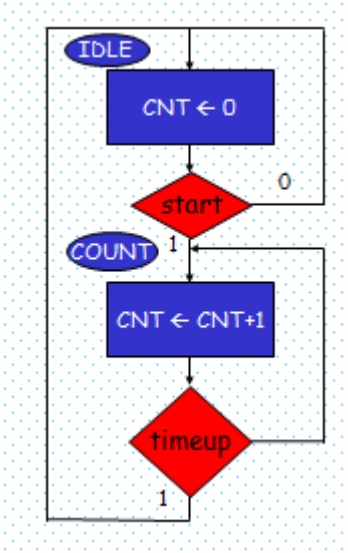
Traffic Control Example - Schematic



Note that **`present_state`**, **`start_timer`** and **`timed`** outputs are normally not shown (they are internal). Here, they are used for debugging

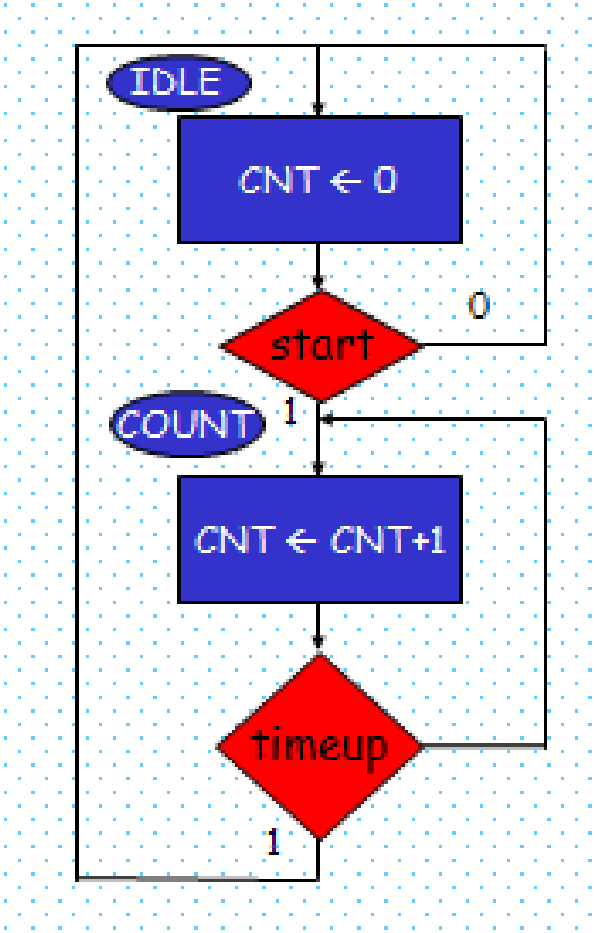
Verilog Code – Counter 1/2

```
`timescale 1ns / 1ps

module Counter(input clk, reset, start, output reg timeup);
  reg state;
  reg [2:0] CNT;

  parameter IDLE = 1'b0, COUNT = 1'b1, LIMIT = 3'd6;
  //Sequential part
  always @ (posedge clk or posedge reset)
  if (reset)
    state <= IDLE; CNT <= 0;
  else case(state)
    IDLE:
     CNT <= 0;
     if(start) state <= COUNT; else state <= IDLE;
    COUNT:
     CNT <= CNT+1;
      if(timeup) state <= IDLE; else state <= COUNT;
   default:
       CNT <= CNT;
       state <= IDLE;
    endcase
```



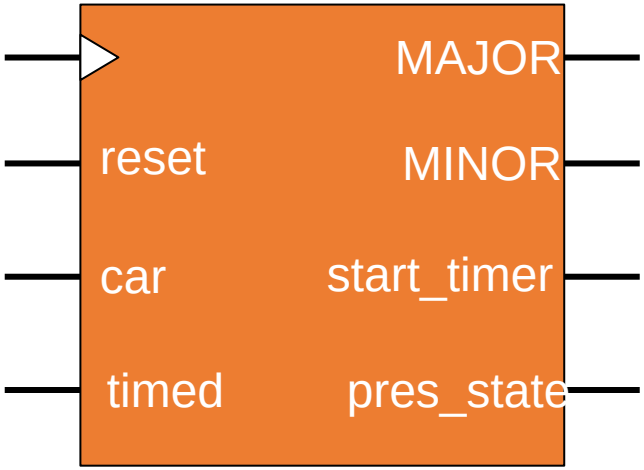
Verilog Code – Counter 2/2

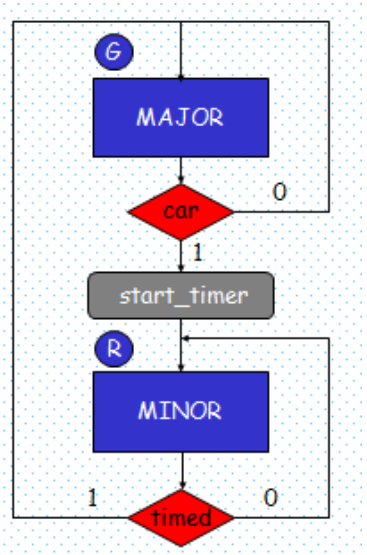
```
...

  //Combinational output circuit
  always@(*)
    case (state)
      COUNT:
        if(CNT == LIMIT)
          timeup = 1'b1;
        else
          timeup = 1'b0;
      default:
        timeup = 1'b0;
    endcase

endmodule
```



Verilog Code –Trafic Light 1/4

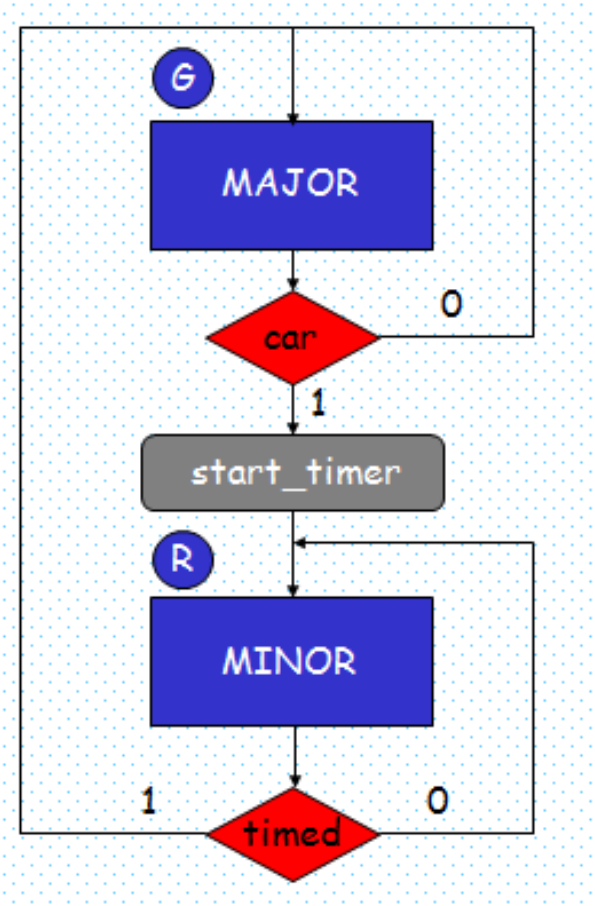
```
`timescale 1ns / 1ps
module TraficLight (clk, reset, car, timed, MAJOR, MINOR, start_timer, pres_state);

input clk, reset, car, timed;
output MAJOR, MINOR, start_timer, pres_state;

reg next_state;
reg pres_state;

reg MAJOR;
reg MINOR;
reg start_timer;

parameter
   ST_G = 1'b0,
   ST_R = 1'b1;

// FSM starts
//Sequential circuit state transitions
always@(posedge clk or posedge reset)
begin
    if(reset)
       pres_state <= ST_G;
    else
       pres_state <= next_state;
end
...
```





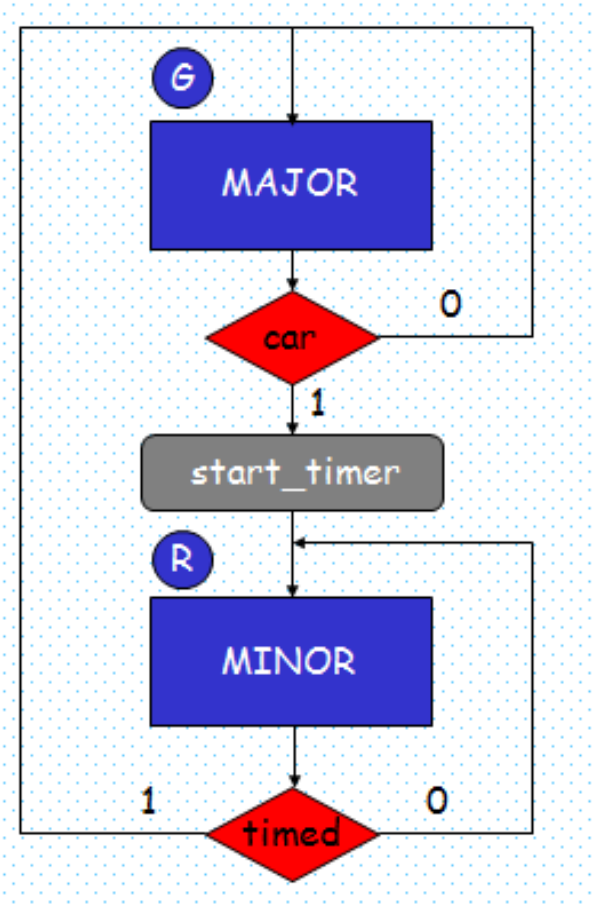
Verilog Code –Trafic Light 2/4

```
...
//Combinational circuit for state transition
always@(*)
   case (pres_state)
      ST_G:
        if(car)
           next_state = ST_R;
        else
           next_state = pres_state;
      ST_R:
        if(timed)
           next_state = ST_G;
        else
           next_state = pres_state;

      default:
         next_state = ST_G;

   endcase
...
```



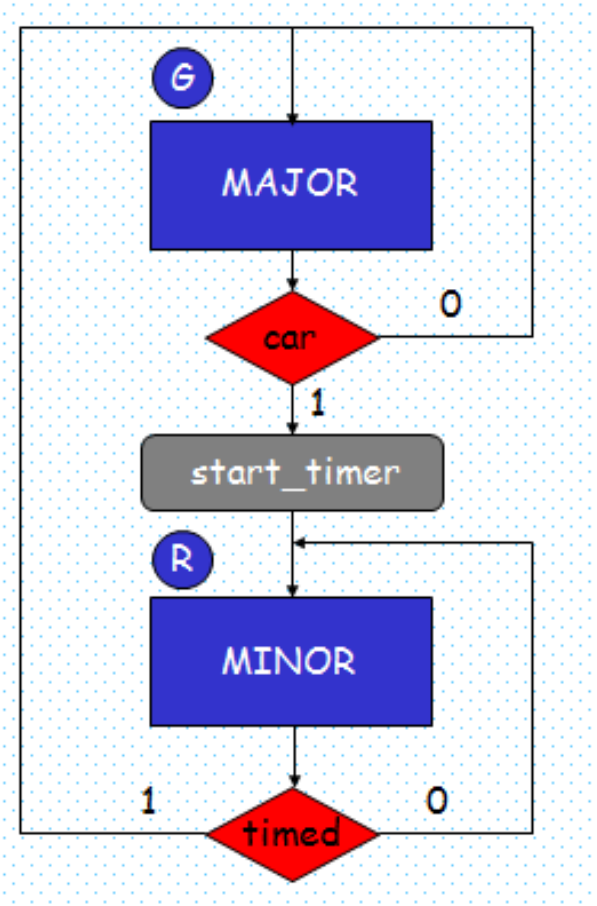
Verilog Code –Trafic Light 3/4

```
...
//Combinational circuits outputs
always@(*)
    case(pres_state)
        ST_G:
            if(car)
                start_timer = 1; // Mealy output
            else
                start_timer = 0;// Mealy output
        default:
            start_timer = 0;
    endcase

// FSM ends
...
```

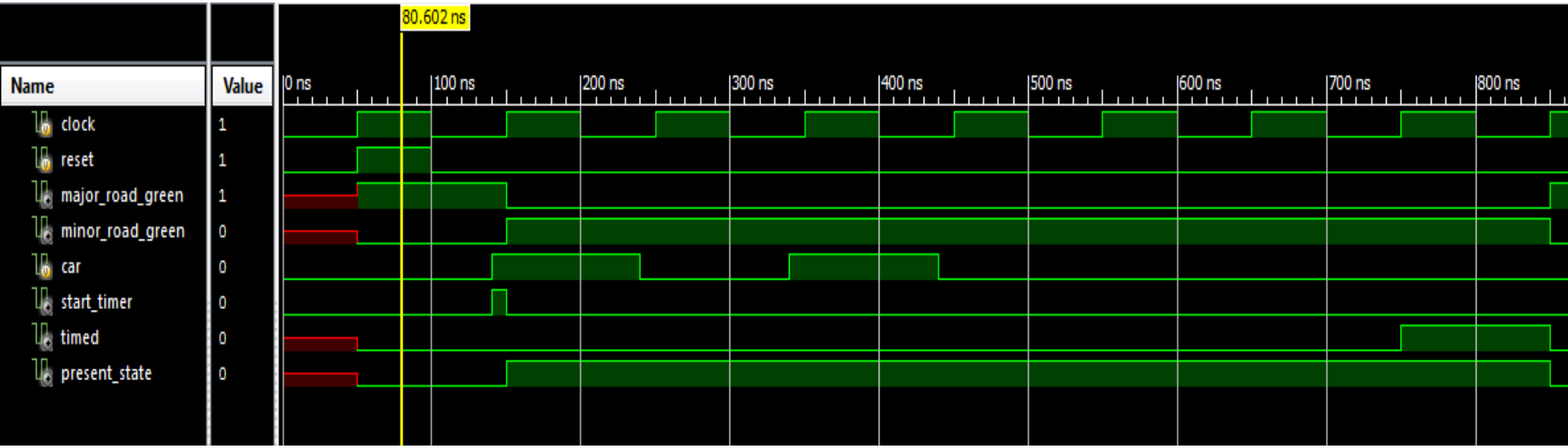


Verilog Code –Trafic Light 4/4

```
...
// DATAPATH STARTS: We have 2 register at datapath MAJOR and MINOR
//DATAPATH starts

always@(*)
begin
    case(pres_state)
         ST_R:
          begin
             MAJOR = 0;    // Moore output
             MINOR = 1;    // Moore output
          end
       default:
          begin
             MAJOR = 1;    // Moore output
             MINOR = 0;    // Moore output
          end
    endcase
end

//DATAPATH ends

endmodule
```



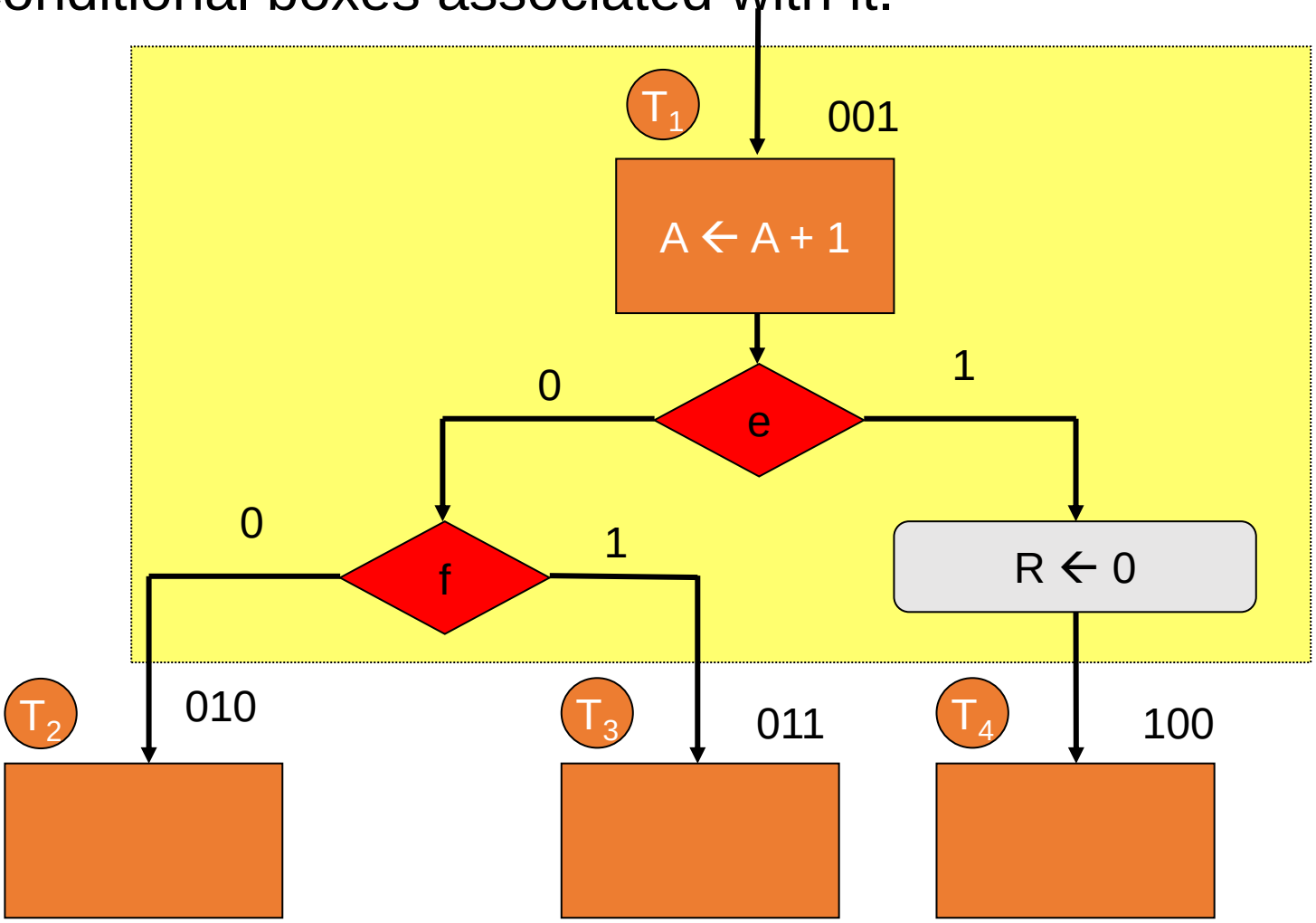
Traffic Control Example - Simulation



- Note 1: 0 is state Green and 1 is state Red for **`present_state`**.
- Note 2: Second time the "**`car`**" signal is asserted, the system is not affected, because the system is designed in this way (i.e., the **`car`** signal is effective only when the present state is Green).
- Please see the files "TrafficLight.v" and "Counter.v" for verilog codes of the traffic control example.

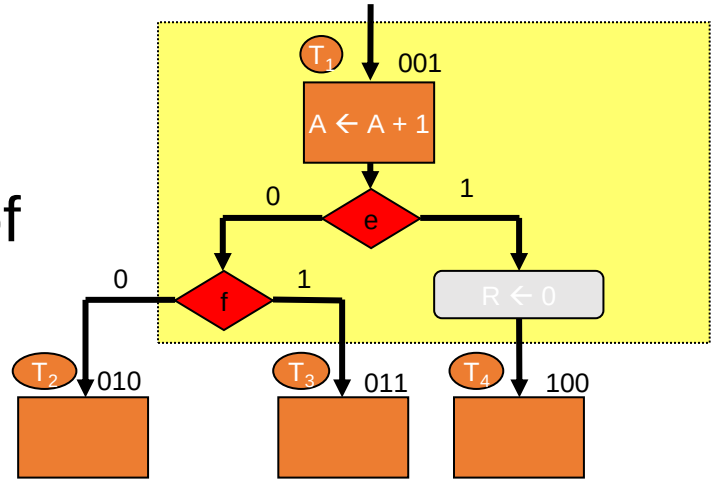
ASM Block

- A structure consisting of one state box and all the decision and conditional boxes associated with it.



ASM Block

- One input path, any number of exit paths
- Each ASM block describes the state of a system during one clock-pulse interval
  - The register operations within the state and conditional boxes in the example are executed with a common clock pulse when the system in this state (e.g. T_1 in the example)
  - The same clock pulse transfer the system controller to one of the next states (e.g. T_2, T_3, or T_4)

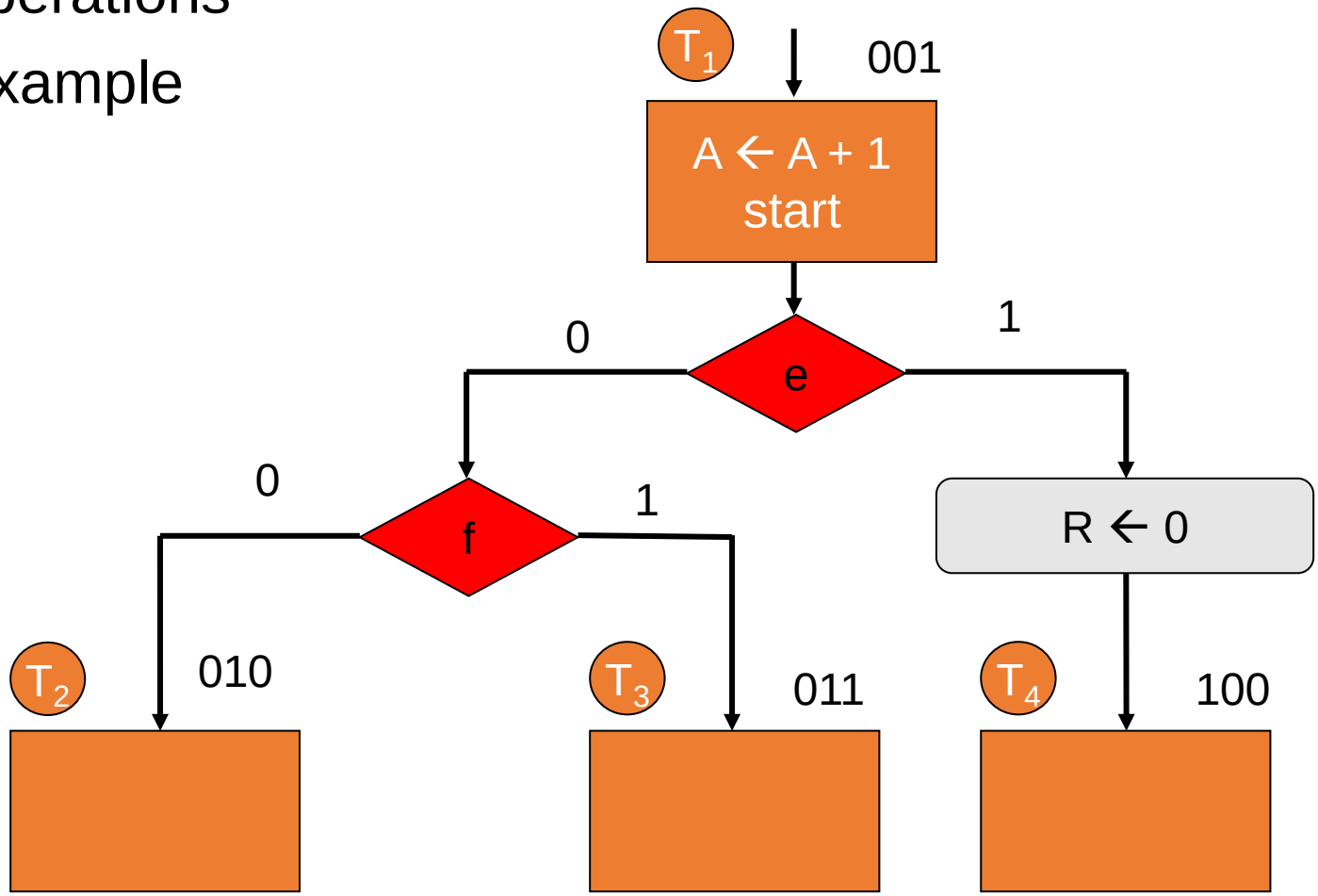


Timing Considerations 1/4

- The pulses of the common clock are applied to
  - registers in the datapath
  - all flip-flops in the control
- We can assume that inputs are also synchronized with the clock
  - Since they are the outputs of another circuit working with the same common clock.
  - Synchronous inputs

Timing Considerations 2/4

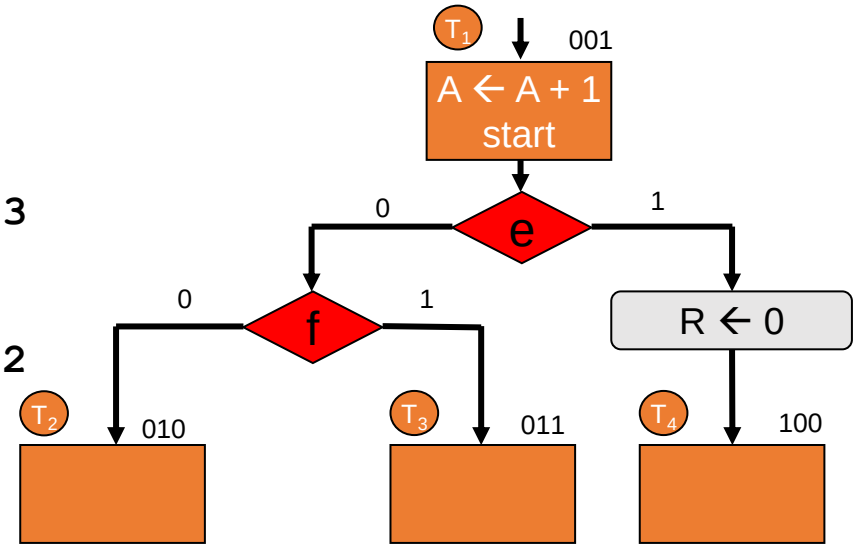
- Major difference between a conventional flow chart and ASM chart is in the time relations among the various operations
- Example



Timing Considerations 3/4

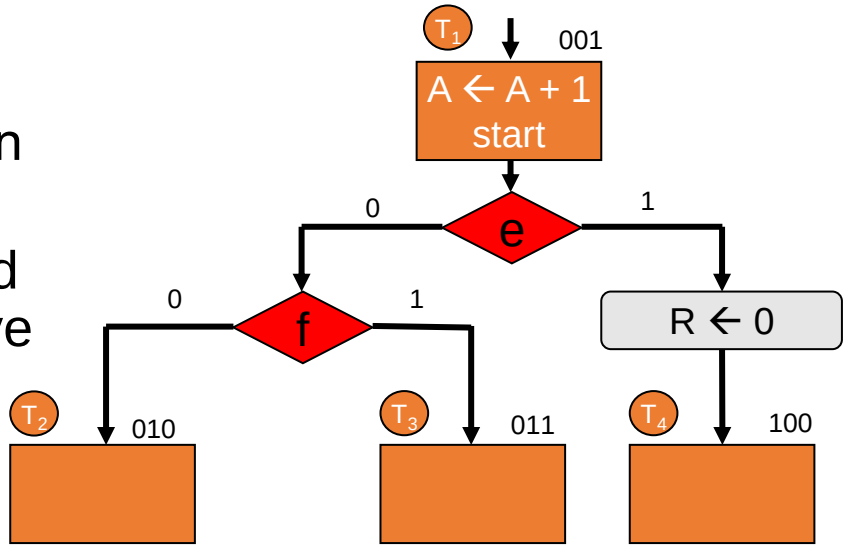
- If it were a conventional flowchart

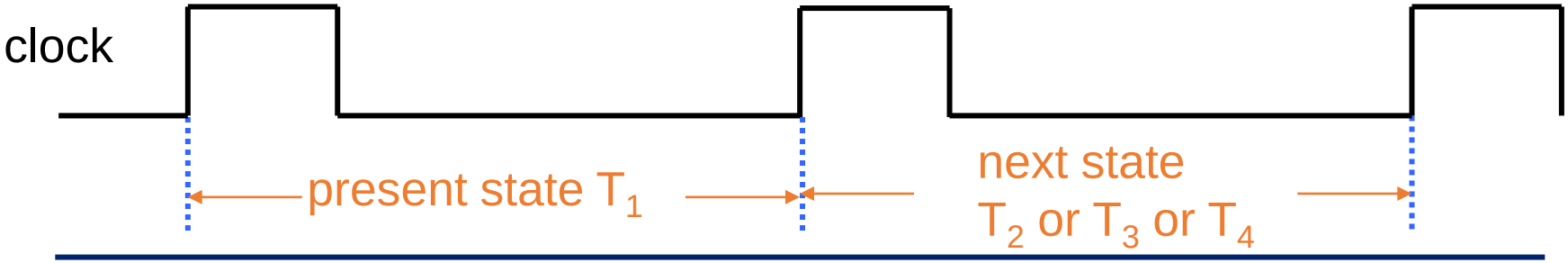
```
1. A ← A + 1
2. start = 1
3. if e = 1 then
     R ← 0
     next state is T4
4. else
     if f = 1 then
       next state is T3
     else
       next state is T2
```



Timing Considerations 4/4

- But, in ASM chart, interpretation is different
  - all operations in a block occur in synchronism with the clock
  - "start" is asserted in T_1
  - input signals "e" and "f" are checked in T_1
  - The following operations are executed simultaneously during the next positive edge of clock
    - A ← A + 1
    - R ← 0 (if e = 1)
    - control transfer to the next state





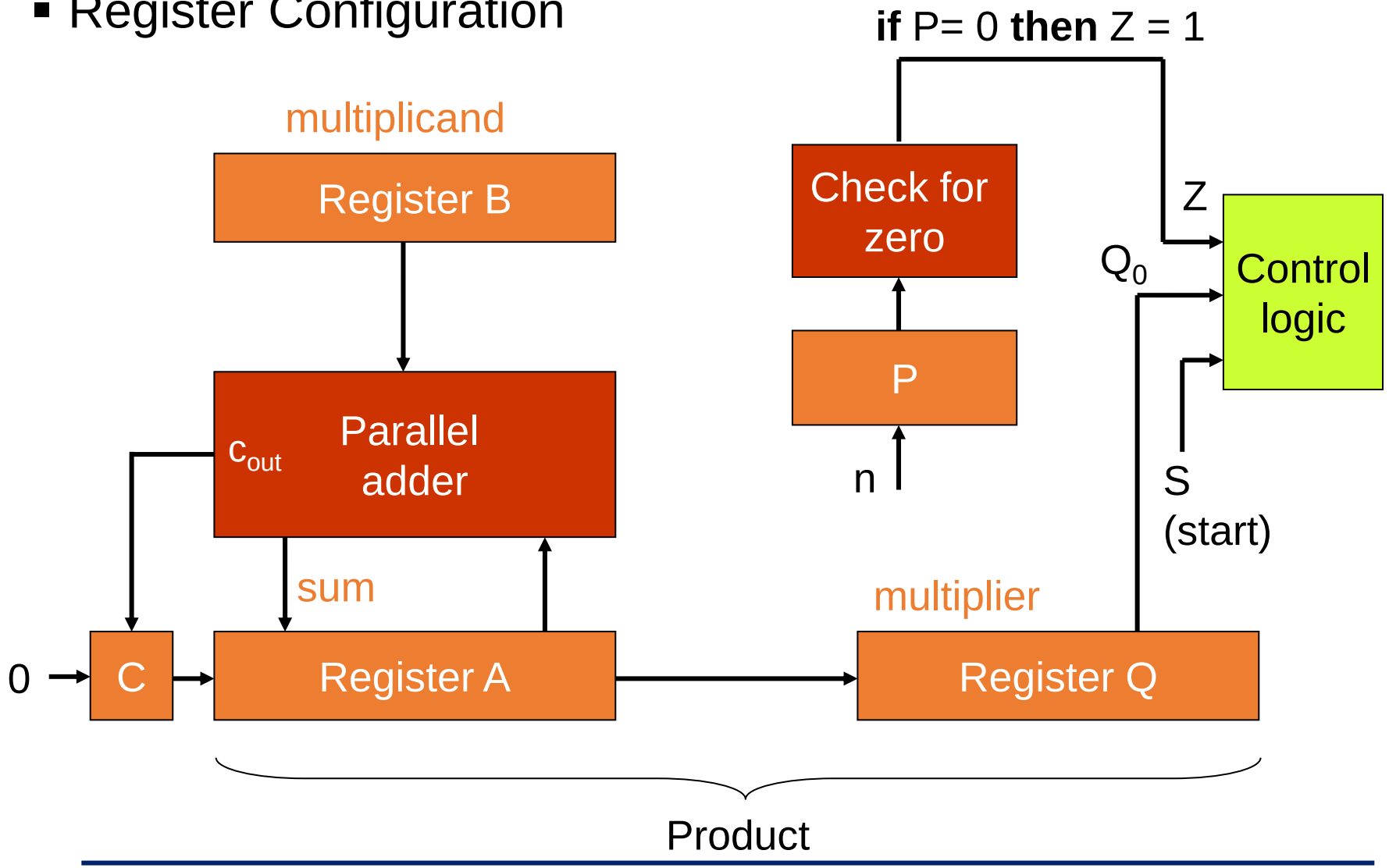
Example: Binary Multiplier

- Sequential multiplier
- Algorithm: successive additions and shifting

multiplicand				y_3	y_2	y_1	y_0
multiplier			×	x_3	x_2	x_1	x_0
partial product			$x_0 y_3$	$x_0 y_2$	$x_0 y_1$	$x_0 y_0$	
		$x_1 y_3$	$x_1 y_2$	$x_1 y_1$	$x_1 y_0$		
	$x_2 y_3$	$x_2 y_2$	$x_2 y_1$	$x_2 y_0$			
+	$x_2 y_3$	$x_3 y_2$	$x_3 y_1$	$x_3 y_0$			
	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
	product							

Schematic of Binary Multiplier

- Register Configuration



Hardware Algorithm for n-bit Multiplier

Input: X, Y, $n = \lceil \log_2 X \rceil$

Output: $Z = X \times Y$

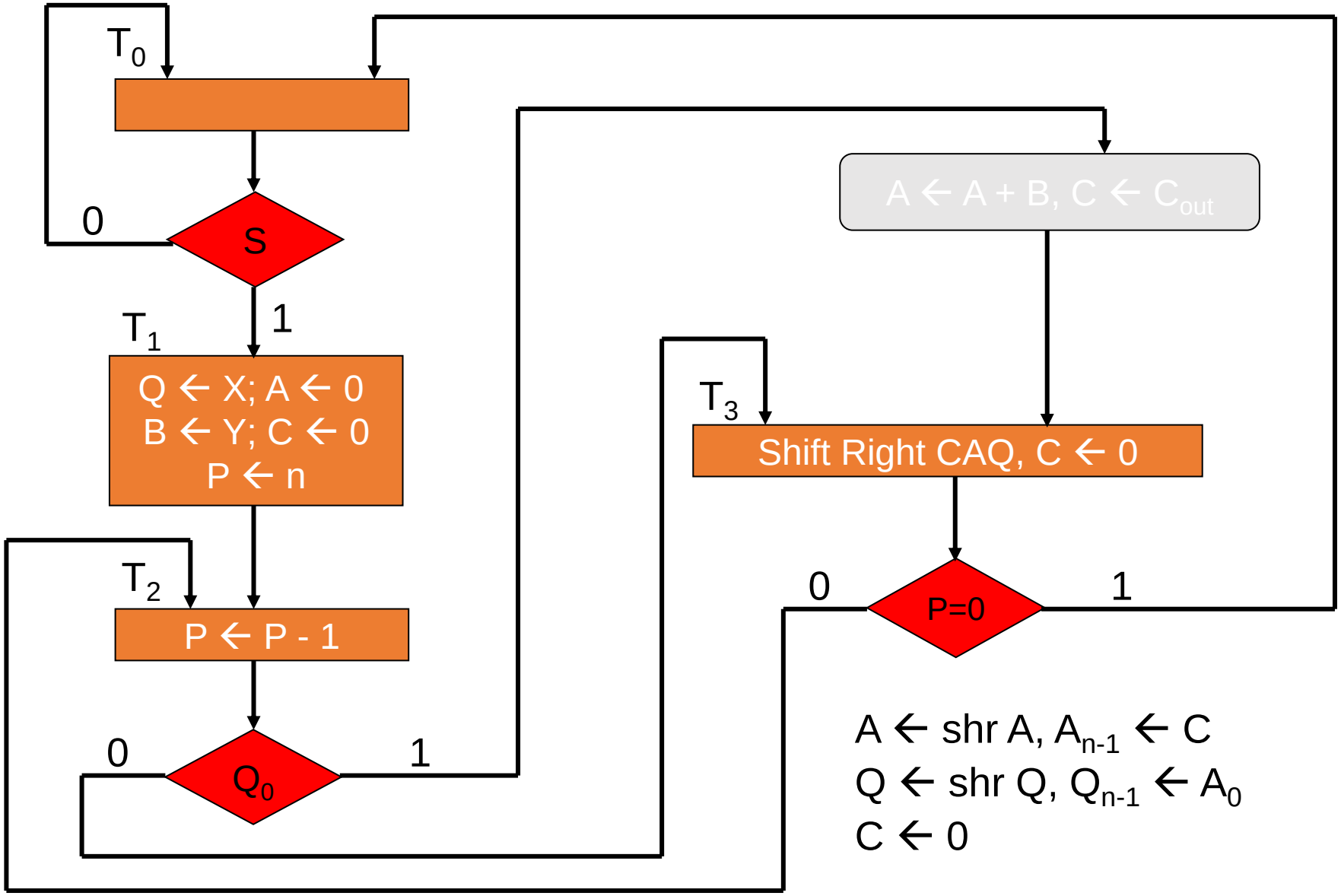
Step 0. if (Start == 0) do nothing
else go to Step 1;

Step 1. Q ← X; B ← Y; A ← 0; C ← 0; P ← n;
go to Step 2;

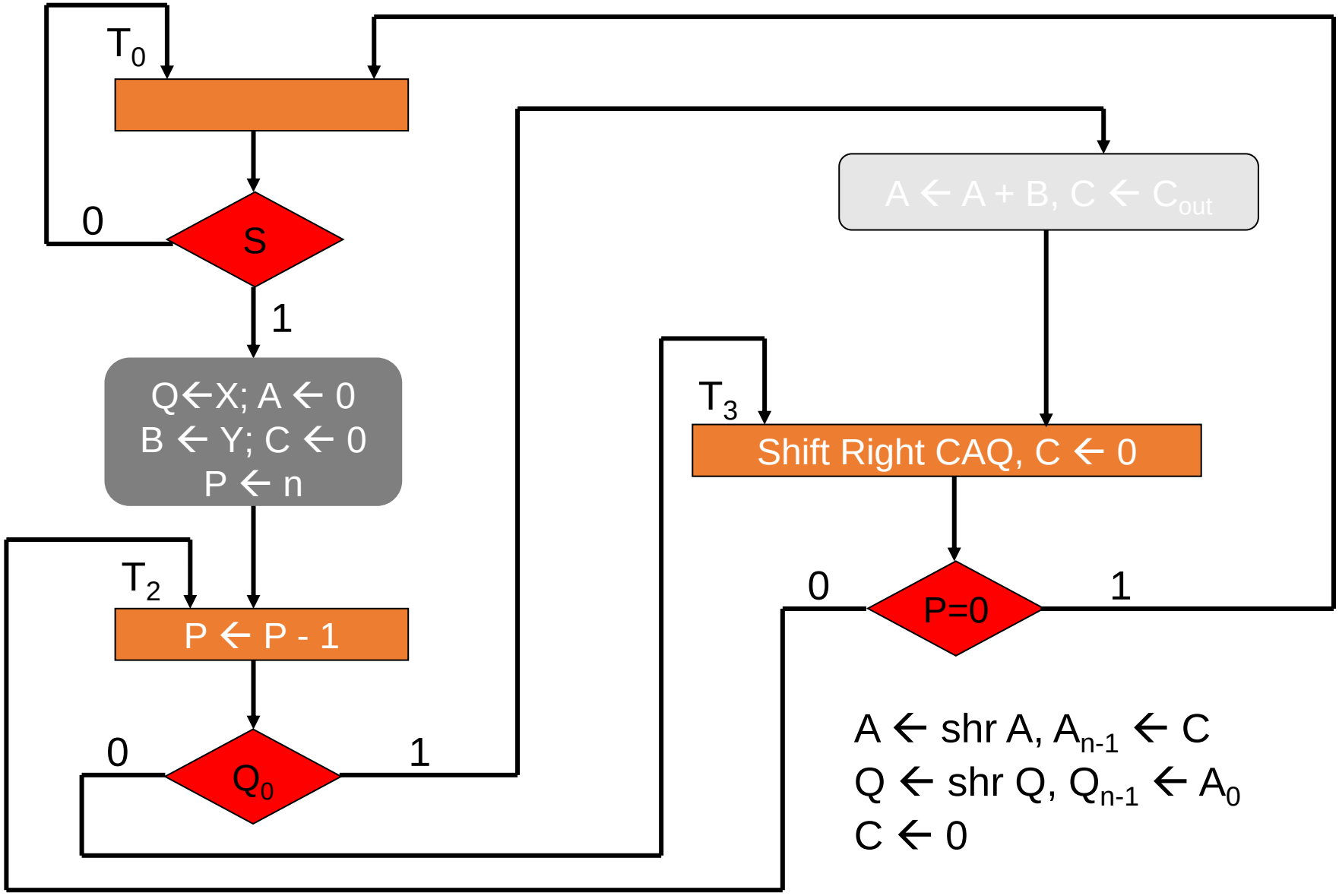
Step 2. P ← P-1;
if(Q_0 == 1)
A ← A+B; C ← Carry(A+B);
go to Step 3;

Step 3. CAQ ← shr(CAQ); C ← 0;
if (P == 0) go to Step 0;
else go to Step 2;

Example: ASM Chart



Example: ASM Chart



Multiplier – Verilog Code 1/7

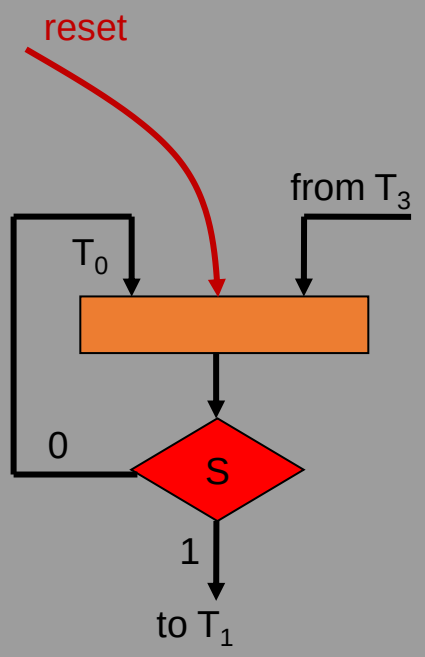
```
module TopModule(x, y, n, clk, start, reset, out, state, p);

input [3:0] x, y, n;
input clk, start, reset;

output [8:0] out;
output [1:0] state;
output [3:0] p;   // counter

reg [8:0] out;
reg [1:0] state;

reg [3:0] b, a, q, p;
reg c;

parameter t0 = 0, t1 = 1, t2 = 2, t3 = 3;
parameter zero = 0;
...
```

Multiplier – Verilog Code 2/7

```
...
module TopModule(x, y, n, clk, start, reset, out, state, p);
...
// combinational part for output
always @ *
     begin
          case (state)
               t0:
                    out = {c, a, q};
               t1:
                    out = {c, a, q};
                    t2:
                    out = {c, a, q};
               t3:
                    out = {c, a, q};
               default:
                    out = {c, a, q};
          endcase
     end
...
```

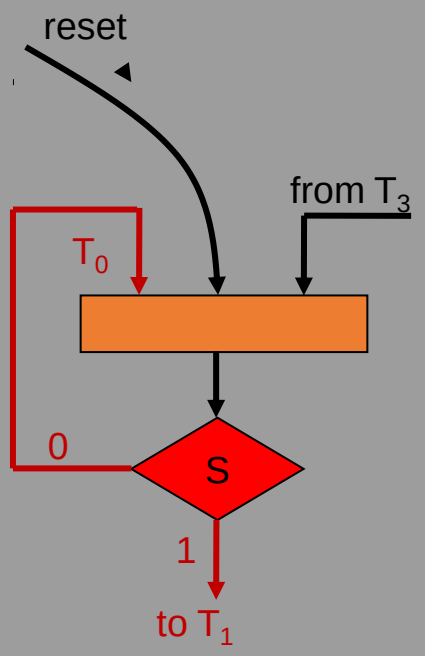
Multiplier – Verilog Code 3/7

```
...
// Sequential part
always @(posedge clk or posedge reset)
  if (reset)
  begin
        state <= t0;
        a <= zero;
        b <= zero;
        c <= zero;
        q <= zero;
        p <= zero;   // counter
  end
  else
...
```



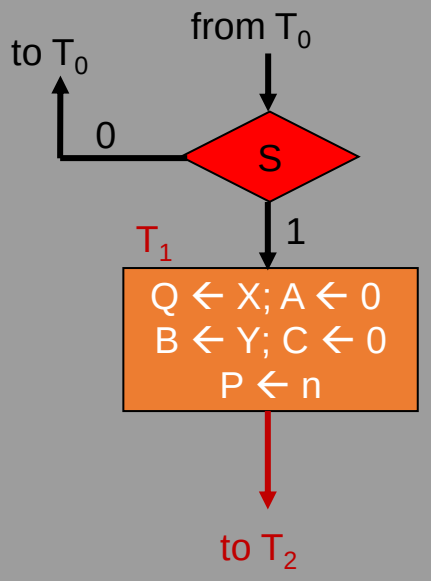
Multiplier – Verilog Code 4/7

```
...
// Sequential part (cont.)
always @(posedge clk or posedge reset)
  if (reset)
  ...
  else
        case (state)
            t0:
                if(start)
                    state <= t1;
                else
                    state <= t0;

...
```



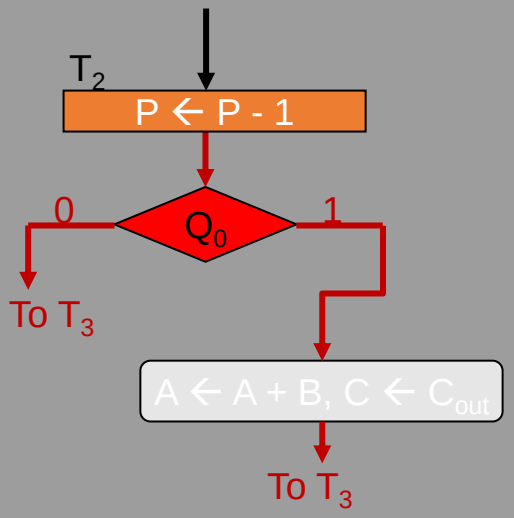
Multiplier – Verilog Code 5/7

```
...
always @(posedge clk or posedge reset)
  if (reset)
  ...
   else
        case (state)
        ...
            t1:
                begin
                    b <= y;
                     q <= x;
                     p <= n;
                     state <= t2;
                end

...
```



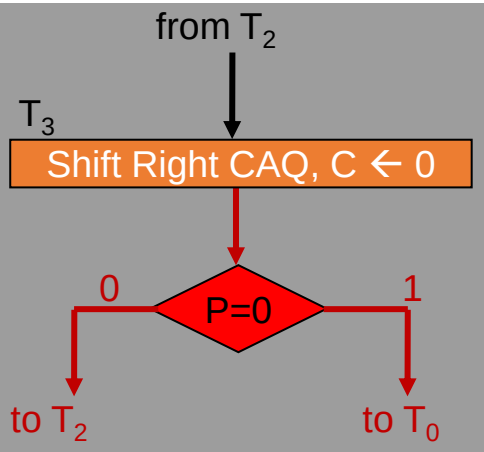
Multiplier – Verilog Code 6/7

```
...
always @(posedge clk or posedge reset)
  if (reset)
  ...
  else
      case (state)
      ...
          t2:
               begin
                     p <= p - 1;
                     if(q[0] == 1'b0)
                         state <= t3;
                     else
                         begin
                              {c, a} <= a + b;
                              state <= t3;
                          end
                end
...
```



Multiplier – Verilog Code 7/7

```
...
always @(posedge clk or posedge reset)
   if (reset)
   ...
   else
       case (state)
       ...
           t3:
               begin
                       {c, a, q} <= {1'b0, c, a, q[3:1]};
                       if(p == 0)
                           state <= t0;
                       else
                           state <= t2;
                   end
                   default:
                       if(start) state <= t1;
                       else state <= t0;
           endcase
endmodule
```



Multiplier – Verilog Code 1/6

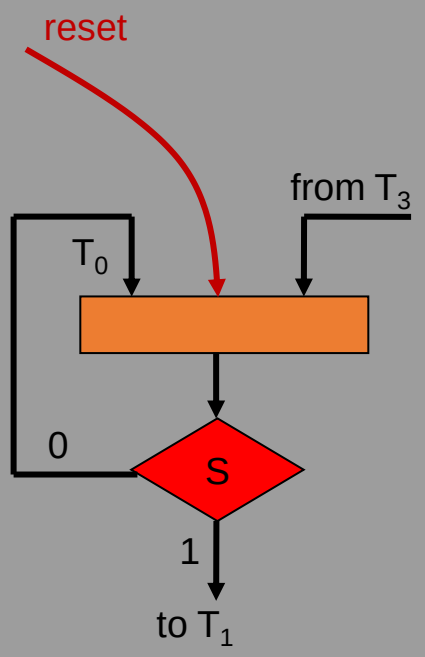
```
module TopModule(x, y, n, clk, start, reset, out, state, p);

input [3:0] x, y, n;
input clk, start, reset;

output [8:0] out;
output [1:0] state;
output [3:0] p;       // counter

reg [8:0] out;
reg [1:0] state;

reg [3:0] b, a, q, P;
reg c;

parameter t0 = 0, t2 = 2, t3 = 3;   // we skip the state T1
parameter zero = 0;
...
```

Multiplier – Verilog Code 2/6

```
...
module TopModule(x, y, n, clk, start, reset, out, state, p);
...
// combinational part for output
always @ *
     begin
          case (state)
               t0:
                    out = {c, a, q};
                t2:
                    out = {c, a, q};
               t3:
                    out = {c, a, q};
               default:
                    out = {c, a, q};
          endcase
     end
...
```

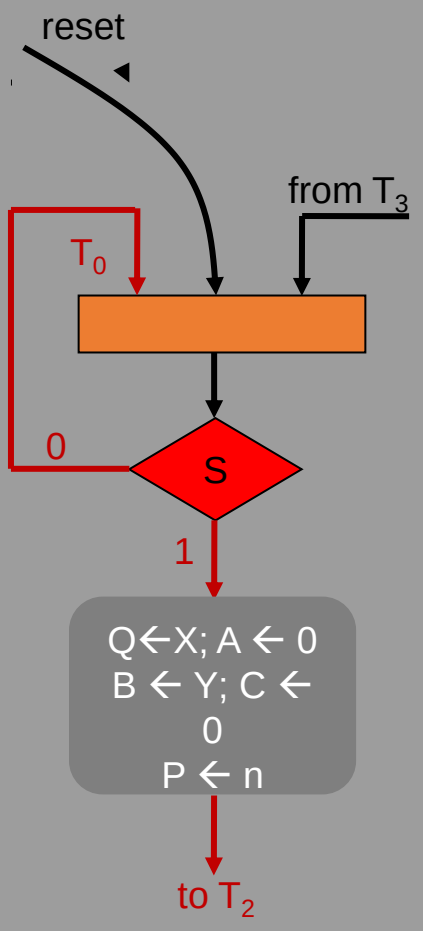
Multiplier – Verilog Code 3/6

```
...
// Sequential part
always @(posedge clk or posedge reset)
  if (reset)
  begin
        state <= t0;
        a <= zero;
        b <= zero;
        c <= zero;
        q <= zero;
        p <= zero;
  end
  else
...
```



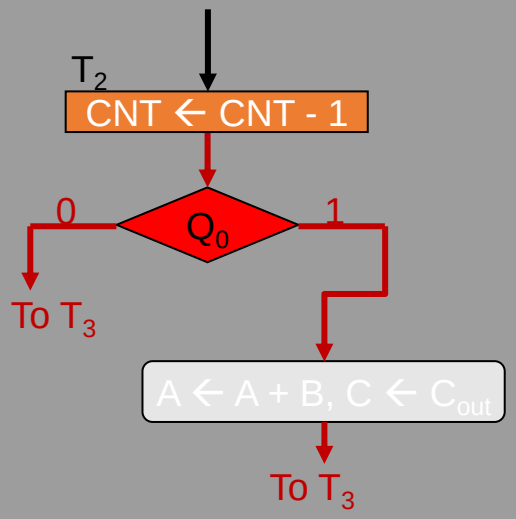
Multiplier – Verilog Code 4/6

```
...
// Sequential part (cont.)
always @(posedge clk or posedge reset)
  if (reset)
  ...
  else
        case (state)
            t0:
                if(start)
                    b <= y;
                    q <= x;
                    P <= n;
                    state <= t2;
                else
                    state <= t0;

...
```



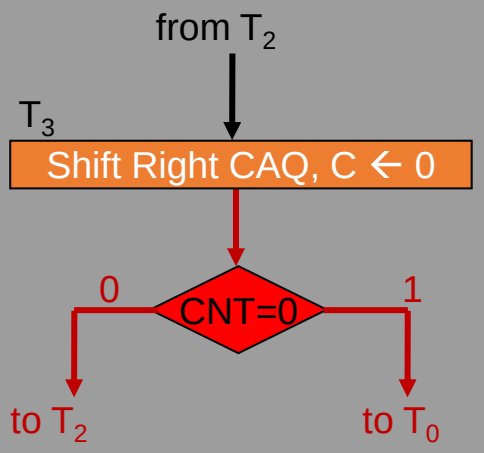
Multiplier – Verilog Code 5/6

```
...
always @(posedge clk or posedge reset)
  if (reset)
  ...
  else
      case (state)
      ...
          t2:
               begin
                    p <= p - 1;
                    if(q[0] == 1'b0)
                        state <= t3;
                    else
                        begin
                            {c, a} <= a + b;
                            state <= t3;
                         end
                end
...
```



Multiplier – Verilog Code 6/6

```
...
always @(posedge clk or posedge reset)
   if (reset)
   ...
   else
       case (state)
       ...
           t3:
               begin
                       {c, a, q} <= {1'b0, c, a, q[3:1]};
                       if(p == 0)
                           state <= t0;
                       else
                           state <= t2;
                   end
                   default:
                       if(start) state <= t1;
                       else state <= t0;
           endcase
endmodule
```



Multiplier – Test Code 1/4

```
`timescale 1ns / 1ps
module topmoduleTest;

  // Inputs
  reg [3:0] x;
  reg [3:0] y;
  reg [3:0] n;
  reg clk;
  reg start;
  reg reset;

  // Outputs
  wire [8:0] out;
  wire [1:0] state;
  wire [3:0] p;
...
```

Multiplier – Test Code 2/4

```
`timescale 1ns / 1ps
module topmoduleTest;
...
// Instantiate the Unit Under Test (UUT)
  TopModule uut (
        .x(x),
        .y(y),
        .n(n),
        .clk(clk),
        .start(start),
        .reset(reset),
        .out(out),
        .state(state),
        .p(p)
  );
...
```

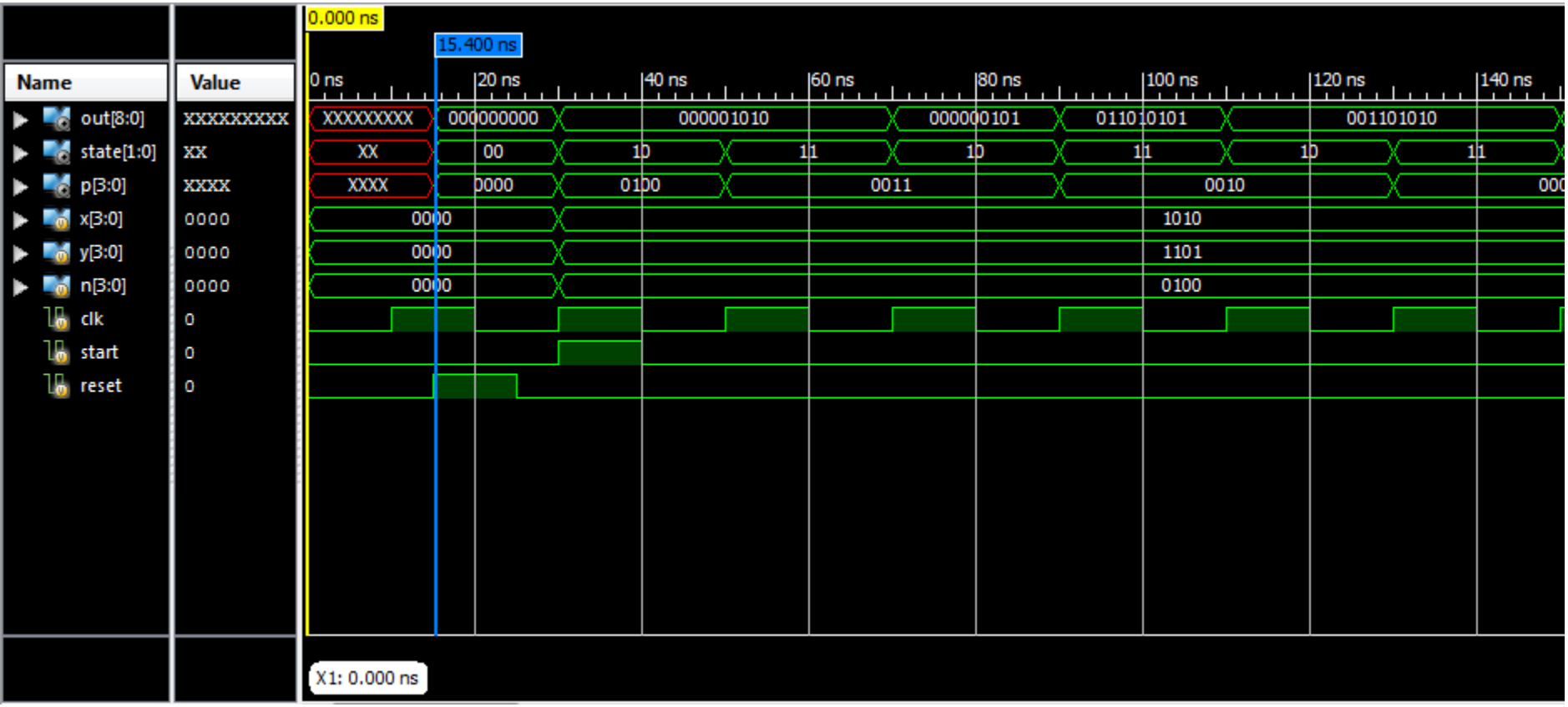
Multiplier – Test Code 3/4

```
`timescale 1ns / 1ps
module topmoduleTest;
...
initial begin
    // Initialize Inputs
    x = 0;
    y = 0;
    n = 0;
    clk = 0;
    start = 0;
    reset = 0;
...
```

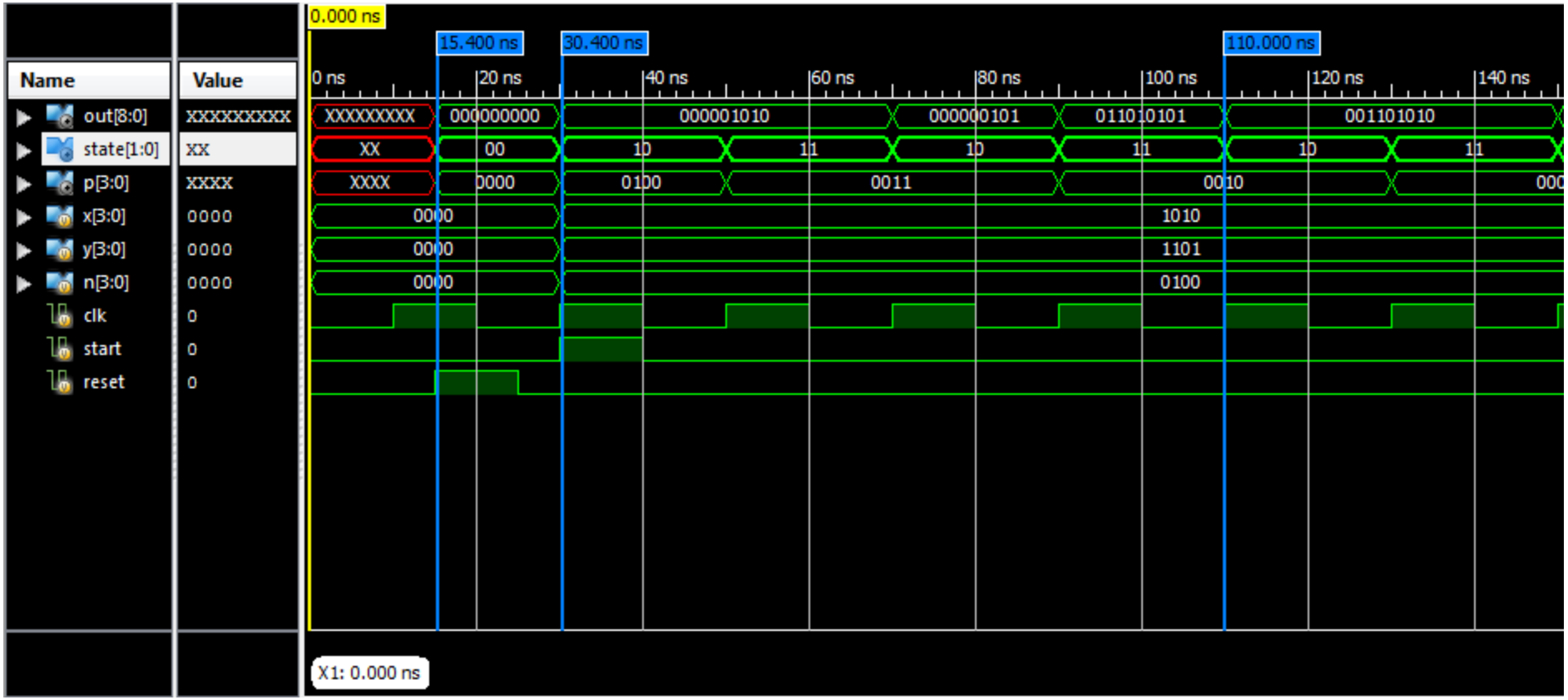
Multiplier – Test Code 4/4

```
`timescale 1ns / 1ps
module topmoduleTest;
...
initial begin
...
    // Wait 15 ns for global reset
    #15
    reset = 1;
    #10
    reset = 0;
    #5
    start = 1;
    n = 4;
    x = 10;
    y = 13;
    #10
    start = 0;
    #200;
end
always #10 clk = ~clk;
endmodule
```

Multiplier – Simulation 1/3



Multiplier – Simulation 2/3



Multiplier – Simulation 3/3