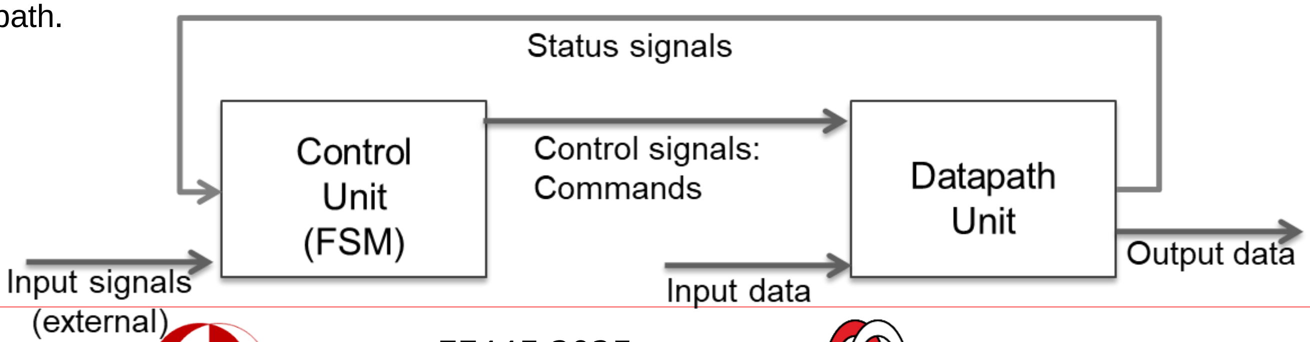
Algorithmic State Machine

Reference

- Digital Design, 4th Edition, M. Morris R. Mano,Michael D. Ciletti, 2007
- Chapter 8

Sequential Machine Implementation

- Binary information in a digital system:
  - Data: manipulated by arithmetic and logic operations implemented by adders, decoders, multiplexers, counters, shift registers
  - Control: Command signals that coordinate and execute the data processing
- Design of the digital system:
  - Data Path Unit:
    - Functional blocks
    - Operates on words of data.
    - Contains structures such as memories, registers, ALUs, and multiplexers.
    - Example: 32-bit ARM architecture, has a 32-bit datapath.
  - Control Unit:
    - receives the status signals (later: current instruction) from the datapath
    - Sends control signals (later: commands how to execute that instruction) to the datapath
    - produces multiplexer select, register enable, and memory write signals to control the operation of the datapath.



Finite State Machines

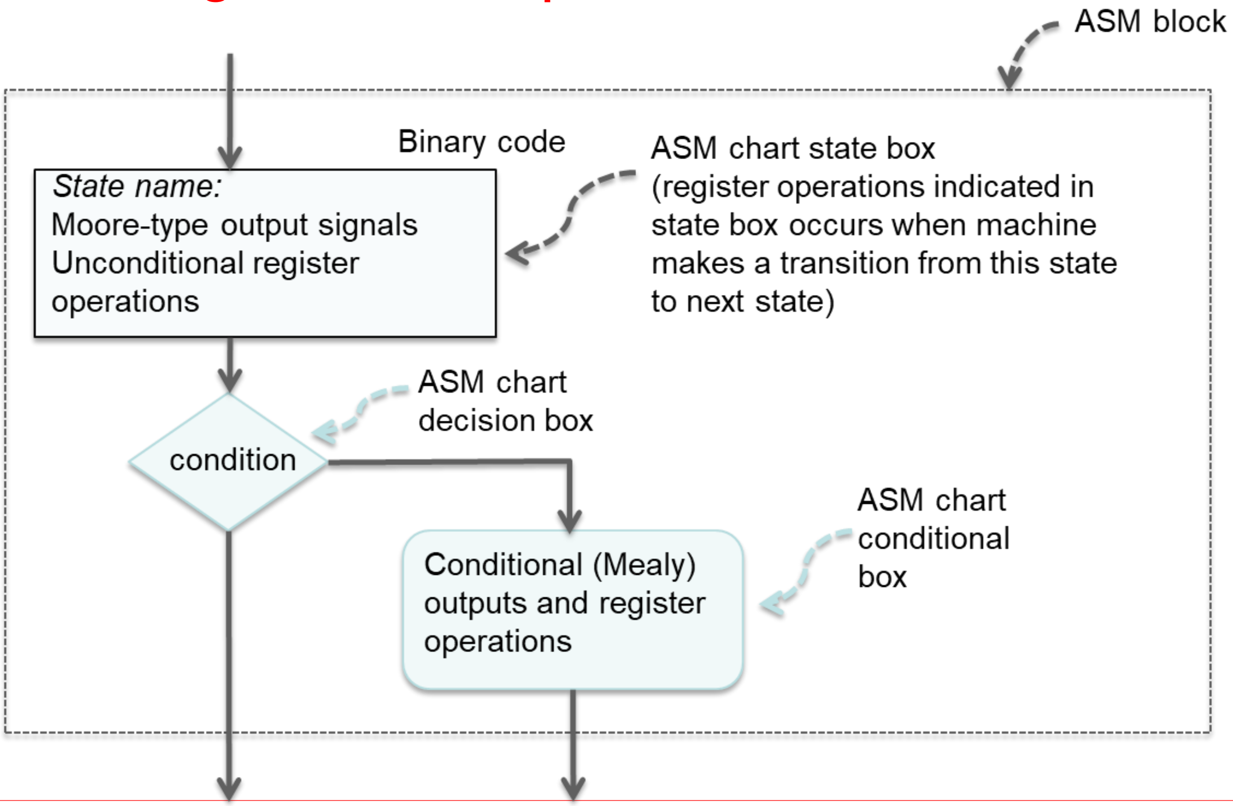
- Digital Systems and especially their Controllers can be described as Finite State Machines (FSMs).
- FSMs are sequential circuits.
- Finite State Machines can be represented using
  - **State Diagrams and State Tables** - suitable for simple digital systems with a relatively few inputs and outputs
  - **Algorithmic State Machine (ASM) Charts** - suitable for complex digital systems with a large number of inputs and outputs

Algorithmic State Machine (ASM)

- A special flowchart to define hardware algorithms
- Describes:
  - the sequence of events
  - the timing relationship between the states of the sequential controller
  - the events that occur while going from one state to another

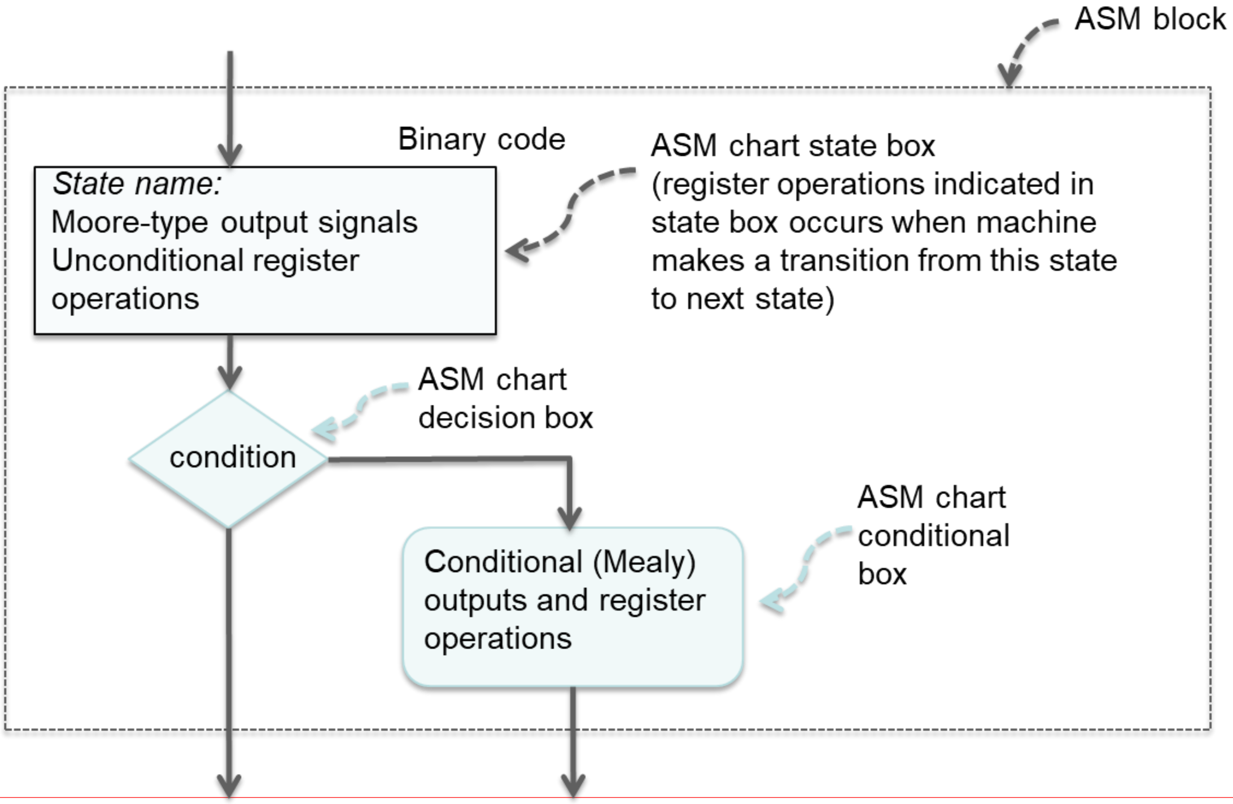
ASM Block

- Describes the state of the system and the machine operation during one clock pulse interval


ASM block
Binary code
State name:
Moore-type output signals
Unconditional register operations
ASM chart state box (register operations indicated in state box occurs when machine makes a transition from this state to next state)
ASM chart decision box
condition
ASM chart conditional box
Conditional (Mealy) outputs and register operations

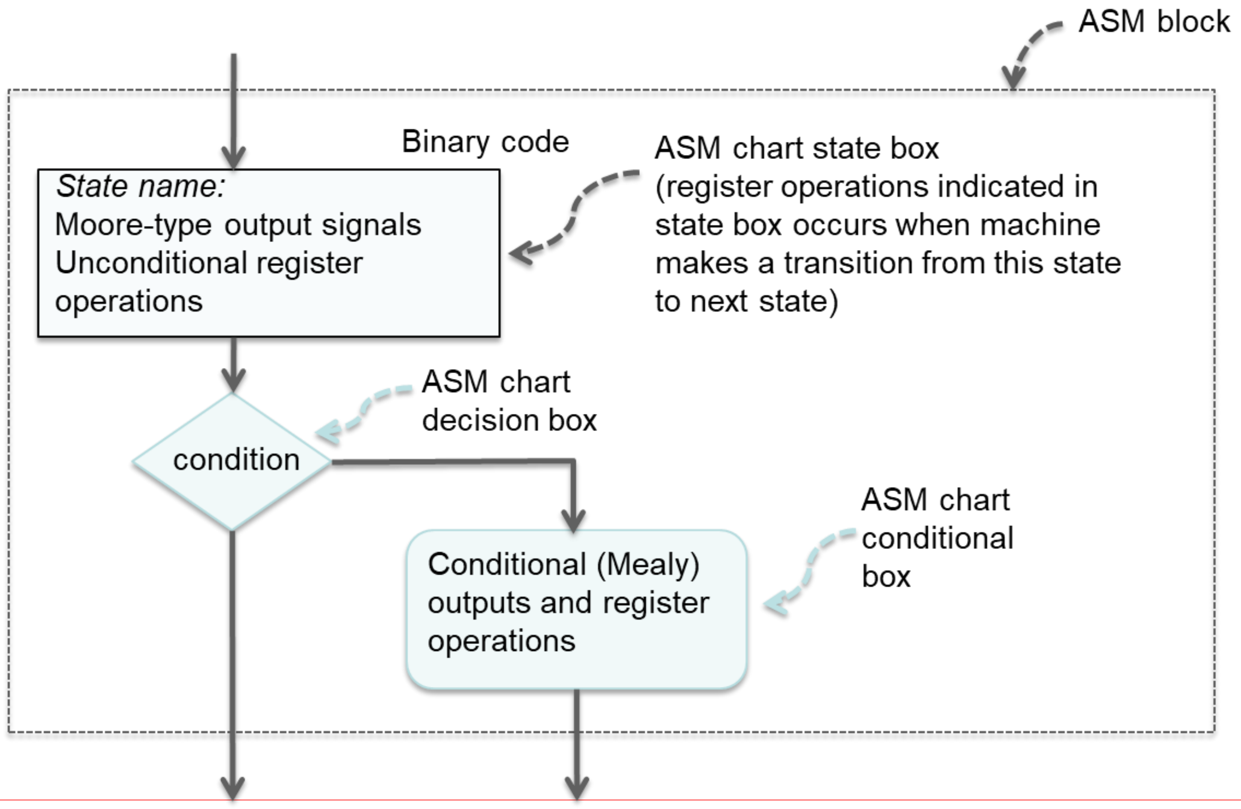
ASM Block

- Contains exactly one state box together with decision boxes and conditional output boxes associated with that state


ASM block
Binary code
State name:
Moore-type output signals
Unconditional register operations
ASM chart state box (register operations indicated in state box occurs when machine makes a transition from this state to next state)
ASM chart decision box
condition
Conditional (Mealy) outputs and register operations
ASM chart conditional box

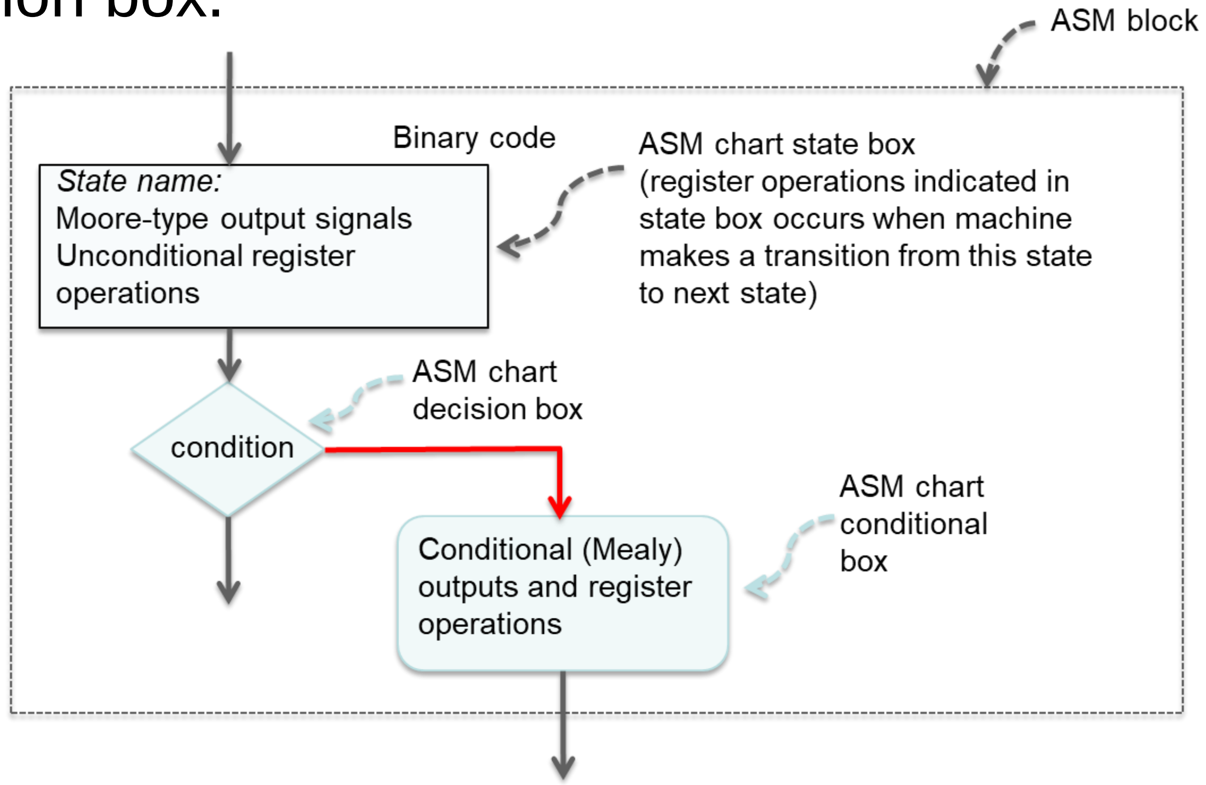
ASM Block

- Has exactly one entrance path and one or more exit paths.

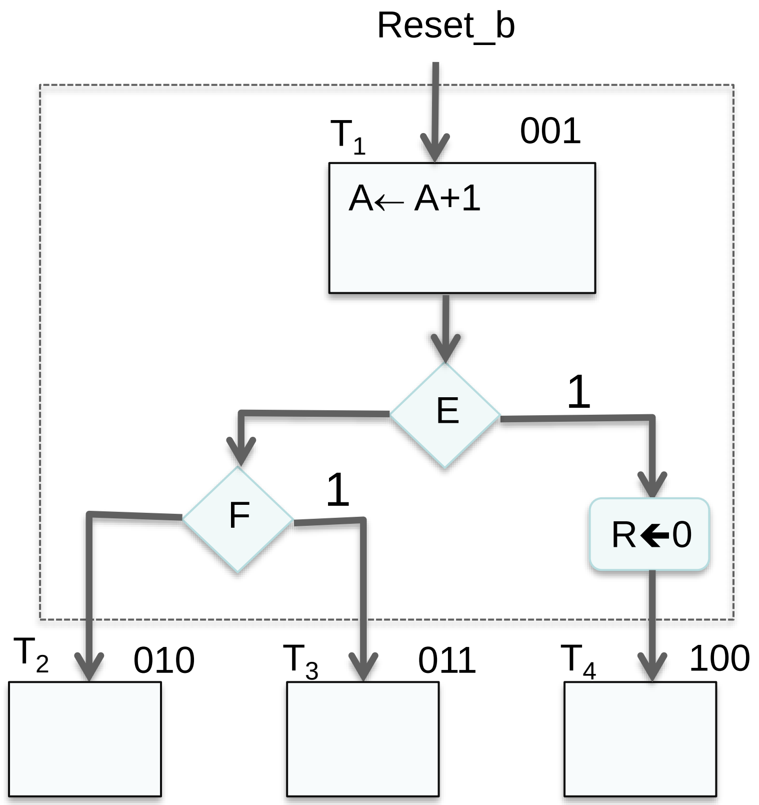

ASM block
Binary code
State name:
Moore-type output signals
Unconditional register operations
ASM chart state box (register operations indicated in state box occurs when machine makes a transition from this state to next state)
ASM chart decision box
condition
ASM chart conditional box
Conditional (Mealy) outputs and register operations

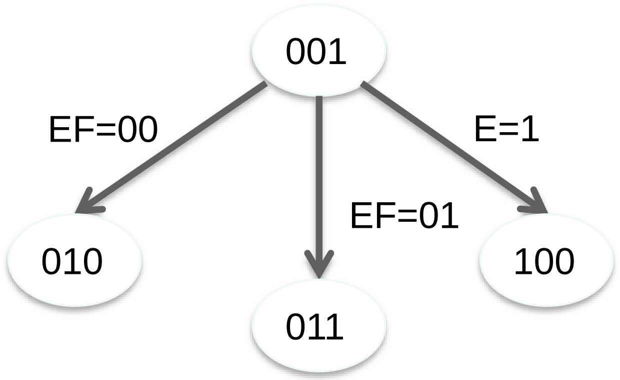
ASM Block

- The input path to a conditional box must come from a decision box.

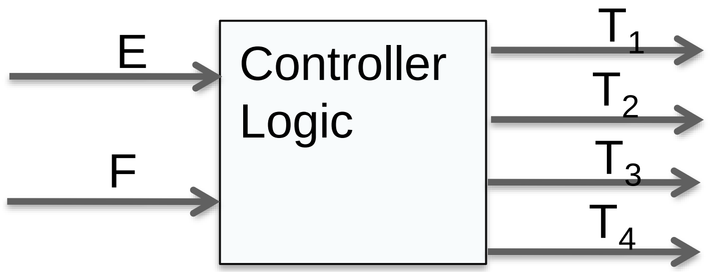

ASM block
Binary code
State name:
Moore-type output signals
Unconditional register operations
ASM chart state box (register operations indicated in state box occurs when machine makes a transition from this state to next state)
ASM chart decision box
condition
Conditional (Mealy) outputs and register operations
ASM chart conditional box

Example



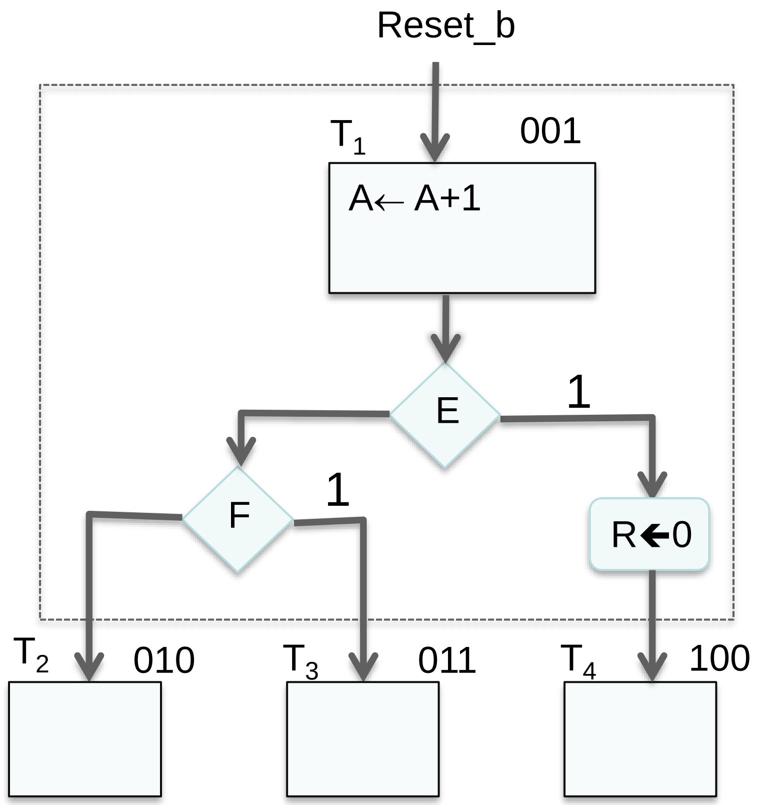


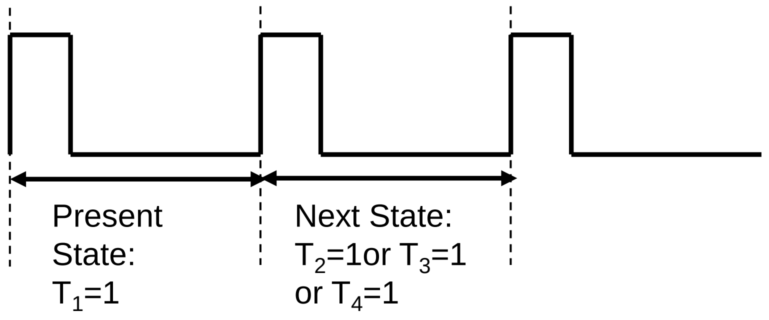
Equivalent State Transition Diagram



Only one of the outputs is 1 during a clock pulse

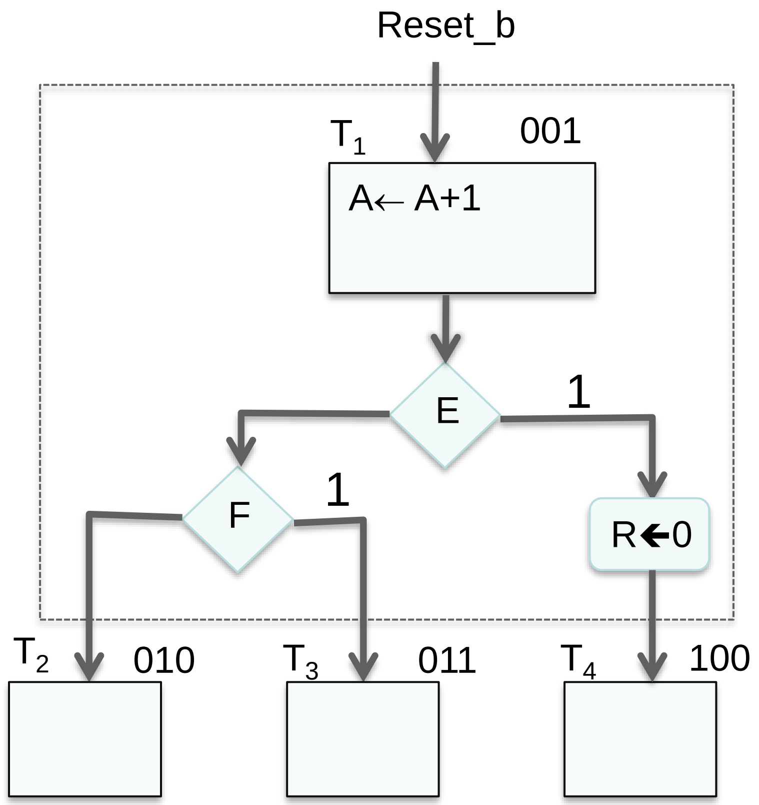
Example



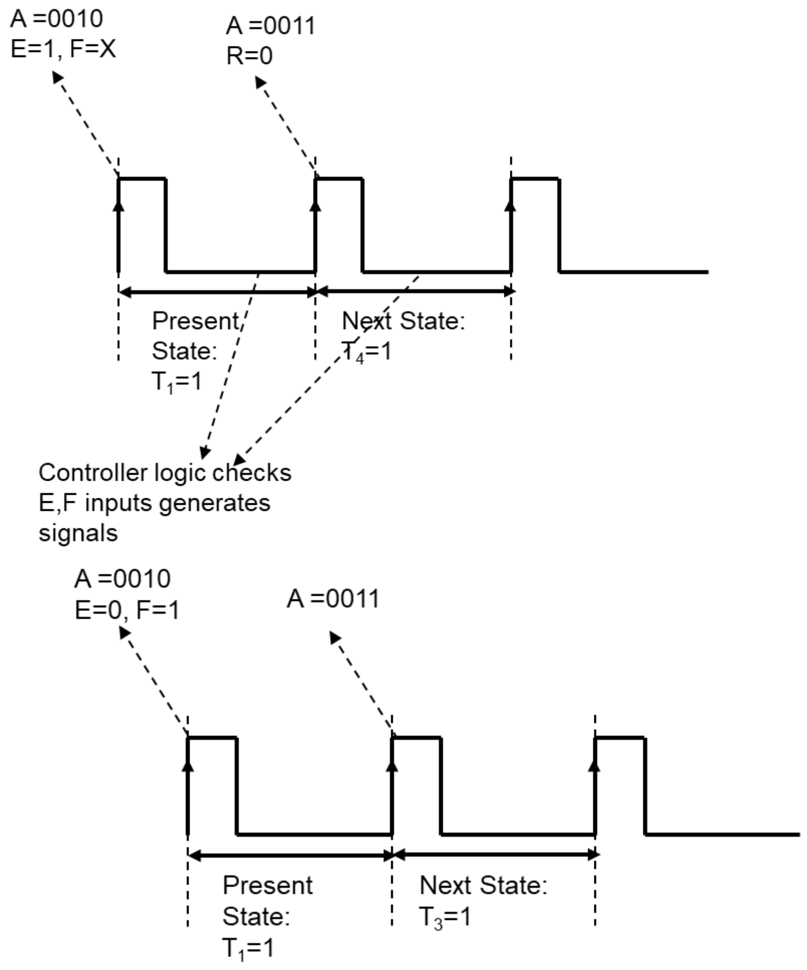


Timing: All unconditional/conditional register operations specified within the block T_1 occur synchronously at the edge transition of the same clock pulse while the system goes from T_1 to the next state

Example

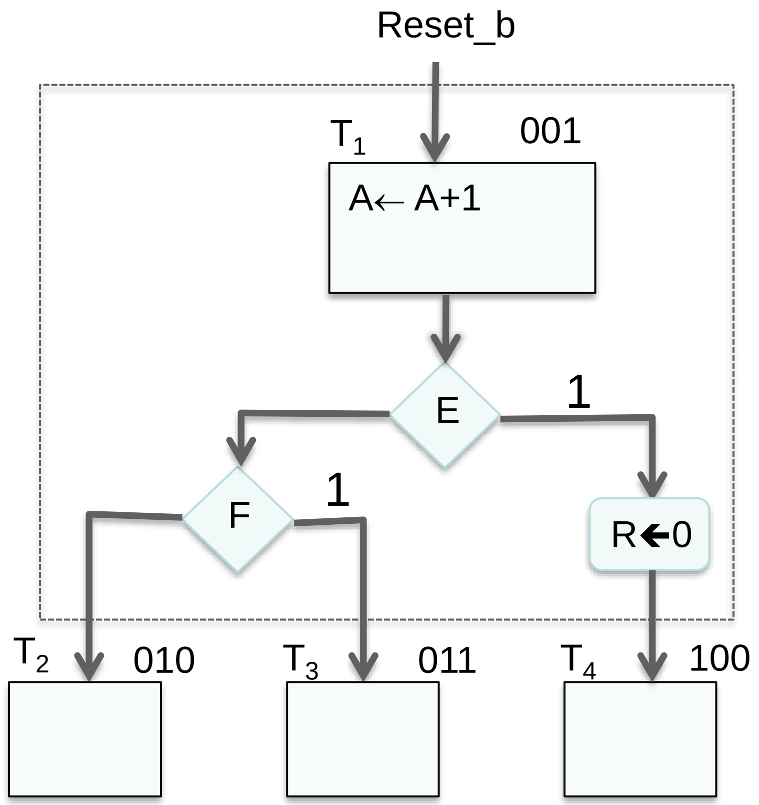


Reset_b
T_1
001
$A \leftarrow A+1$
E
1
F
1
$R \leftarrow 0$
T_2
010
T_3
011
T_4
100



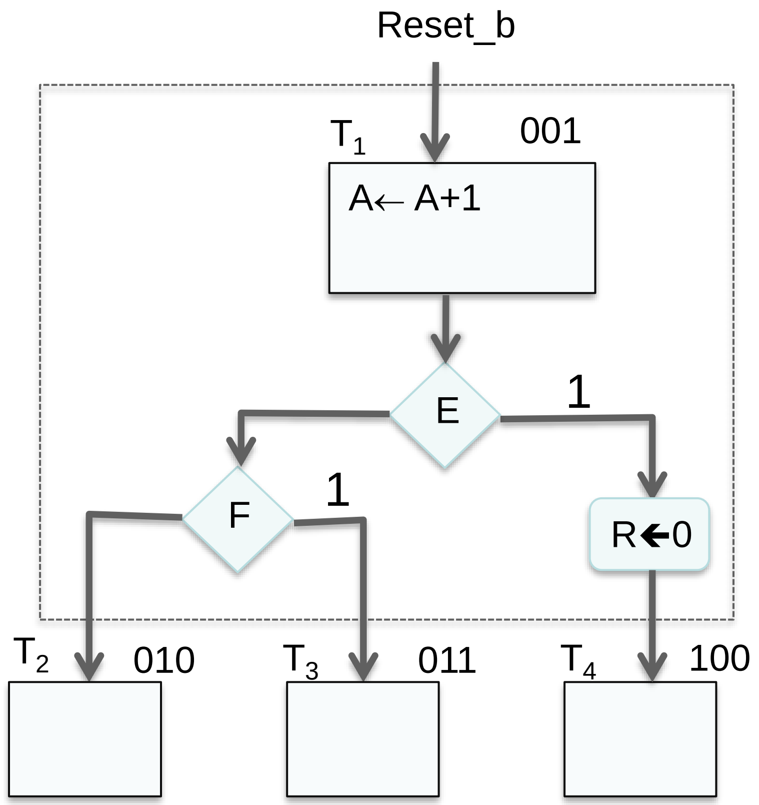
A =0010
E=1, F=X
A =0011
R=0
Present State: T_1=1
Next State: T_4=1
Controller logic checks E,F inputs generates signals
A =0010
E=0, F=1
A =0011
Present State: T_1=1
Next State: T_3=1

Summary

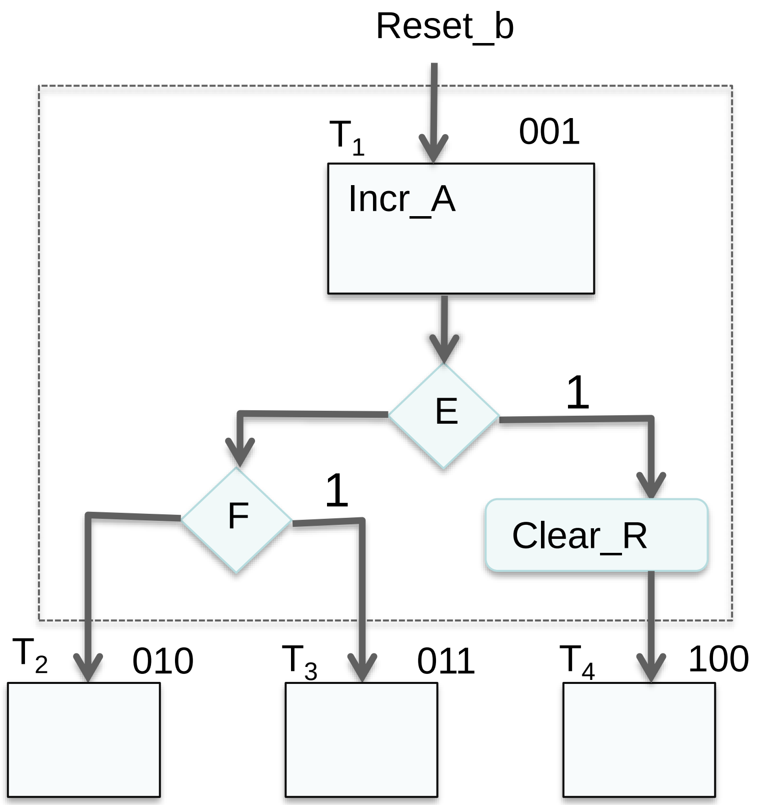


- Transitions are governed by clock
- Transitions take place between ASM blocks
- For a given input combination, there is one unique exit path from the current ASM block
- The exit path of an ASM block must always lead to a state box.
- The state box can be the state box of the current ASM block or a state box of another ASM block.

ASM Chart Styles

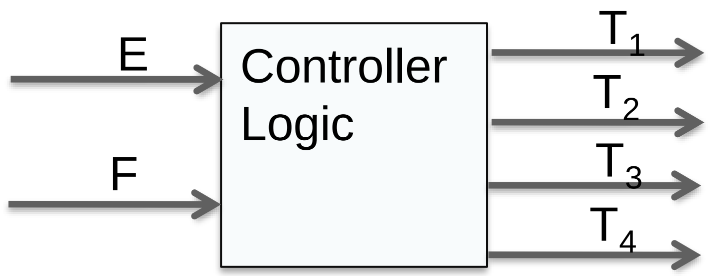


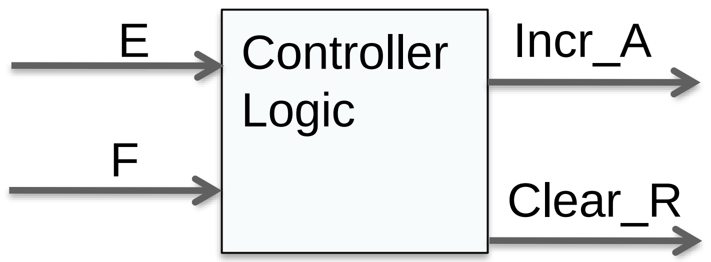
ASM chart showing Datapath and controller



ASM chart showing control signals

Controller Styles





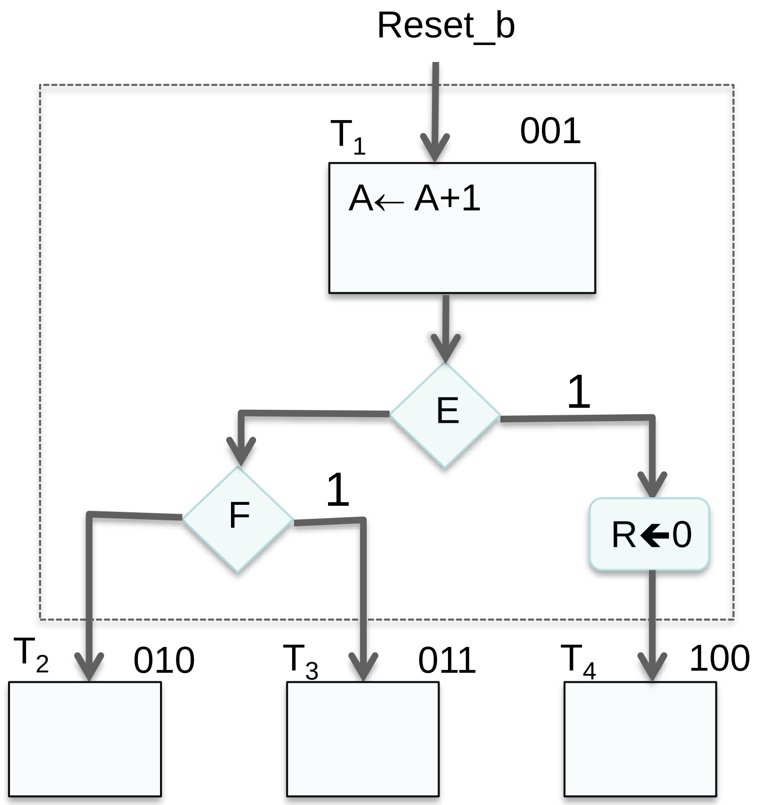
- Controller Logic creates the State Signals
- More structured

Controller Logic creates the Control (output) Signals

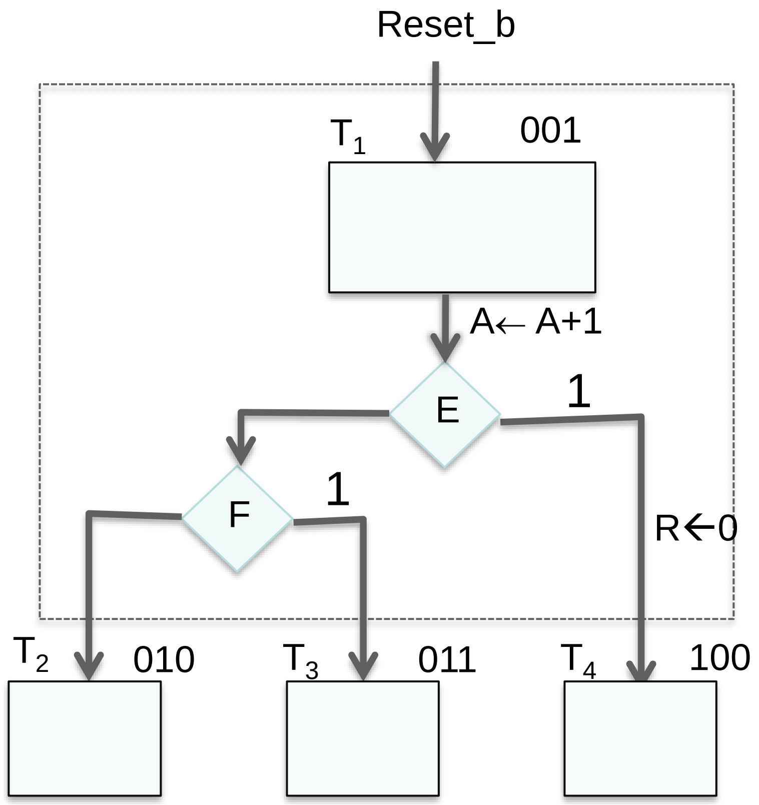
Algorithmic State Machine and Datapath (ASMD) Chart

- Associates register operations with state transitions rather than with states
  - Register operations are not shown within a state box
  - The edges are annotated with register operations that are concurrent with state transition indicated by the edge
  - Conditional boxes identify the signals which control the register operations that annotate the edges of the chart.

ASMD Chart



ASM chart

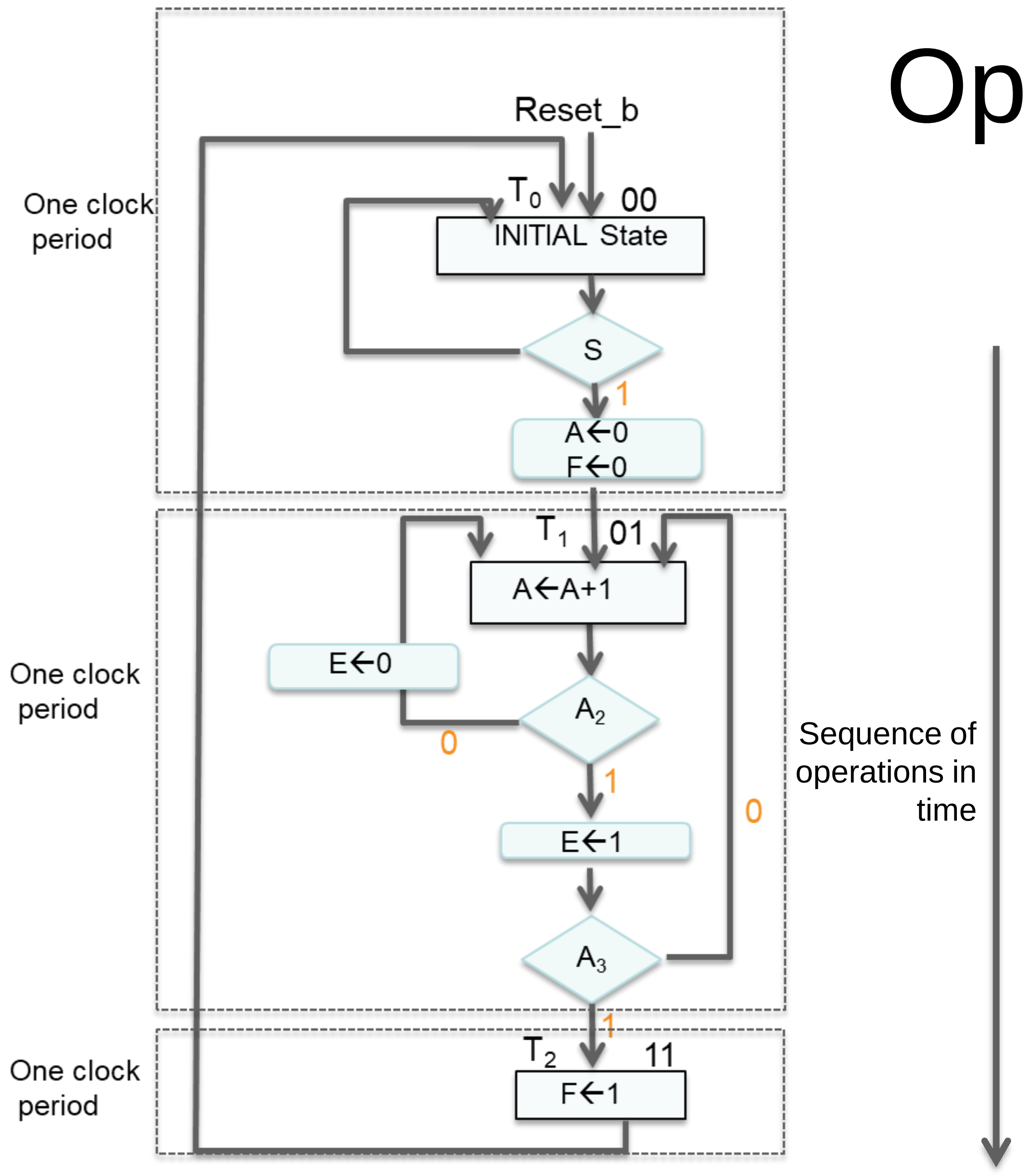


ASMD chart

Example 1

- Given: Two JK FFs E and F, and one four-bit binary counter A[3:0].
- A signal, Start, initiates the system's operation by clearing the counter A and FF F.
- At each **subsequent** clock pulse, the counter is incremented by 1 until the operations stop.
- Counter bits A_2 and A_3 determine the sequence of operations
- If A_2=0, E is cleared to 0 and the count continues.
- If A_2=1, E is set to 1 and
  - if A_3=0, the count continues
  - if A_3=1, F is set to 1 **on the next clock pulse** and the system stops counting (and goes back to the initial state).
- If Start = 0, the system remains in the initial state
- If Start =1, the operation cycle repeats

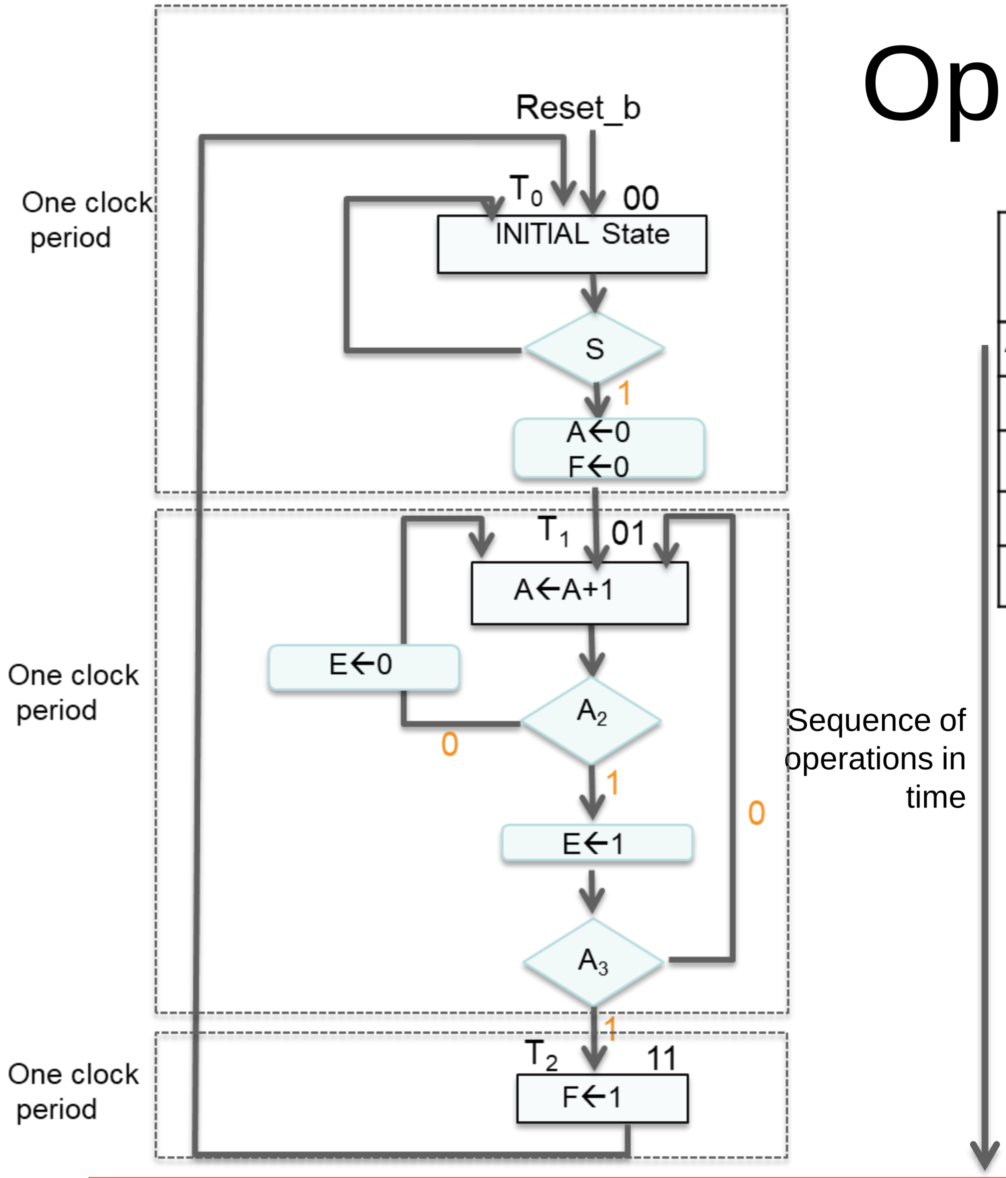
ASM Chart and Operation Sequence



- If Start = 0, the system remains in the initial state
- Start, initiates the system's operation: A←0, F ← 0
- At each **subsequent** clock pulse, the counter is incremented by 1 until the operations stop.
- If A_2=0, E←0 (E is reset), the count continues.
- If A_2=1, E←1 (E is set) and
  - if A_3=0, the count continues
  - if A_3=1, F is set to 1 **on the next clock pulse** and the system stops counting (and goes back to the initial state).
- If Start = 0, the system remains in the initial state
- If Start =1, the operation cycle repeats

ASM Chart and Operation Sequence

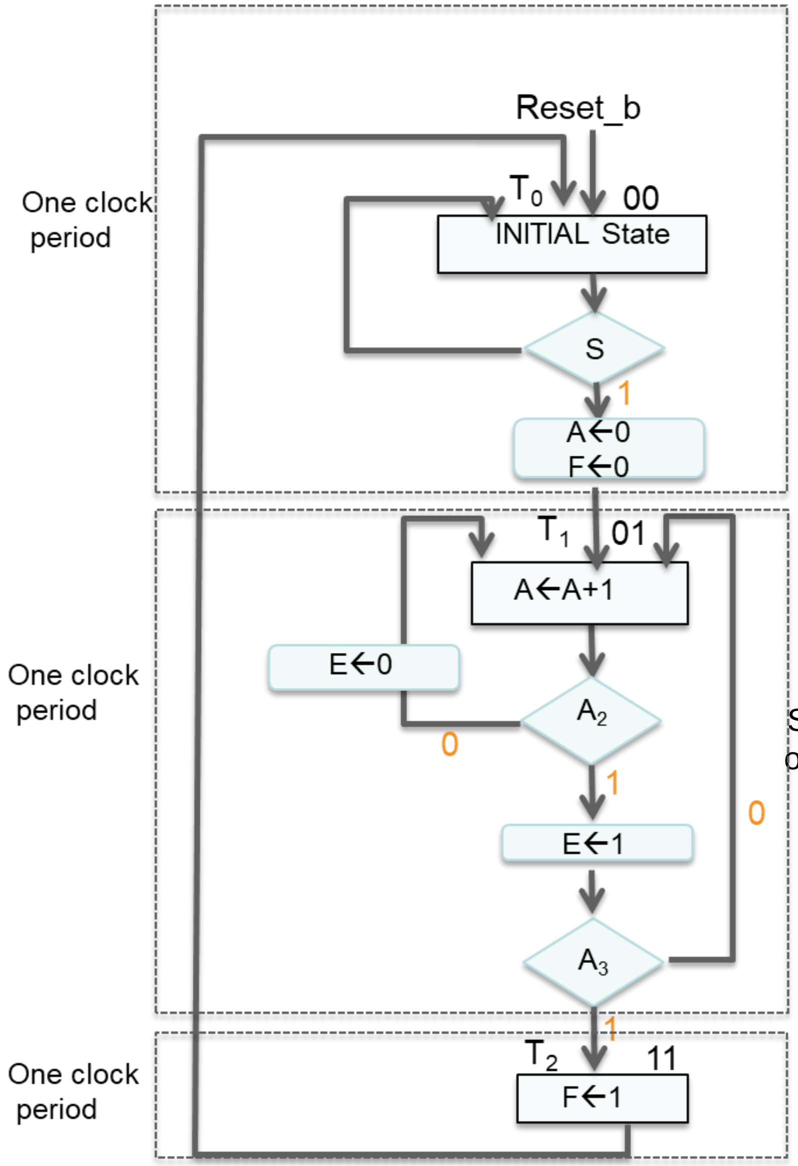
S←1, A ← 0, F ← 0



Counter				Flip Flops		Present State	Conditions	Next State	Command Signals
A_3	A_2	A_1	A_0	E	F				
0	1	0	0	0	0	T_1	A_2=1, A_3=0	T_1	set_E incr_A
0	1	0	1	1	0				
1	0	0	0	1	0	T_1	A_2=0, A_3=1	T_1	clr_E incr_A
1	0	0	1	0	0				

- All decision boxes are checked with the values at the beginning of the clock pulse
- Register/FF operations are executed during the transition of the clock pulse and the updated values are effective at the beginning of the next clock pulse

ASM Chart and Operation Sequence

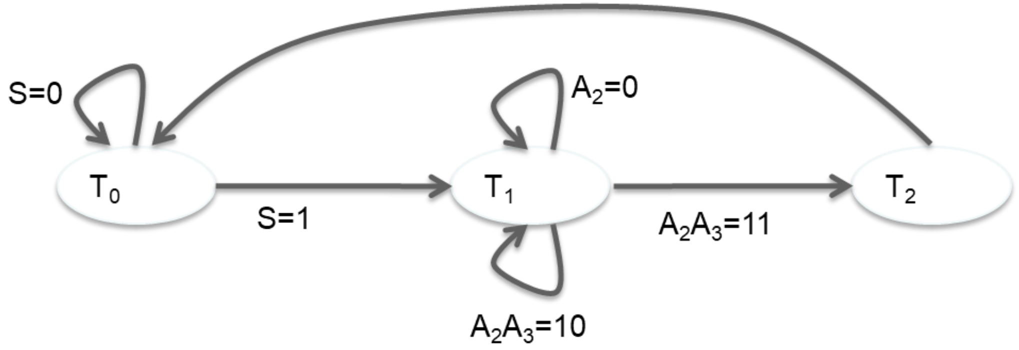


Sequence of operations in time

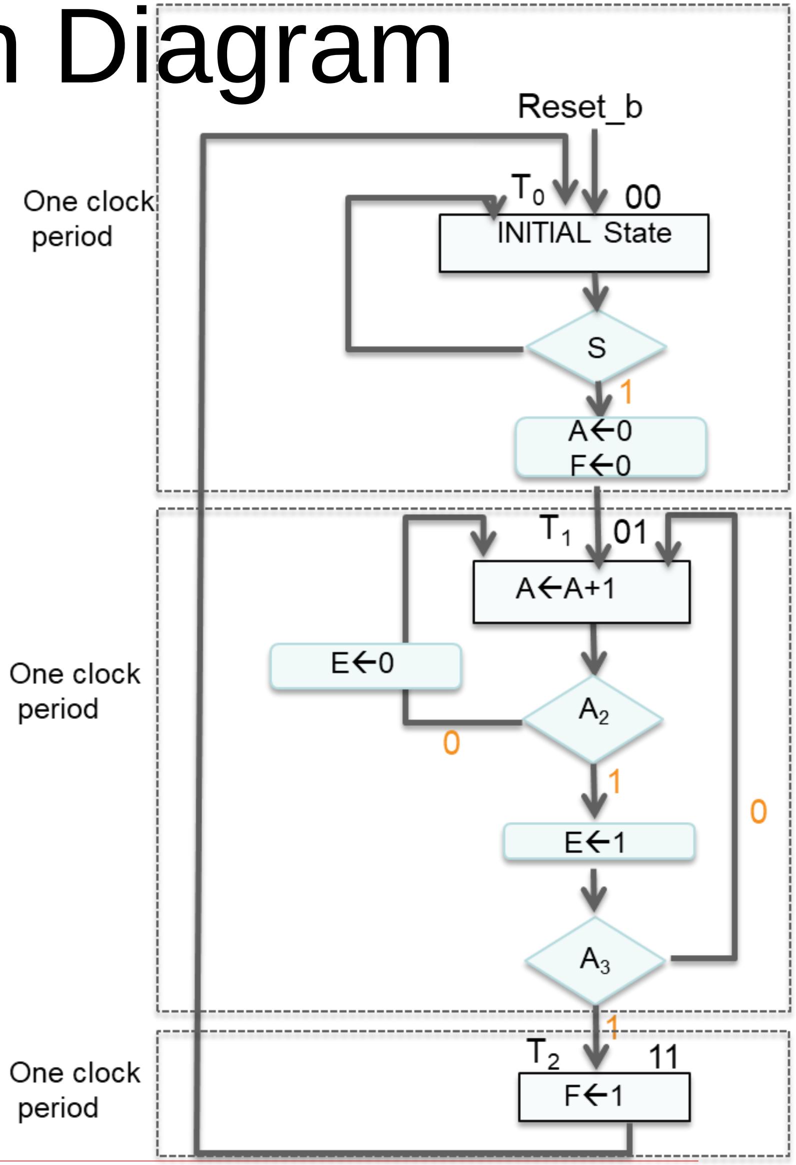
$S \leftarrow 1$, $A \leftarrow 0$, $F \leftarrow 0$

Counter				Flip Flops		Present State	Conditions	Next State	Command Signals
A_3	A_2	A_1	A_0	E	F				
0	0	0	0	X	0	T_1	$A_2=0, A_3=0$	T_1	clr_E incr_A
0	0	0	1	0	0				
0	0	1	0	0	0				
0	0	1	1	0	0				
0	1	0	0	0	0	T_1	$A_2=1, A_3=0$	T_1	set_E incr_A
0	1	0	1	1	0				
0	1	1	0	1	0				
0	1	1	1	1	0				
1	0	0	0	1	0	T_1	$A_2=0, A_3=1$	T_1	clr_E incr_A
1	0	0	1	0	0				
1	0	1	0	0	0				
1	0	1	1	0	0				
1	1	0	0	0	0	T_1	$A_2=1, A_3=1$	T_2	set_E incr_A
1	1	0	1	1	0	T_2		T_0	set_F
1	1	0	1	1	1	T_0		Depends on S	clr_A_F

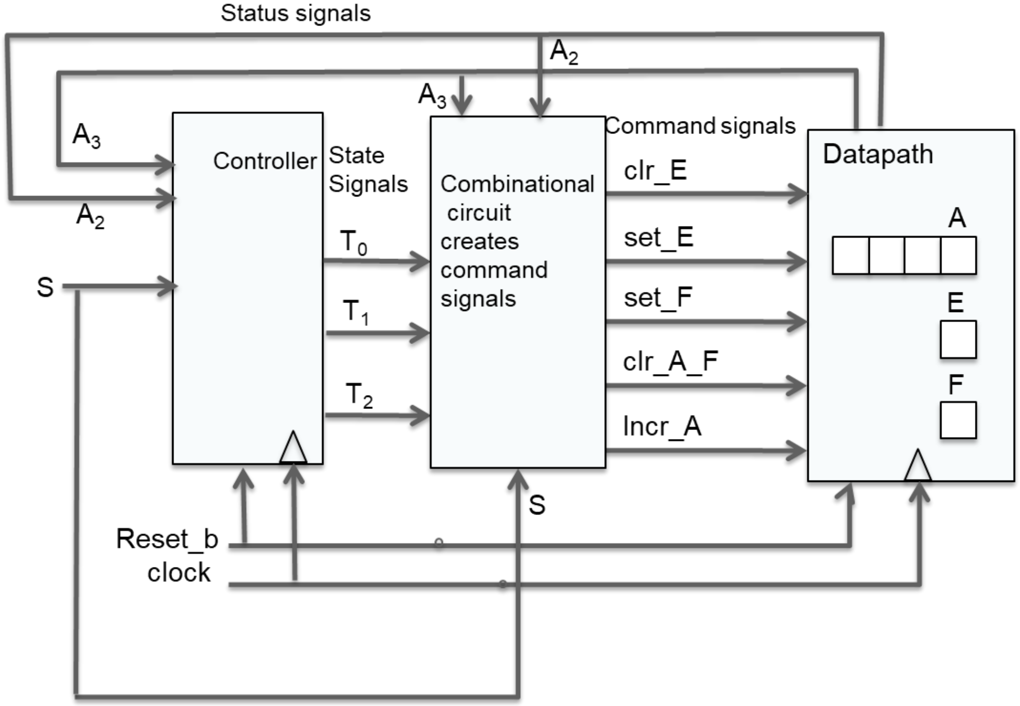
State Transition Diagram

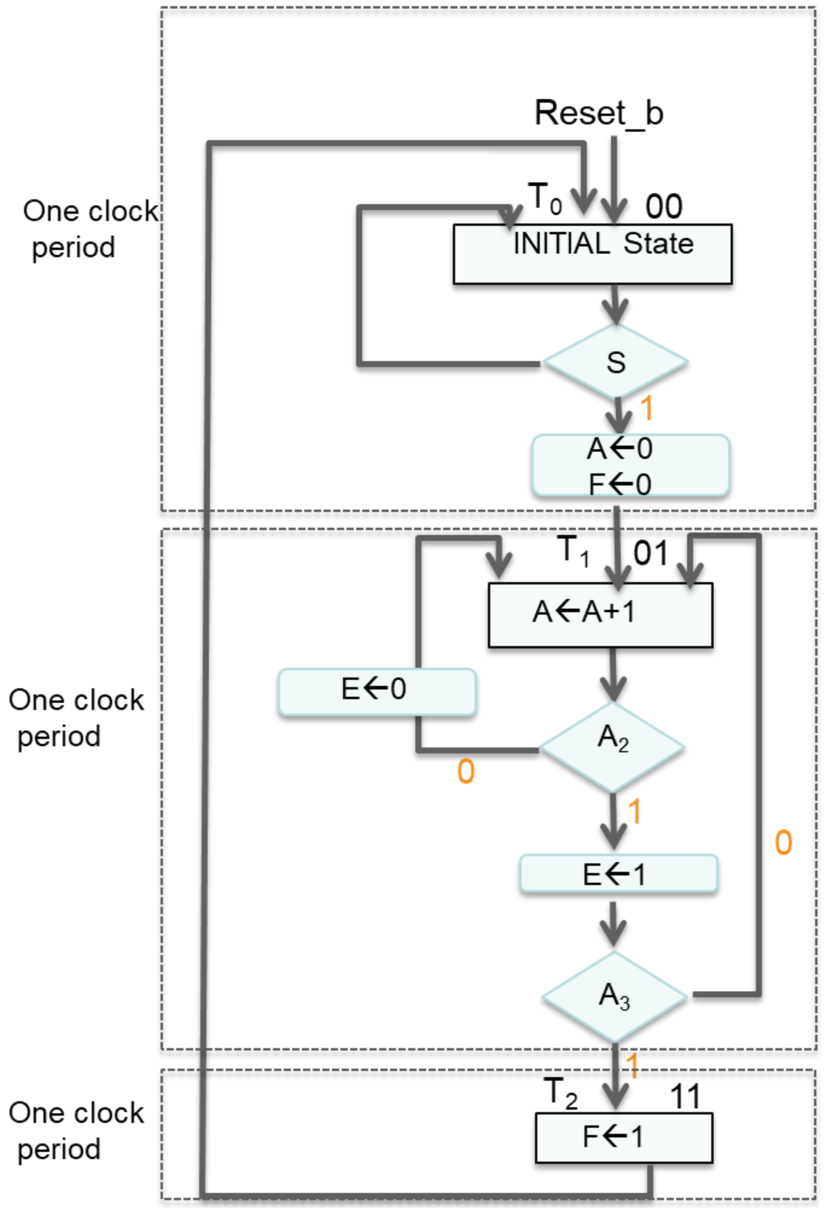


CONTROL	DATAFLOW
$T_0 \rightarrow T_1$, clr_A_F:	$A \leftarrow 0, F \leftarrow 0$
$T_1 \rightarrow T_1$, incr_A:	$A \leftarrow A+1$
if (A2=1) then set_E:	$E \leftarrow 1$
if (A2=0) then clr_E:	$E \leftarrow 0$
$T_1 \rightarrow T_2$, incr_A, set_E :	$A \leftarrow A+1, E \leftarrow 1$
$T_2 \rightarrow T_0$, set_F:	$F \leftarrow 1$



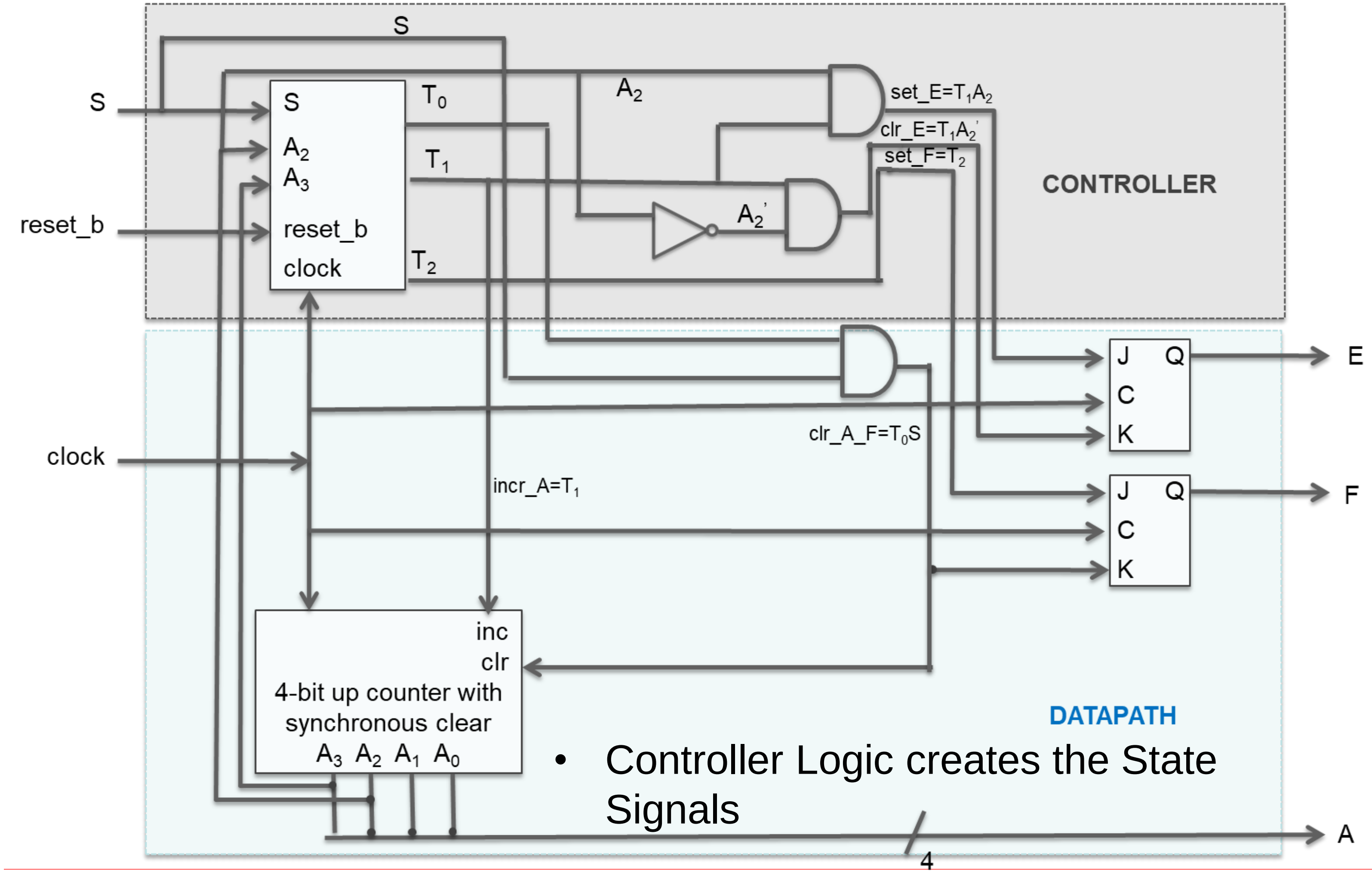
ASM Chart and the Controller (1)





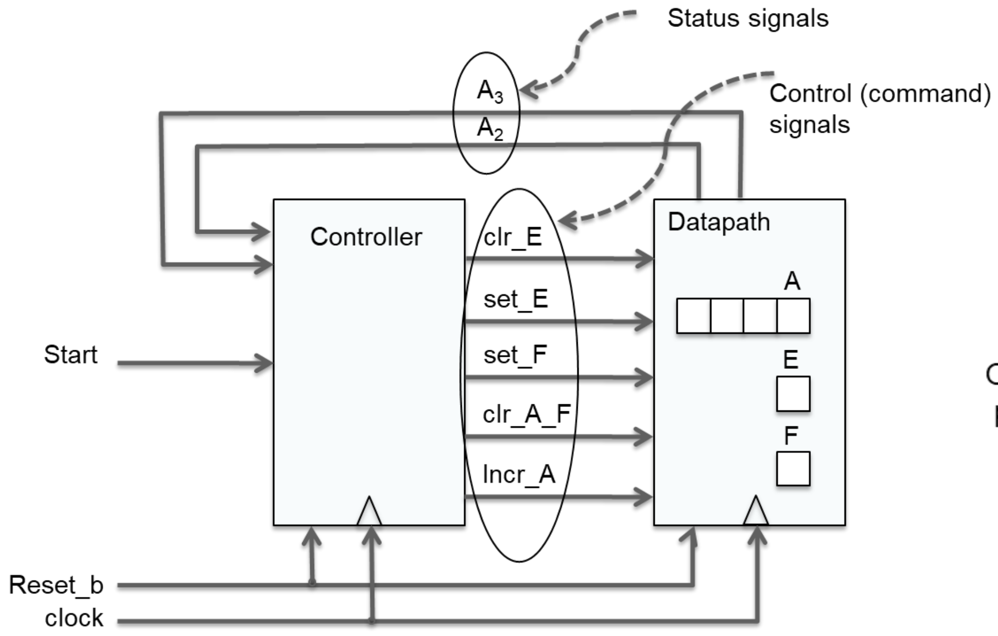
- Controller Logic creates the State Signals

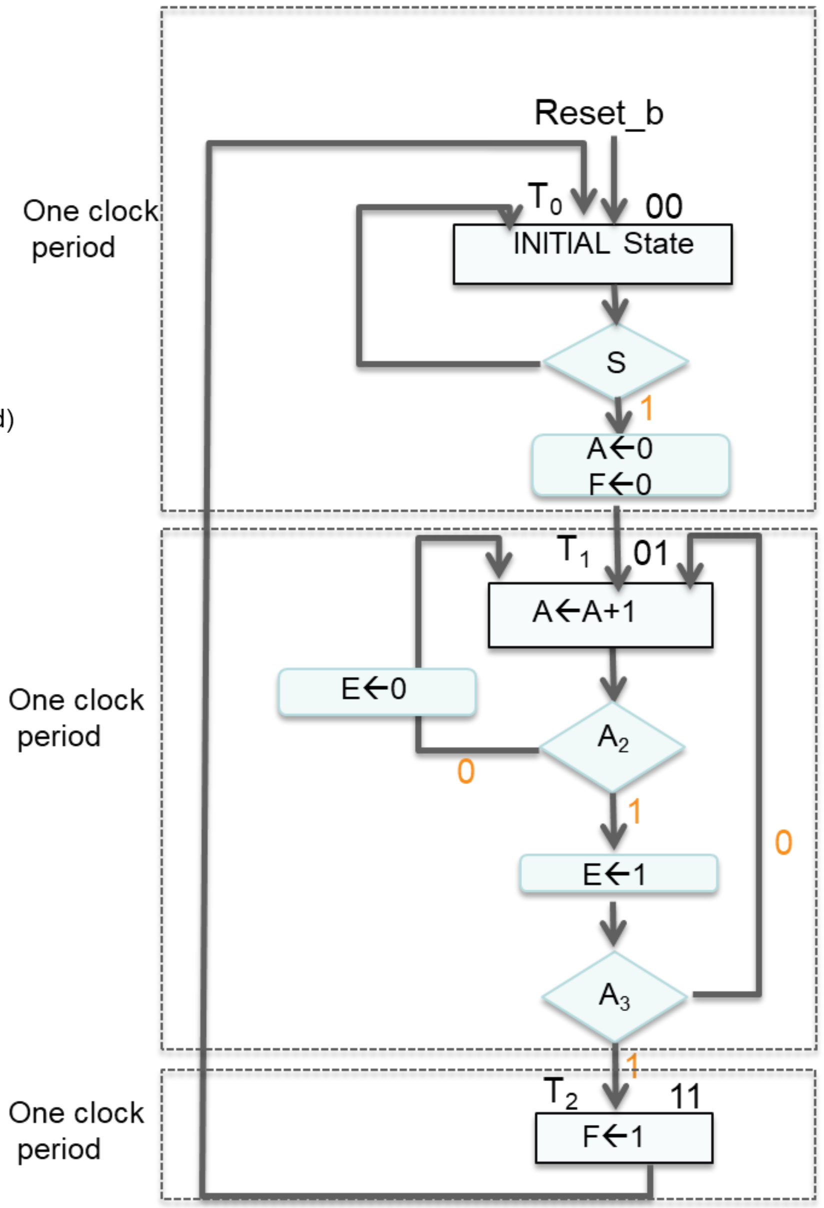
Controller and Datapath (1)



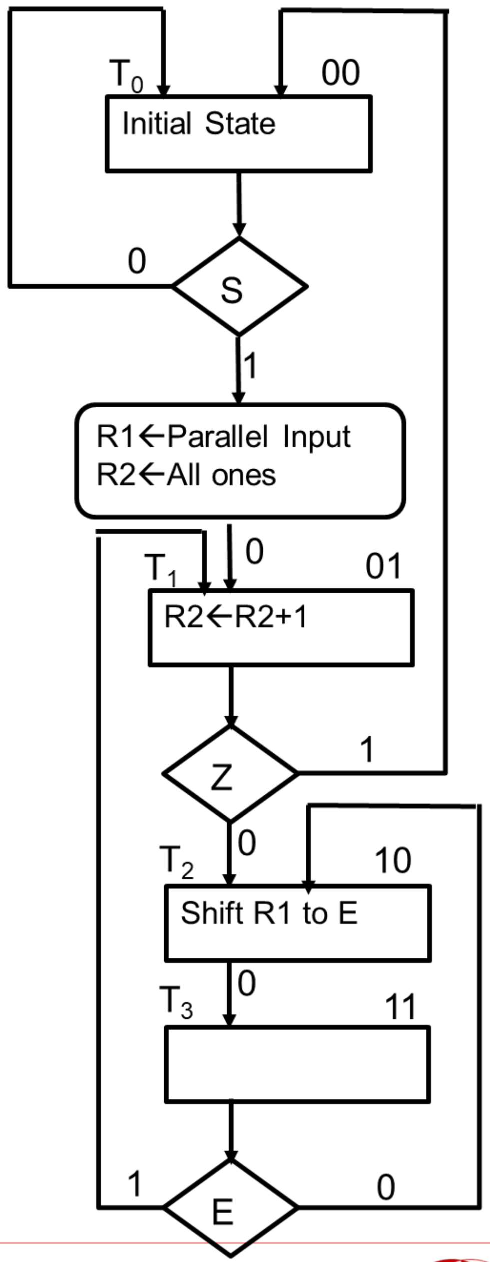
- Controller Logic creates the State Signals

ASM Chart and the Controller (2)





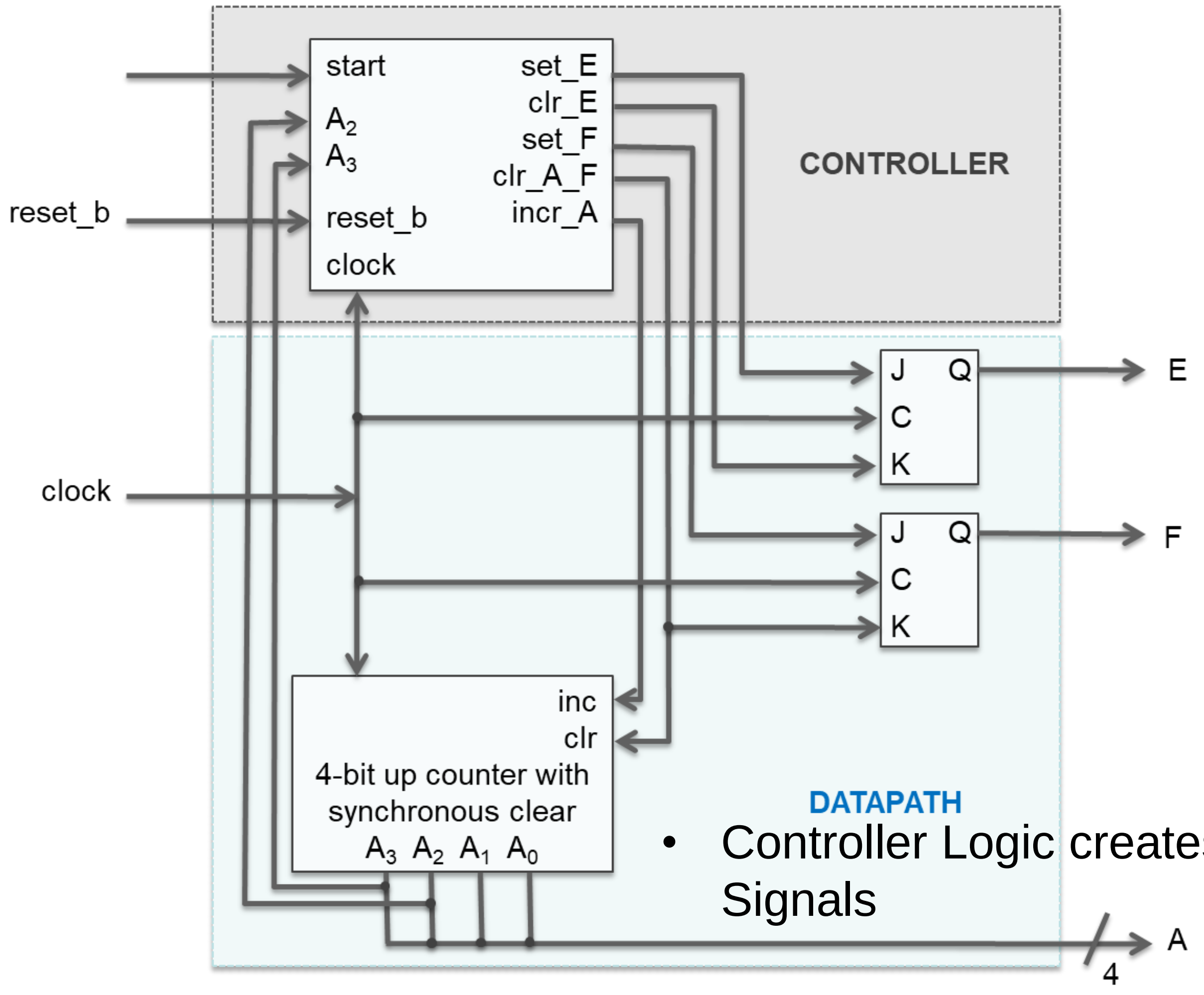
- Controller Logic creates the output Signals



A Note on Controller Styles

- For this ASM Chart T_3 only spends a clock pulse, no output is created.
- Controller Logic that creates the State Signals is a more structured representation.

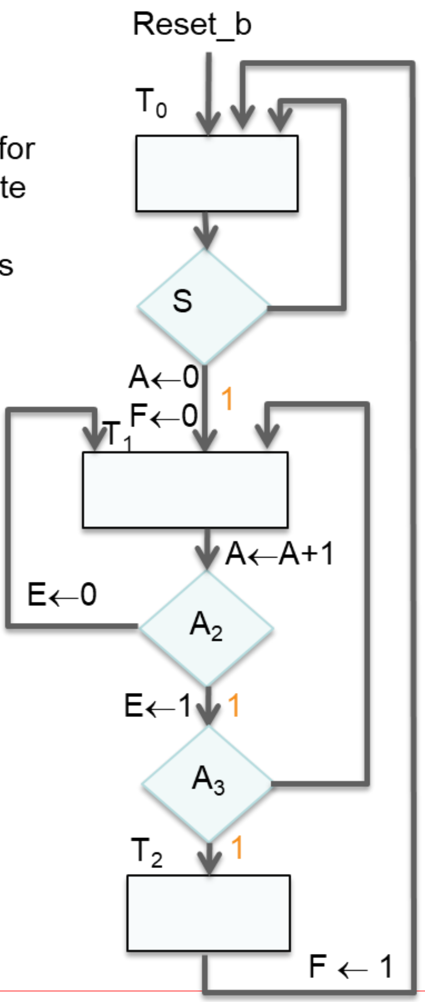
Controller and Datapath (2)



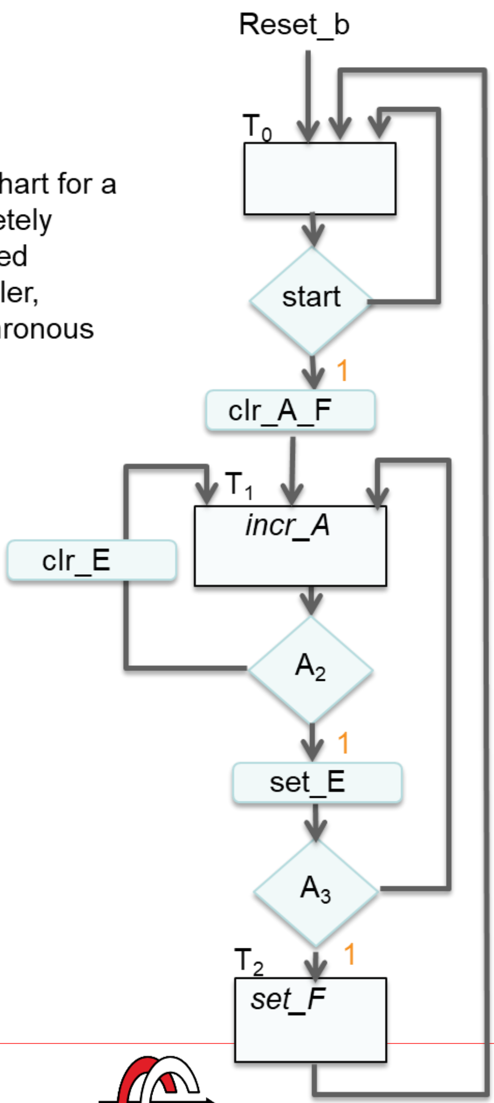
- Controller Logic creates the output Signals

Other ASM Chart Styles

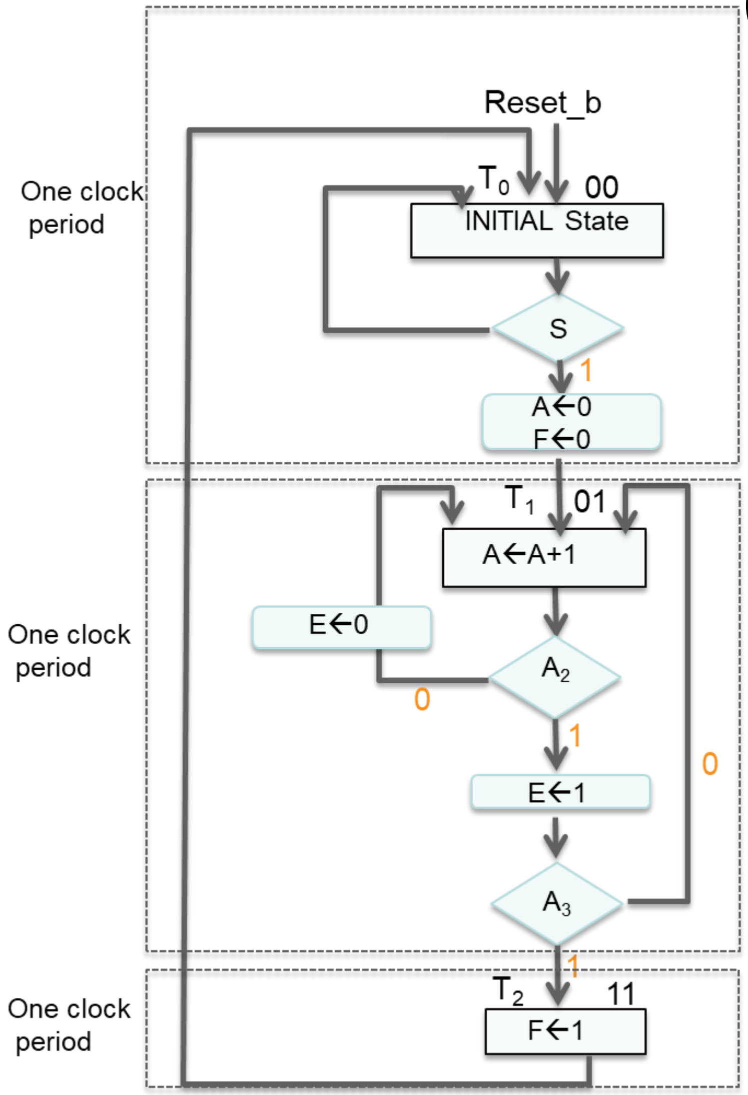
ASMD chart for controller state transitions, asynchronous reset



ASM chart for a completely specified controller, asynchronous reset



Controller State Table



	Present State		Inputs				Next state		Outputs				
Present State Symbols	G_1	G_0	S	A_2	A_3	Next State Symbols	$\overline{G}_1$	$\overline{G}_0$	Set_E	Clr_E	Set_F	Clr_A_F	Incr_A
T_0	0	0	0	x	x	T_0	0	0	0	0	0	0	0
T_0	0	0	1	x	x	T_1	0	1	0	0	0	1	0
T_1	0	1	x	0	x	T_1	0	1	0	1	0	0	1
T_1	0	1	x	1	0	T_1	0	1	1	0	0	0	1
T_1	0	1	x	1	1	T_2	1	1	1	0	0	0	1
T_2	1	1	x	x	x	T_0	0	0	0	0	1	0	0

State assignment:

T_0:	00 ($G_1'G_0'$)
T_1:	01 ($G_1'G_0$)
Unused:	10 (G_1G_0')
T_2:	11 (G_1G_0)

Controller Implementation Styles

1. Formal sync. cct. design using any type of FF (use the excitation tables for the used FFs and obtain the minimal cct.)
2. D-FFs and decoder
3. One FF per state (one hot FF-state assignment)
4. Multiplexer (modified D-FF)

- We will do the implementation for the controller that generates the state signals (T_2, T_1, T_0).
- The output signals are:
  - $Set\_E=T_1A_2$
  - $Clr\_E=T_1A_2'$
  - $Set\_F=T_2$
  - $Clr\_A\_F=T_0S$
  - $Incr\_A=T_1$

Formal Sequential Circuit

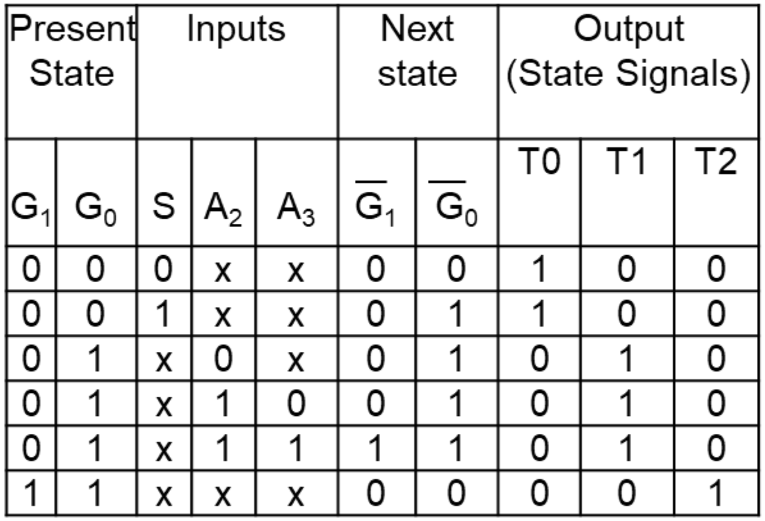
- 2 state variables $G_1G_0 \rightarrow$ Use 2 JK FFs with inputs J_1K_1, J_0K_0
- 5 Inputs: Present State (G_1G_0), S(tart), $A_2A_3 \rightarrow$ 5-input K-maps for next state and outputs
- 3 Outputs: Present state ($T_0T_1T_2$), only one state signal is 1 during the clock pulse

Present State		Inputs			Next state		Output (State Signals)		
G_1	G_0	S	A_2	A_3	$\overline{G}_1$	$\overline{G}_0$	T0	T1	T2
0	0	0	x	x	0	0	1	0	0
0	0	1	x	x	0	1	1	0	0
0	1	x	0	x	0	1	0	1	0
0	1	x	1	0	0	1	0	1	0
0	1	x	1	1	1	1	0	1	0
1	1	x	x	x	0	0	0	0	1

7 K-maps with 5 inputs each

Formal Sequential Circuit

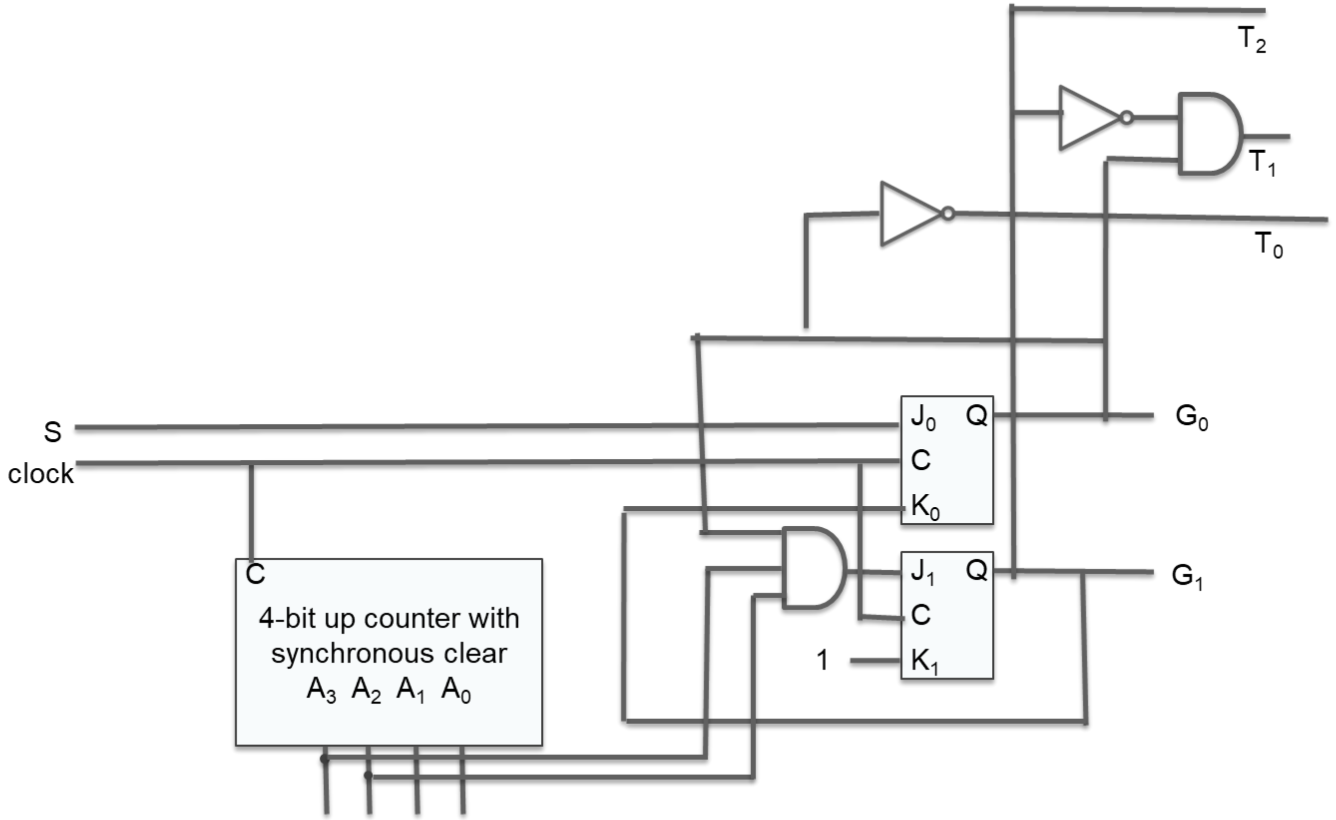
- $J_1=G_0A_2A_3$
- $K_1=1$
- $J_0 = S$
- $K_0 = G_1$
- $T0=G_0'$
- $T1=G_1'G_0$
- $T2=G_1$



Present State		Inputs			Next state		Output (State Signals)		
G_1	G_0	S	A_2	A_3	$\overline{G_1}$	$\overline{G_0}$	T0	T1	T2
0	0	0	x	x	0	0	1	0	0
0	0	1	x	x	0	1	1	0	0
0	1	x	0	x	0	1	0	1	0
0	1	x	1	0	0	1	0	1	0
0	1	x	1	1	1	1	0	1	0
1	1	x	x	x	0	0	0	0	1

Formal Sequential Circuit

- $J_1 = G_0 A_2 A_3$
- $K_1 = 1$
- $J_0 = S$
- $K_0 = G_1$
- $T0 = G_0'$
- $T1 = G_1' G_0$
- $T2 = G_1$



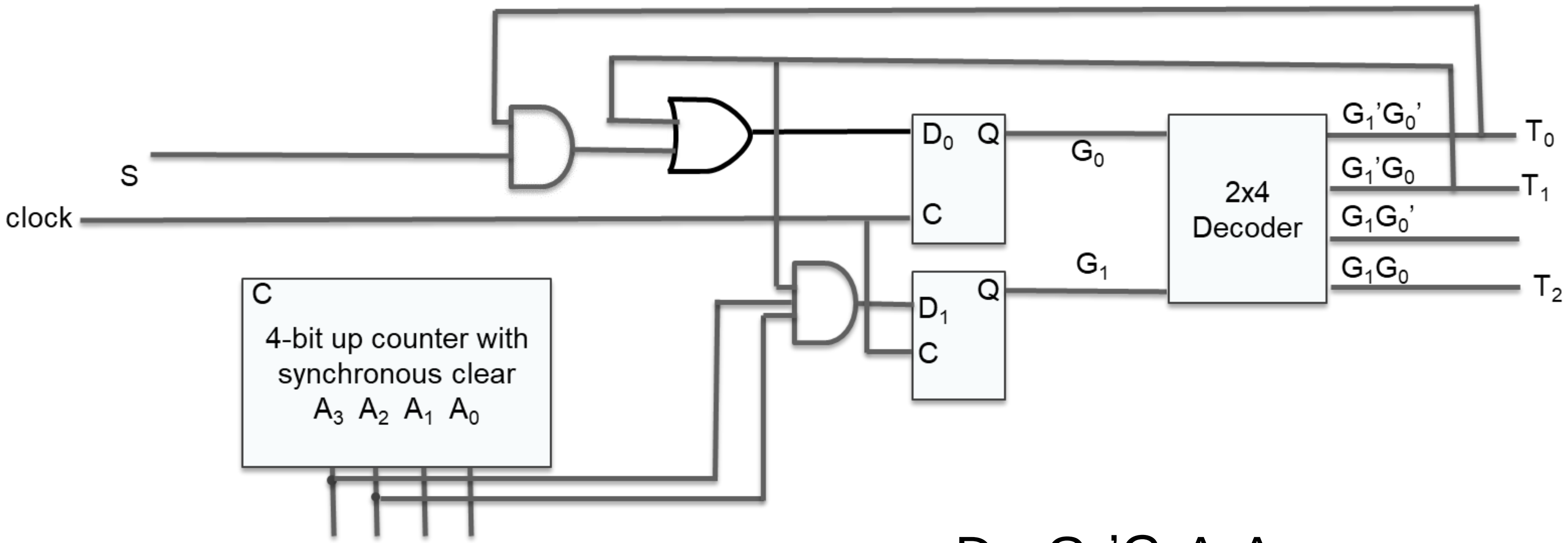
Reset_b is not shown. It is connected to asynchronous clear of FFs

DFFs and a decoder

Present State		Inputs			Next state		Output (State Signals)		
							T0	T1	T2
G_1	G_0	S	A_2	A_3	$\overline{G_1}$	$\overline{G_0}$			
0	0	0	x	x	0	0	1	0	0
0	0	1	x	x	0	1	1	0	0
0	1	x	0	x	0	1	0	1	0
0	1	x	1	0	0	1	0	1	0
0	1	x	1	1	1	1	0	1	0
1	1	x	x	x	0	0	0	0	1

- The next state is D-FF input
- 2 D-FFs with inputs D_1, D_0
- $D_1=G_1'G_0A_2A_3$
- $D_0=G_1'G_0'S+G_1'G_0$

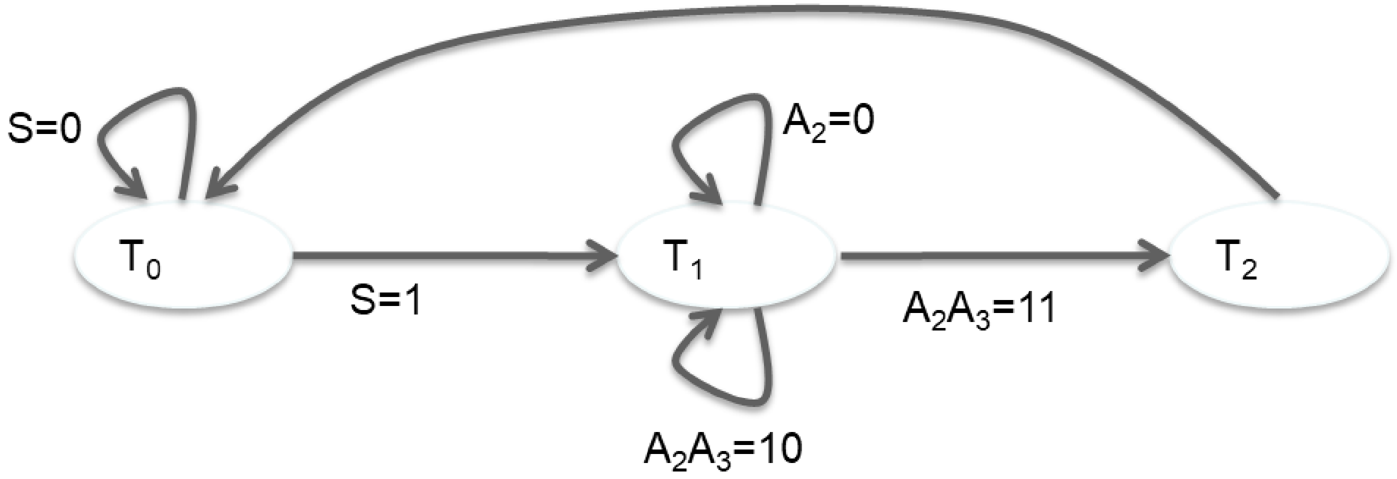
DFFs and a decoder



- $D_1 = G_1'G_0A_2A_3$
- $D_0 = G_1'G_0'S + G_1'G_0$

Reset_b is not shown. It is connected to asynchronous clear of FFs

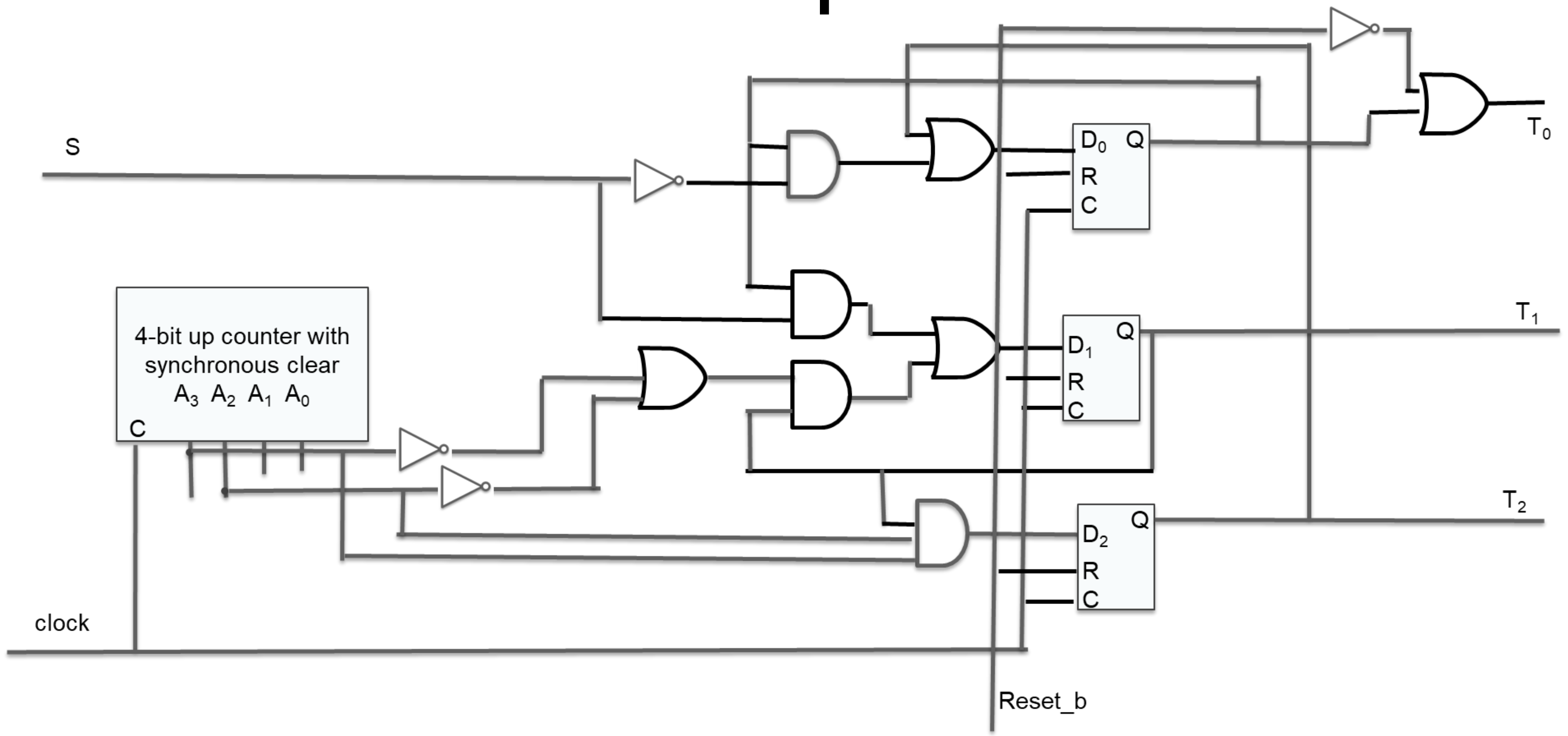
One FF per State



- $D_0 = S'T_0 + T_2$
- $D_1 = ST_0 + A_2A_3'T_1 + A_2'T_1$

  $= ST_0 + (A_2' + A_3')T_1$
- $D_2 = A_2A_3T_1$

One FF per State



- $D_0=S'T_0+T_2$
- $D_1=ST_0+A_2A_3'T_1+A_2'T_1$
  $=ST_0+(A_2'+A_3')T_1$
- $D_2=A_2A_3T_1$

We get $T_0T_1T_2=100$ as initial state by applying Reset_b

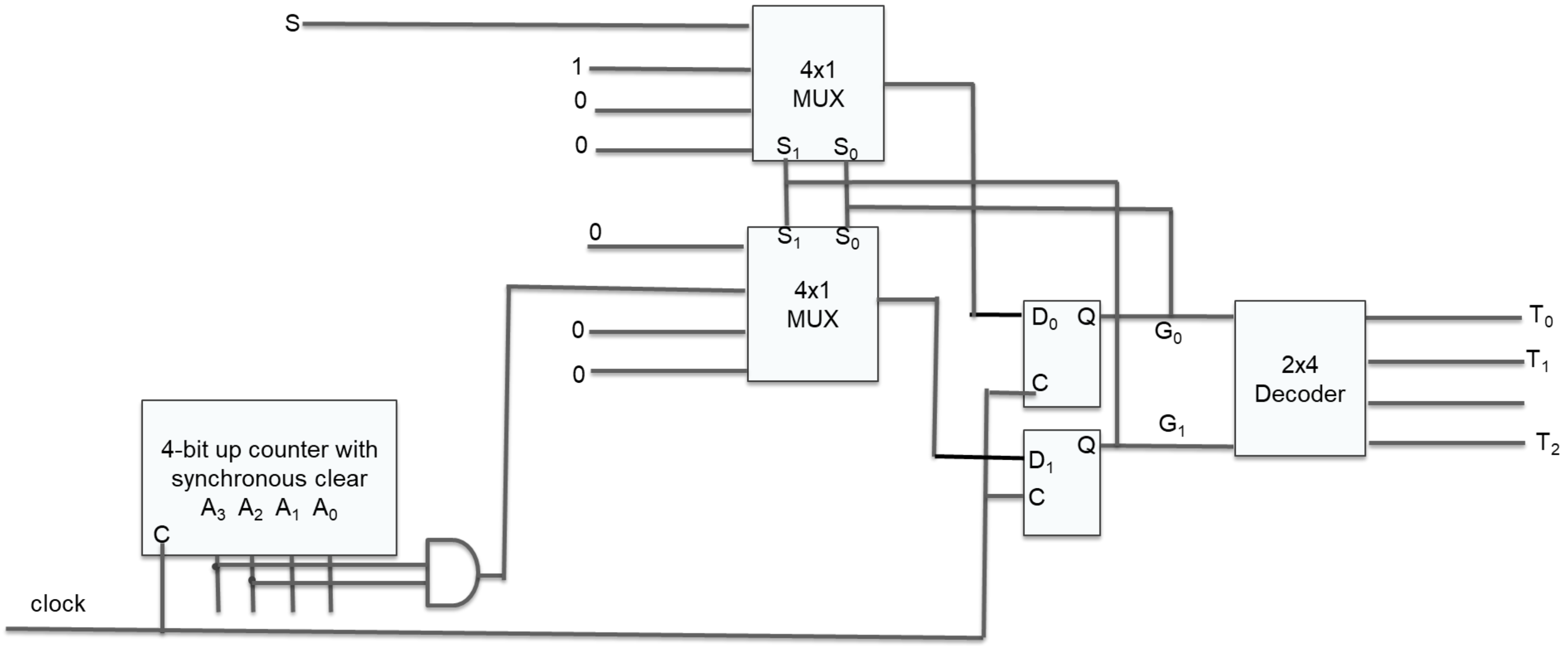
Multiplexer (modified D-FF)

- Use n 2^nx1 multiplexers and a n$\times 2^n$decoder
- n: number of state variables,
- state variables are the mux selects
- Mux outputs are the next states

Multiplexer (modified D-FF)

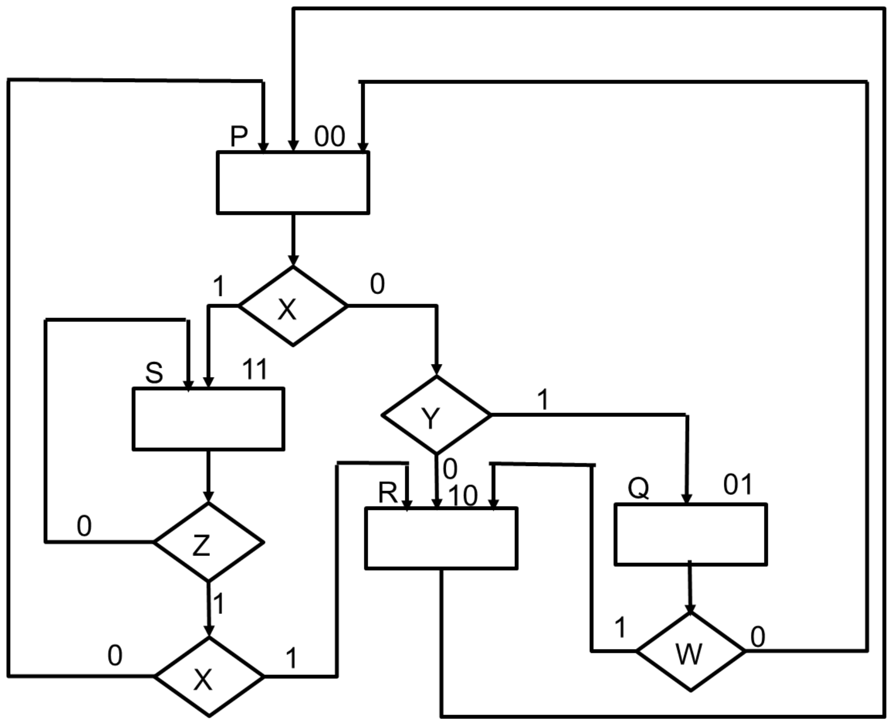
- Our example:
  - n=2, selects: G_1G_0
  - From DFF and a decoder design:
  - $D_1=G_1'G_0A_2A_3$
  - $D_0=G_1'G_0'S+G_1'G_0$
- $D_1=G_1'G_0'0+G_1'G_0A_2A_3 +G_1G_0'0+G_1G_00$
- $D_0=G_1'G_0'S+G_1'G_01 +G_1G_0'0+G_1G_00$

Multiplexer (modified D-FF)



Reset_b is not shown. It is connected to asynchronous clear of FFs

Example 2



Present State		Next State		Input Conditions	x	y	z	w
Name	G_1G_0	Name	$\overline{G_1}\overline{G_0}$					
P	00	Q	01	x'y	0	1	x	x
	00	R	10	x'y'	0	0	x	x
	00	S	11	x	1	x	x	x
Q	01	P	00	w'	x	x	x	0
	01	R	10	w	x	x	x	1
R	10	P	00	Clock pulse	x	x	x	x
S	11	P	00	x'z	0	x	1	x
	11	R	10	xz	1	x	1	x
	11	S	11	z'	x	x	0	x

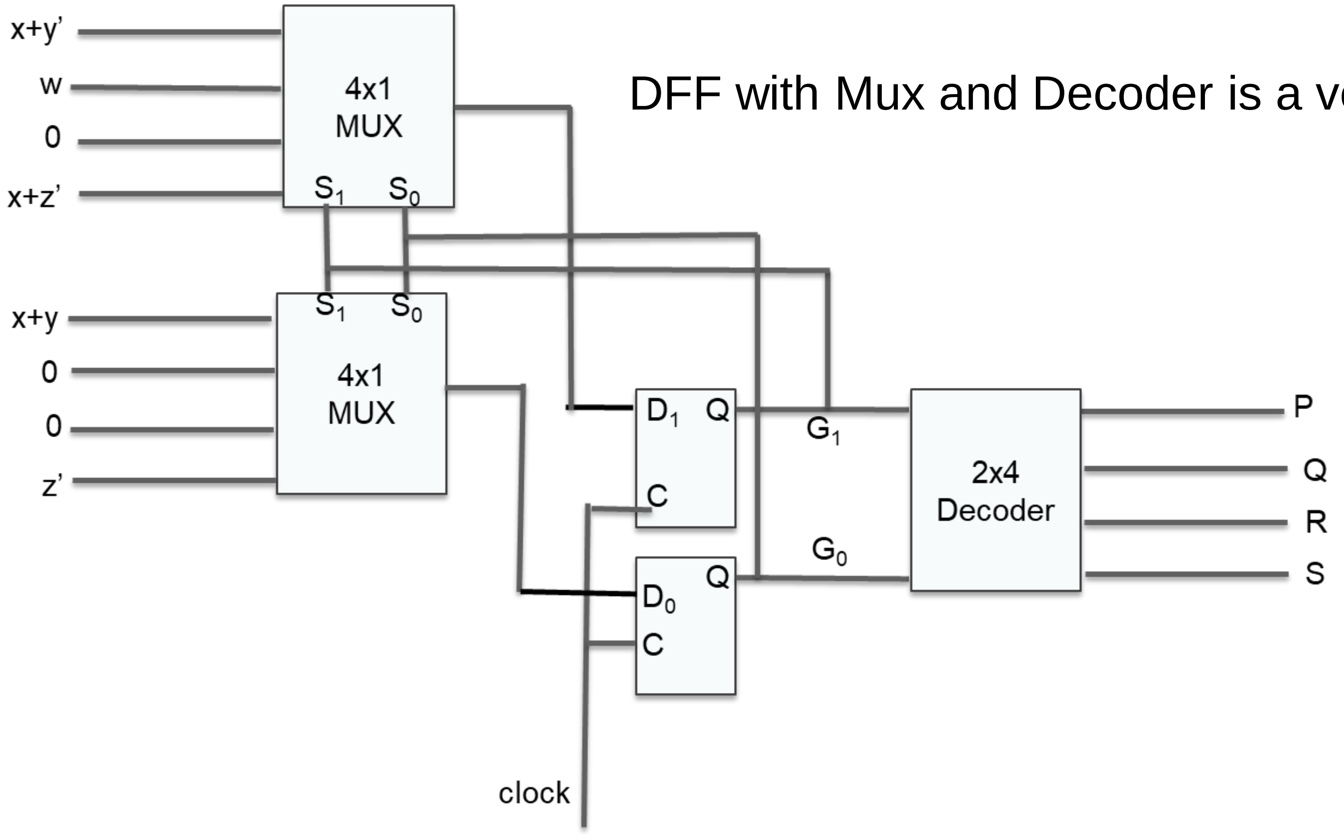
Example 2

Present State		Next State		Input Conditions	x	y	z	w
Name	G_1G_0	Name	$\overline{G_1}\,\overline{G_0}$					
P	00	Q	01	x'y	0	1	x	x
	00	R	10	x'y'	0	0	x	x
	00	S	11	x	1	x	x	x
Q	01	P	00	w'	x	x	x	0
	01	R	10	w	x	x	x	1
R	10	P	00	Clock pulse	x	x	x	x
S	11	P	00	x'z	0	x	1	x
	11	R	10	xz	1	x	1	x
	11	S	11	z'	x	x	0	x

- $DG_1=G_1'G_0'(x'y'+x)+G_1'G_0w+G_1G_0'0+G_1G_0(xz+z')$
  $=G_1'G_0'(x+y')+G_1'G_0w+G_1G_0'0+G_1G_0(x+z')$
- $DG_0=G_1'G_0'(x'y+x)+G_1'G_00+G_1G_0'0+G_1G_0z'$
  $=G_1'G_0'(x+y)+G_1'G_00+G_1G_0'0+G_1G_0z'$

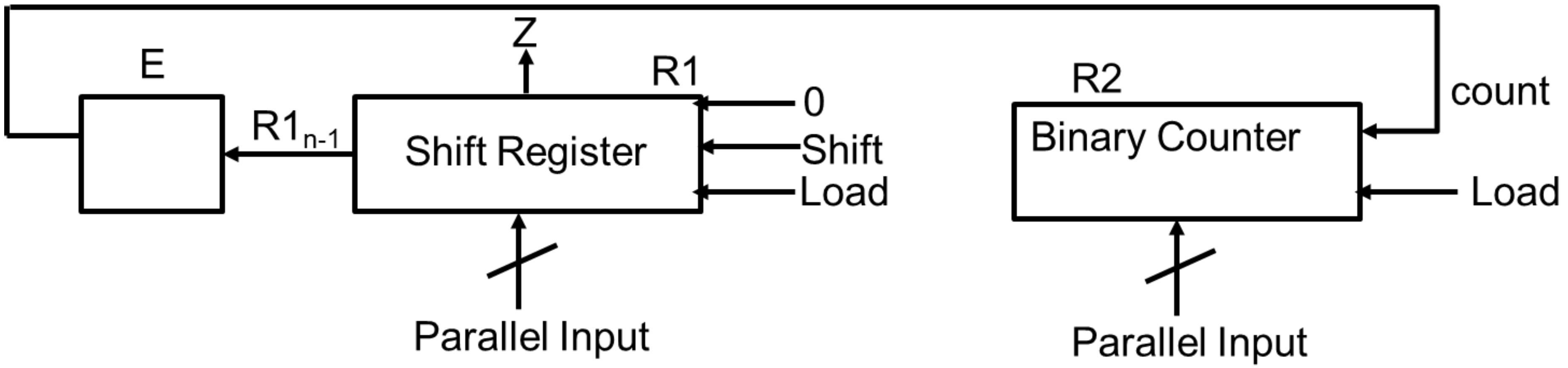
Example 2 Controller

DFF with Mux and Decoder is a very systematic method !



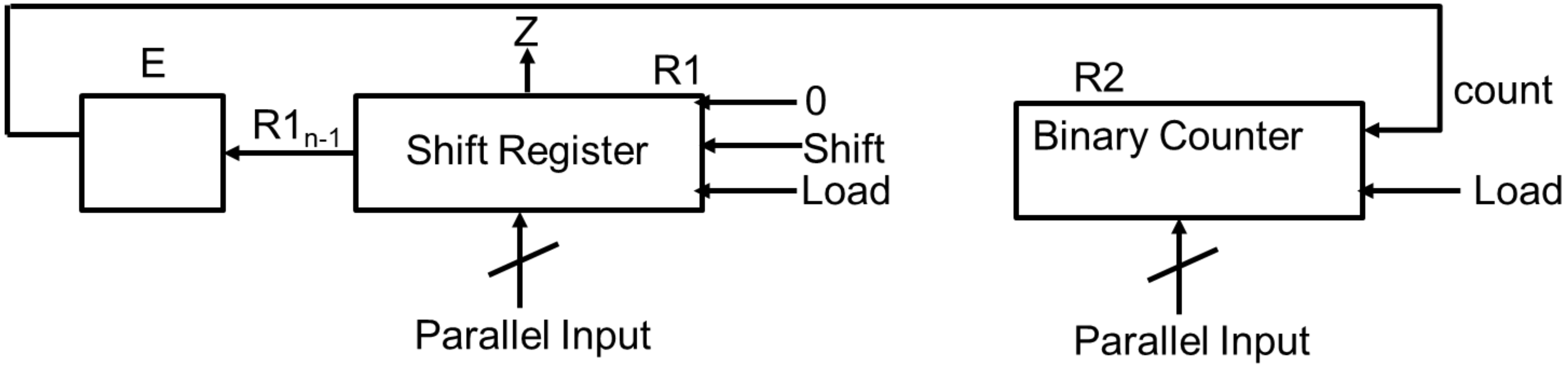
- $DG_1=G_1'G_0'(x+y')+G_1'G_0w+G_1G_0'0+G_1G_0(x+z')$
- $DG_0=G_1'G_0'(x+y)+G_1'G_00+G_1G_0'0+G_1G_0z'$

Example 3



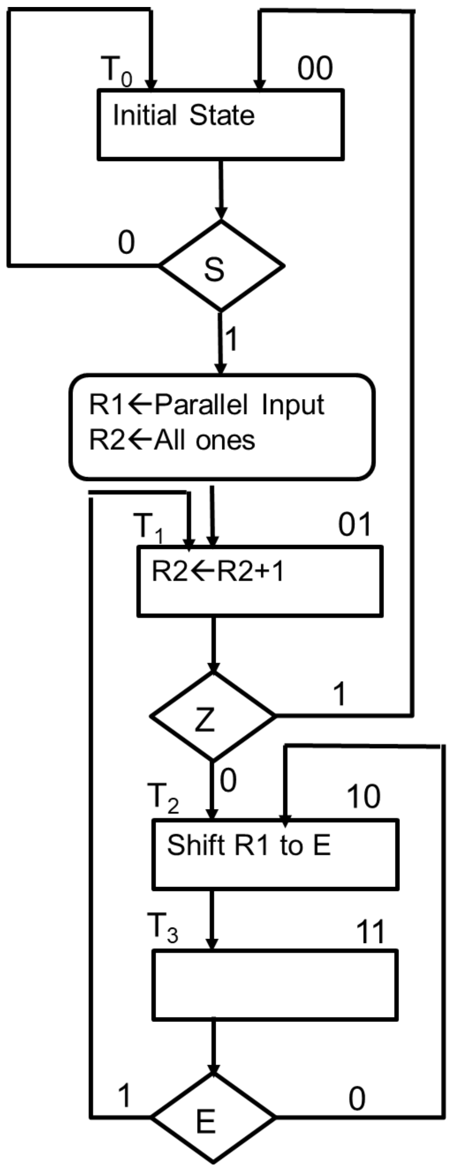
- The circuit counts the number of 1s in the number loaded in R1 and sets R2 to the binary representation of that value.
- To do this the shifted value from R1 to E is checked and count signal for R2 is generated
- Shifting R1 to E: Loads $R1_{n-1}$ (the most significant bit of R1) in E and R1 is shifted left in the same clock pulse. E is cleared otherwise.
- E acts as the n^{th} bit of the shift register

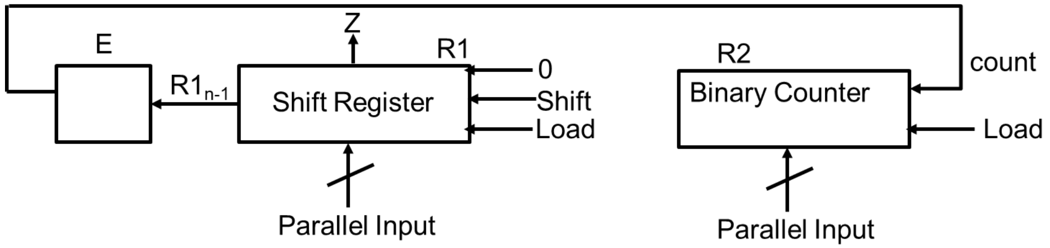
Example 3



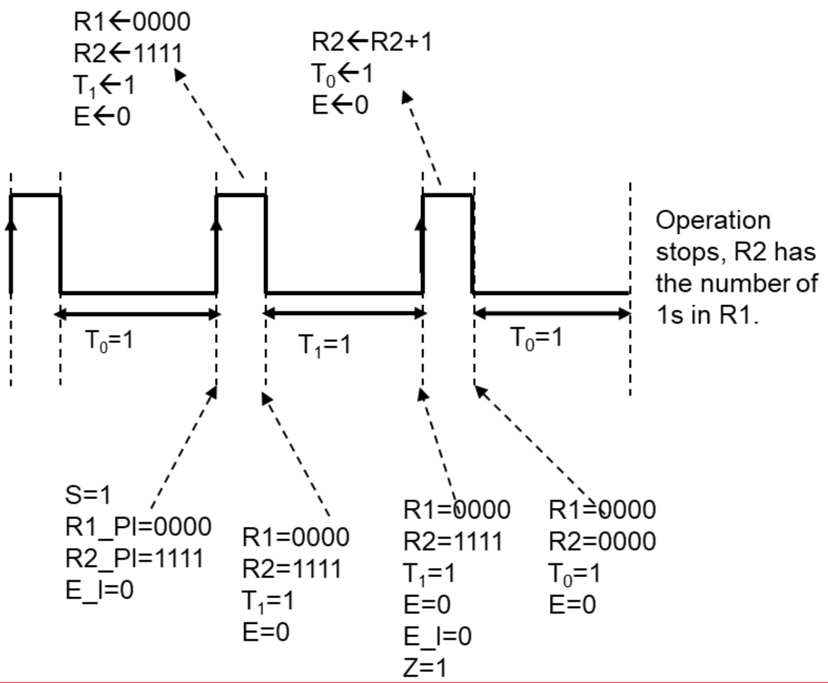
- The operation of the circuit starts with the external input S
- E: The state of the E FF
- Z: Signal to indicate whether the R1 contains all zeros or not,
- i.e. Z = 1 if R1 = 0 and Z=0 if R1 ≠ 0

ASM Chart for Example 3

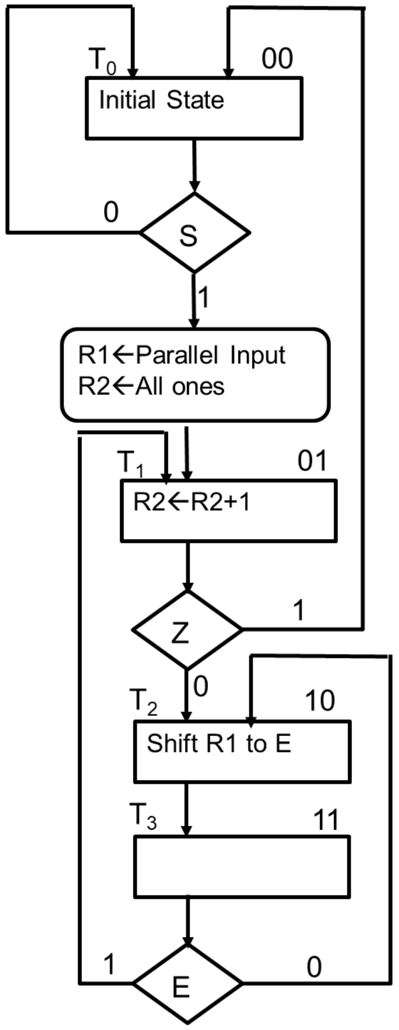


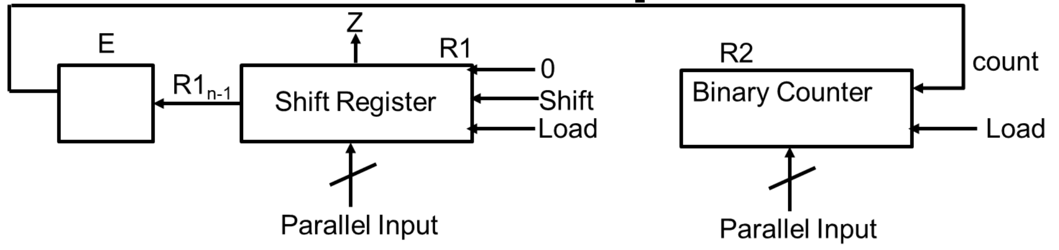


Example: R1, R2 4 bit registers.

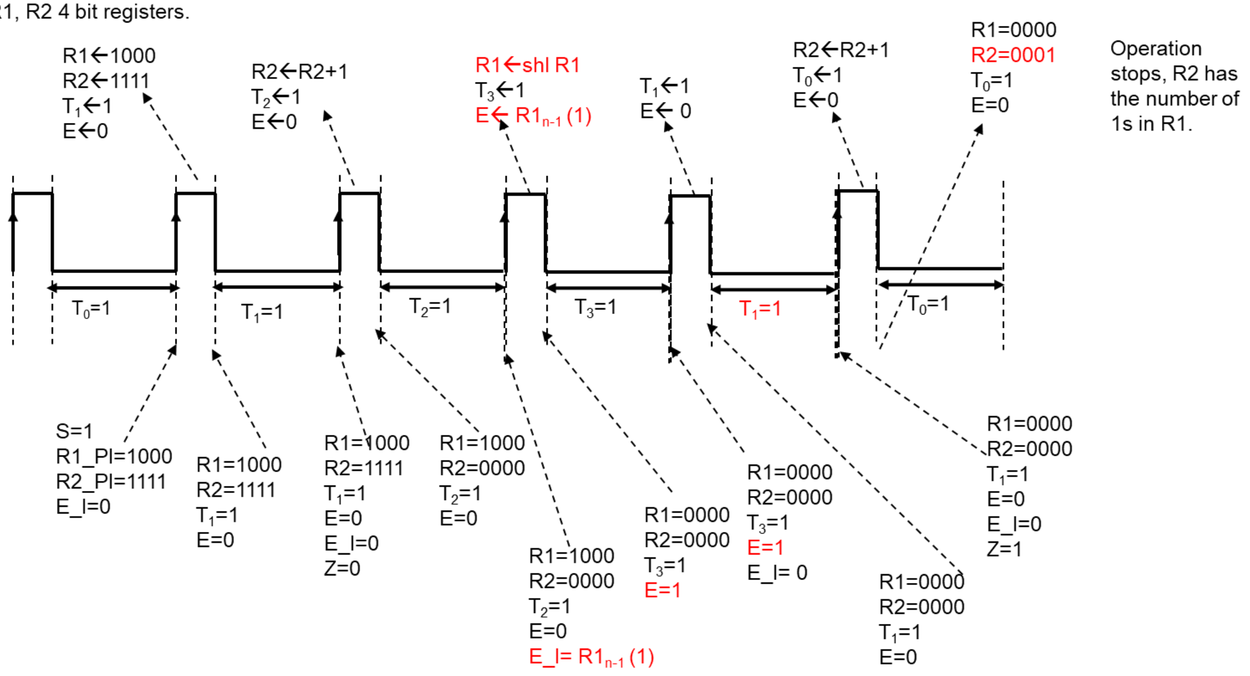


ASM Chart for Example 3

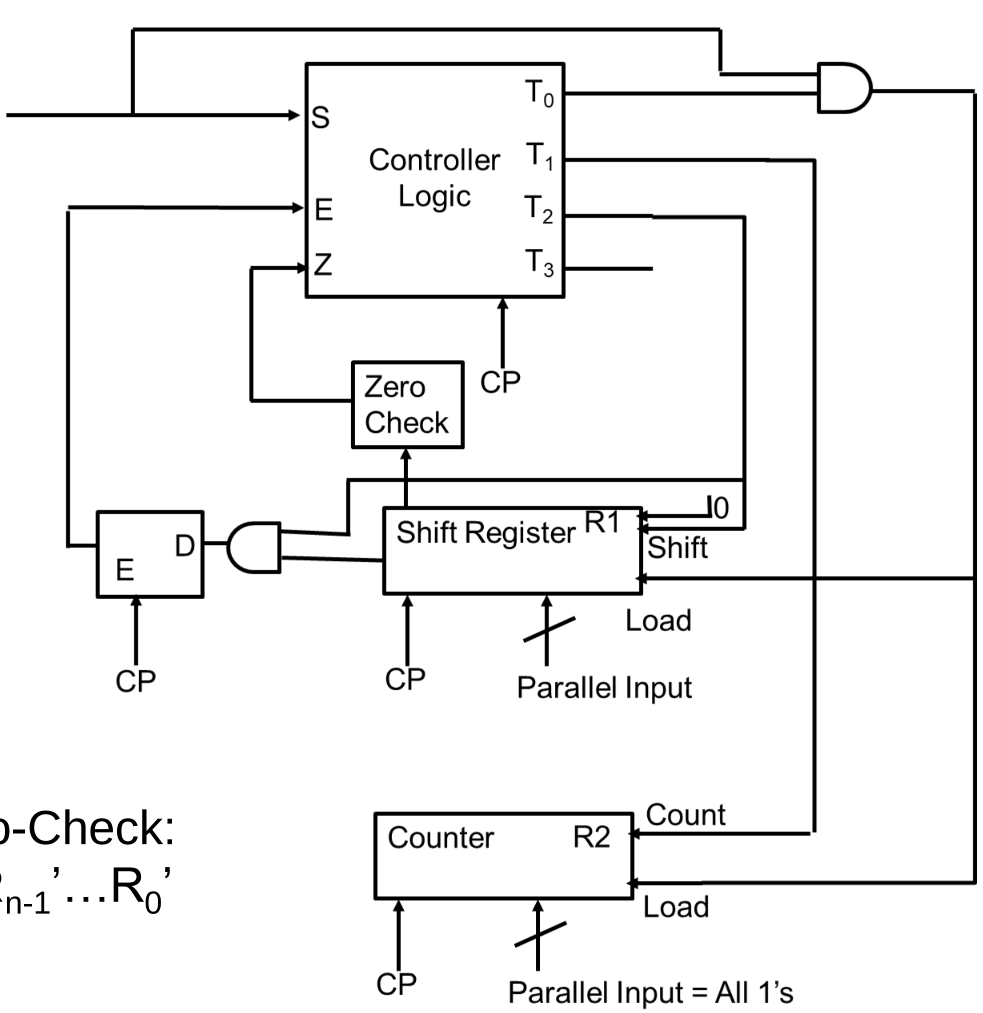


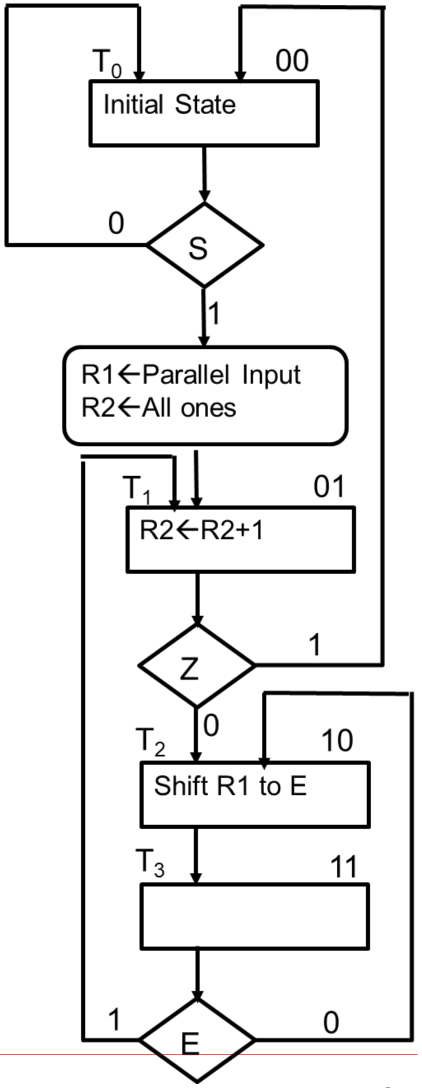


Example: R1, R2 4 bit registers.



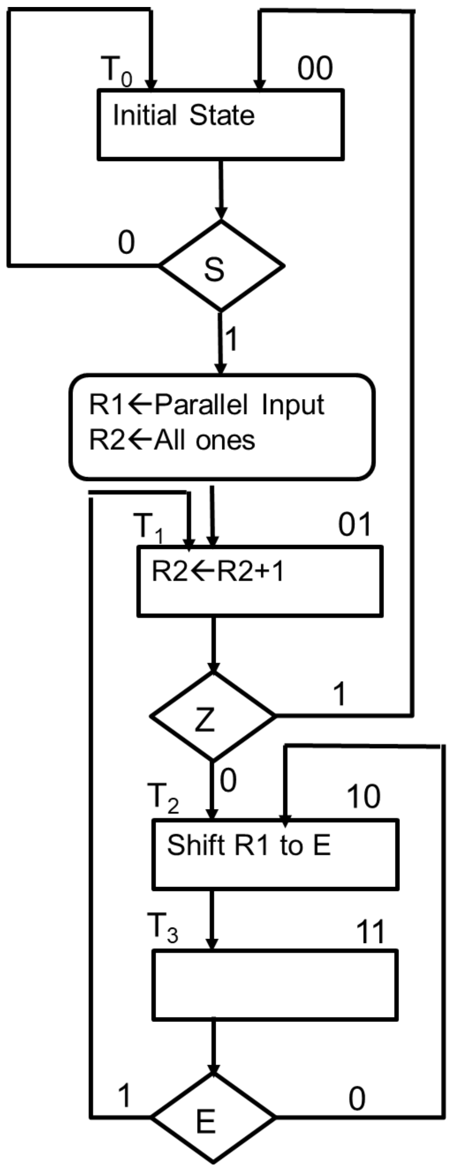
Datapath for Example 3





Zero-Check:
$Z=R_{n-1}'\ldots R_0'$

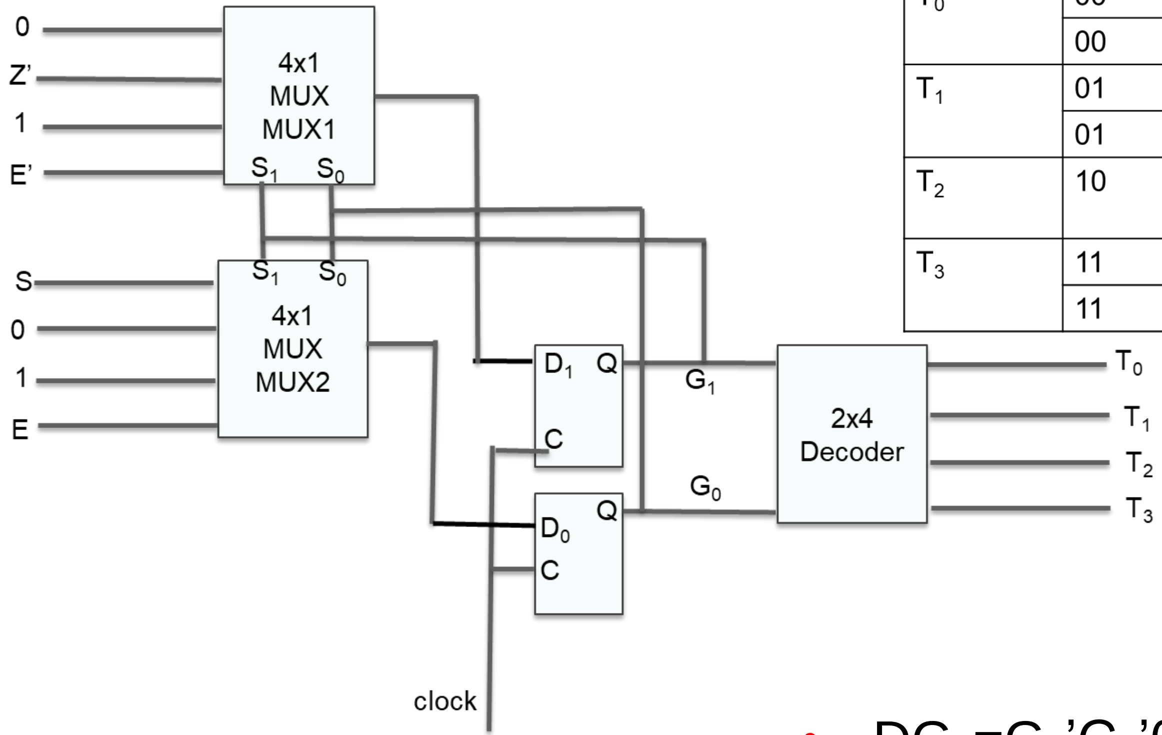
Example 3



Present State		Next State		Input Conditions	MUX1	MUX0
Name	G_1G_0	Name	$\overline{G}_1\overline{G}_0$			
T_0	00	T_0	00	S’	0	S
	00	T_1	01	S		
T_1	01	T_0	00	Z	Z’	0
	01	T_2	10	Z’		
T_2	10	T_3	11	Clock pulse	1	1
T_3	11	T_1	01	E	E’	E
	11	T_2	10	E’		

- $DG_1=G_1'G_0'0+G_1'G_0Z'+G_1G_0'1+G_1G_0E'$
- $DG_0=G_1'G_0'S+G_1'G_00+G_1G_0'1+G_1G_0E$

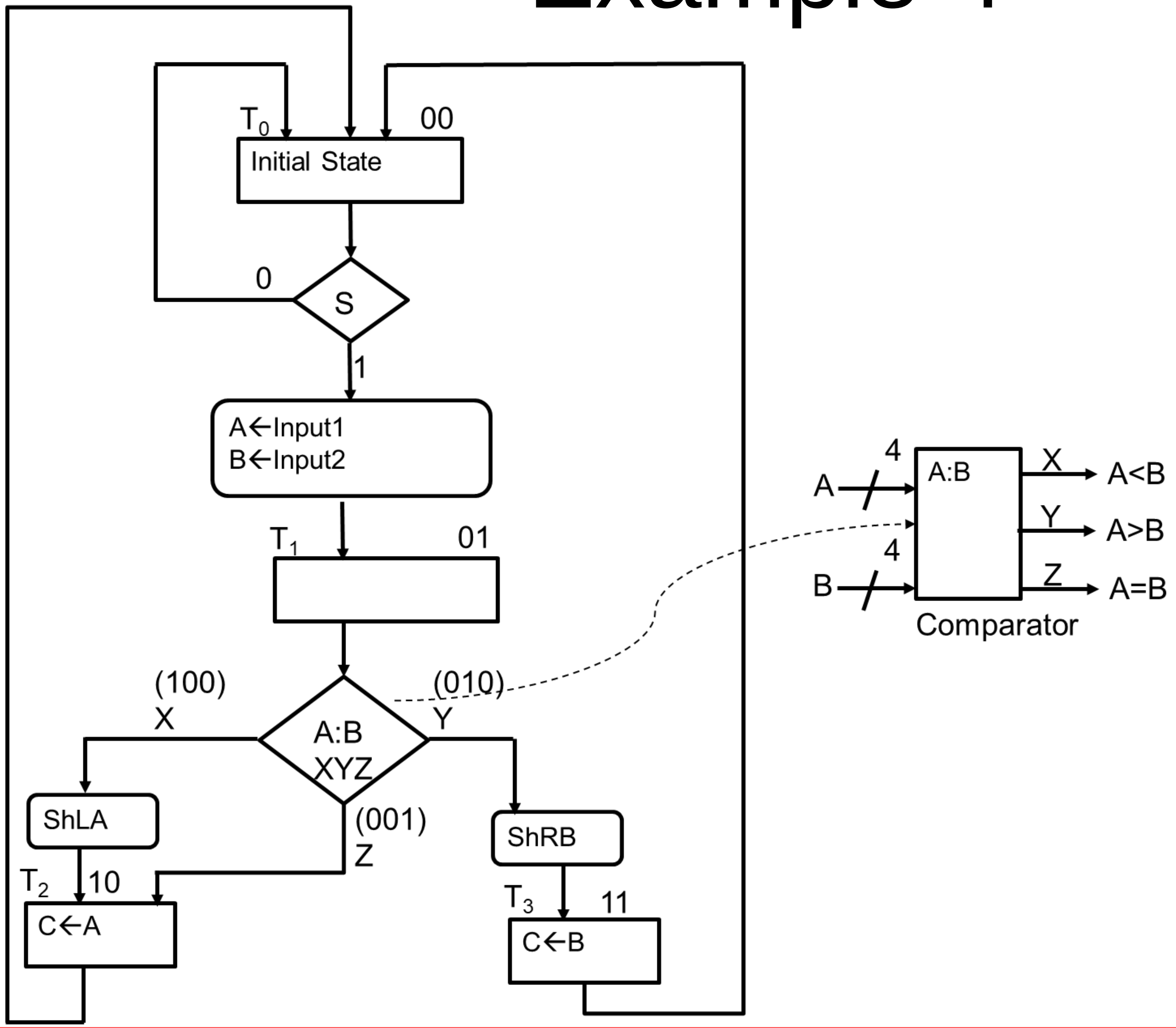
Controller for Example 3



Present State		Next State		Input Conditions	MUX1	MUX0
Name	G_1G_0	Name	$\overline{G_1}\overline{G_0}$			
T_0	00	T_0	00	S'	0	S
	00	T_1	01	S		
T_1	01	T_0	00	Z	Z'	0
	01	T_2	10	Z'		
T_2	10	T_3	11	Clock pulse	1	1
T_3	11	T_1	01	E	E'	E
	11	T_2	10	E'		

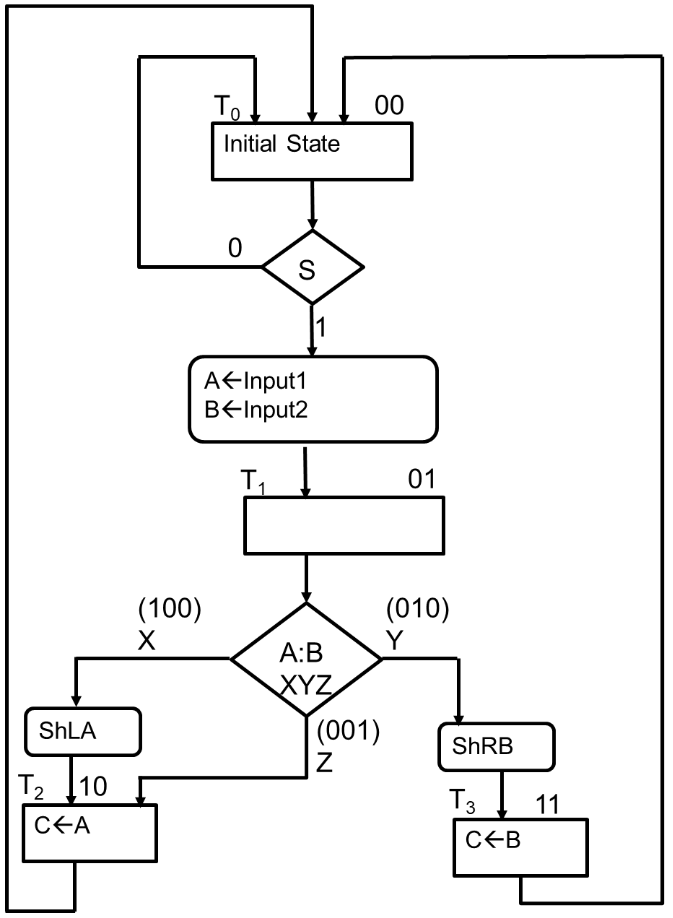
- $DG_1=G_1'G_0'0+G_1'G_0Z'+G_1G_0'1+G_1G_0E'$
- $DG_0=G_1'G_0'S+G_1'G_00+G_1G_0'1+G_1G_0E$

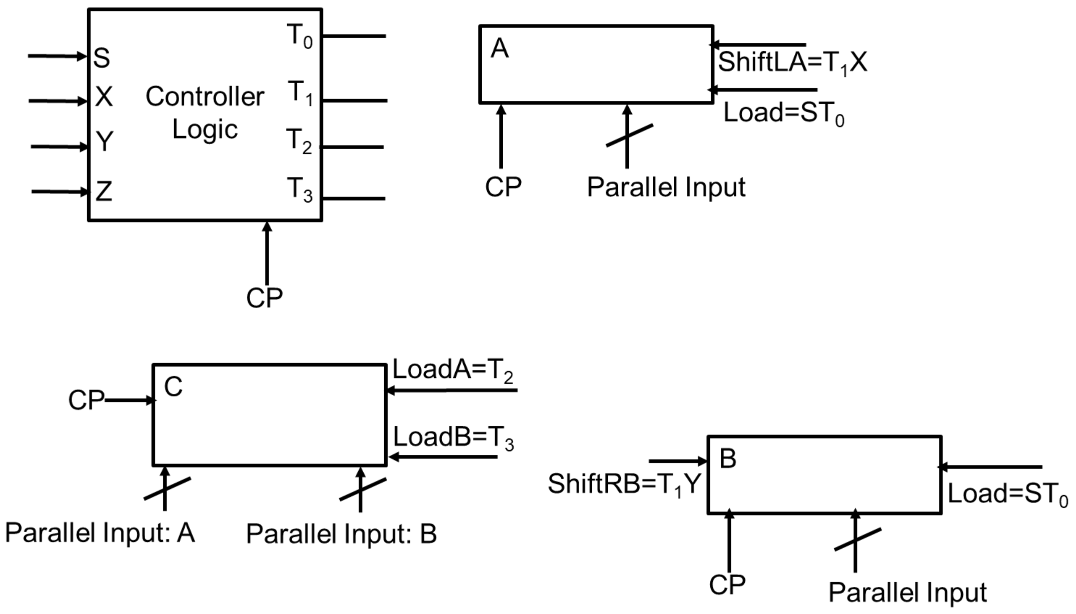
Example 4



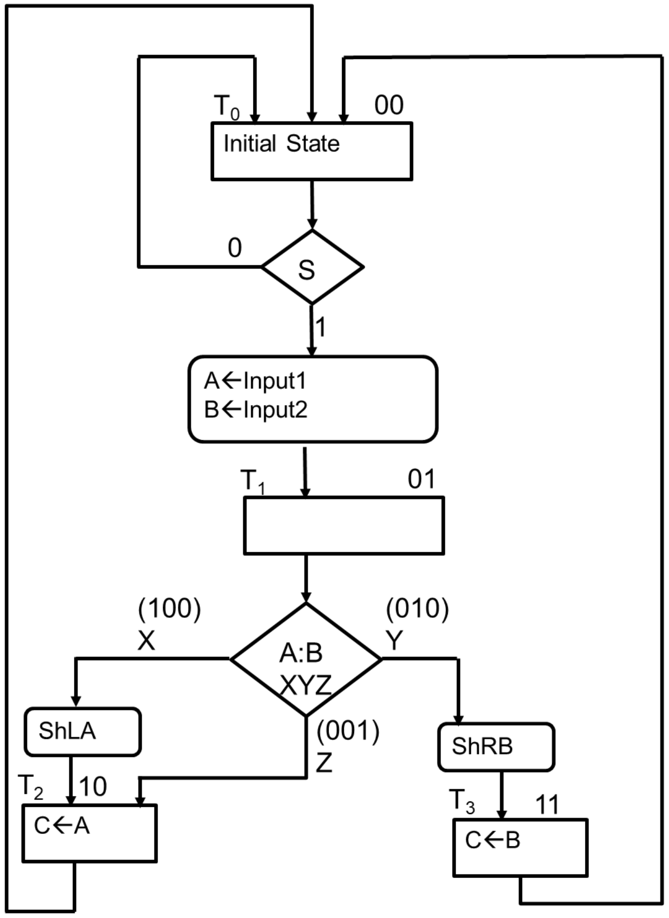
T0
00
Initial State
0
S
1
A←Input1
B←Input2
T1
01
(100)
X
(010)
Y
A:B
XYZ
(001)
Z
ShLA
ShRB
T2
10
C←A
T3
11
C←B
4
A
A:B
X
A<B
Y
A>B
4
B
Z
A=B
Comparator

Example 4 Datapath





Example 4 Controller



Present State		Next State		Input Conditions	MUX1	MUX0
Name	G_1G_0	Name	$\overline{G_1}\overline{G_0}$			
T_0	00	T_0	00	S'	0	S
	00	T_1	01	S		
T_1	01	T_2	10	X	1	Y
	01	T_3	11	Y		
	01	T_2	10	Z		
T_2	10	T_0	00	Clock pulse	0	0
T_3	11	T_0	00	Clock pulse	0	0

- $DG_1=G_1'G_0'0+G_1'G_01+G_1G_0'0+G_1G_00$
- $DG_0=G_1'G_0'S+G_1'G_0Y+G_1G_0'0+G_1G_00$

Example 5 Sequential Binary Multiplier

Algorithm M_1:

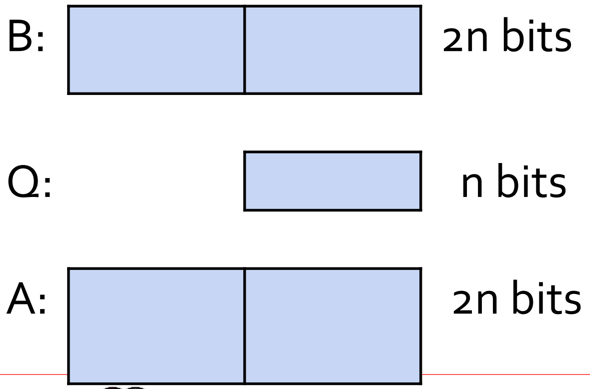
```
A←0, i=n;
repeat
{ if (Q[0]=1 then A←A+B;
  Q←shr Q;
  B←shl B; (insert 0 from right)
  dec i;
} until i=0
```

```
                1  1  1  0  : B
           x    1  1  0  1  : Q
             ----------------
                1  1  1  0
             0  0  0  0
          1  1  1  0
  +    1  1  1  0
  ----------------------------
    1  0  1  1  0  1  1  0  :A
```

A← B*Q (B: 2n bits, Q: n bits, A: 2n bits)

The above algorithm requires all registers to have shift capability

Register sizes required



Example 5 Sequential Binary Multiplier

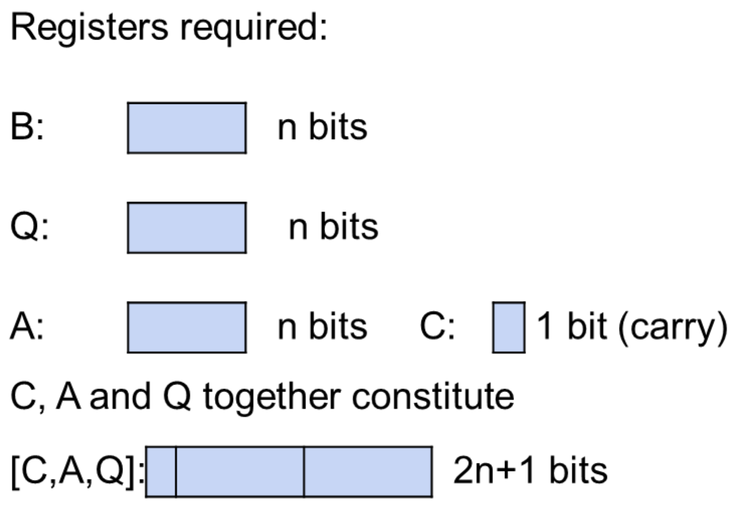
$$\begin{array}{ccccccccccl}
 & & & & & 1 & 1 & 1 & 0 & \rightarrow B \\
 & & & & \times & 1 & 1 & 0 & 1 & \rightarrow Q \\
\hline
 & & & & & 1 & 1 & 1 & 0 & \\
 & & & & 0 & 0 & 0 & 0 & & \\
 & & & 1 & 1 & 1 & 0 & & & \\
+ & & 1 & 1 & 1 & 0 & & & & \\
\hline
 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \rightarrow AQ
\end{array}$$

Alternatively:

- Instead of shifting the multiplicand towards left, shift the partial products towards right
- when LSB of Q=0, do not add

Algorithm M_2:

```
A←0, C←0; i=n;
repeat
{ dec i;
  if (Q[0]=1 then
     CA←A+B;
  CAQ←shr CAQ
} until i=0
```



Example 5 Sequential Binary Multiplier

$$
\begin{array}{rcccccccl}
 & & & & & 1 & 1 & 1 & 0 \rightarrow B \\
 & & & & \times & 1 & 1 & 0 & 1 \rightarrow Q \\
\hline
 & & & & & 1 & 1 & 1 & 0 \\
 & & & & 0 & 0 & 0 & 0 & \\
 & & & 1 & 1 & 1 & 0 & & \\
+ & & 1 & 1 & 1 & 0 & & & \\
\hline
 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \rightarrow AQ
\end{array}
$$

C	A				Q			
0	0	0	0	0	1	1	0	1

Q[0]=1 then
CA←A+B;

C	A				Q			
0	1	1	1	0	1	1	0	1

CAQ←shr CAQ (insert 0 from left)

C	A				Q			
0	0	1	1	1	0	1	1	0

<u>Algorithm M_2:</u>

```
A←0, C←0; i=n;
repeat
{ dec i;
  if (Q[0]=1 then
     CA←A+B;
  CAQ←shr CAQ (insert 0 from left)
} until i=0
```

Example 5 Sequential Binary Multiplier

C	A				Q			
0	0	1	1	1	0	1	1	0

Q[0]=0
CAQ←shr CAQ (insert 0 from left)

C	A				Q			
0	0	0	1	1	1	0	1	1

Q[0]=1 then
CA←A+B;

C	A				Q			
1	0	0	0	1	1	0	1	1

CAQ←shr CAQ (insert 0 from left)

C	A				Q			
0	1	0	0	0	1	1	0	1

```
            1 1 1 0  → B
          x 1 1 0 1  → Q
          ---------
            1 1 1 0
          0 0 0 0
        1 1 1 0
  +   1 1 1 0
  -----------------
    1 0 1 1 0 1 1 0  → AQ
```

Algorithm M_2:

```
A←0, C←0; i=n;
repeat
{ dec i;
  if (Q[0]=1 then
     CA←A+B;
  CAQ←shr CAQ (insert 0 from left)
} until i=0
```

Example 5 Sequential Binary Multiplier

C	A				Q			
0	1	0	0	0	1	1	0	1

Q[0]=1 then
    CA←A+B;

C	A				Q			
1	0	1	1	0	1	1	0	1

CAQ←shr CAQ (insert 0 from left)

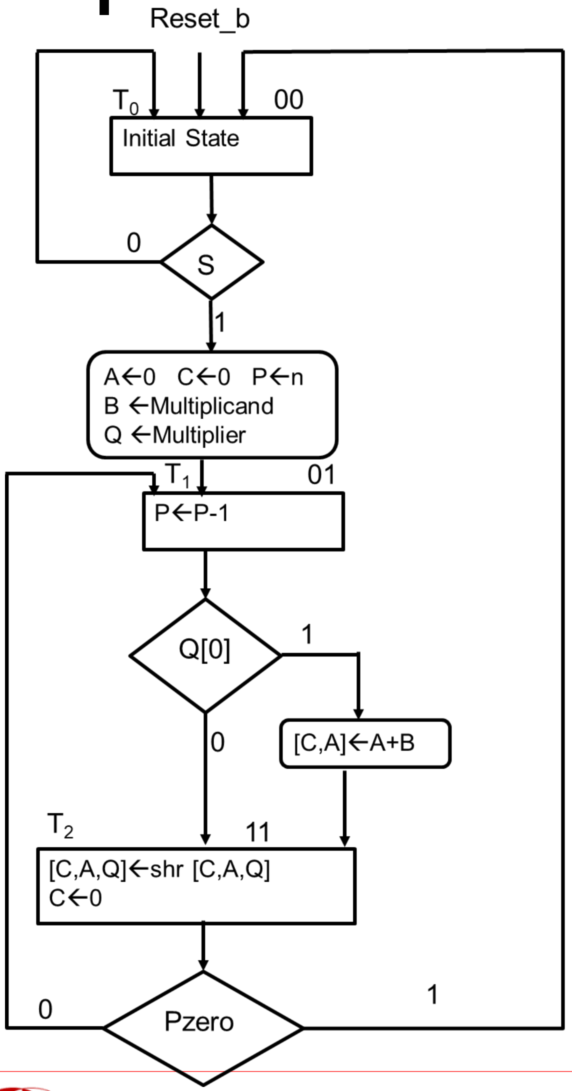
C	A				Q			
0	1	0	1	1	0	1	1	0

$$
\begin{array}{cccccccccl}
 & & & & & 1 & 1 & 1 & 0 & \rightarrow B \\
 & & & & \text{x} & 1 & 1 & 0 & 1 & \rightarrow Q \\
\hline
 & & & & & 1 & 1 & 1 & 0 & \\
 & & & & 0 & 0 & 0 & 0 & & \\
 & & & 1 & 1 & 1 & 0 & & & \\
+ & & 1 & 1 & 1 & 0 & & & & \\
\hline
 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \rightarrow AQ
\end{array}
$$

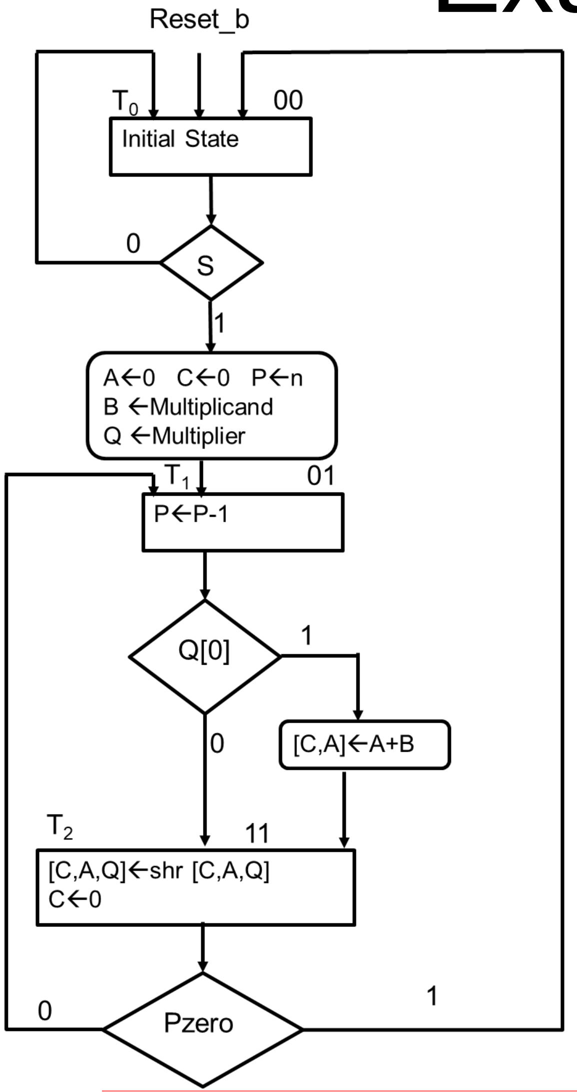
Algorithm M_2:

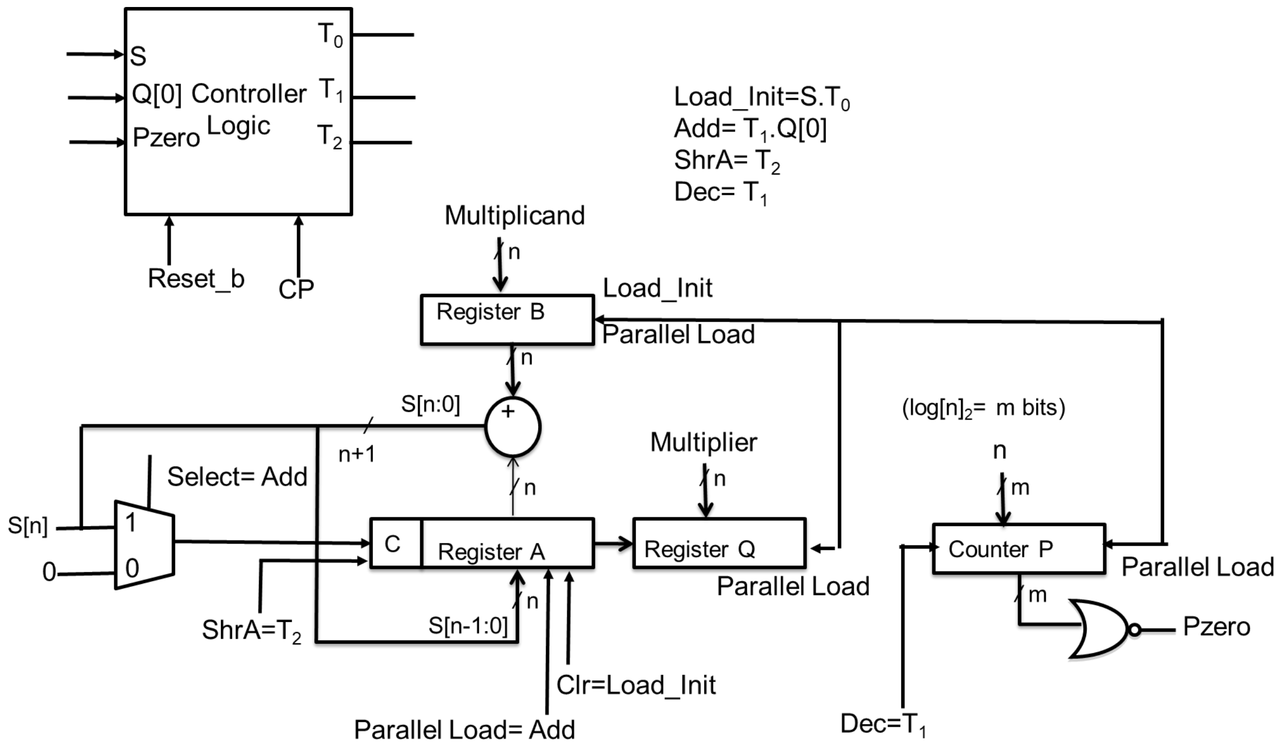
```
A←0, C←0; i=n;
repeat
{ dec i;
  if (Q[0]=1 then
     CA←A+B;
  CAQ←shr CAQ (insert 0 from left)
} until i=0
```

Example 5 ASM Chart



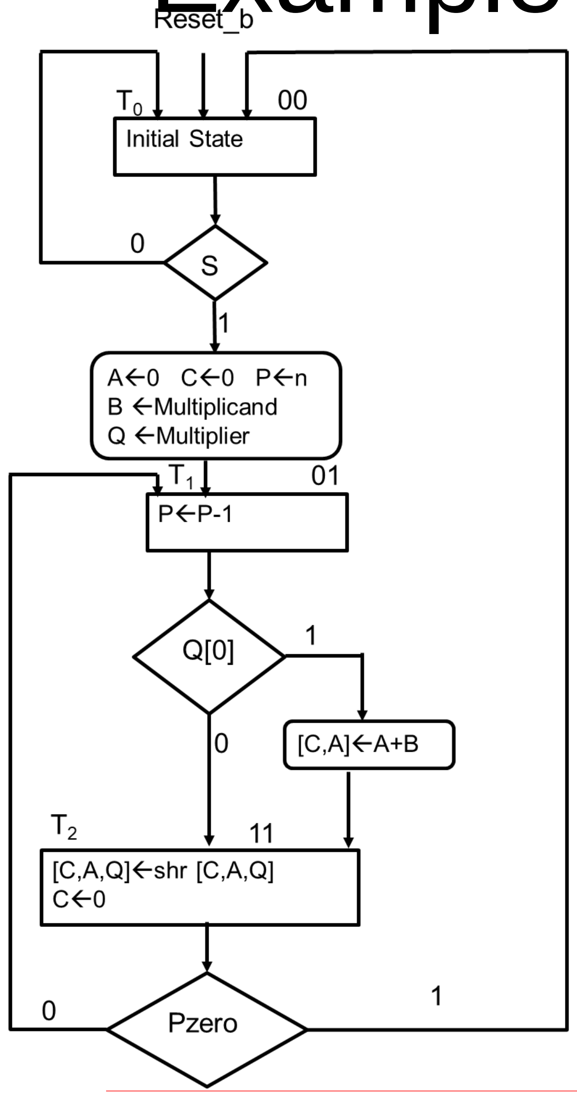
Example 5 Data Path





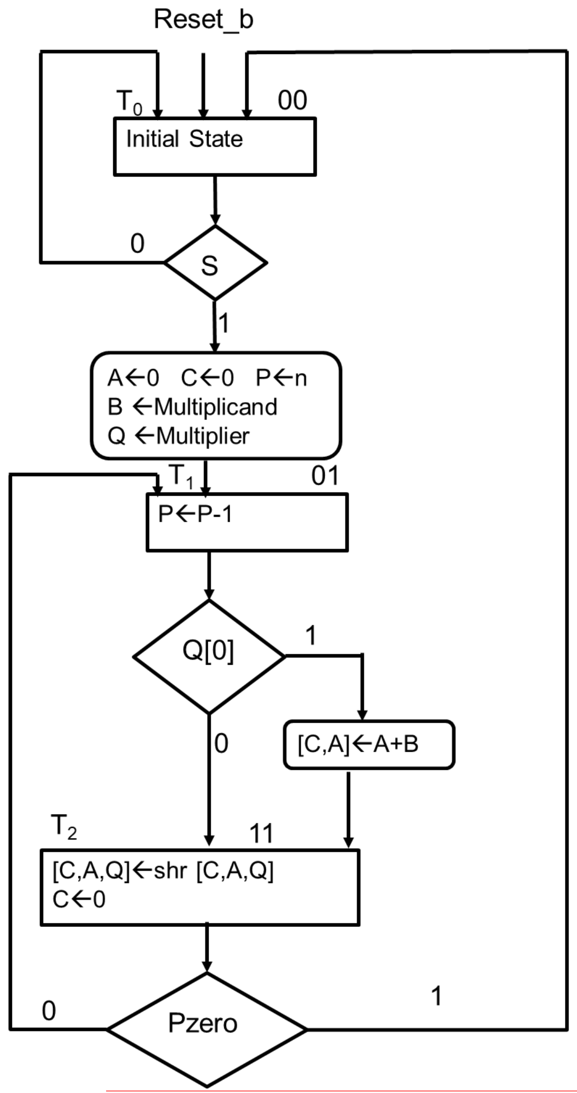
Load_Init=S.T_0
Add= T_1.Q[0]
ShrA= T_2
Dec= T_1

Example 5 Controller Logic: MUX

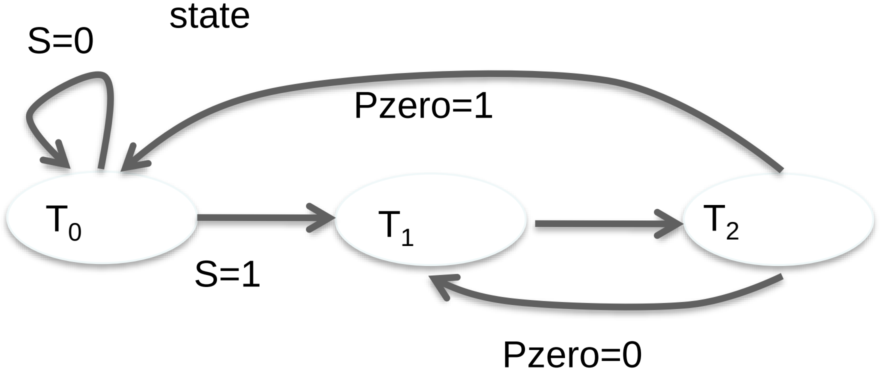


Present State		Next State		Input Conditions	MUX1	MUX0
Name	G_1G_0	Name	$\overline{G_1}\overline{G_0}$			
T_0	00	T_0	00	S'	0	S
	00	T_1	01	S		
T_1	01	T_2	10	Clock Pulse	1	0
T_2	10	T_1	01	Pzero'	0	Pzero'
	10	T_0	00	Pzero		

Example 5 Controller Logic: 1FF/State



1FF per state



- $D_0=S'T_0+PzeroT_2$
- $D_1=ST_0+ Pzero'T_2$
- $D_2=T_1$

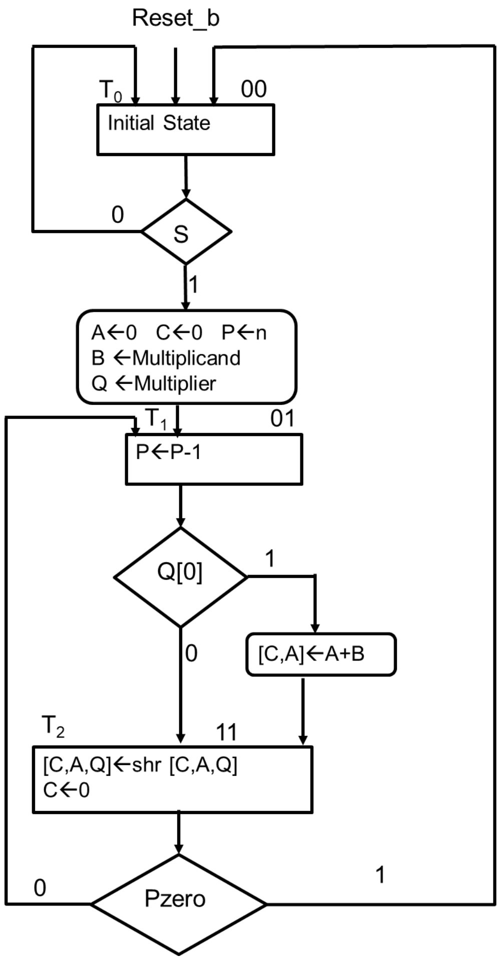
Load_Init=S.T_0
Add= T_1.Q[0]
ShrA= T_2
Dec= T_1

Execution

Multiplicand B=1110   Multiplier Q=1101 n=4 S=1

Each row shows the state at the beginning of the clock pulse.
The control signals take effect at the end of the clock pulse.

PS	C	A	Q	B	P	status		NS	outputs			
						Q[0]	Pzero		Load_Init	Add	ShrA	Dec
T_0								T_1	1	0	0	0
T_1	0	0000	1101	1110	100	1	0	T_2	0	1	0	1
T_2	0	1110	1101	1110	011	1	0	T_1	0	0	1	0
T_1	0	0111	0110	1110	011	0	0	T_2	0	0	0	1
T_2	0	0111	0110	1110	010	0	0	T_1	0	0	1	0
T_1	0	0011	1011	1110	010	1	0	T_2	0	1	0	1
T_2	1	0001	1011	1110	001	1	0	T_1	0	0	1	0
T_1	0	1000	1101	1110	001	1	0	T_2	0	1	0	1
T_2	1	0110	1101	1110	000	1	1	T_0	0	0	1	0
T_0	0	1011	0110	1110	000	0	1					



Algorithmic State Machine