Register Transfer Level Design

Chun-Jen Tsai
National Chiao Tung University
12/20/12

Review of Digital Systems

- A digital system is a sequential logic system constructed with flip-flops and gates
- To specify a large digital system with a state table is very difficult
- A large system is usually decomposed into modular subsystems
  - Each subsystem is composed of basic components such as: registers, decoders, multiplexers, arithmetic elements, and control logic.
  - These basic components are interconnected with datapaths and control signals.

Descriptions of Digital Designs

- Structural
  - The lowest and most detailed level
  - Specified in terms of physical components and their interconnection
- Register Transfer Level
  - Imply a certain hardware configuration
  - Specified in terms of the registers, operations performed, and control that sequences the operations.
- Algorithmic-based behavioral
  - The most abstract level
  - Most appropriate for simulating complex to verify design ideas and explore tradeoffs

Register Transfer Level Notation (1/2)

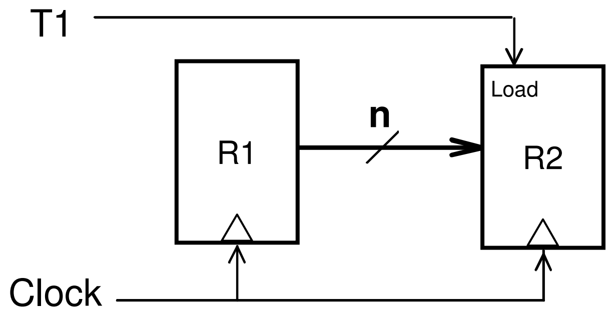
- A digital system is represented at the register transfer level (RTL) when it is specified by the following three components:
  - The set of **registers** in the system.
  - The **operations** performed on the data in the registers.
  - The **controllers** that supervise the sequence of operations.
- Notations:
  - Letters and numbers denotes a register (ex. R2, PC, …)
  - Parentheses denotes register bits (ex. R1(7:0), R2(4))
  - Arrow (←) denotes data transfer (ex. R1←R2)
  - Comma separates parallel operations (ex. R1←R2, R0←R3)
  - Brackets [ ] – Specifies a memory address (ex. R0←M[100])

Register Transfer Level Notation (2/2)

- A conditional statement governing a register transfer operation is symbolized with an if-then statement
  - Example:

    If (T1 = 1) then (R2 $\leftarrow$ R1)

    where T1 is a control signal generated in the controller.
  - The statement can also be expressed as T1: (R2 $\leftarrow$ R1)
  - A logic diagram for the statement is as follows:

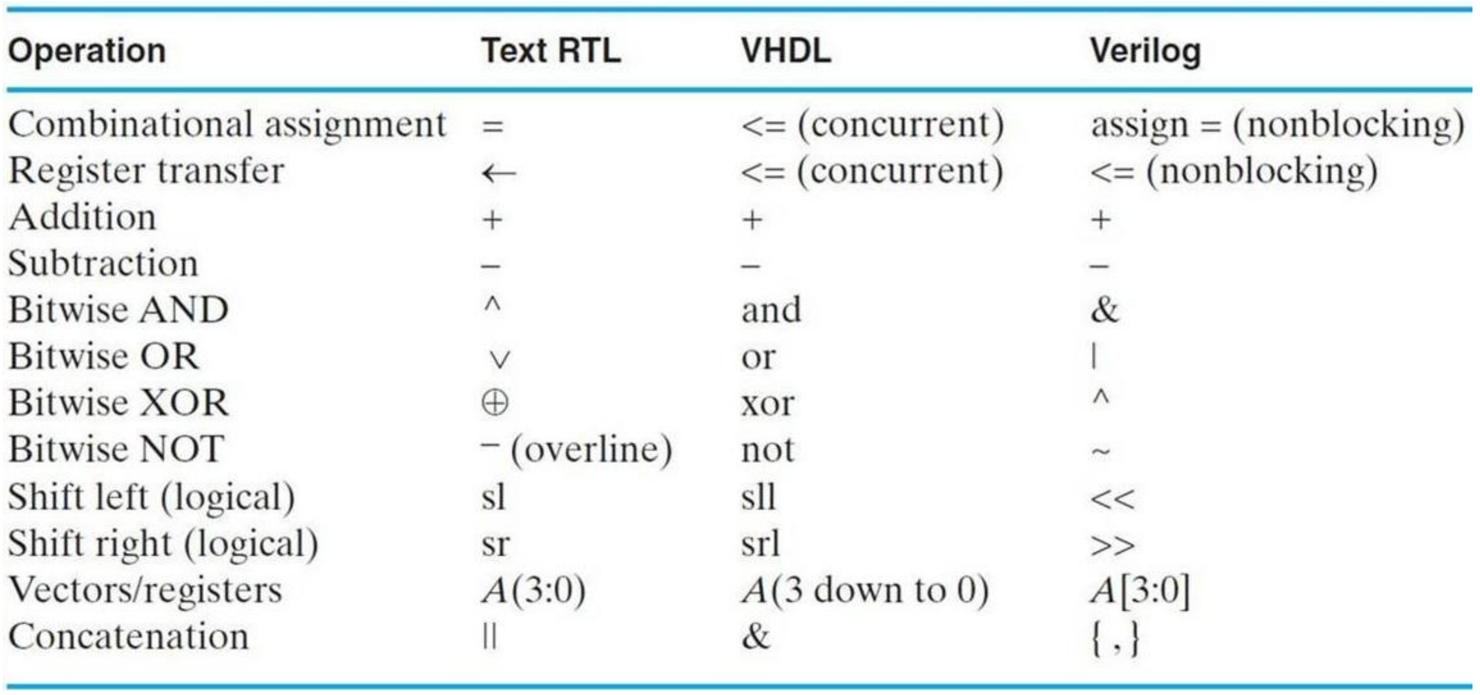


Register Operations

- The type of operations most often encountered in digital systems are as follows:
  - Transfer: transfer data from one register to another.
  - Arithmetic: perform arithmetic on data in registers.
  - Logic: perform bit manipulation of data in registers.
  - Shift: shift data between registers.
- Example of operations:
  - $R1 \leftarrow R1 + R2$ // Add contents of R2 to R1
  - $R3 \leftarrow R3 + 1$ // Increment R3 by 1
  - $R4 \leftarrow \text{shr } R4$ // Shift right R4
  - $R5 \leftarrow 0$ // Clear R5 to 0

Operator Notations in HDL

- The notation for various operations in RTL models (text, VHDL, and Verilog) are as follows:



Operation	Text RTL	VHDL	Verilog
Combinational assignment	=	<= (concurrent)	assign = (nonblocking)
Register transfer	←	<= (concurrent)	<= (nonblocking)
Addition	+	+	+
Subtraction	−	−	−
Bitwise AND	∧	and	&
Bitwise OR	∨	or	|
Bitwise XOR	⊕	xor	^
Bitwise NOT	¯ (overline)	not	~
Shift left (logical)	sl	sll	<<
Shift right (logical)	sr	srl	>>
Vectors/registers	*A*(3:0)	*A*(3 down to 0)	*A*[3:0]
Concatenation	||	&	{ , }

RTL Model and Logic Synthesis

- An RTL model is a “program” that describes the operations (behavior) of a digital circuit based on register operations
- An RTL model can be translated into a gate-level model by a process called “logic synthesis.”
  - A gate-level model usually goes through the mapping and the place-and-route processes to become an implementation model for a specific target hardware.
- A continuous assignment in an RTL model is synthesized into combinational circuit:

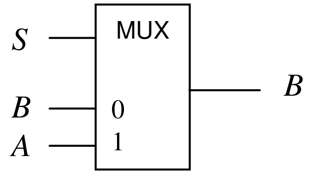
```
assign S = A + B;
```

where `S` is a signal wire.

Logic Synthesis of Cyclic Behavior

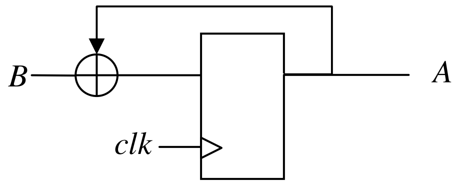
- A cyclic behavior may imply a combinational or a sequential circuit, depending on whether the event control expression is level sensitive or edge sensitive
  - Examples:

```
always@(A or B or S)
begin
    if (S) Y = A;
    else Y = B;
end
```



synthesized into a combinational circuit block.

```
always@(posedge clk)
begin
    A <= A + B;
end
```

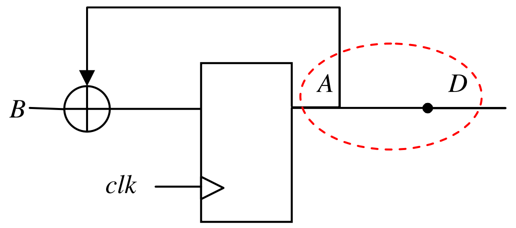


translates into a synchronous sequential circuit block.

Blocking vs. Non-blocking Behavior

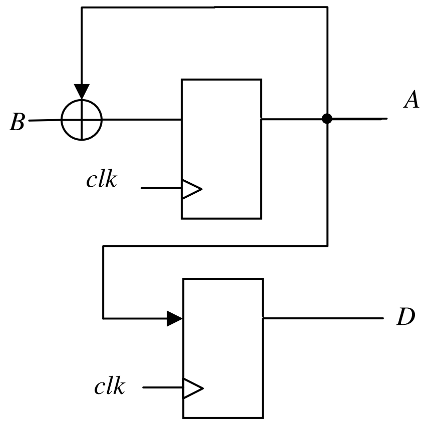
- Blocking operation is for combinational behavior:

```
always@(posedge clk)
begin
     A = A + B;
     D = A;
end
```

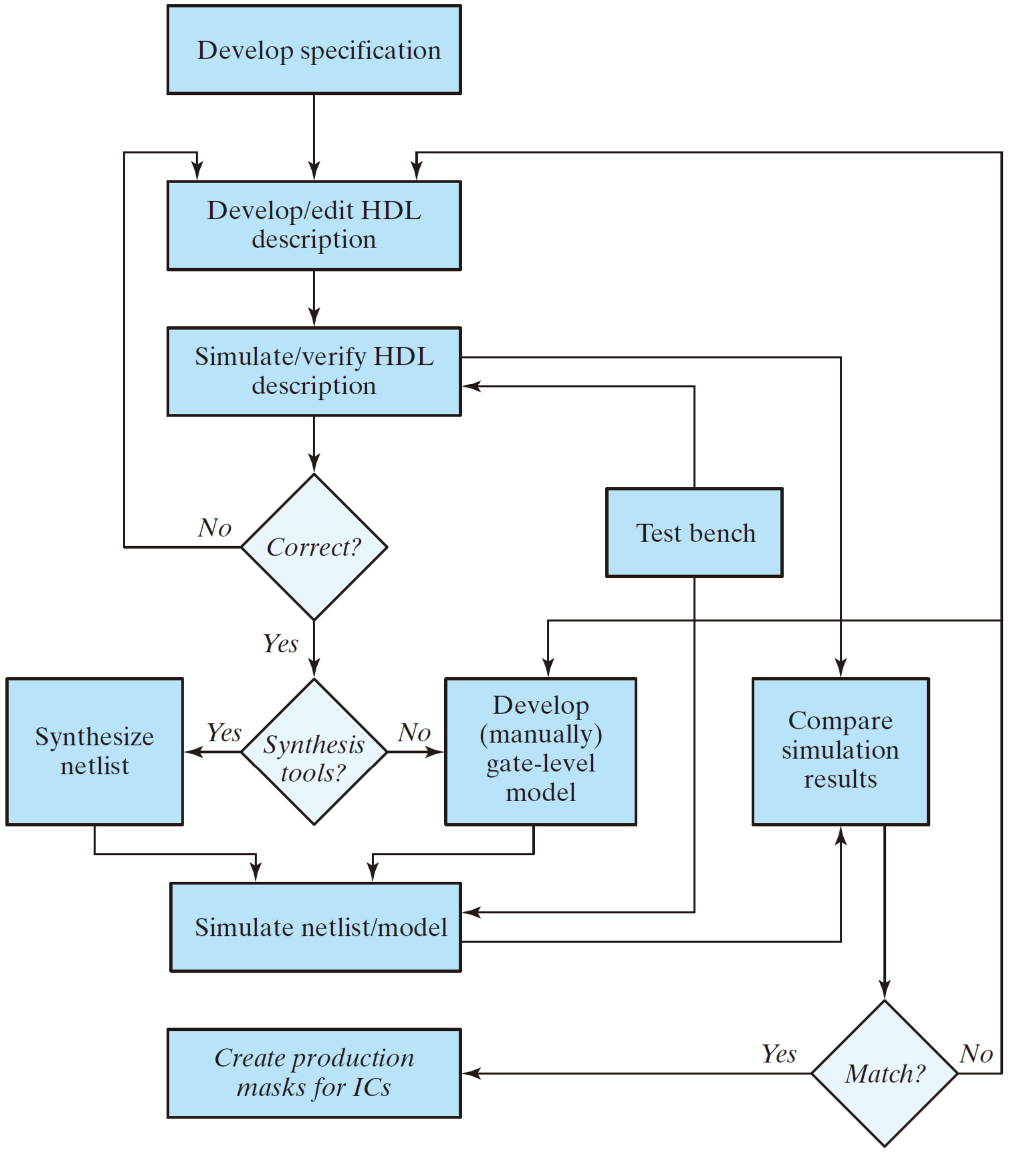


- Non-blocking operation is for sequential behavior:

```
always@(posedge clk)
begin
     A <= A + B;
     D <= A;
end
```

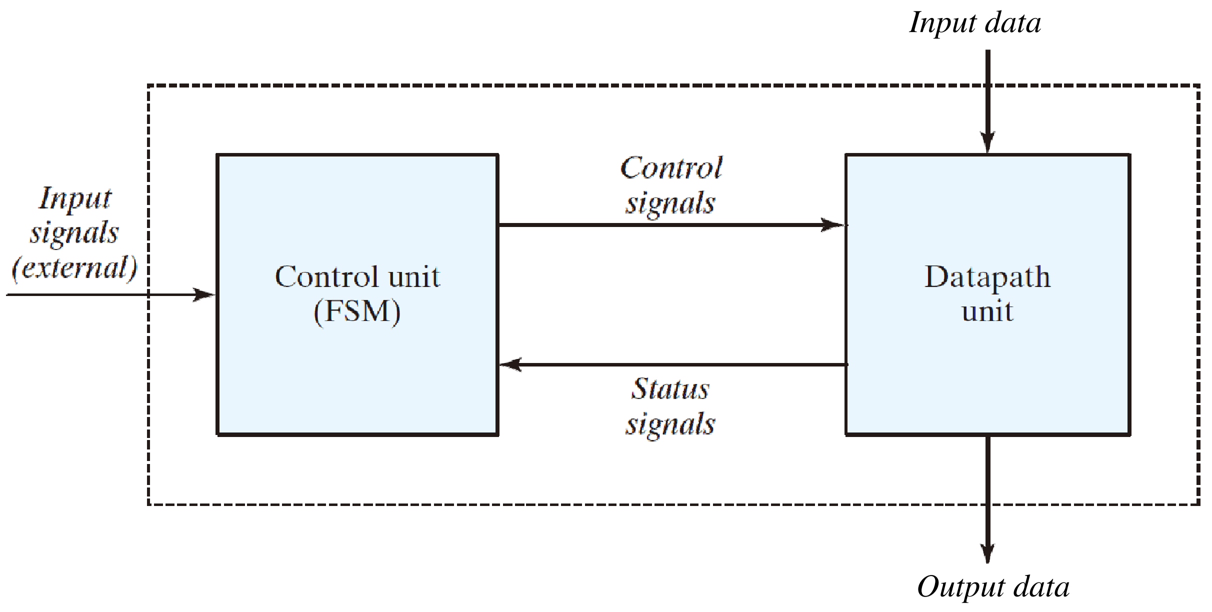


HDL-based Design Flow


Develop specification
Develop/edit HDL description
Simulate/verify HDL description
No
Correct?
Yes
Test bench
Synthesize netlist
Yes
Synthesis tools?
No
Develop (manually) gate-level model
Compare simulation results
Simulate netlist/model
Create production masks for ICs
Yes
Match?
No

Digital Systems = FSM + DP (1/2)

- Logic design of digital system can be divided into two distinct parts:
  - Datapath design – the design of the digital circuits that perform the data-processing operations
  - Controller design – the design of the control circuits that arrange the sequence in which the actions are performed.

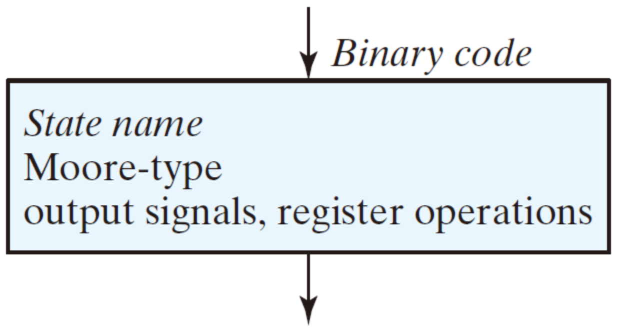


Digital Systems = FSM + DP (2/2)

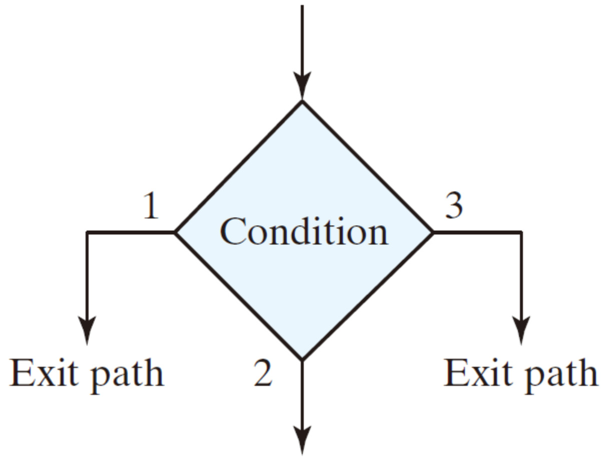
- The control logic that generates the signals for sequencing the operations in the data path unit is a finite state machine (FSM).
  - The control sequence and datapath tasks of the digital system are specified by means of a hardware algorithm.
- A flowchart that has been developed specifically to define digital hardware algorithm is called an algorithmic state machine (ASM) chart.

ASM Charts

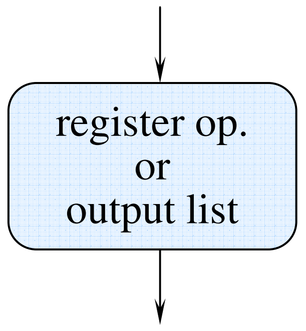
- ❑ The ASM chart is composed of three basic elements:
  - State box
  - Decision box
  - Conditional box
- ❑ They are connected by directed edges indicating the sequential precedence and evolution of the states as the machine operates.



State box

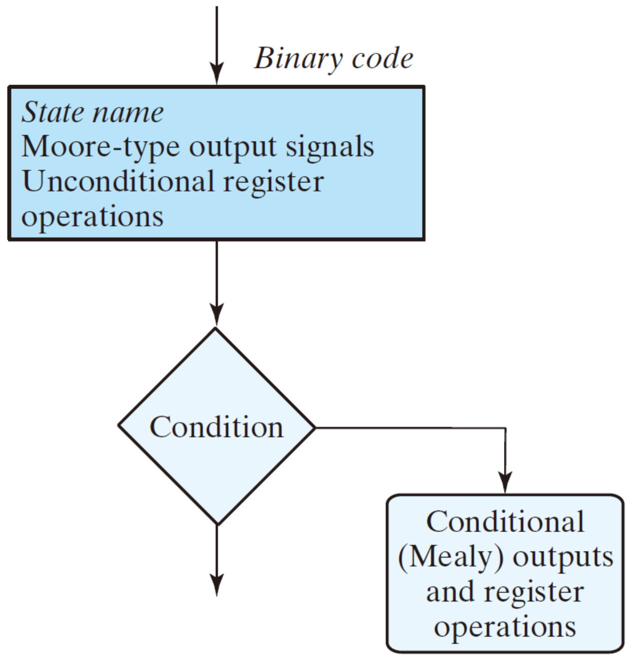


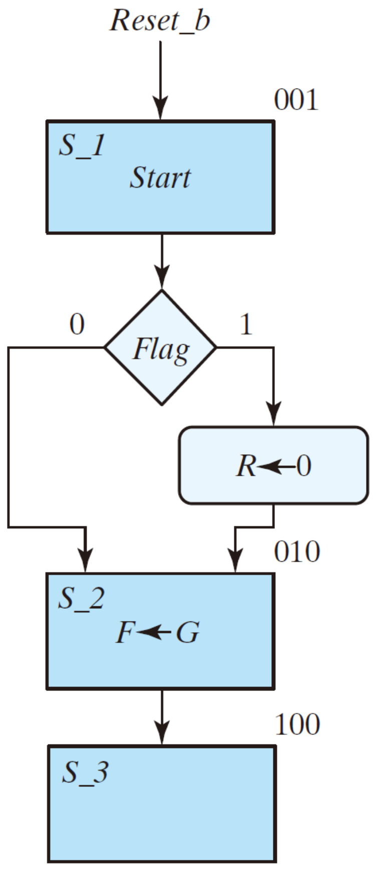
Decision box

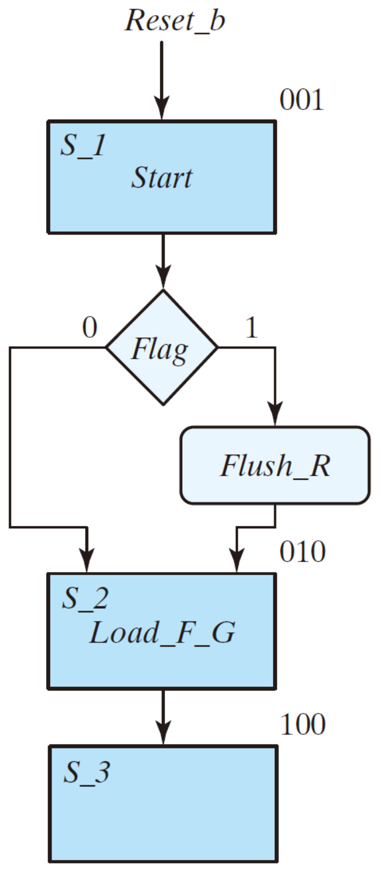


Conditional box

AMS Chart Examples

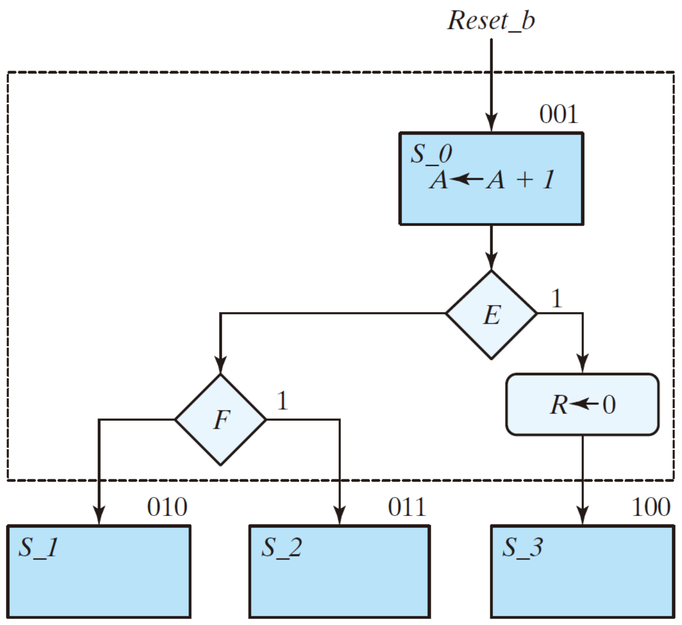

Binary code
State name
Moore-type output signals
Unconditional register operations
Condition
Conditional (Mealy) outputs and register operations

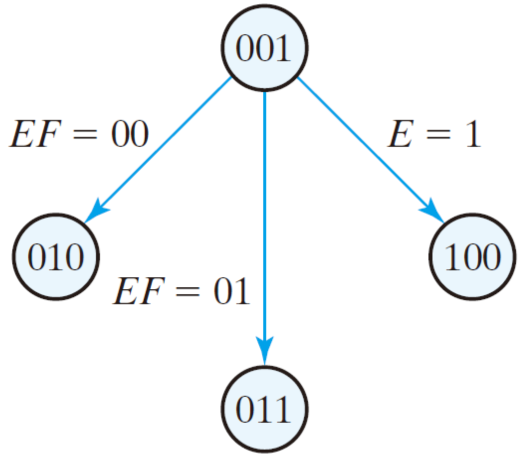




ASM Blocks

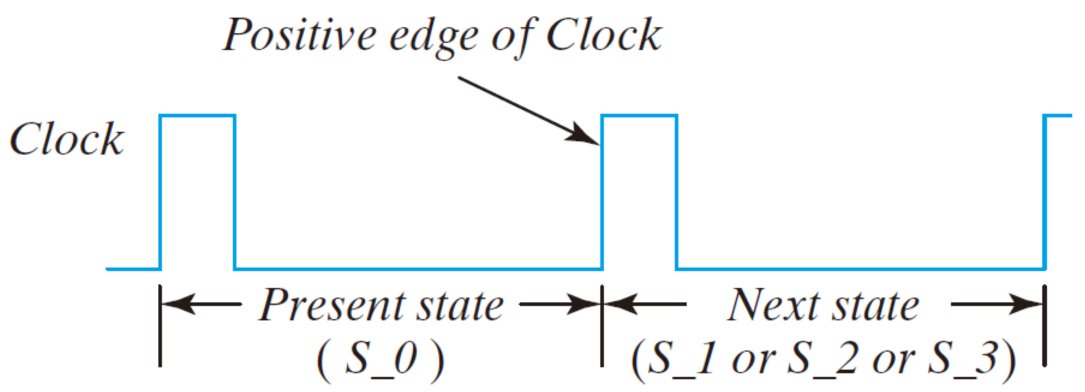
- An ASM chart is composed of many ASM blocks
  - Each block contains exactly one state box and describes the state of the system during one clock-pulse interval.
  - May have decision boxes and conditional boxes.
  - One entrance path, one or more exit paths.





Timing Considerations

- The timing for all register and flip-flop in digital system is controlled by a master-clock generator
- All the operations within an ASM block must be completed within one clock period
  - All operations in a block are concurrent



ASMD Chart

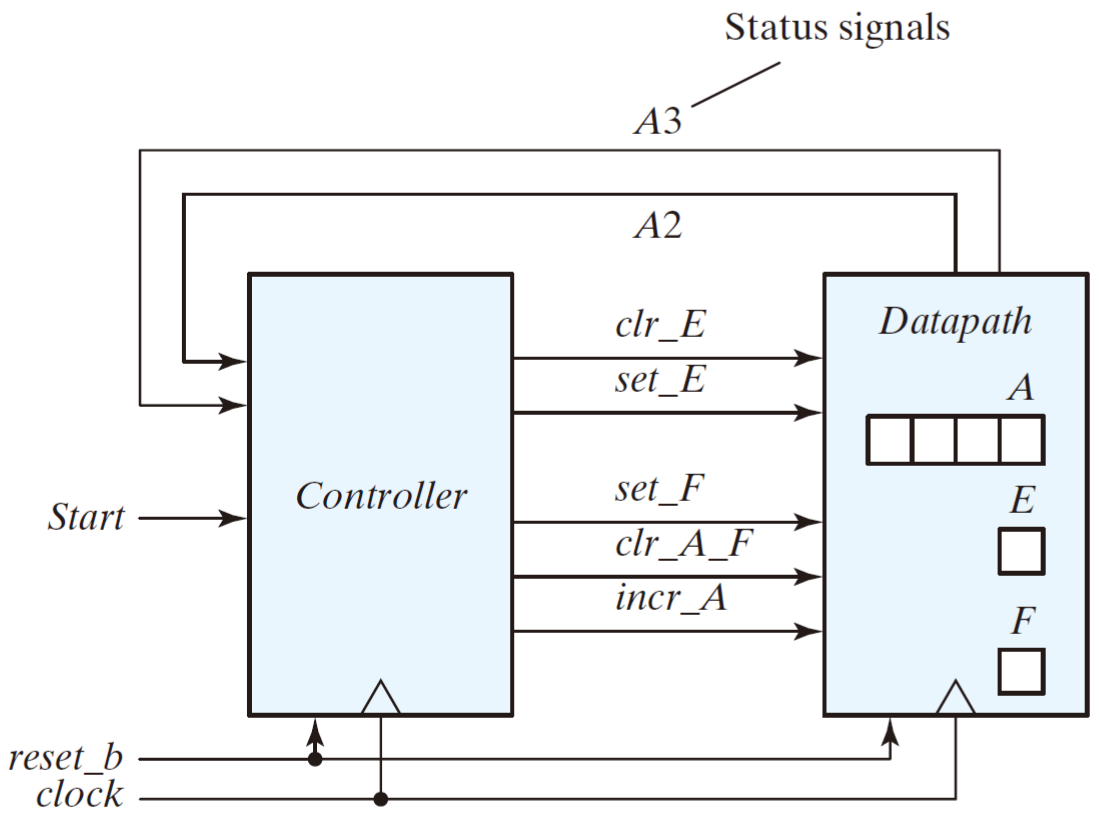
- An Algorithmic State Machine and Datapath (ASMD) chart is a variant to a ASM chart:
    - ASMD charts do not list register operations in state boxes.
    - The edges of an ASMD charts are annotated with register operations that are concurrent with the state transition indicated by the edge.
    - An ASMD chart includes conditional boxes identifying the signals which control the register operations that annotate the edges of the chart.
    - An ASMD chart associates register operations with state transitions rather than with state.

ASMD Chart Design

- Designing an ASMD chart has three steps:
  - Form an ASM chart displaying only how the inputs to the controller determine its state transitions.
  - Convert the ASM chart to an ASMD chart by annotating the edges of ASM chart to indicate the concurrent register operations of the datapath unit.
  - Modify the ASMD chart to identify the control signals that are generated by the controller which cause the indicated register operations in the datapath unit.

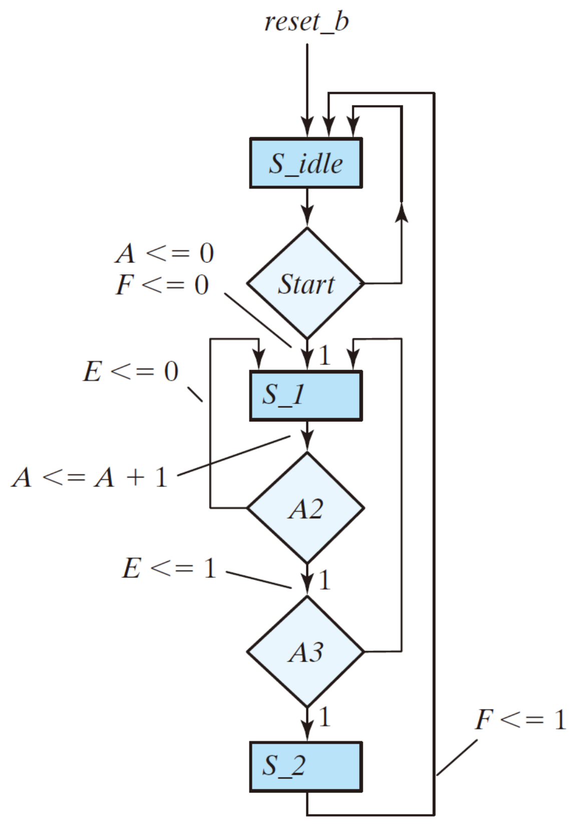
ASMD Chart Design Example

- A binary counter with two flags:

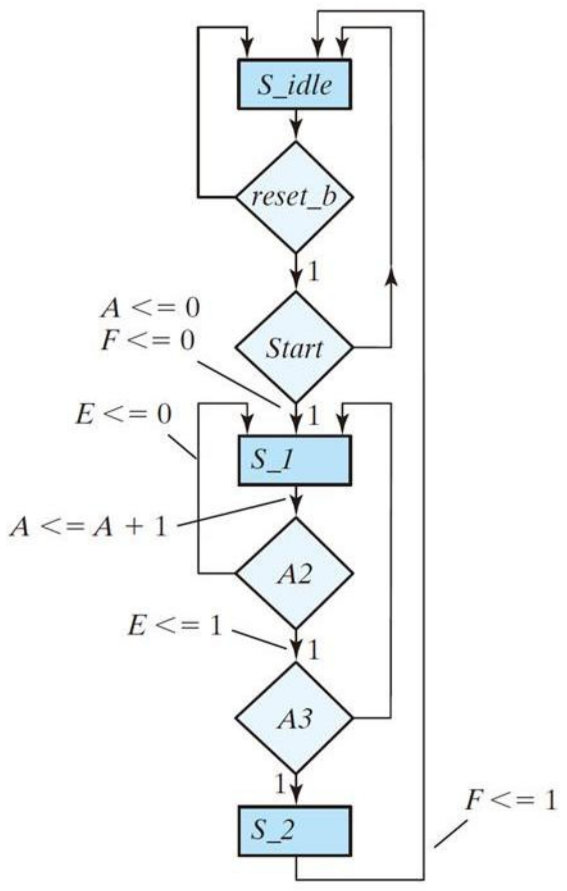


ASM Chart for State Transitions

- State transitions with different reset behaviors:



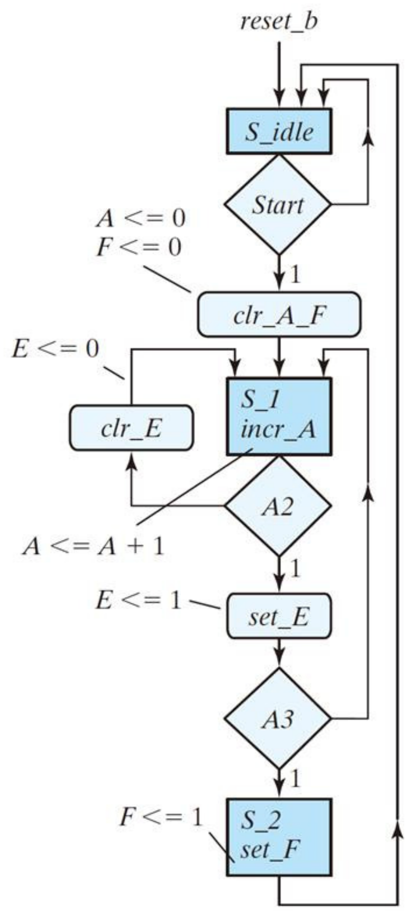
asynchronous reset



synchronous reset

Complete Controller ASM Chart

- A complete controller generates various control signals throughout state transitions:



Example of Operations

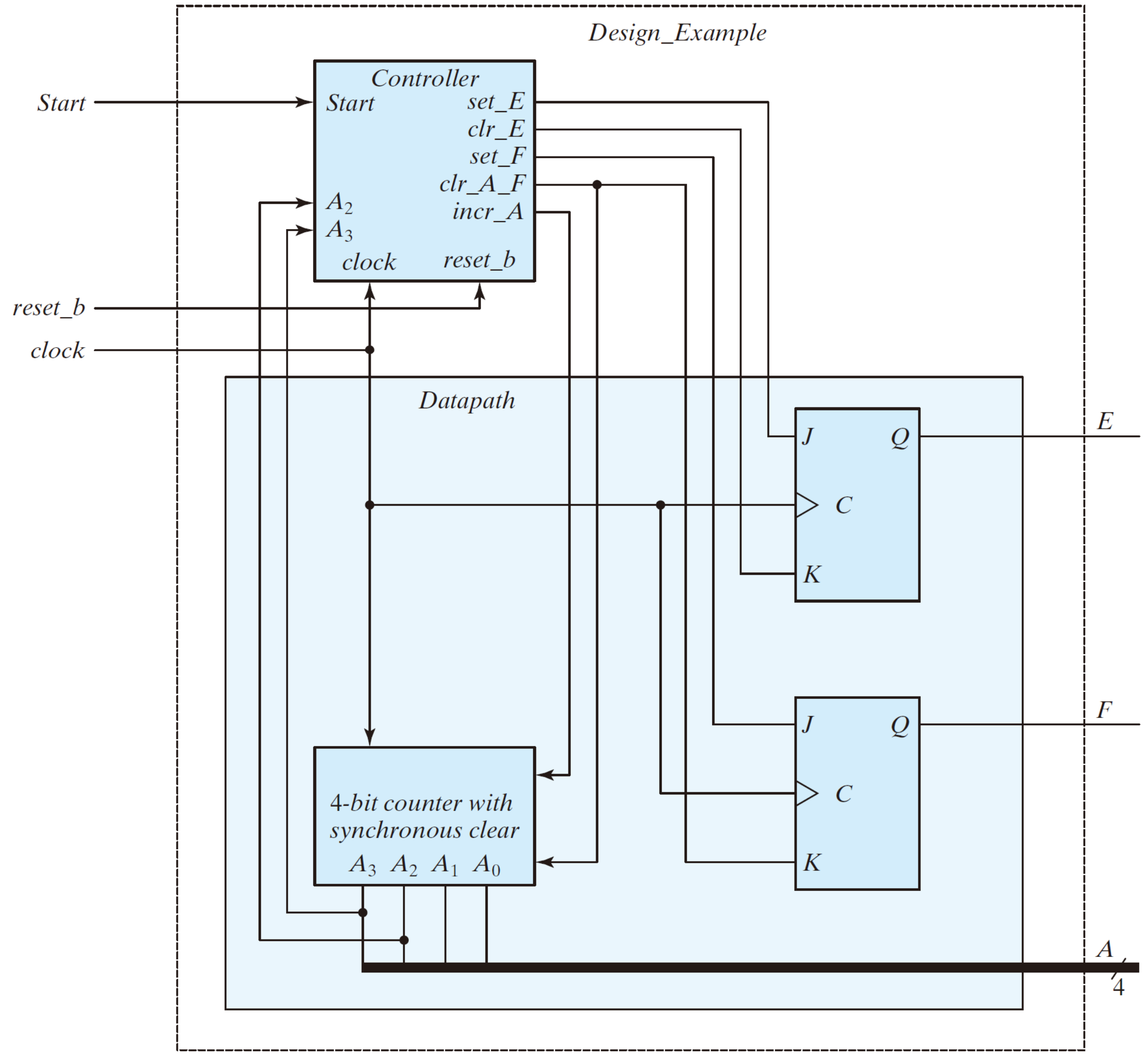
- An example run of controller operation is as follows:

progress of timing clock ticks

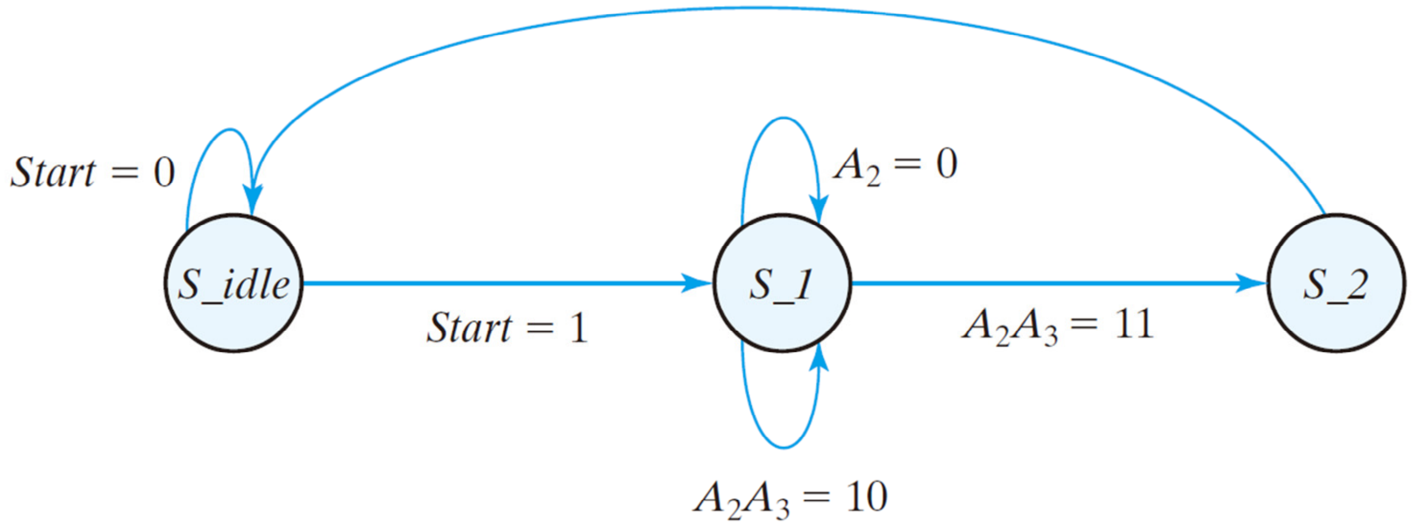
Sequence of Operations for Design Example

Counter				**Flip-Flops**			
A_3	A_2	A_1	A_0	E	F	**Conditions**	**State**
0	0	0	0	1	0	$A_2 = 0,\ A_3 = 0$	*S_1*
0	0	0	1	0	0		
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	$A_2 = 1,\ A_3 = 0$	
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	$A_2 = 0,\ A_3 = 1$	
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	$A_2 = 1,\ A_3 = 1$	
1	1	0	1	1	0		*S_2*
1	1	0	1	1	1		*S_idle*

Datapath and Controller



Register Transfer Level Description



$S\_idle \longrightarrow S\_1, clr\_A\_F$:		$A \longleftarrow 0, F \longleftarrow 0$
$S\_1 \longrightarrow S\_1, incr\_A$:		$A \longleftarrow A + 1$
	if $(A_2 = 1)$ then set_E:	$E \longleftarrow 1$
	if $(A_2 = 0)$ then clr_E:	$E \longleftarrow 0$
$S\_2 \longrightarrow S\_idle, set\_F$:		$F \longleftarrow 1$

State Table and FF Input Equations

- State flip-flops (D-type) input equations:
  - G_1 input equation: $D_{G_1} = S\_1 \cdot A_2 \cdot A_3$
  - G_0 input equation: $D_{G_0} = Start \cdot S\_idle + S\_1$

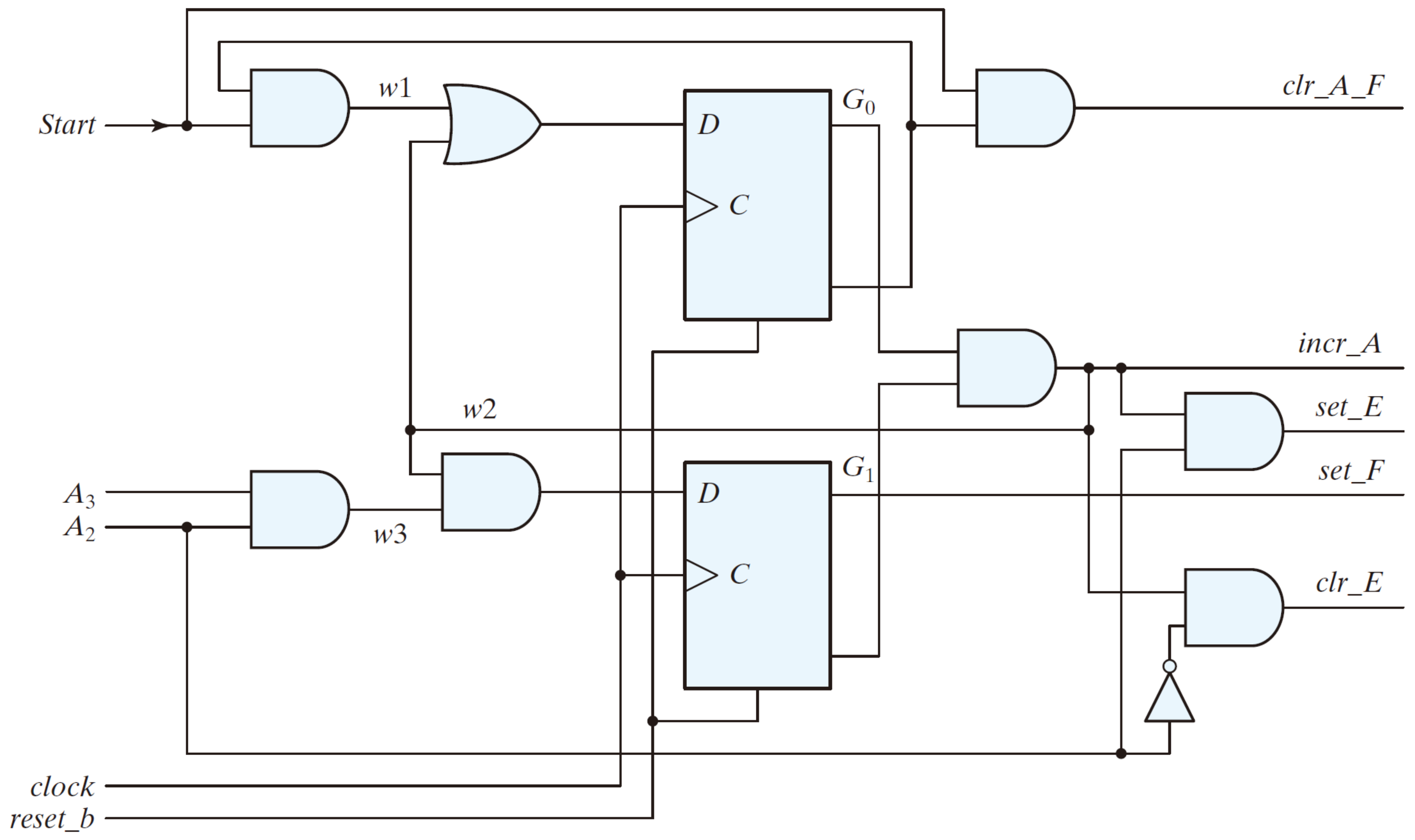
State Table for the Controller

	Present State		Inputs			Next State		Outputs				
Present-State Symbol	G_1	G_0	*Start*	A_2	A_3	G_1	G_0	*set_E*	*clr_E*	*set_F*	*clr_A_F*	*incr_A*
S_idle	0	0	0	X	X	0	0	0	0	0	0	0
S_idle	0	0	1	X	X	0	1	0	0	0	1	0
S_1	0	1	X	0	X	0	1	0	1	0	0	1
S_1	0	1	X	1	0	0	1	1	0	0	0	1
S_1	0	1	X	1	1	1	1	1	0	0	0	1
S_2	1	1	X	X	X	0	0	0	0	1	0	0

Output Equations

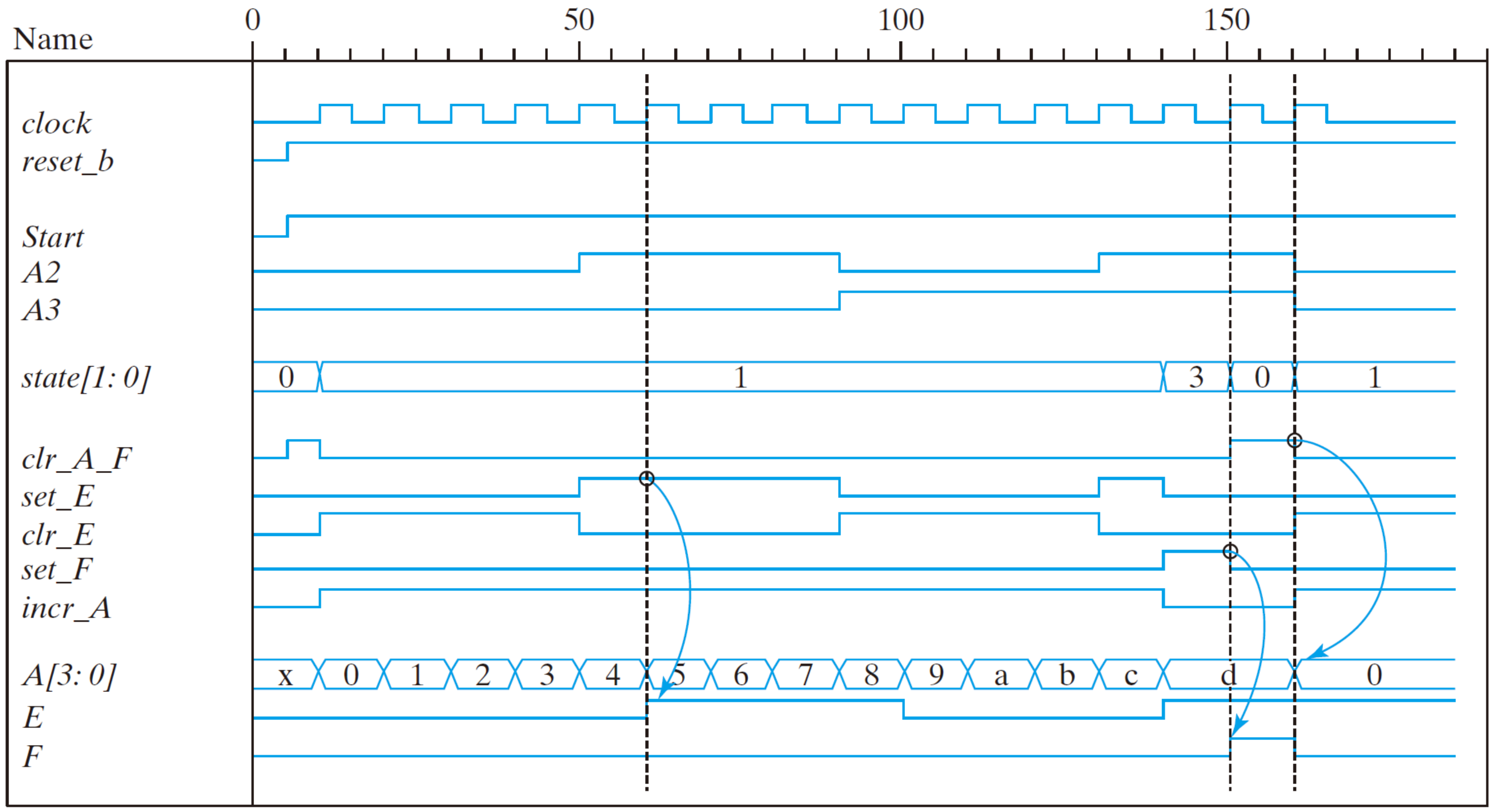
- To derive the five output function, we can exploit the fact that state 10 is not used, which simplifies the equation for clr_A_F :
  - $set\_E = S\_1 \cdot A_2$
  - $clr\_E = S\_1 \cdot A'_2$
  - $set\_F = S\_2$
  - $clr\_A\_F = Start \cdot S\_idle$
  - $incr\_A = S\_1$

Logic Diagram of the Controller



Waveform Simulation

- Verification of a digital design begins with functional waveform simulation:



Sequential Binary Multiplier

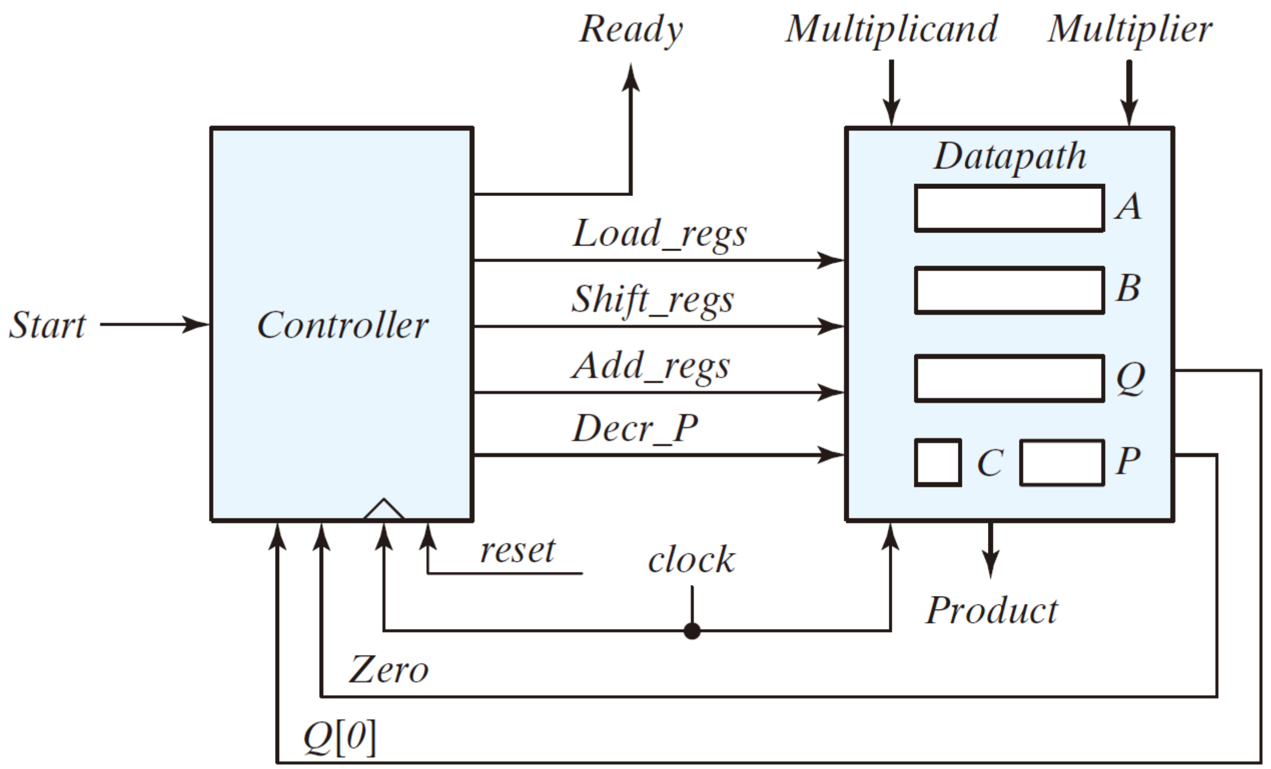
- Example behavior of sequential binary multiplication of 10111 and 10011 is as follows:

```
          10111   → multiplicand
 ×        10011   → multiplier
 ----------------
          10111
         10111
        00000
       00000
 +    10111
 ----------------
      110110101   → product
```

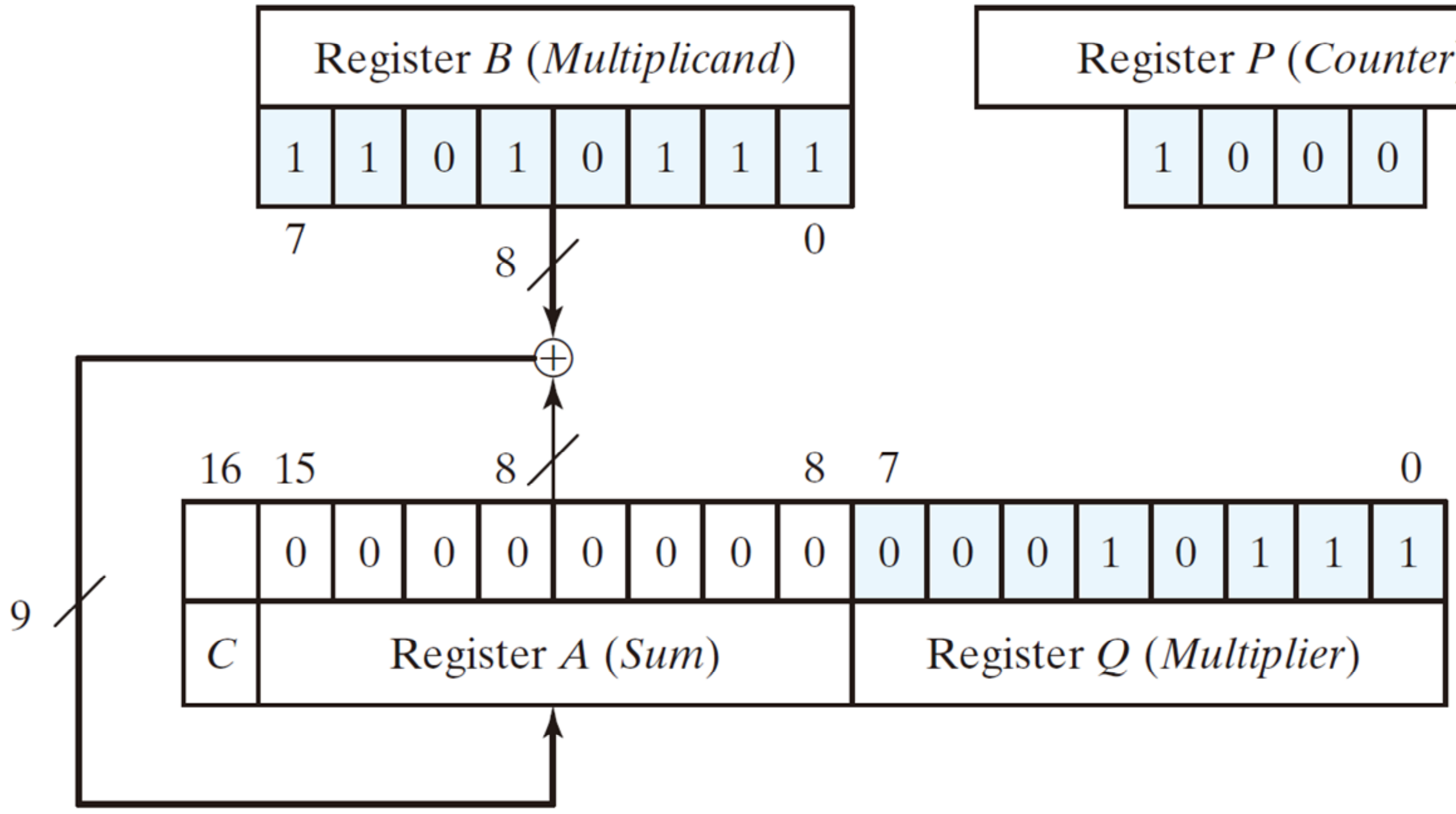
The product obtained from the multiplication of two binary numbers of n bits each can have up to $2n$ bits.

Register Configuration

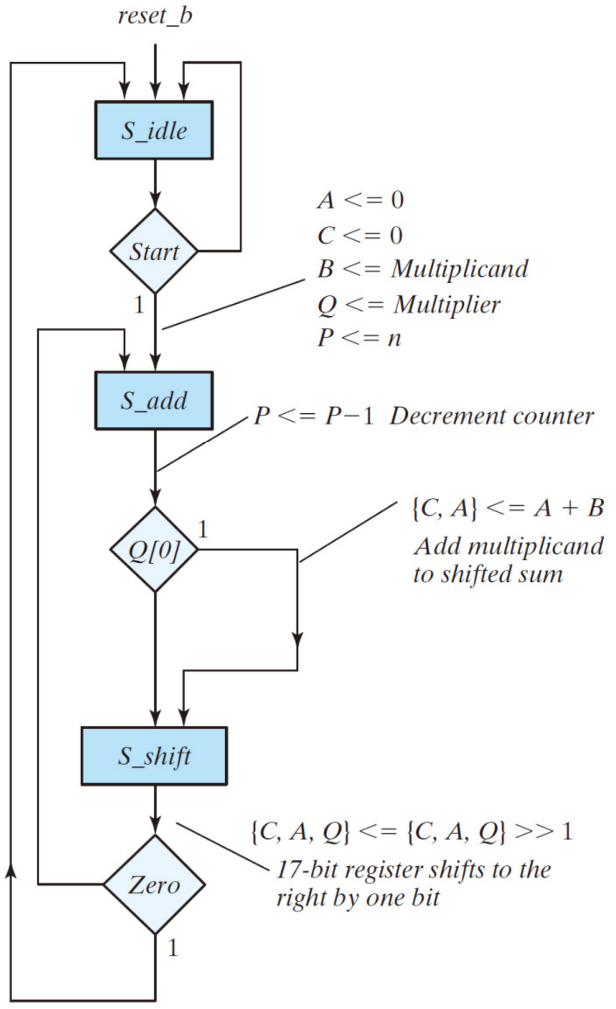
- The block diagram of the sequential multiplier:



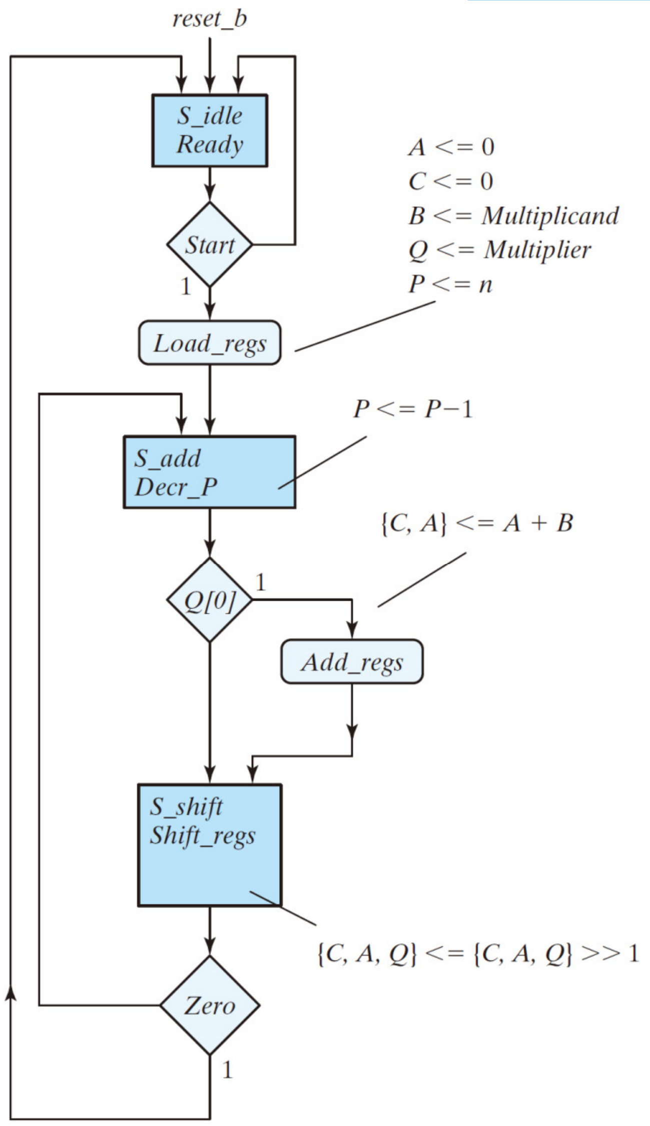
Datapath of the Multiplier



ASMD Chart of the Multiplier



reset_b
S_idle
Start
1
A <= 0
C <= 0
B <= Multiplicand
Q <= Multiplier
P <= n
S_add
P <= P−1 Decrement counter
Q[0]
1
{C, A} <= A + B
Add multiplicand to shifted sum
S_shift
{C, A, Q} <= {C, A, Q} >> 1
17-bit register shifts to the right by one bit
Zero
1



Register Notations for Shifting

- The shift operator $>>$ in the chart for concatenated registers $\{C, A, Q\}$ is equivalent to the following register transfer notation:

$$\text{Shift right } CAQ,\ C \leftarrow 0$$

- In terms of individual register symbols, the shift operation can be described by the following register operations:

$$\begin{aligned}&A \leftarrow \text{shr } A,\ A_{n-1} \leftarrow C\\&Q \leftarrow \text{shr } Q,\ Q_{n-1} \leftarrow A_0\\&C \leftarrow 0\end{aligned}$$

Example of Operations

- The registers change during a multiplication process:

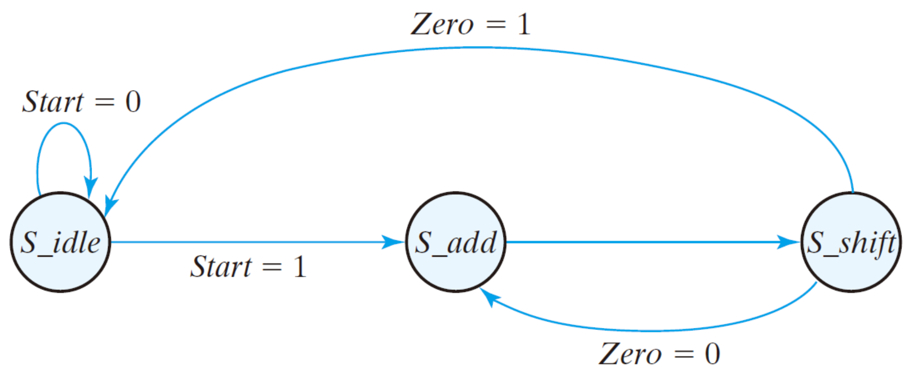
Numerical Example For Binary Multiplier

Multiplicand $B = 10111_2 = 17_H = 23_{10}$ **Multiplier $Q = 10011_2 = 13_H = 19_{10}$**

	C	A	Q	P
Multiplier in Q	0	00000	10011	101
$Q_0 = 1$; add B		$\underline{10111}$		
First partial product	0	10111		100
Shift right CAQ	0	01011	11001	
$Q_0 = 1$; add B		$\underline{10111}$		
Second partial product	1	00010		011
Shift right CAQ	0	10001	01100	
$Q_0 = 0$; shift right CAQ	0	01000	10110	010
$Q_0 = 0$; shift right CAQ	0	00100	01011	001
$Q_0 = 1$; add B		$\underline{10111}$		
Fifth partial product	0	11011		
Shift right CAQ	0	01101	10101	000
Final product in $AQ = 0110110101_2 = 1b5_H$				

Multiplier Control Logic

- The controller specification of the multiplier:



State Transition		Register Operations
From	To	
S_idle		Initial state
S_idle	*S_add*	$A <= 0, C <= 0, P <= dp\_width$
S_add	*S_shift*	$P <= P - 1$ if ($Q[0]$) then ($A <= A + B, C <= C_{out}$)
S_shift		shift right $\{CAQ\}, C <= 0$

State Assignment and State Table

- Possible state assignments for the multiplier:

State Assignment for Control

State	Binary	Gray Code	One-Hot
S_idle	00	00	001
S_add	01	01	010
S_shift	10	11	100

- State table (with binary states):

State equations:

$D_{G_1} = S\_add$

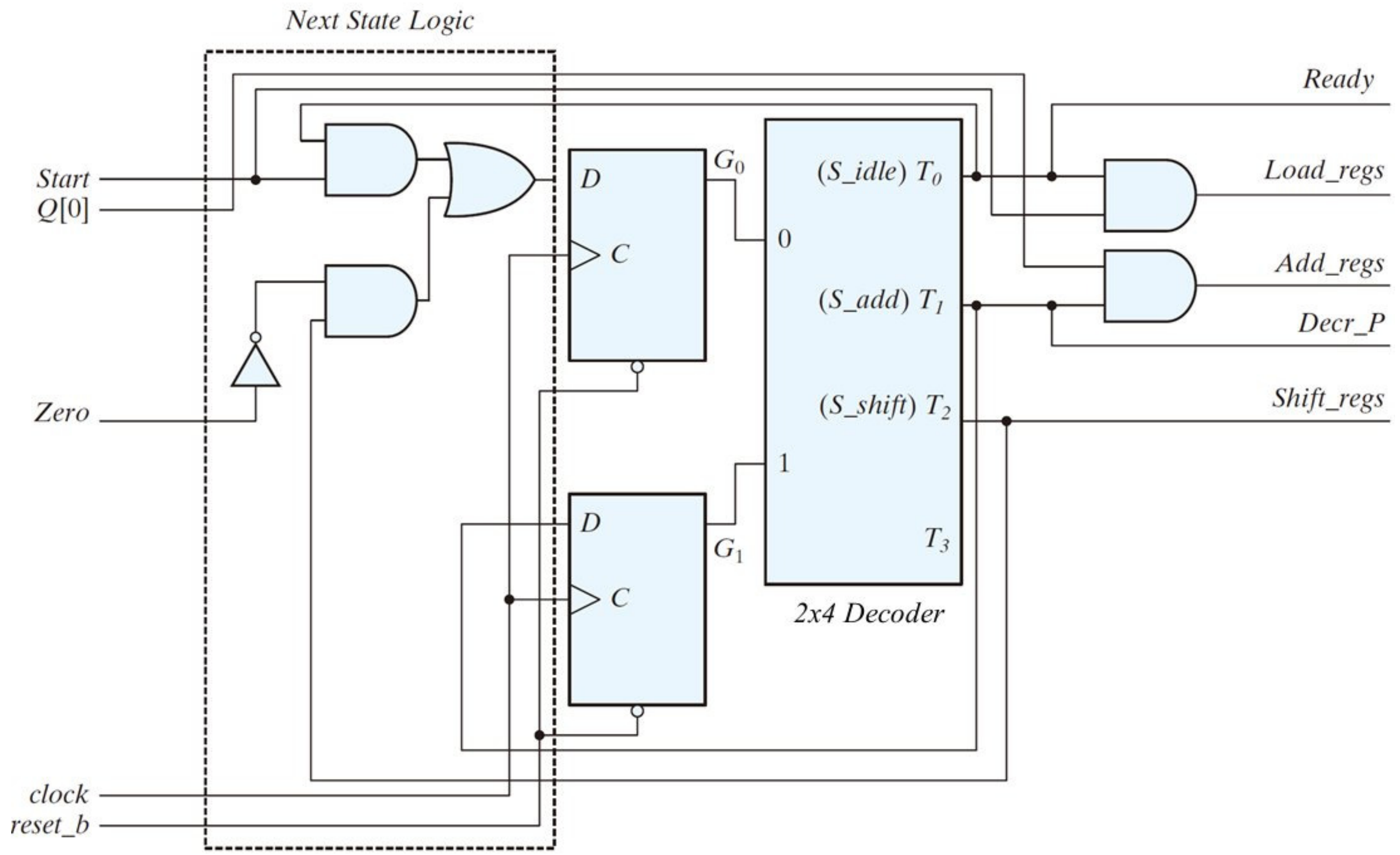
$D_{G_0} = S\_idle \cdot Start' + S\_shift \cdot Zero'$

State Table for Control Circuit

	Present State		Inputs			Next State						
Present-State Symbol	G_1	G_0	*Start*	*Q[0]*	*Zero*	G_1	G_0	*Ready*	*Load_regs*	*Decr_P*	*Add_regs*	*Shift_regs*
S_idle	0	0	0	X	X	0	0	1	0	0	0	0
S_idle	0	0	1	X	X	0	1	1	1	0	0	0
S_add	0	1	X	0	X	1	0	0	0	1	0	0
S_add	0	1	X	1	X	1	0	0	0	1	1	0
S_shift	1	0	X	X	0	0	1	0	0	0	0	1
S_shift	1	0	X	X	1	0	0	0	0	0	0	1

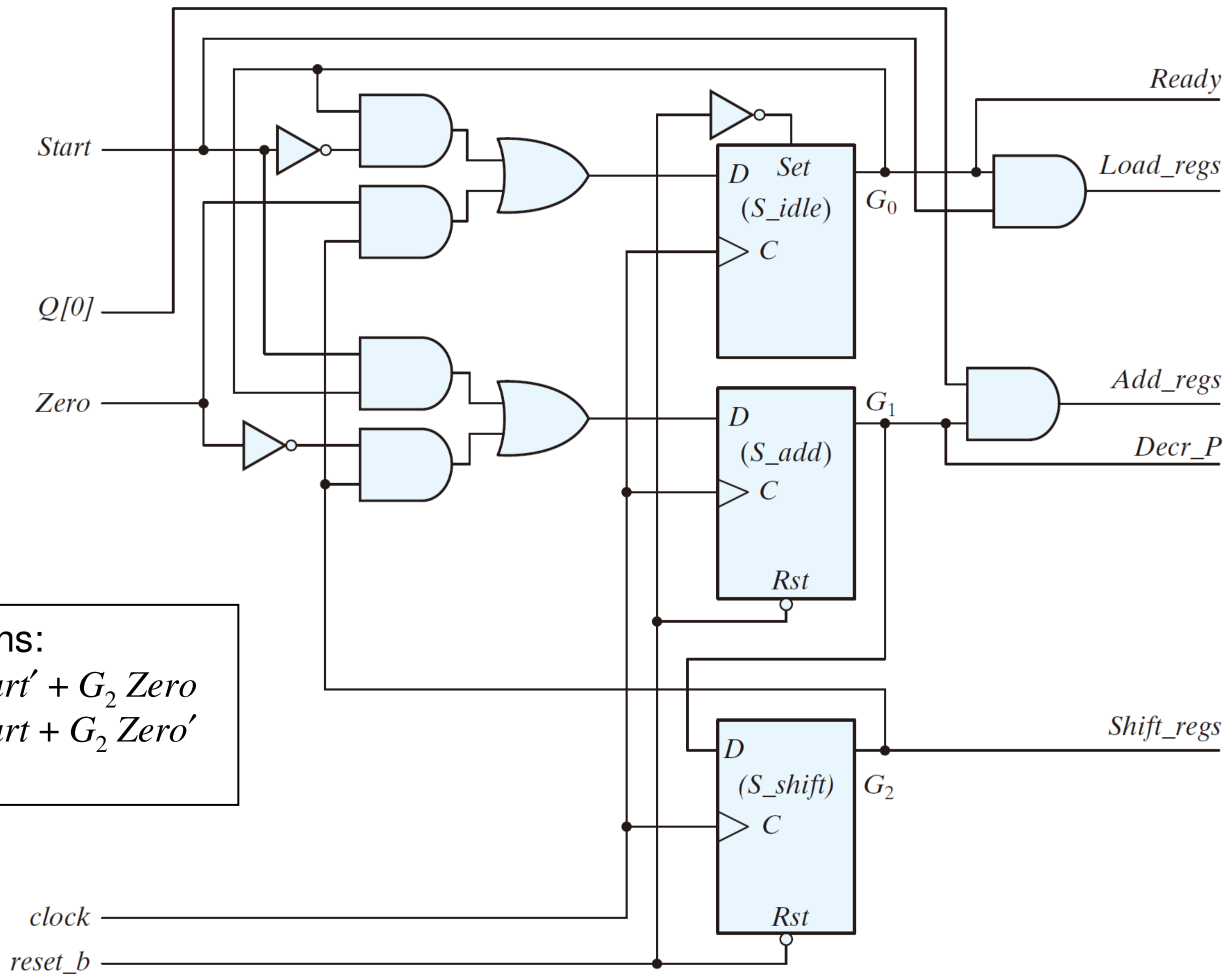
Sequence Register and Decoder

- Logic diagram of binary multiplier controller using a sequence register and decoder



One-Hot Controller State

- If one-hot state ID is used, the controller becomes:

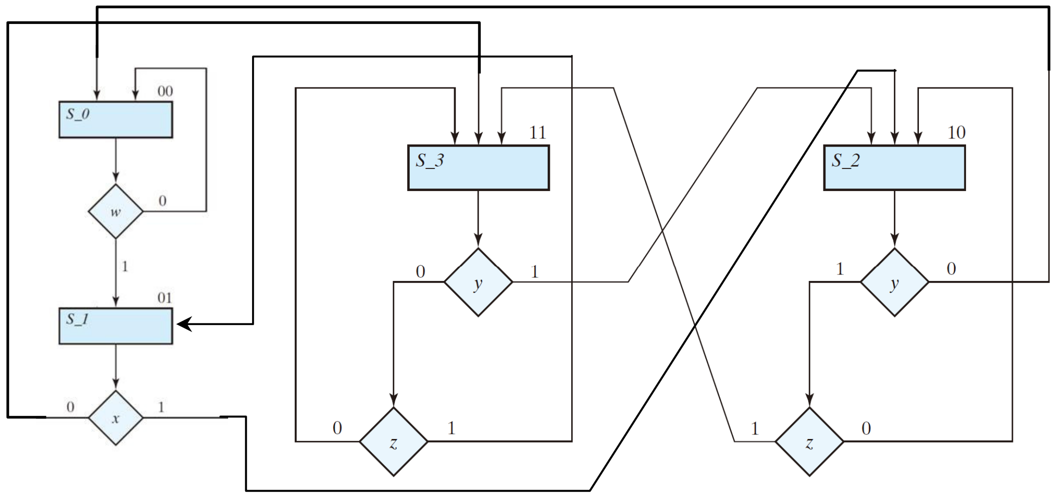


State equations:

$$D_{G_0} = G_0\ Start' + G_2\ Zero$$
$$D_{G_1} = G_0\ Start + G_2\ Zero'$$
$$D_{G_2} = G_1$$

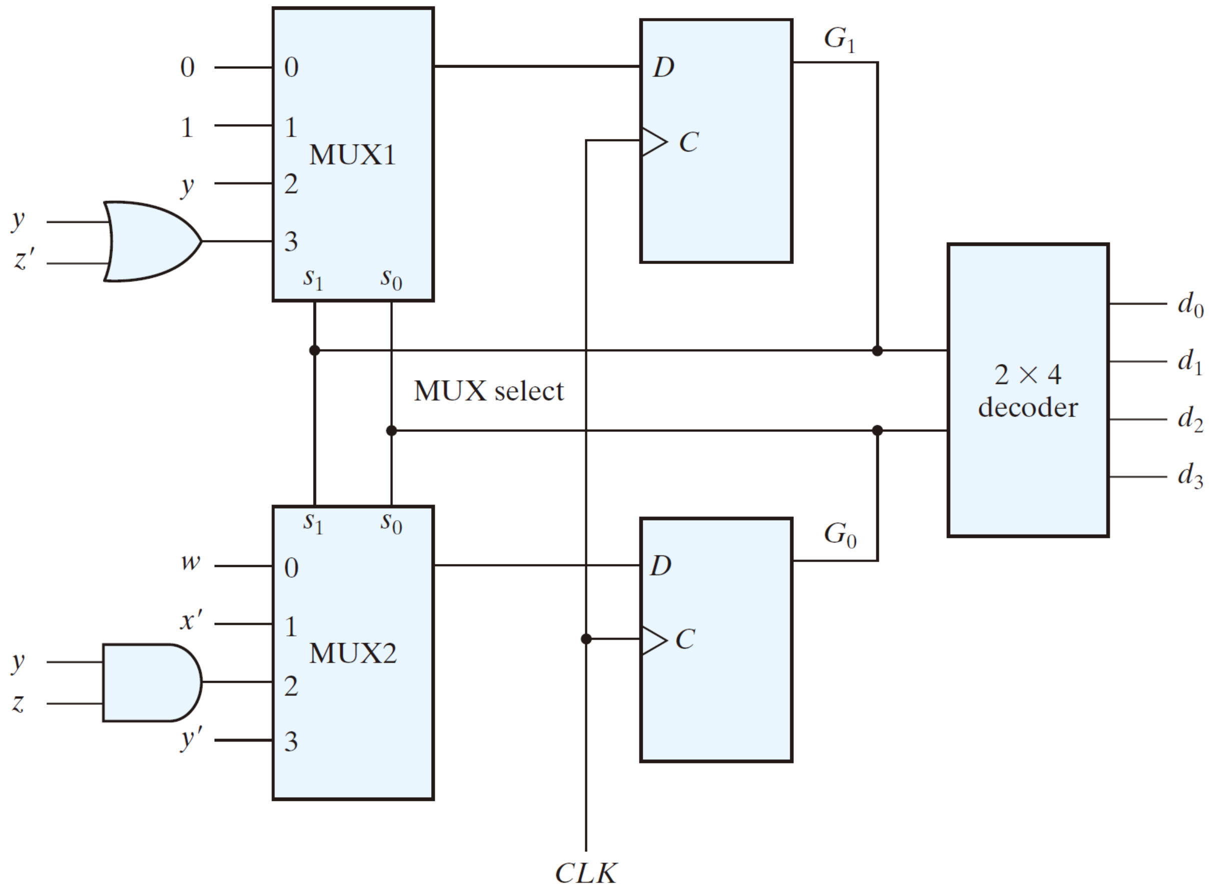
Design with Multiplexers

- ❑ Replacing the gates with multiplexers results in a regular pattern of three levels of components.
  - Level 1:multiplexers that determine the next state.
  - Level 2: registers that hold the present binary state.
  - Level 3: a decoder that asserts a unique output line for each control state.
- ❑ Design example: 4 states, 4 control inputs

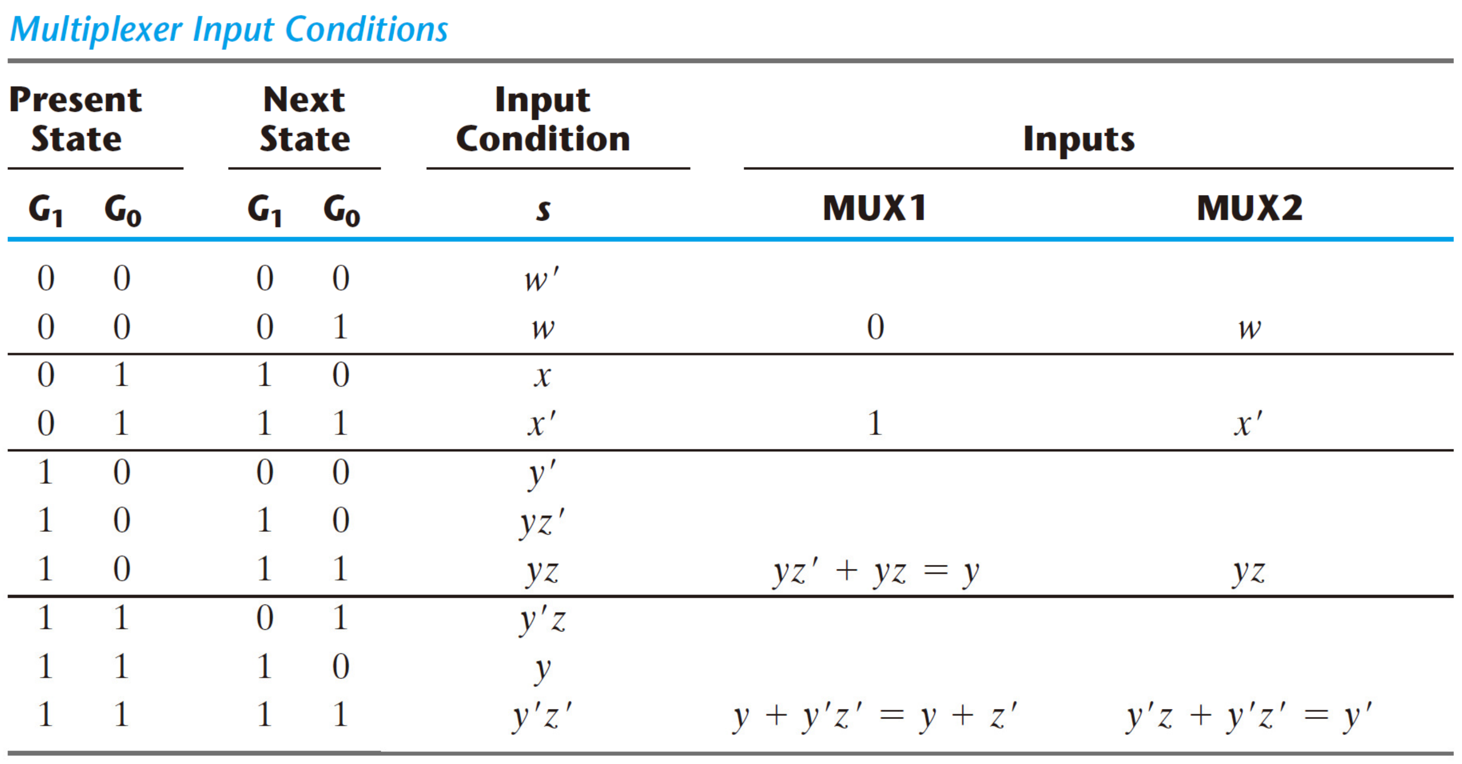


Control Implementation with MUX

- The multiplexors implement the state equations:
  - Current state selects the equations for next state.



State Table and MUX Inputs

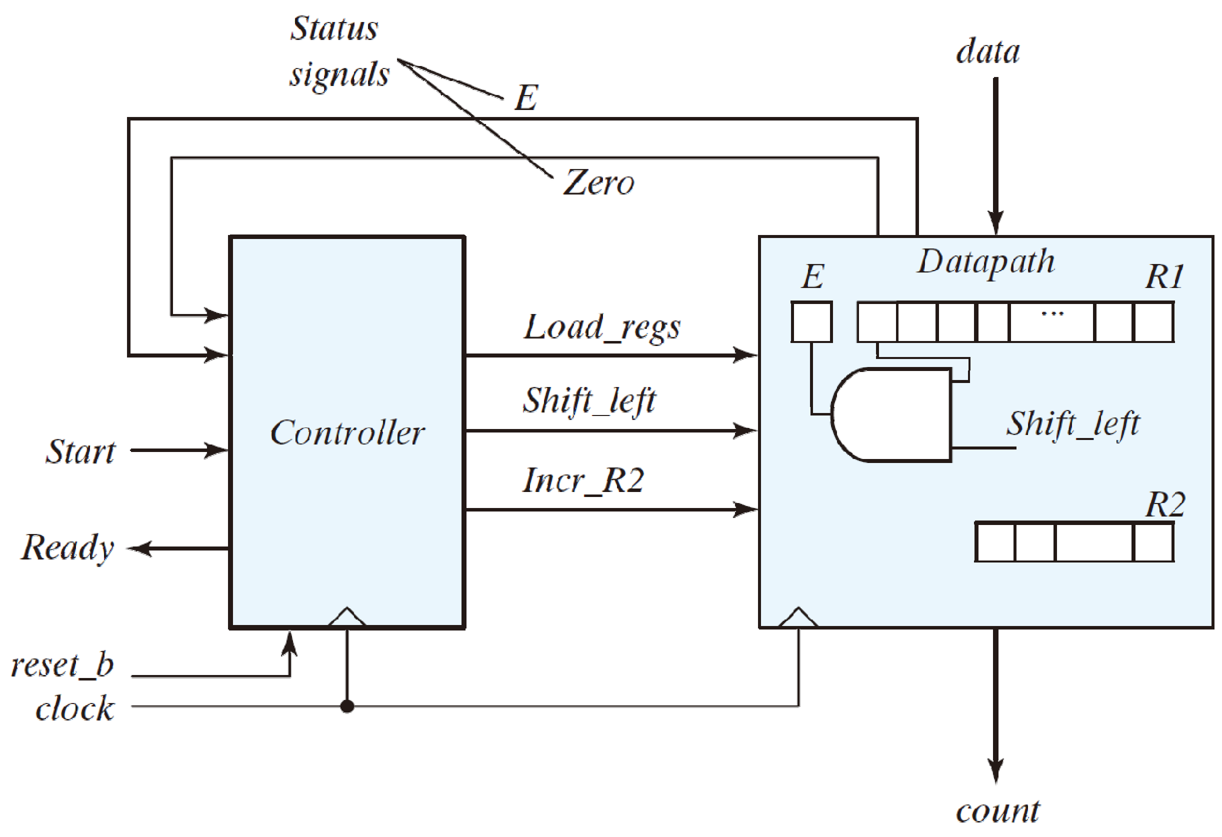


Multiplexer Input Conditions

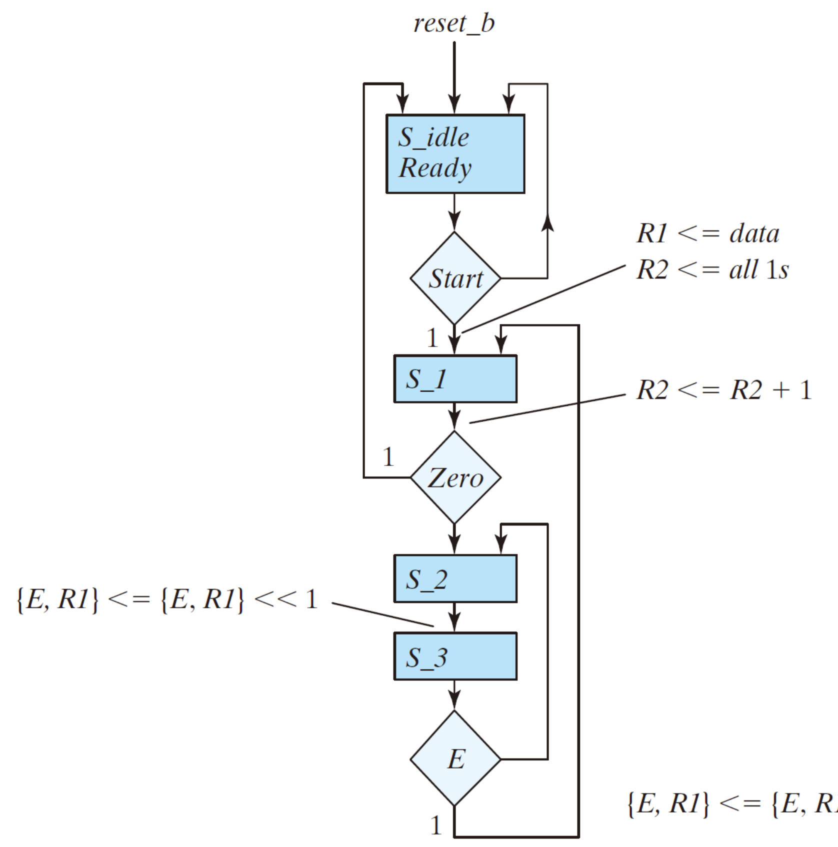
Present State		**Next State**		**Input Condition**	**Inputs**	
G_1	G_0	G_1	G_0	s	**MUX1**	**MUX2**
0	0	0	0	w'		
0	0	0	1	w	0	w
0	1	1	0	x		
0	1	1	1	x'	1	x'
1	0	0	0	y'		
1	0	1	0	yz'		
1	0	1	1	yz	$yz' + yz = y$	yz
1	1	0	1	$y'z$		
1	1	1	0	y		
1	1	1	1	$y'z'$	$y + y'z' = y + z'$	$y'z + y'z' = y'$

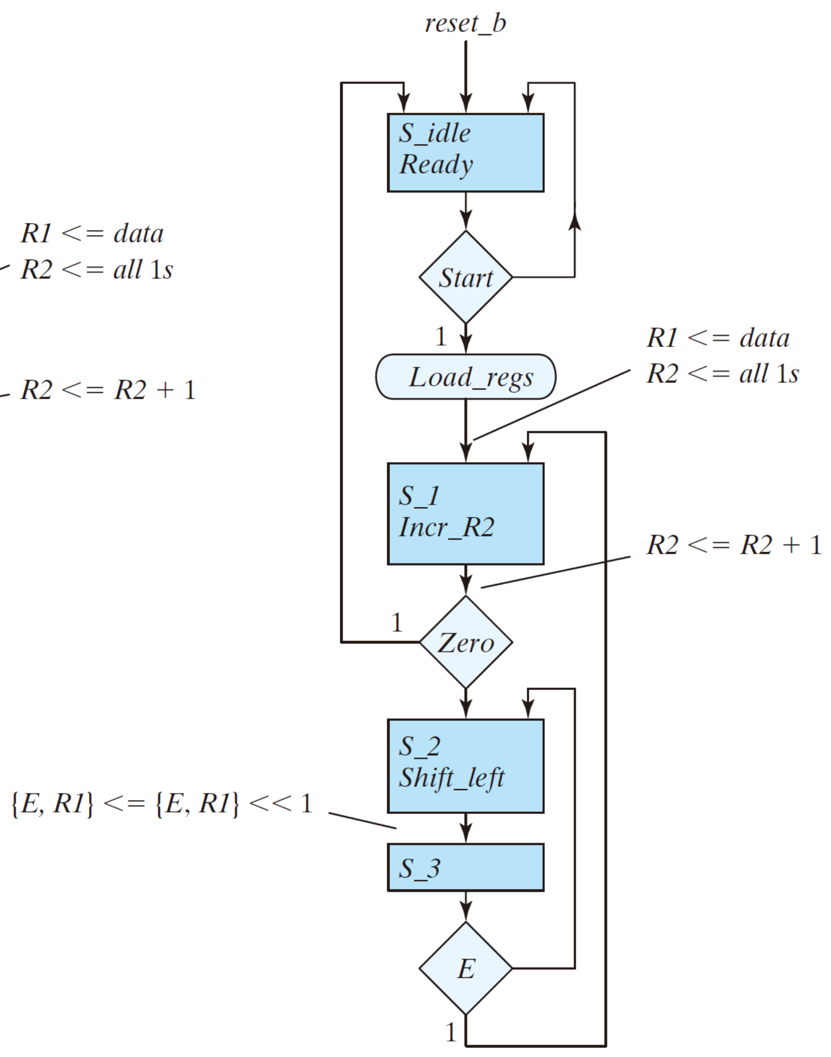
Design Example: 1’s Counter

- A system that counts the number of 1’s in a word:



ASMD Chart of the 1's Counter



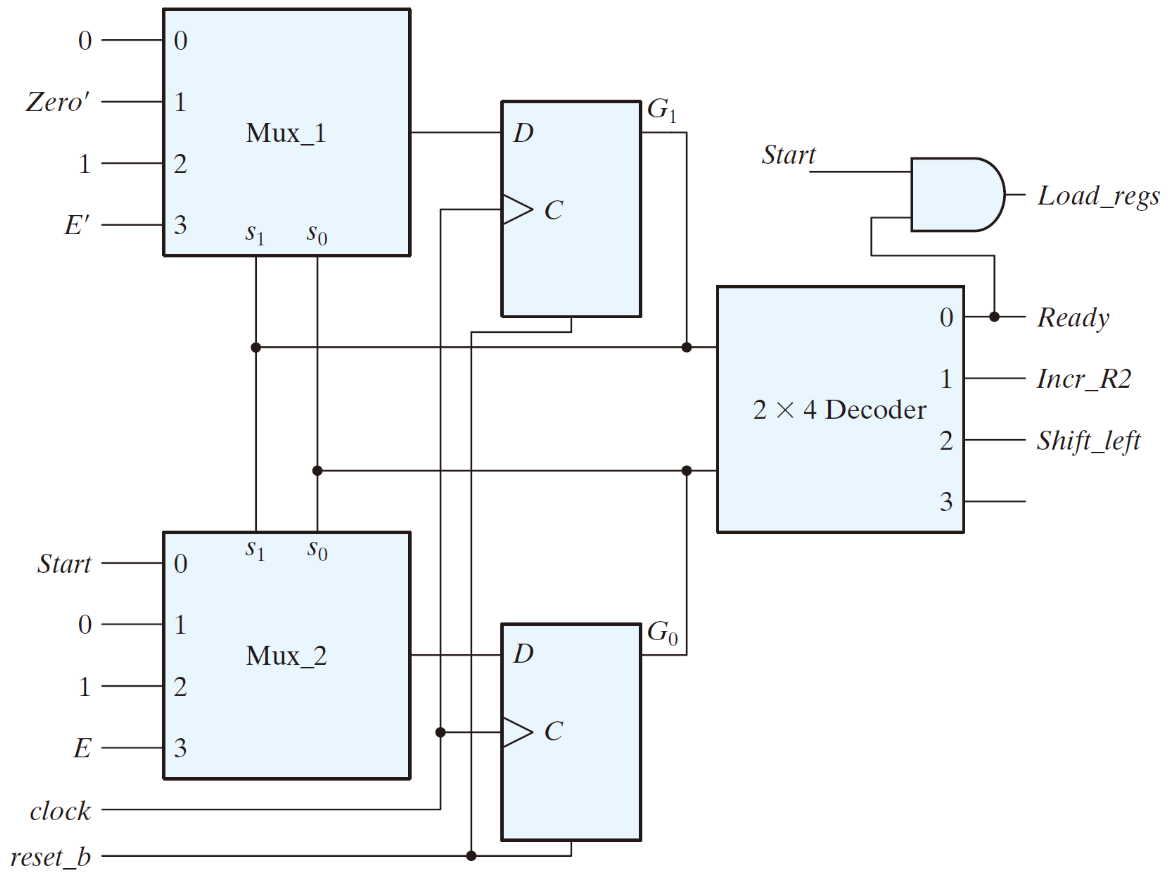


State Table of the 1's Counter

Multiplexer Input Conditions for Design Example

Present State		Next State		Input Conditions	Multiplexer Inputs	
G_1	G_0	G_1	G_0		MUX1	MUX2
0	0	0	0	*Start'*		
0	0	0	1	*Start*	0	*Start*
0	1	0	0	*Zero*		
0	1	1	0	*Zero'*	*Zero'*	0
1	0	1	1	*None*	1	1
1	1	1	0	*E'*		
1	1	0	1	*E*	*E'*	*E*

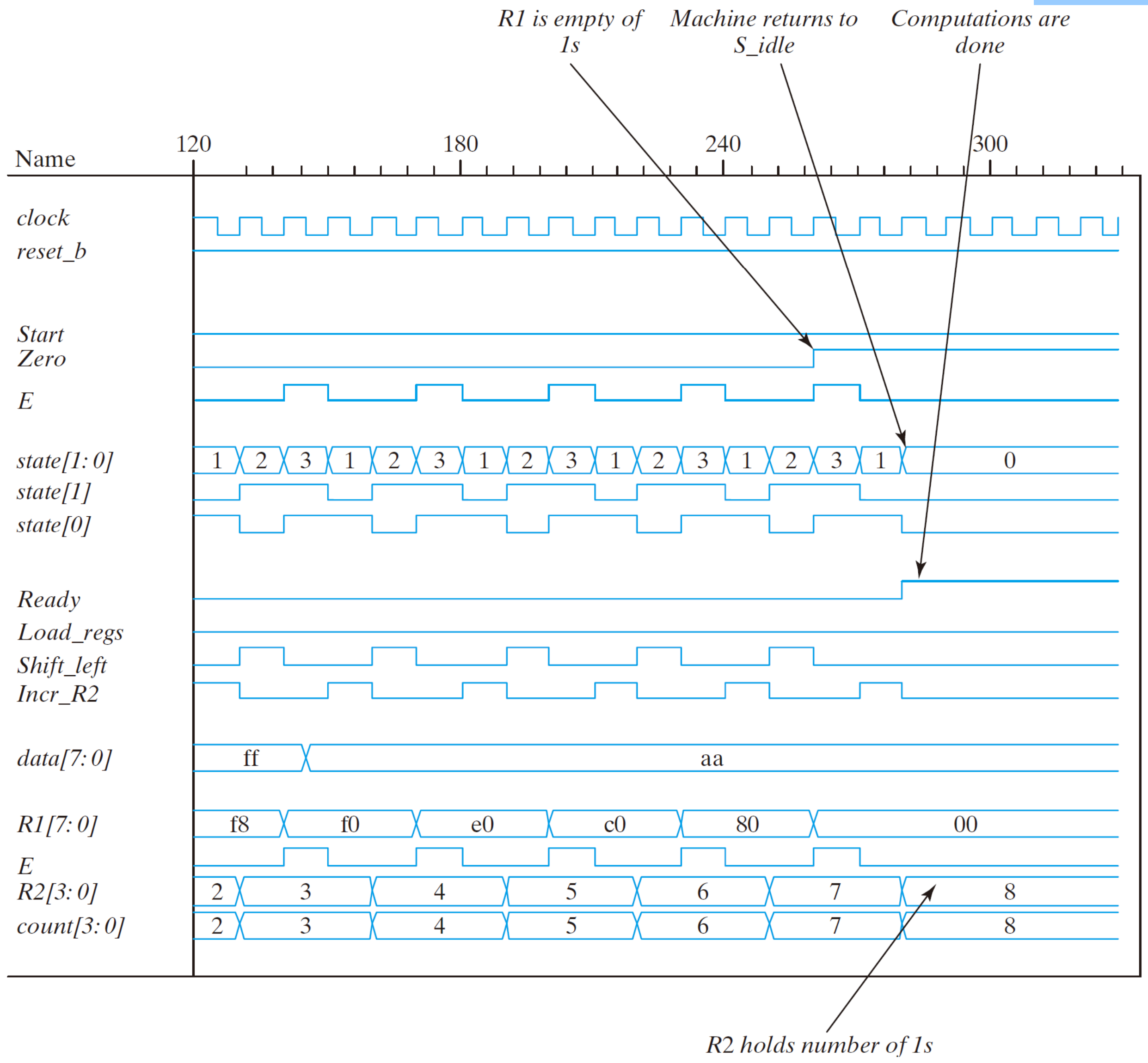
Controller of the 1's Counter Circuit



Race-Free Design

- Theoretically, any simulations of the behavior model of a circuit should match
  - In reality, software-based simulation may produce mismatch
- Potential cause of problem:
  - A physical feedback path exists between a datapath unit and control unit whose inputs include status signals fed back from the datapath unit.
  - Blocked propagation assignments execute immediately, and behavioral models simulate with 0 propagation delays, effectively creating immediate changes in the outputs of the combinational logic when its inputs change.
  - The order in which a simulator executes multiple blocked assignments to the same variable at a given time step of the simulation is indeterminate.

Waveform Simulation of 1's Counter



Latch-Free Design

- A feedback-free continuous assignment will synthesize to combinational logic, and the input-output relationship of the logic is automatically sensitive to all of the inputs of the circuit.
- If a level-sensitive cyclic behavior is used to describe combinational logic, it is essential that the sensitivity list include every variable; if the list is incomplete, the logic described by the behavior will be synthesized with latches at the output logic.