

# DESIGN OF DIGITAL CIRCUITS AND SYSTEMS

## ASM with Datapath I

**Instructor:** Justin Hsia

**Teaching Assistants:**

Colton Carroll

Grace Zhou

Hemil Patel

Quinlyn Donohue

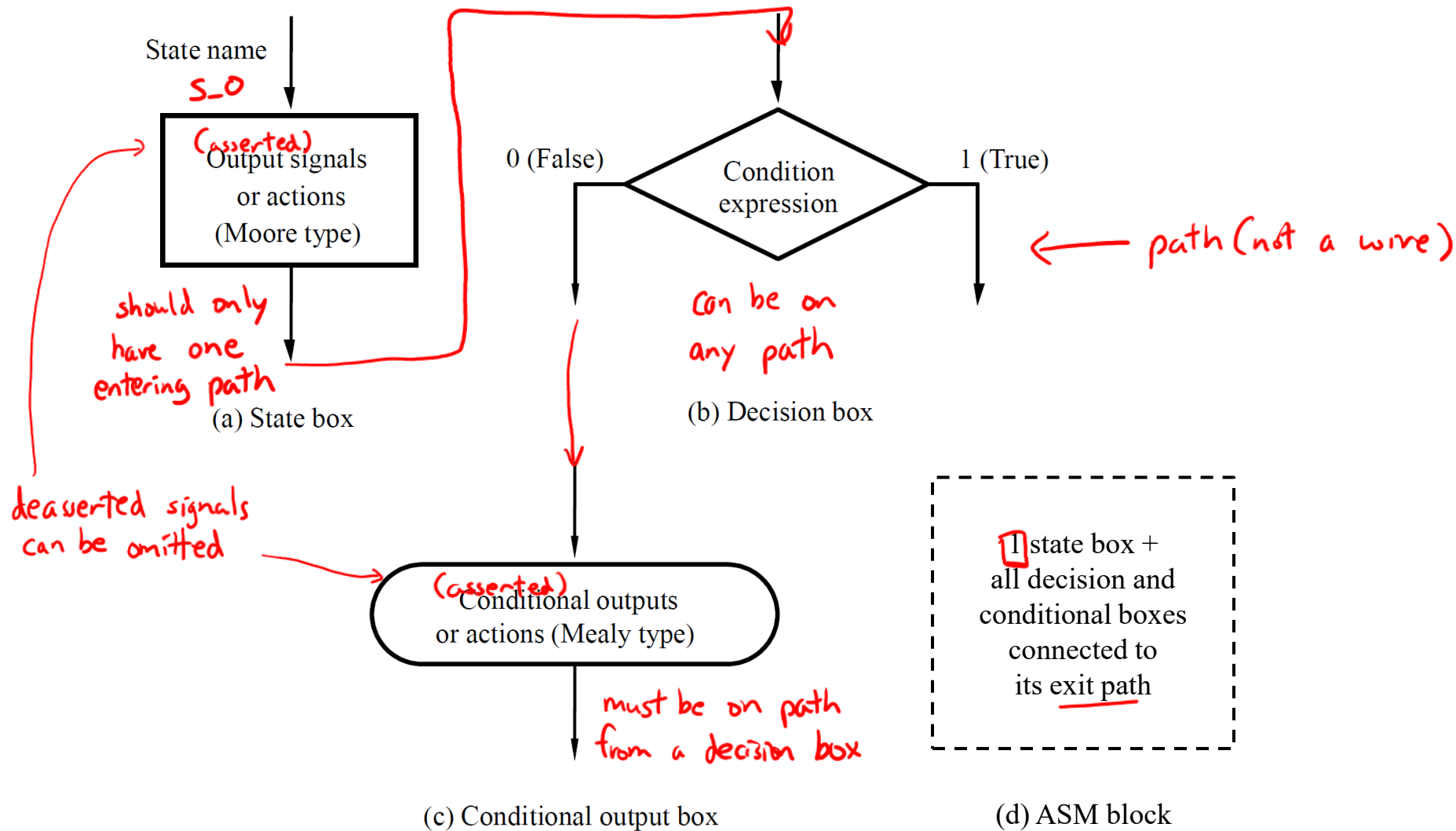
Rasya Fawwaz

Rose Maresh

# Relevant Course Information

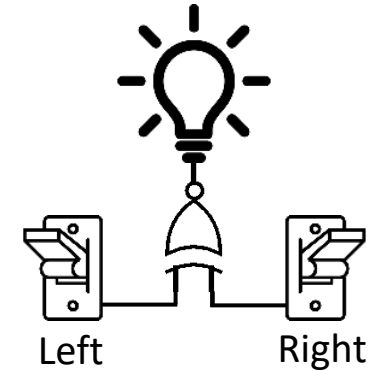
- ❖ Homework 2 late deadline tonight (4/16)
- ❖ Homework 3 due next Friday (4/24)
  - FIFO buffers & ASM charts
- ❖ Lab 2 reports due Friday (4/17), demos 4/20-24
  - Same lab demo slots for whole quarter
- ❖ Lab 3 due 5/1
  - Lab 3 + 4 are really ~1.5 weeks long, so don't wait!
- ❖ Quiz 2 next Thursday (4/23)
  - Memory (ROM, RAM, reg files)

# Review: ASM Chart



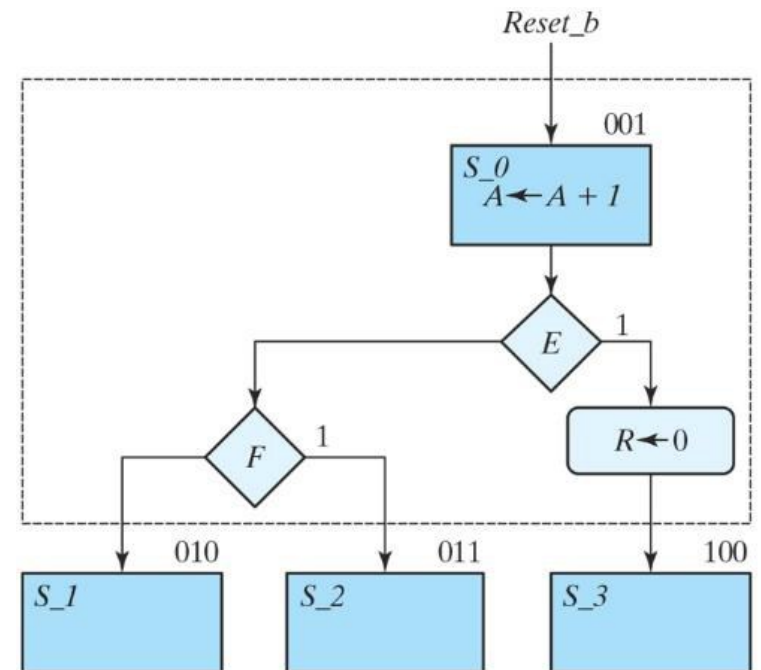
# Review Question: 3-way Switch

- ❖ Create an ASM chart for a 3-way switch system using *Mealy*-type output
  - LTog and RTog pulse 1 when the switch is flipped/toggled
  - Output called light



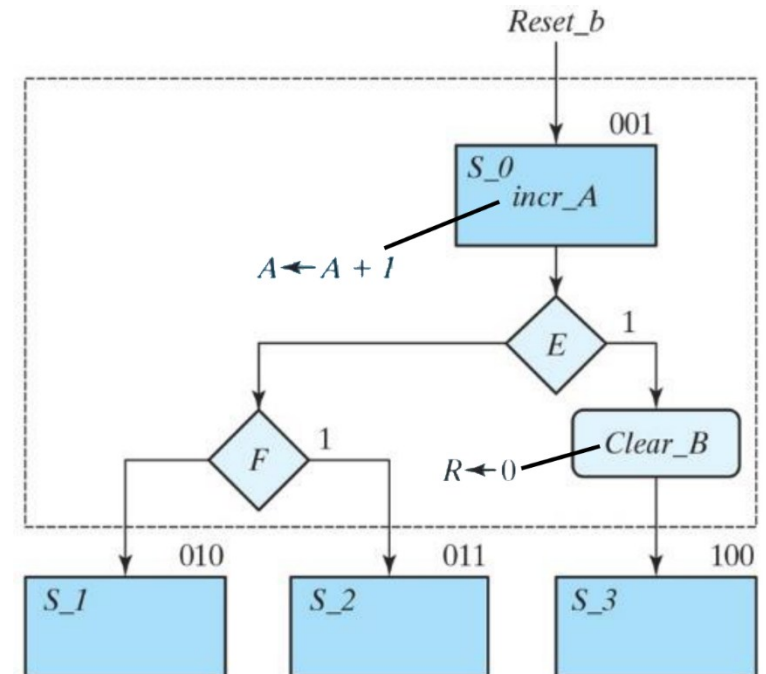
# ASMD Charts (1/2)

- ❖ An **A**lgorithmic **S**tate **M**achine with a **D**atapath chart is created by adding RTL operations to an ASM chart
  - Timing of operations can be confusing – *NOT* a flowchart
- ❖ School of Thought #1:
  - RTL operations are triggered by control signals, so they can appear anywhere an output signal can:



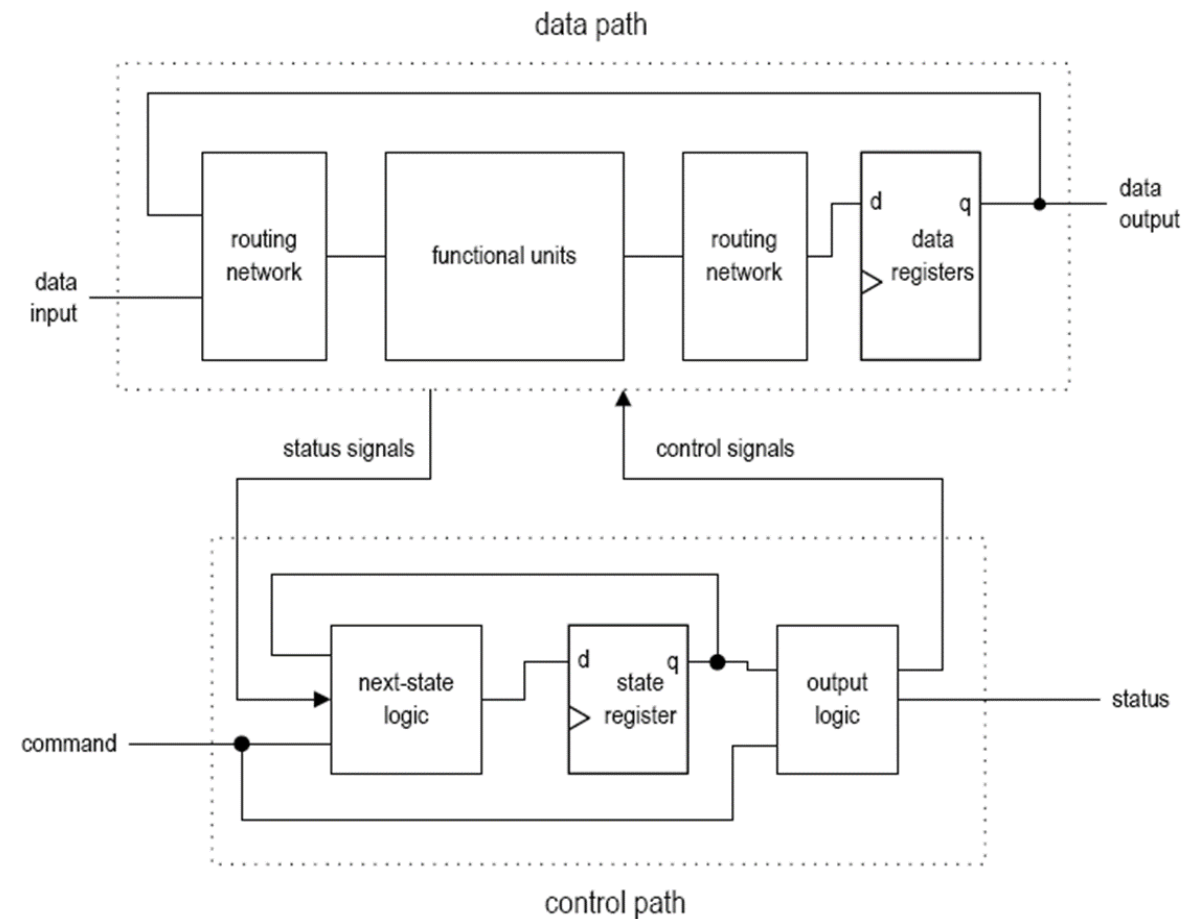
# ASMD Charts (2/2)

- ❖ An **A**lgorithmic **S**tate **M**achine with a **D**atapath chart is created by adding RTL operations to an ASM chart
  - Timing of operations can be confusing – *NOT* a flowchart
- ❖ School of Thought #2:
  - It's clearer to separate control signals (Control) from RTL operations (Datapath)
- ❖ There isn't a set standard
  - You may see both or variants
  - ***We use School of Thought #2***



# ASMD Hardware (1/2)

- ❖ State transitions and RTL operations are both controlled by the clock
  - It's often helpful to remember the underlying hardware – registers!

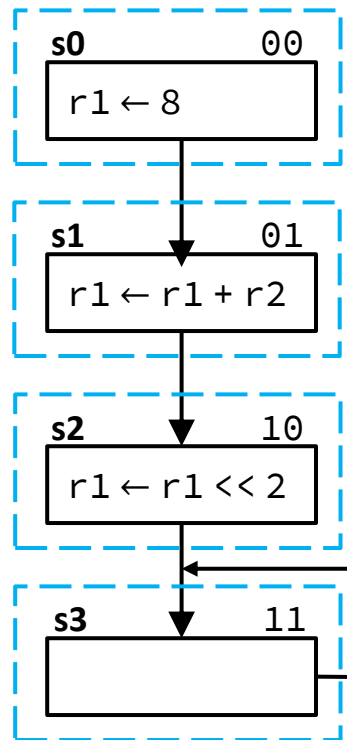


## ASMD Hardware (2/2)

- ❖ State transitions and RTL operations are both controlled by the clock
  - It's often helpful to remember the underlying hardware – registers!
- ❖ The behavior of both state and data registers depend on the current control state
  - Can conceptually think of as a MUX to the registers' inputs that uses the current state as its selector bits

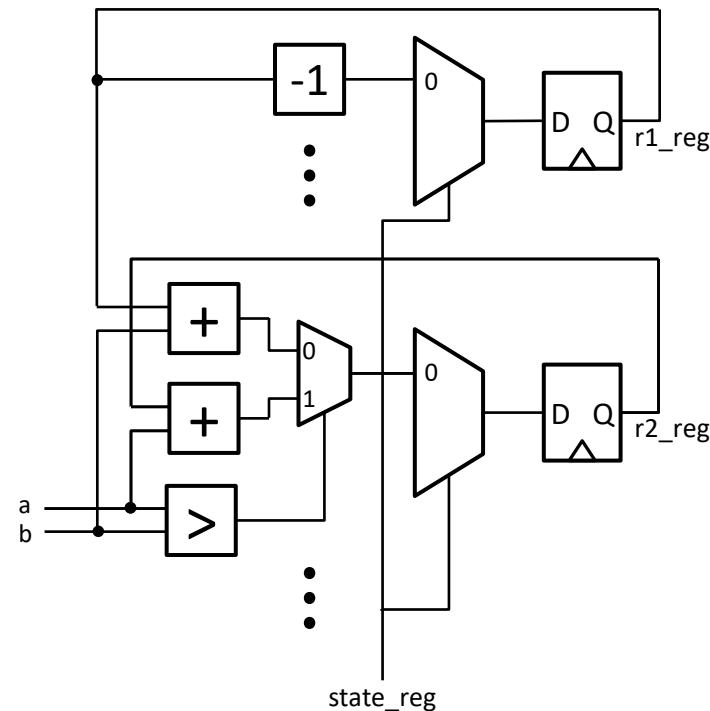
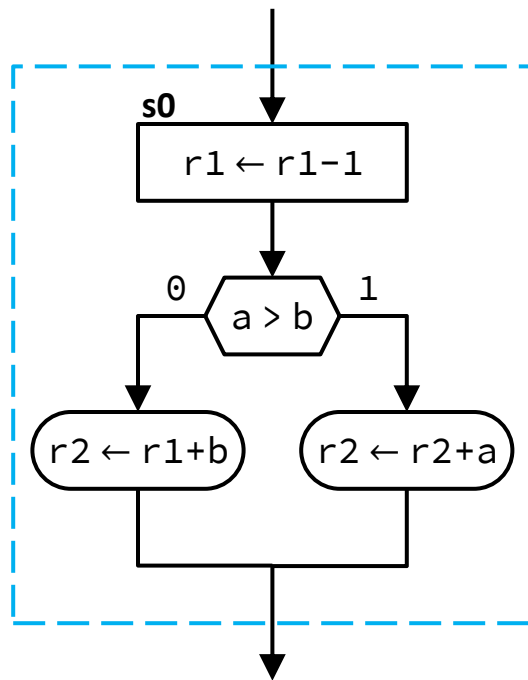
# Hardware Example (1/2)

- ❖ State transitions and RTL operations are both controlled by the clock
  - It's often helpful to remember the underlying hardware – registers!



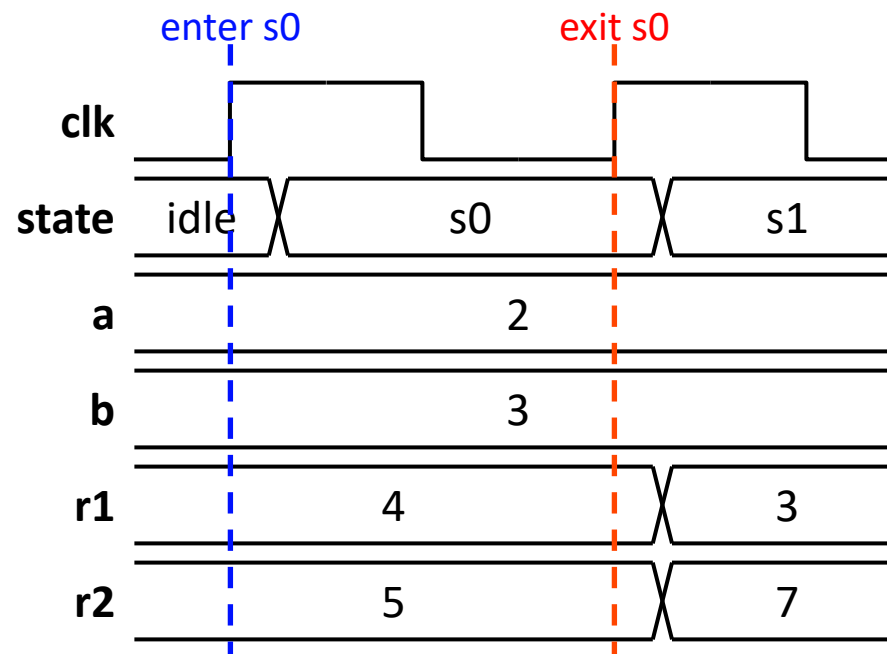
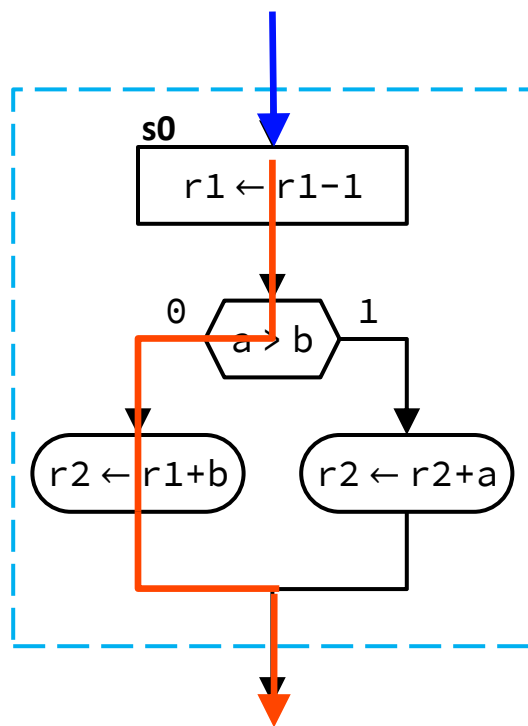
# ASMD Timing (1/2)

- ❖ Everything (registers!) within an ASM block occurs simultaneously at the next clock trigger
  - Differs from a flowchart – changes occur at state exit rather than *entrance*



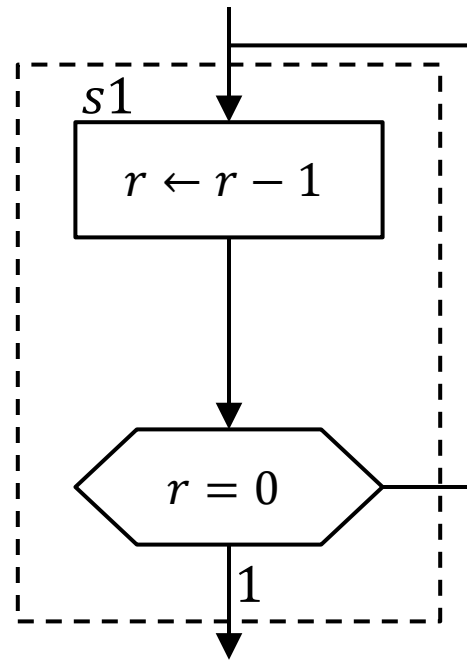
## ASMD Timing (2/2)

- ❖ Everything (registers!) within an ASM block occurs simultaneously at the next clock trigger
  - Differs from a flowchart – changes occur at state exit rather than *entrance*



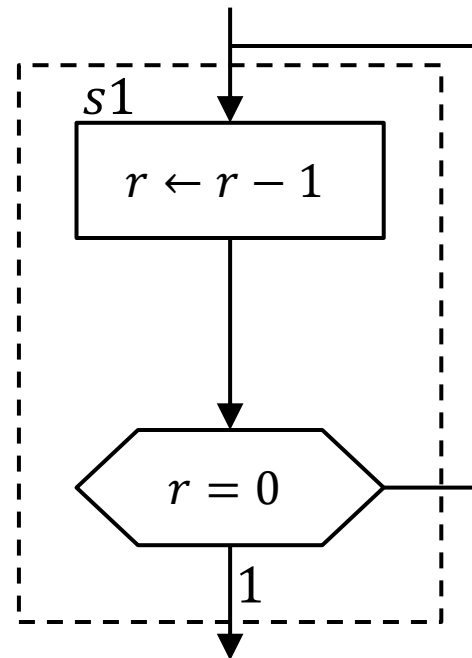
# ASMD Timing Question

- ❖ What value will be stored in  $r$  when we transition from state  $s1$  to the next state? -1, 0, 1

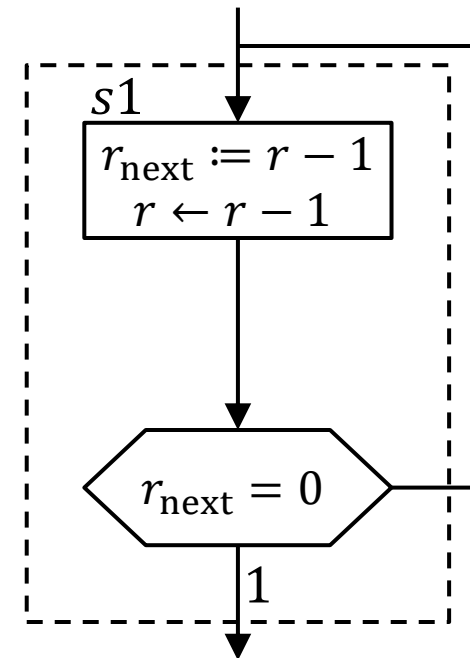


# ASMD Timing Tip

- ❖ When a registered output (*e.g.*,  $r$ ) is used in a decision box, its effect may appear to be delayed by one clock
  - Can define a next-state value (*e.g.*,  $r_{\text{next}}$ ) to use instead



(a) Use old value of  $r$



(b) Use new value of  $r$

SHORT TECH

BREAK

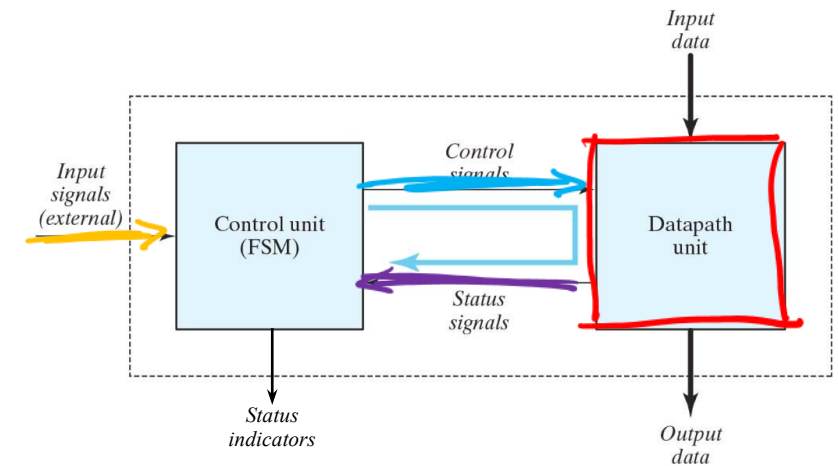
# ASMD Design Procedure

- ❖ From problem description or algorithm pseudocode:
  - 1) Identify necessary datapath components and operations**
  - 2) Identify states and signals that cause state transitions** (external inputs and status signals), based on the necessary sequencing of operations
  - 3) Name the control signals** that are generated by the controller that cause the indicated operations in the datapath unit
  - 4) Form an ASM chart for your controller**, using states, decision boxes, and signals determined above
  - 5) Add the datapath RTL operations** associated with each control signal

# Design Example #1

## ❖ System specification:

- Flip-flops  $E$  and  $F$
- 4-bit binary up-counter  $A = 0bA_3A_2A_1A_0$
- Active-low reset signal  $reset\_b$  puts us in state  $S\_idle$ , where we remain while signal  $Start = 0$
- $Start = 1$  initiates the system's operation by clearing  $A$  and  $F$ . At each subsequent clock pulse, the counter is incremented by 1 until the operations stop.
- Bits  $A_2$  and  $A_3$  determine the sequence of operations:
  - If  $A_2 = 0$ , set  $E$  to 0 and the count continues
  - If  $A_2 = 1$ , set  $E$  to 1; additionally, if  $A_3 = 0$ , the count continues, otherwise, wait one clock pulse to set  $F$  to 1 and stop counting (*i.e.*, back to  $S\_idle$ )



# Design Example #1: ASM $\rightarrow$ ASMD Chart

- ❖ Synchronous or asynchronous reset?

# Design Example #1: Timing

❖ Sequence of operations:

Counter				Flip-Flops		Conditions	State	
$A_3$	$A_2$	$A_1$	$A_0$	$E$	$F$			
X	X	X	X	1	X	<i>Start</i>	<i>S_idle</i>	
0	0	0	0	1	0	$A_2 = 0, A_3 = 0$	<i>S_count</i>	
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	$A_2 = 0, A_3 = 1$		
1	0	0	0	1	0			
1	0	0	1	0	0			
1	0	1	0	0	0			
1	0	1	1	0	0	$A_2 = 1, A_3 = 1$		
1	1	0	0	0	0			
1	1	0	1	1	0			<i>S_F</i>
1	1	0	1	1	1			<i>S_idle</i>

# Design Example #1: Logic

## ❖ Controller:

### ■ State Table:

Present-State Symbol	Present State		Inputs			Next State		Outputs				
	$P_1$	$P_0$	Start	$A_2$	$A_3$	$N_1$	$N_0$	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
$S_{count}$	0	1	X	0	X	0	1	0	1	0	0	1
$S_{count}$	0	1	X	1	0	0	1	1	0	0	0	1
$S_{count}$	0	1	X	1	1	1	1	1	0	0	0	1
$S_F$	1	1	X	X	X	0	0	0	0	1	0	0

### ■ Logic:

$$\begin{aligned}
 N_1 &= & set_F &= \\
 N_0 &= & clr_{A_F} &= \\
 set_E &= & incr_A &= \\
 clr_E &= & &=
 \end{aligned}$$

SHORT TECH

BREAK

# Design Example #1: Controller Outline (1/2)

```
module controller (set_E, clr_E, set_F, clr_A_F,  
                  incr_A, A2, A3, Start, clk,  
                  reset_b);  
  
    // port definitions  
    input  logic Start, clk, reset_b, A2, A3;  
    output logic set_E, clr_E, set_F, clr_A_F, incr_A;  
  
    // define state names and variables  
  
    // controller logic w/synchronous reset
```

# Design Example #1: Controller Outline (2/2)

```
// next state logic
```

```
// output assignments
```

```
endmodule // controller
```

# Design Example #1: Datapath Outline

```
module datapath (A, E, F, clk, set_E, clr_E, set_F, clr_A_F,  
                incr_A);  
  
    // port definitions  
    output logic [3:0] A;  
    output logic E, F;  
    input  logic clk, set_E, clr_E, set_F, clr_A_F, incr_A;  
  
    // datapath logic  
  
endmodule // datapath
```

# Design Example #1: Top-Level Code

```
module top_level (A, E, F, clk, Start, reset_b);  
  
    // port definitions  
    output logic [3:0] A;  
    output logic E, F;  
    input  logic clk, Start, reset_b;  
  
    // internal signals  
    logic set_E, clr_E, set_F, clr_A_F, incr_A;  
  
    // instantiate controller and datapath  
    controller c_unit (.set_E, .clr_E, .set_F,  
                      .clr_A_F, .incr_A, .A2(A[2]),  
                      .A3(A[3]), .Start, .clk,  
                      .reset_b);  
  
    datapath d_unit (.*);  
  
endmodule // top_level
```