DESIGN OF DIGITAL CIRCUITS AND SYSTEMS

ASM with Datapath I

Instructor: Justin Hsia

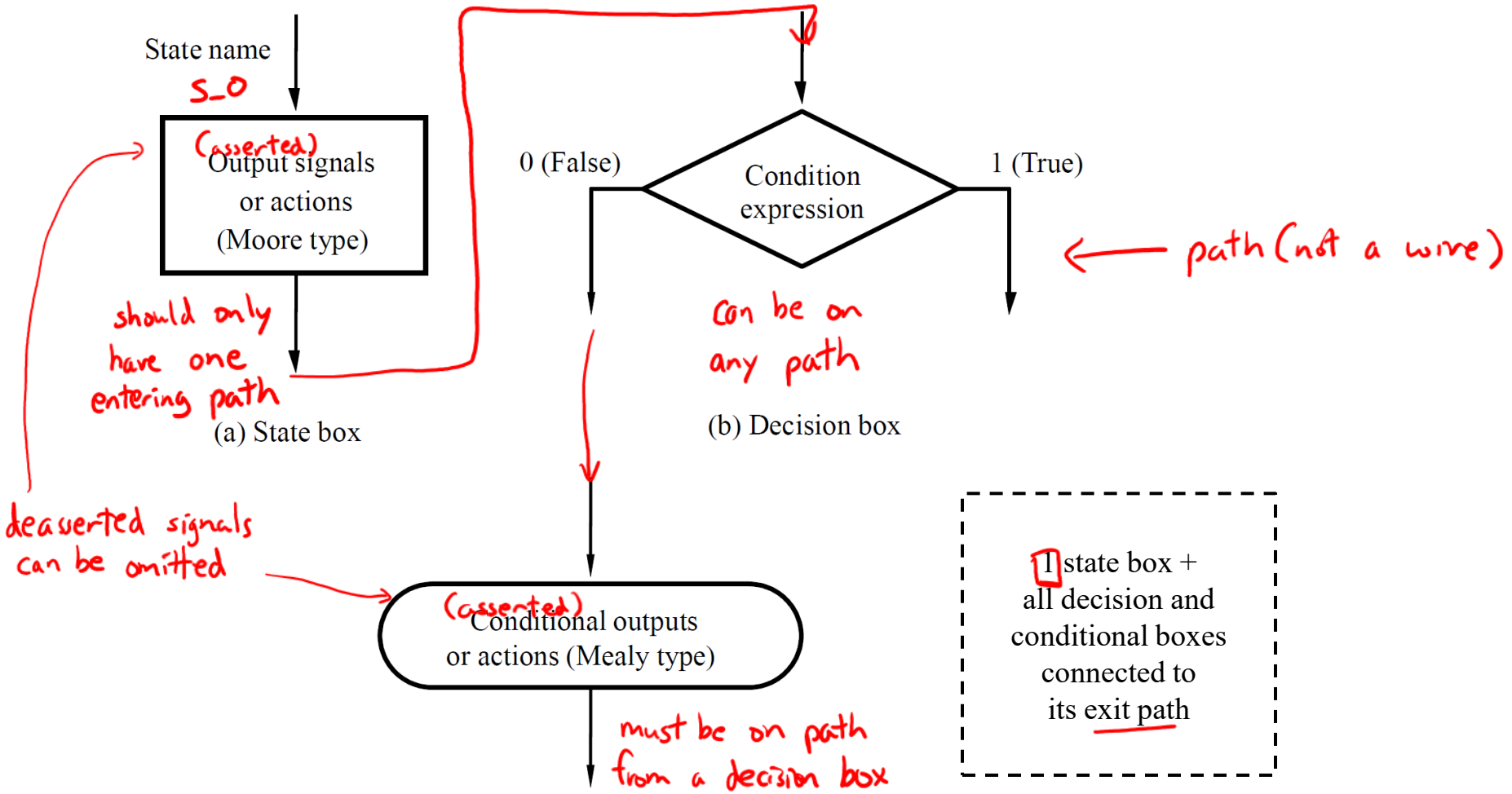
Teaching Assistants:

Colton Carroll
Hemil Patel
Rasya Fawwaz
Grace Zhou
Quinlyn Donohue
Rose Maresh

Relevant Course Information

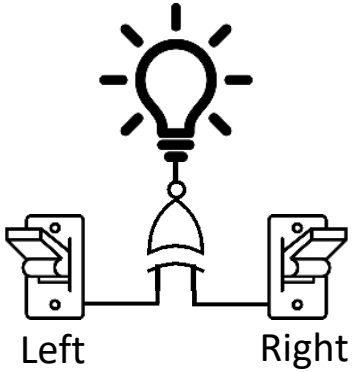
- Homework 2 late deadline tonight (4/16)
- Homework 3 due next Friday (4/24)
  - FIFO buffers & ASM charts
- Lab 2 reports due Friday (4/17), demos 4/20-24
  - Same lab demo slots for whole quarter
- Lab 3 due 5/1
  - Lab 3 + 4 are really ~1.5 weeks long, so don't wait!
- Quiz 2 next Thursday (4/23)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

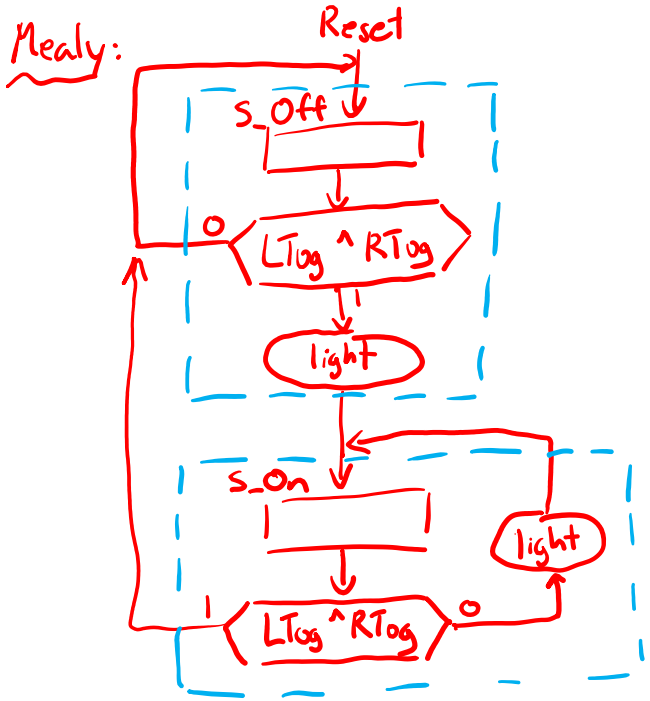
Review: ASM Chart

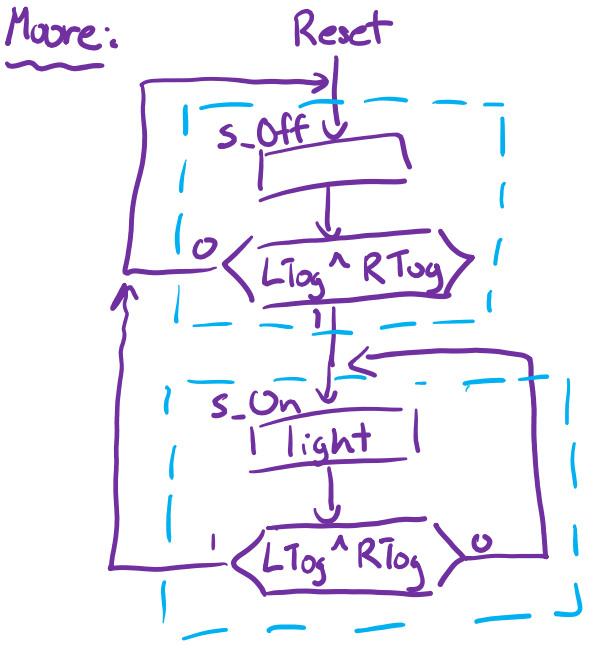


Review Question: 3-way Switch

- 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`



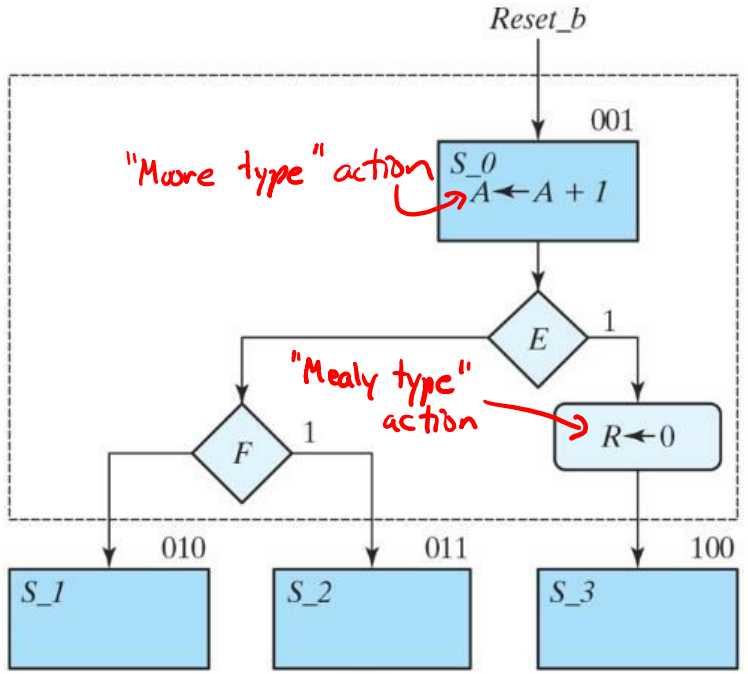




ASMD Charts (1/2)

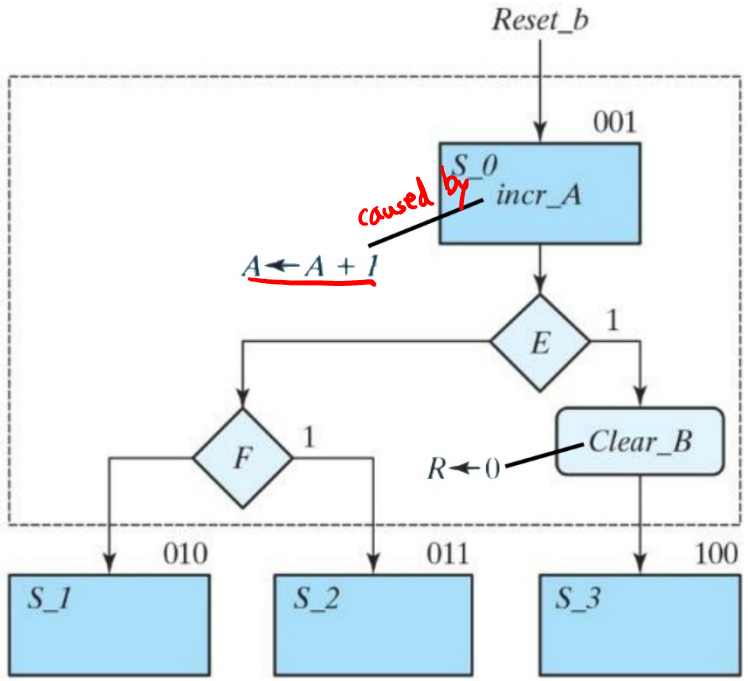
- An **A**lgorithmic **S**tate **M**achine with a **D**atapath chart is created by adding RTL operations to an ASM chart
  - Timing of operations can be confusing – *NOT* a flowchart

- School of Thought #1:
  - RTL operations are triggered by control signals, so they can appear anywhere an output signal can:



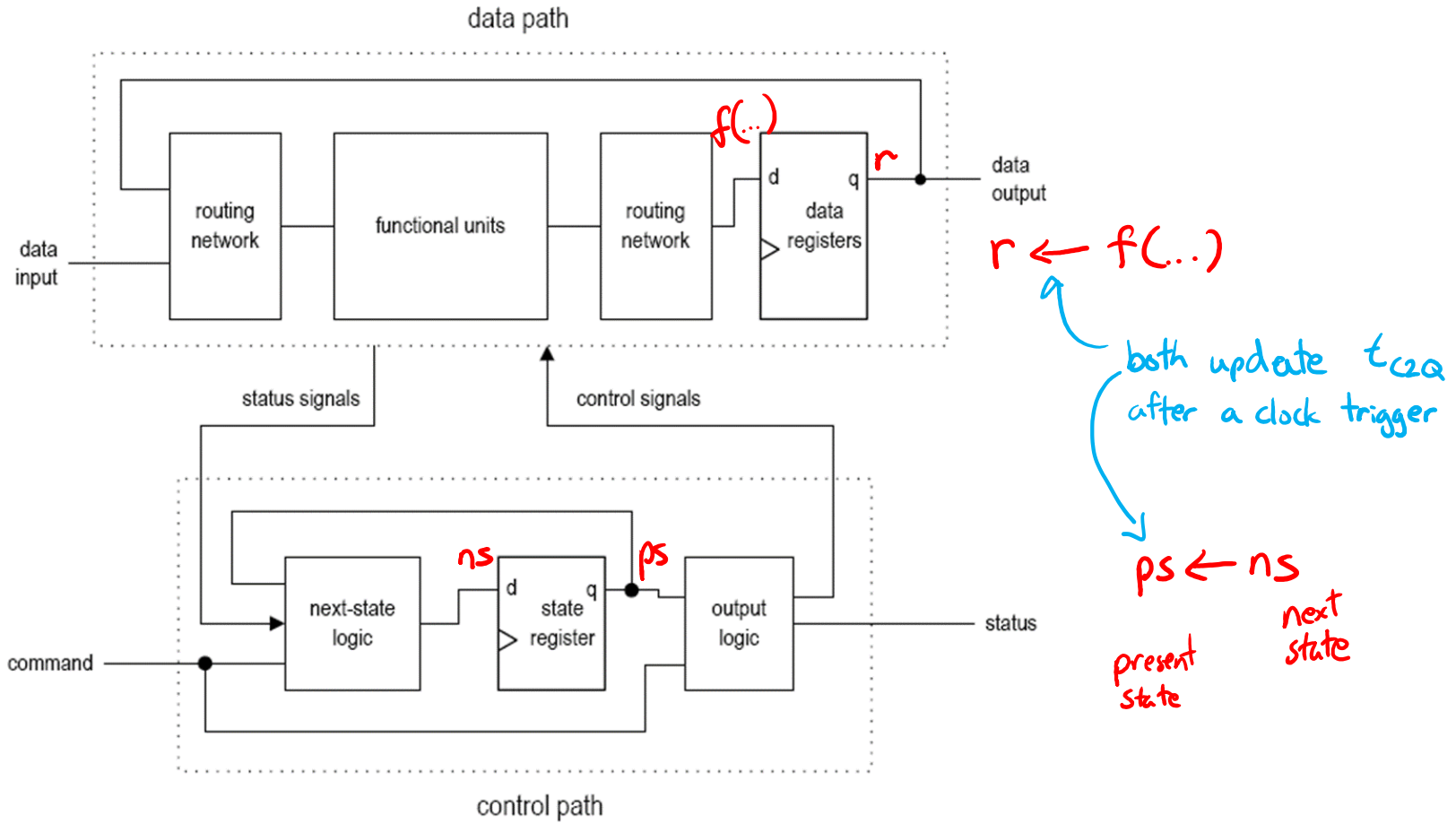
ASMD Charts (2/2)

- 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)

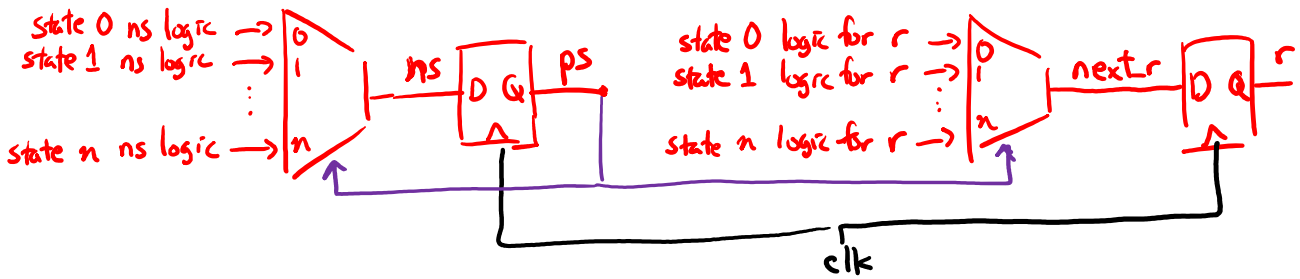
- State transitions (control) and RTL operations (datapath) are both controlled by the clock
  - It's often helpful to remember the underlying hardware – registers!



ASMD Hardware (2/2)

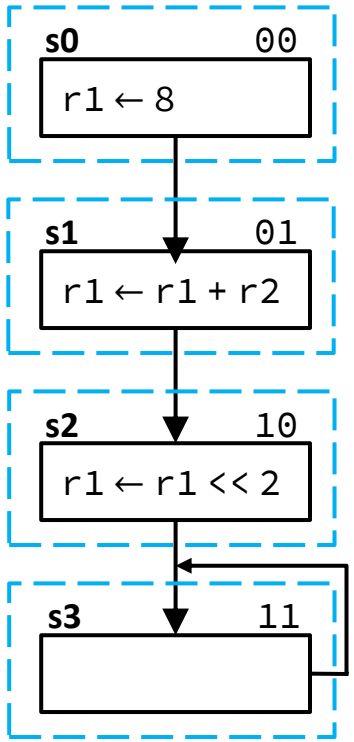
- State transitions and RTL operations are both controlled by the clock
  - It's often helpful to remember the underlying hardware – registers!

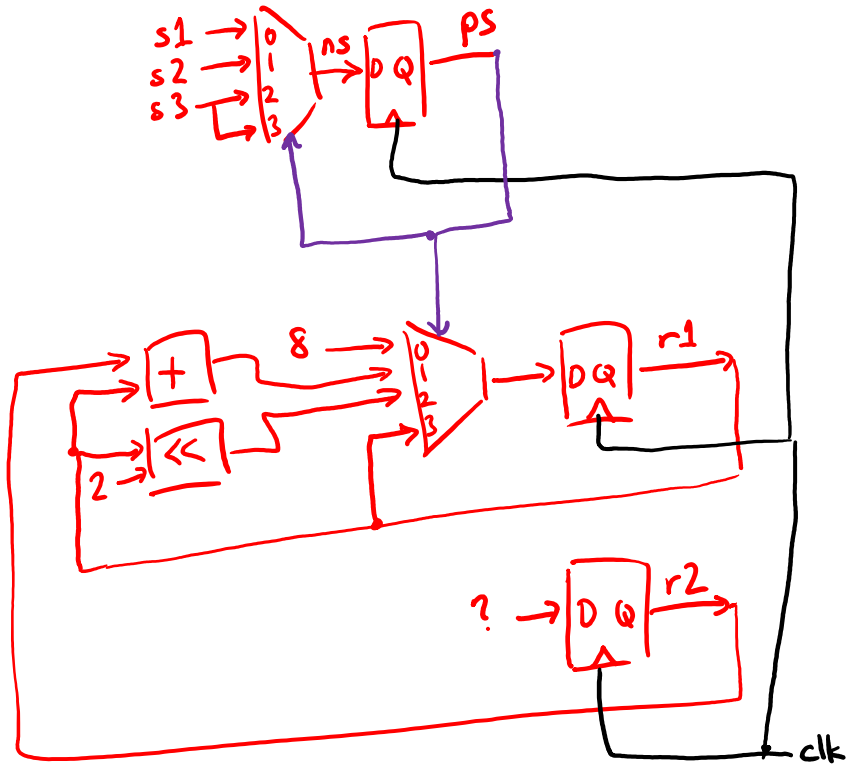
- The behavior of both state and data registers depend on the current control state
  - Can conceptually think of as a MUX to the registers' inputs that uses the current state as its selector bits



Hardware Example (1/2)

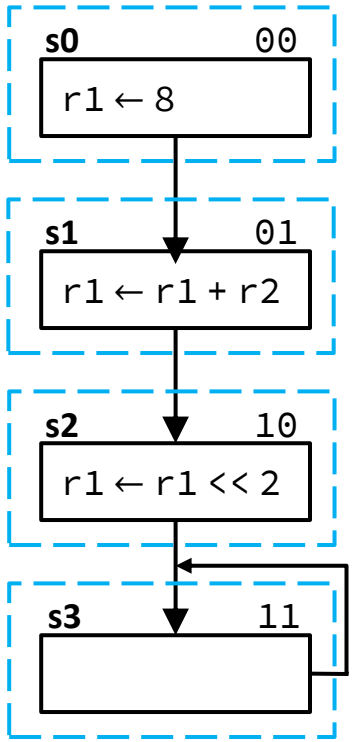
- State transitions and RTL operations are both controlled by the clock
  - It's often helpful to remember the underlying hardware – registers!

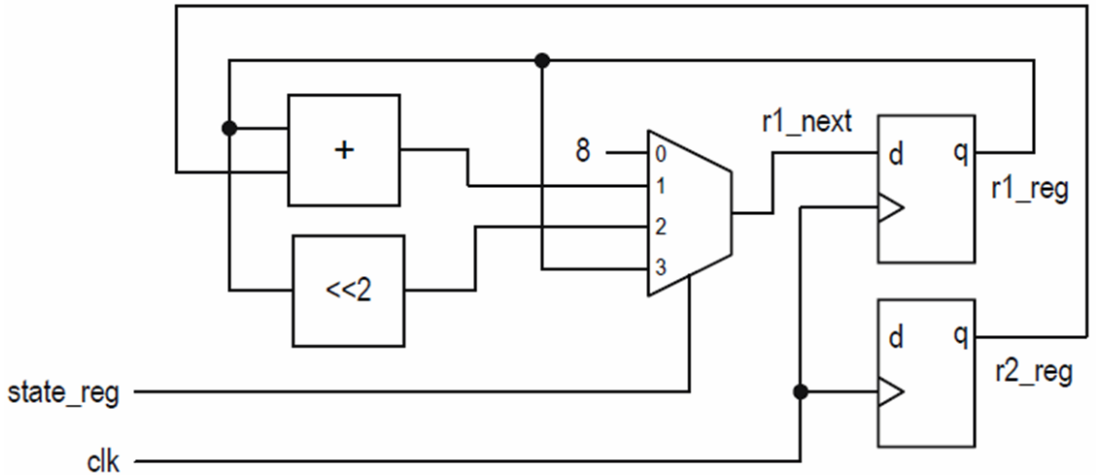




Hardware Example (2/2)

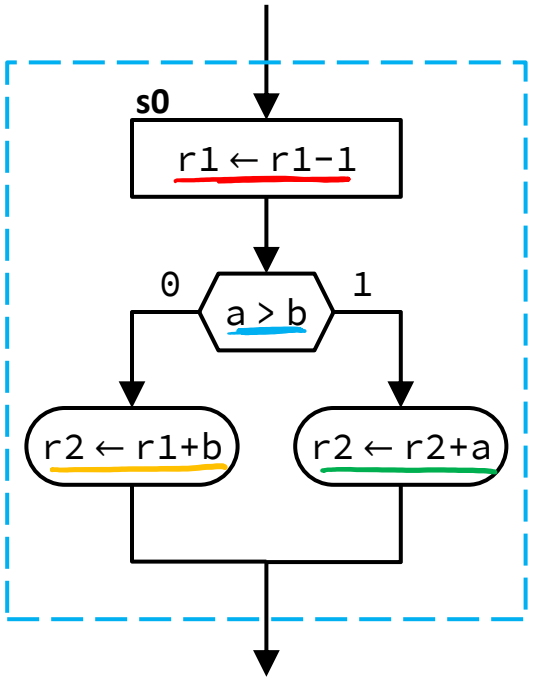
- State transitions and RTL operations are both controlled by the clock
  - It's often helpful to remember the underlying hardware – registers!

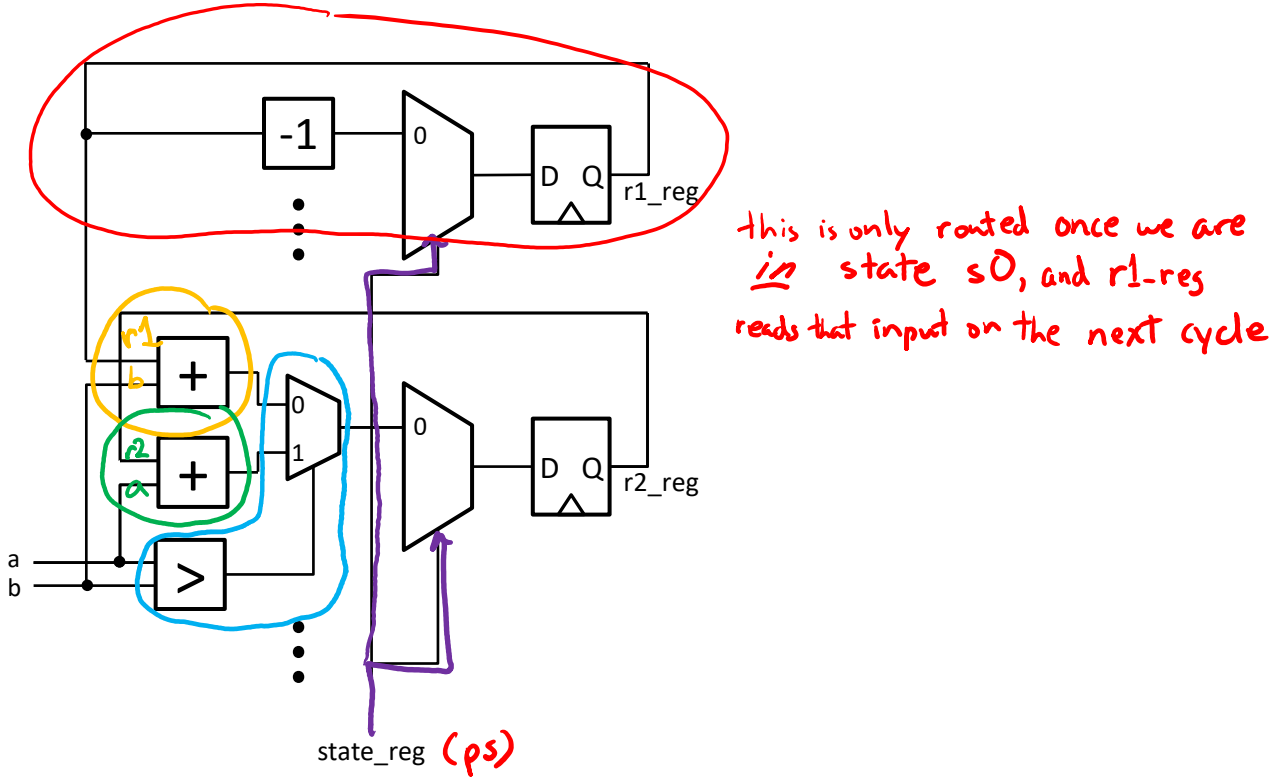




ASMD Timing (1/2)

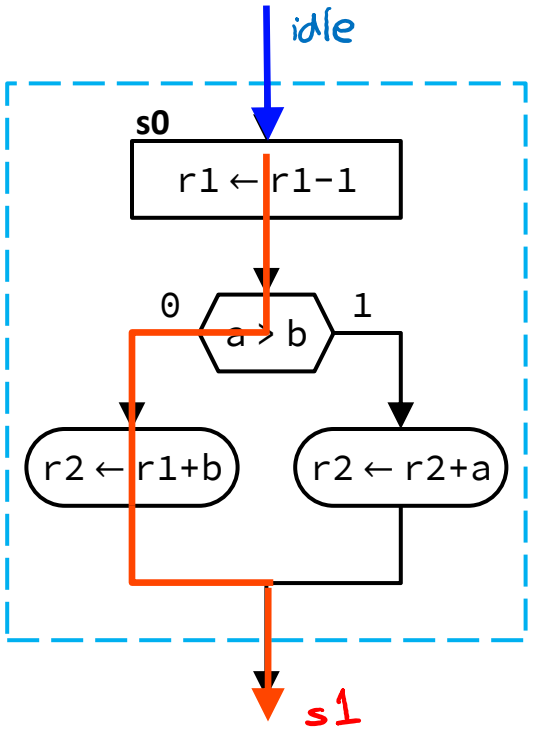
- Everything (registers!) within an ASM block occurs simultaneously at the *next* clock trigger
  - Differs from a flowchart – changes occur at state *exit* rather than *entrance*

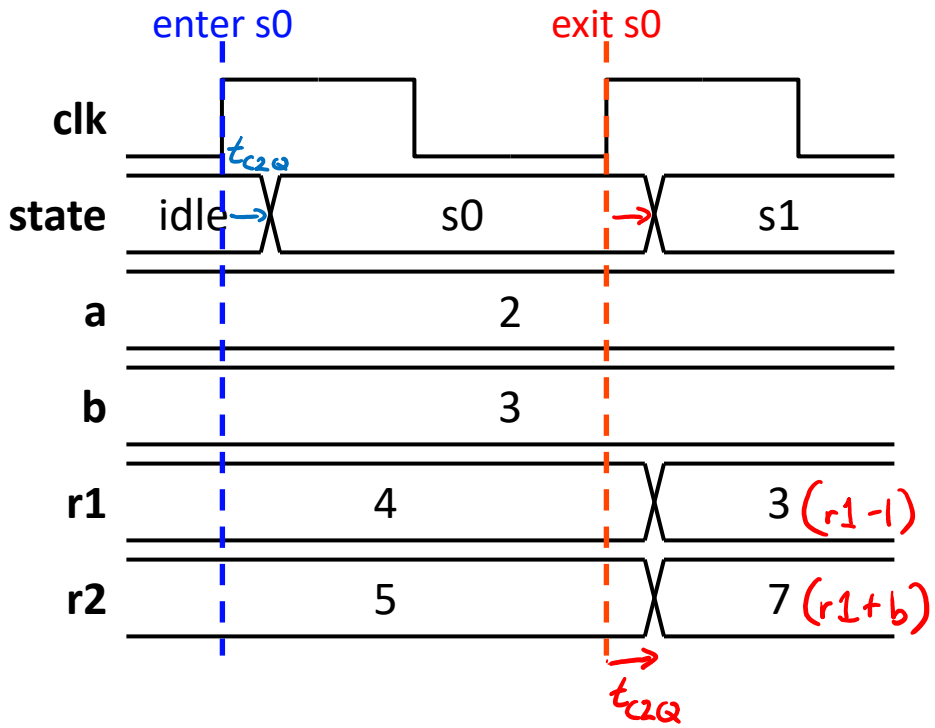




ASMD Timing (2/2)

- 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*

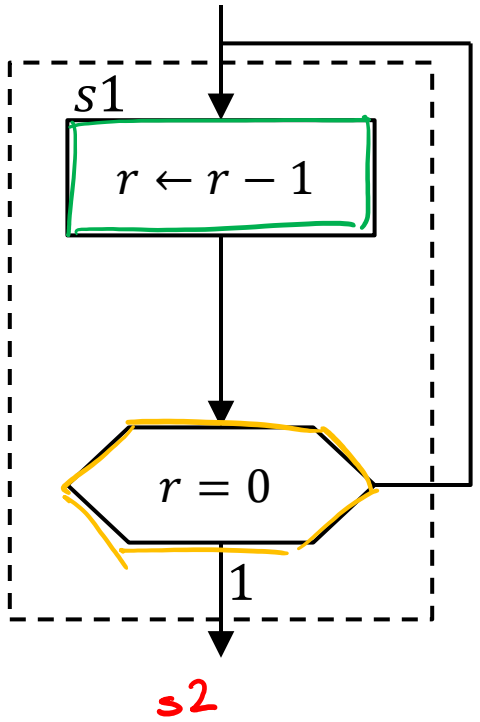


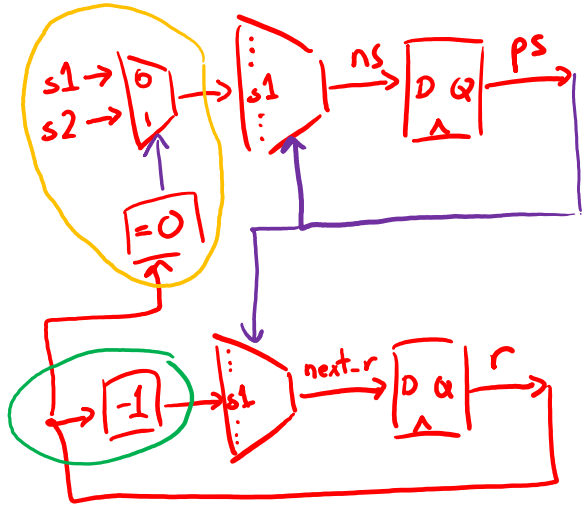


ASMD Timing Question

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

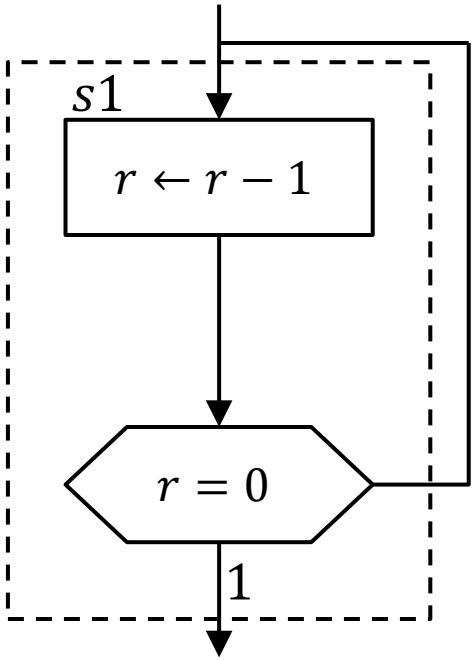
we exit when r=0, which means 0-1 = -1 is sitting on the input to r



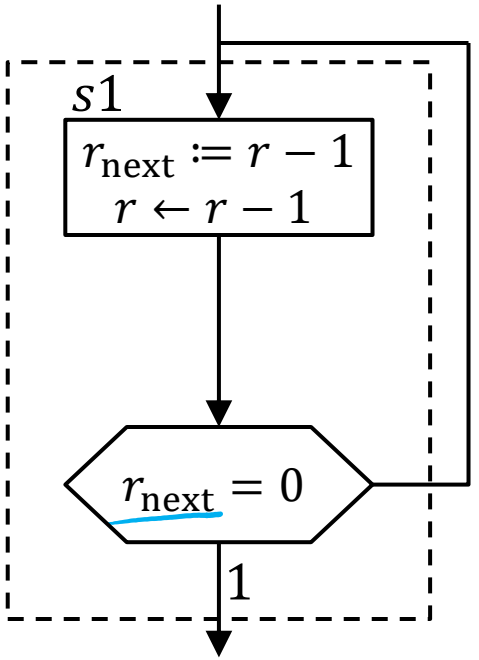


ASMD Timing Tip

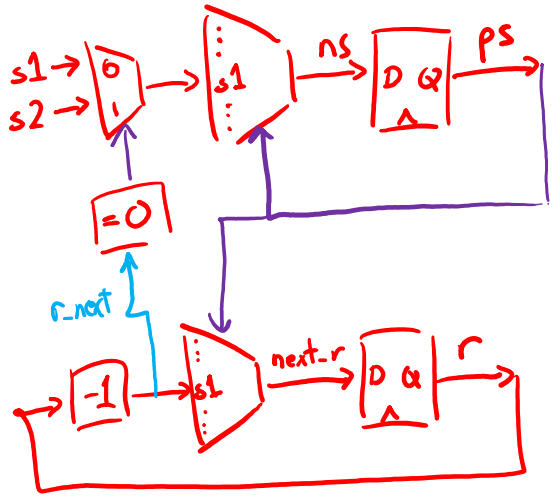
- When a registered output (*e.g.*, r) is used in a decision box, its effect may appear to be delayed by one clock
  - Can define a next-state value (*e.g.*, r_{next}) to use instead



(a) Use old value of r



(b) Use new value of r



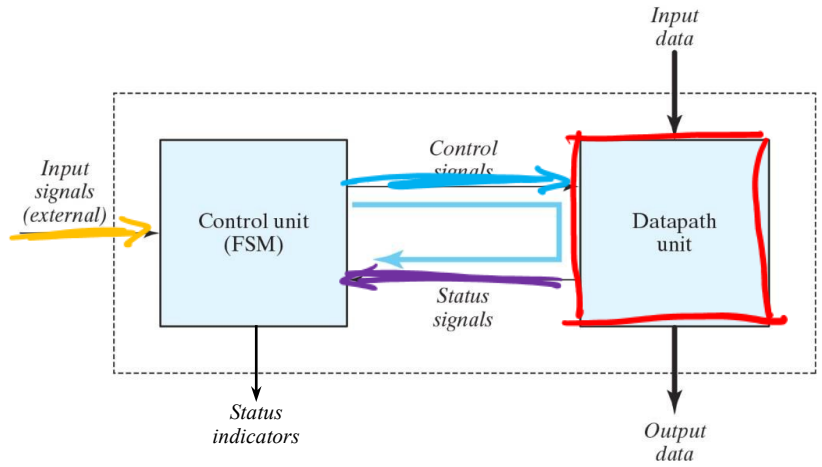
SHORT TECH

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

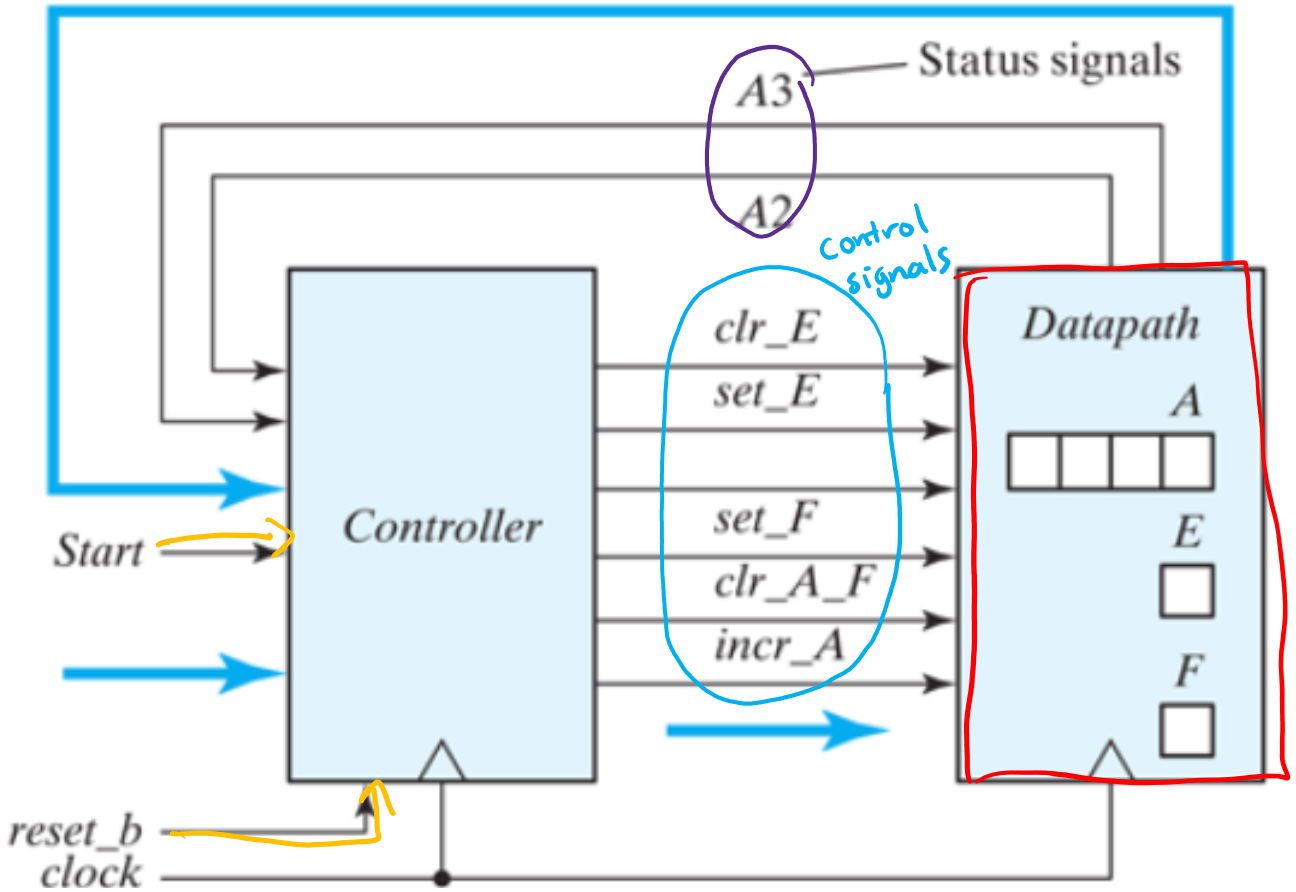
Design Example #1



- System specification:
  - Flip-flops E and F
  - 4-bit binary up-counter $A = 0\text{b}A_3A_2A_1A_0$
  - 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)

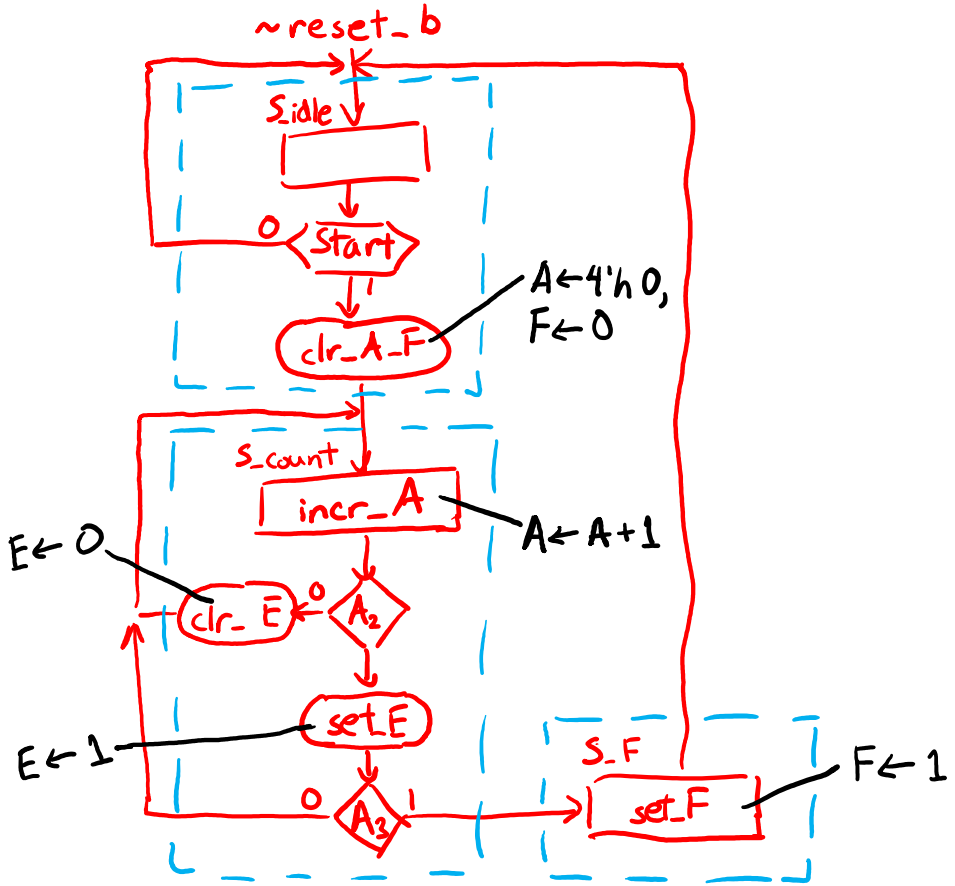
Design Example #1: Block Diagram

- The system can be represented by the following block diagram:

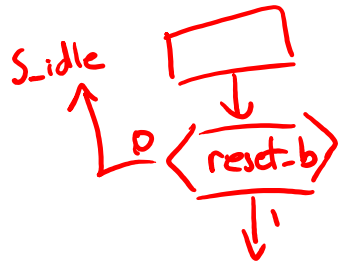


Design Example #1: ASM → ASMD Chart

- Synchronous or asynchronous reset?



for sychronous reset, add decision box on reset_b out of every state box:



Design Example #1: Timing

- Sequence of operations:

	Counter				Flip-Flops			
	A_3	A_2	A_1	A_0	E	F	Conditions	State
	X	X	X	X	1	X	*Start*	*S_idle*
clr_A_F, clr_E	0	0	0	0	1	0	$A_2 = 0, A_3 = 0$	*S_count*
	0	0	0	1	0	0		
	0	0	1	0	0	0		
	0	0	1	1	0	0		
set_E	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		
clr_E	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		
set_E	1	1	0	0	0	0	$A_2 = 1, A_3 = 1$	
	1	1	0	1	1	0		*S_F*
	1	1	0	1	1	1		*S_idle*

incr_A

Design Example #1: Logic

- Controller:
  - State Table:

$S\_idle = \overline{P_1}\,\overline{P_0}$

$S\_count = \overline{P_1}\,P_0$

$S\_F = P_1\,P_0$

state logic inputs ← → state logic outputs

Present-State Symbol	Present State P_1	Present State P_0	Inputs Start	Inputs A_2	Inputs A_3	Next State N_1	Next State N_0	Outputs set_E	Outputs clr_E	Outputs set_F	Outputs clr_A_F	Outputs 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:

$$N_1 = S\_count \cdot A_2\,A_3 \qquad set\_F = S\_F$$

$$N_0 = S\_count + S\_idle \cdot Start \qquad clr\_A\_F = S\_idle \cdot Start$$

$$set\_E = S\_count \cdot A_2 \qquad incr\_A = S\_count$$

$$clr\_E = S\_count \cdot \overline{A_2}$$

SHORT TECH BREAK

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

control signals (out)
status signals (in)
external inputs (in)

```
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
   enum logic [1:0] { S_idle, S_count, S_F =2'b11} ps, ns ;

   // controller logic w/synchronous reset
   always_ff @ (posedge clk)
      if (~reset_b)
         ps <= S_idle;
      else
         ps <= ns;
```

```
//asynchronous reset:
always_ff @ (posedge clk, negedge reset_b)
   if (~reset_b)
      ps <= S_idle;
   else
      ps <= ns;
```

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

control signals (out)
status signals (in)
external inputs (in)

```
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
   enum logic [1:0] {S_idle, S_count, S_F = 2'b11} ps, ns;

   // controller logic w/synchronous reset
   always_ff @(posedge clk)
      if (~reset_b)
         ps <= S_idle;
      else
         ps <= ns;
```

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

```
// next state logic
always_comb
    case (ps)
        S_idle:     ns = Start ? S_count : S_idle;
        S_count:    ns = (A3 & A2) ? S_F : S_count;
        S_F:        ns = S_idle;
    endcase

// output assignments
  assign  set_E = (ps == S_count) & A2;
                .
                .
                .

endmodule  // controller
```

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

```
   // next state logic
   always_comb
      case (ps)
         S_idle:  ns = Start ? S_count : S_idle;
         S_count: ns = (A2 & A3) ? S_F : S_count;
         S_F:              ns = S_idle;
      endcase

   // output assignments
   assign set_E   = (ps == S_count) & A2;
   assign clr_E   = (ps == S_count) & ~A2;
   assign set_F   = (ps == S_F);
   assign clr_A_F = (ps == S_idle) & Start;
   assign incr_A  = (ps == S_count);


endmodule  // controller
```

Design Example #1: Datapath Outline

control signals (in)
status signals (out)
external inputs (in)
external outputs (out)

```
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
   always_ff @ (posedge clk) begin
       if (set_E)   E <= 1;
    else if (clr_E)   E <= 0;
       if (set_F)   F <= 1;
    else if(clr_A_F) begin
             A <= 4'h0;
             F <= 0;
         end
    else if(incr_A)  A <= A + 4'h1;
   end
endmodule  // datapath
```

watch for potential conflicts!

Design Example #1: Datapath Code

control signals (in)
status signals (out)
external inputs (in)
external outputs (out)

```
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
   always_ff @(posedge clk) begin
      if (clr_E)       E <= 1'b0;
      else if (set_E)  E <= 1'b1;
      if (clr_A_F)
         begin
            A <= 4'h0;
            F <= 1'b0;
         end
      else if (set_F)  F <= 1'b1;
      else if (incr_A) A <= A + 4'h1;
   end  // always_ff

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 (control signals and status signals that aren't outputs)
   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
```