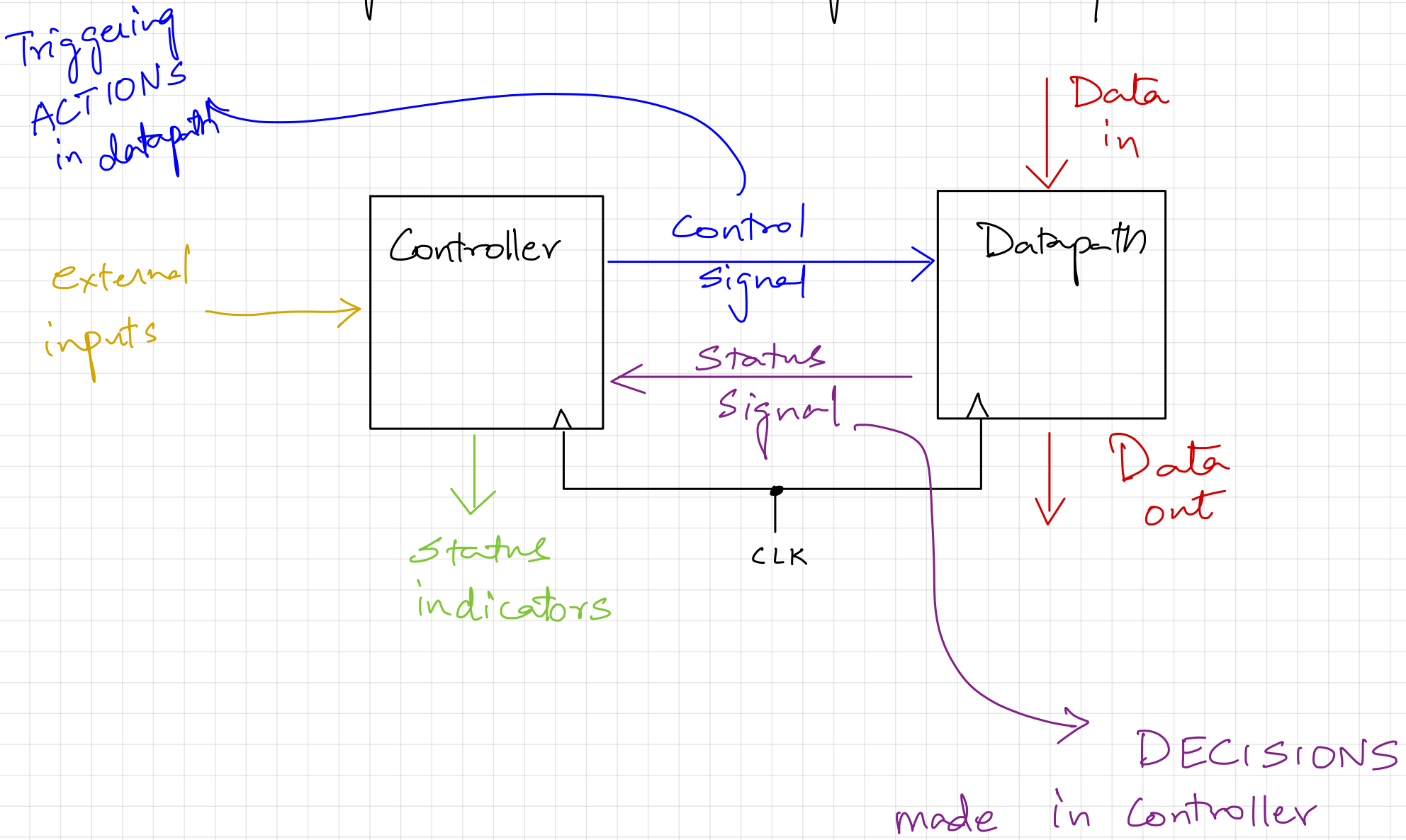
EE/CSE 371
Design of Digital Circuits and Systems

Lecture 7: ASM with Datapath II

SDS

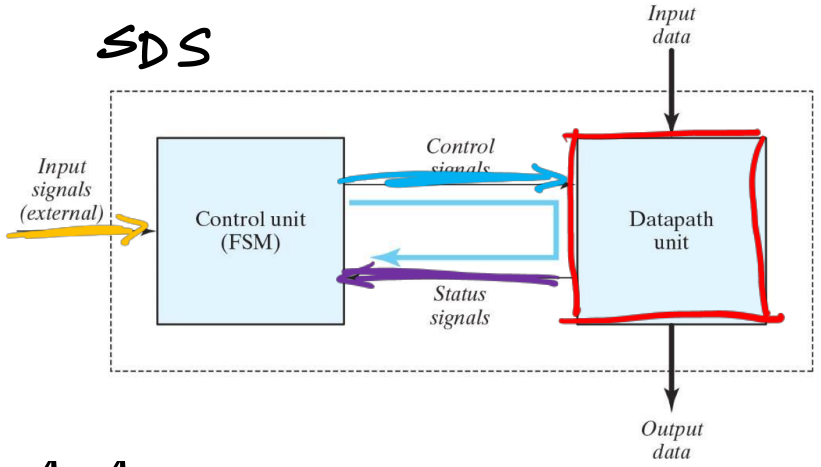
Synchronous Digital System.



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

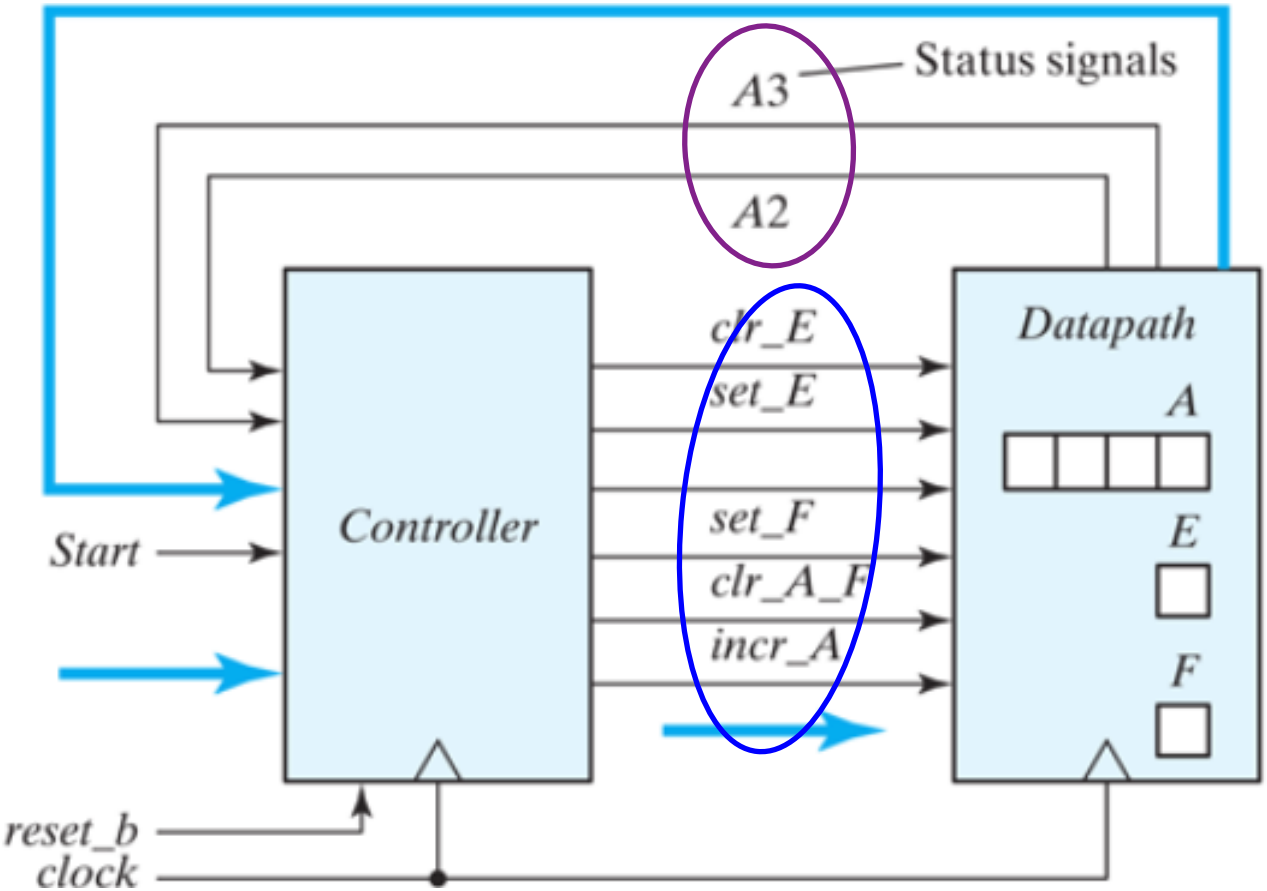
Design Example #1



- System specification:
  - Flip-flops E and F
  - 4-bit binary 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)

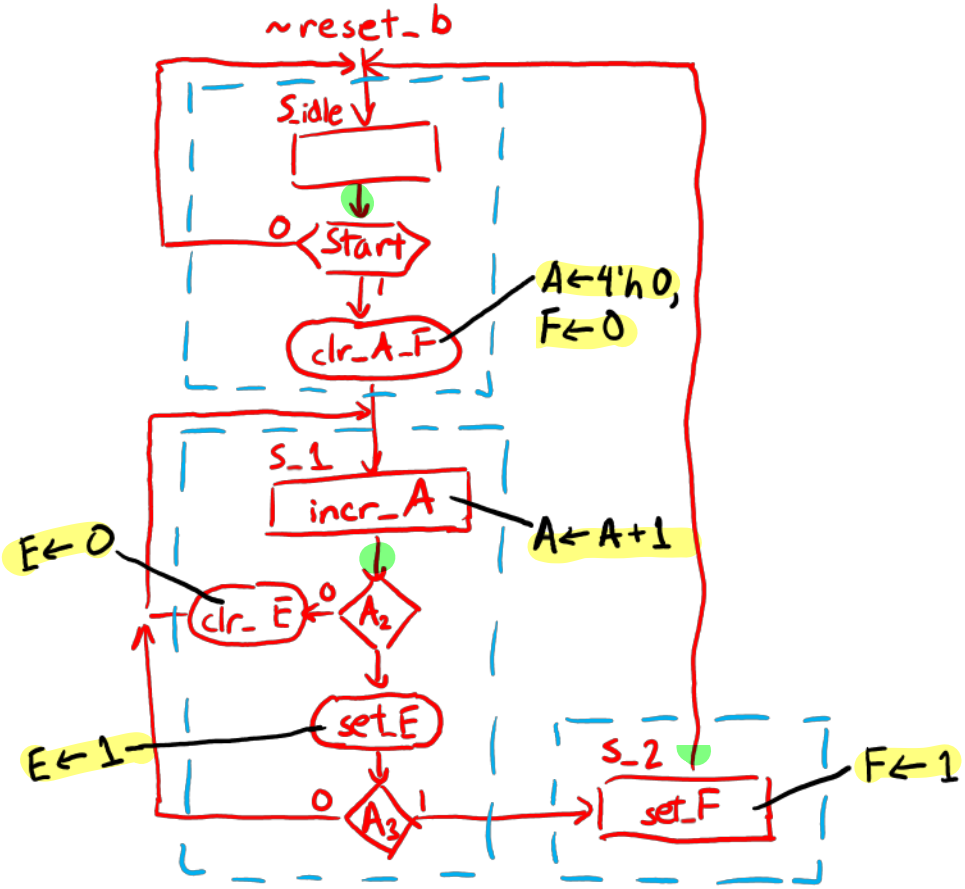
Design Example #1

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

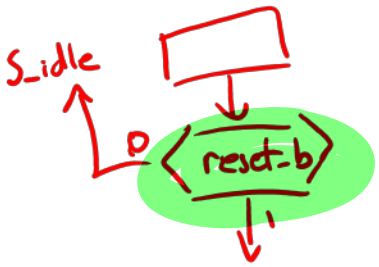


Design Example #1 (ASM → ASMD Chart)

- 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:

Time

clr_A_F

clr_E

Set_E

Counter				Flip-Flops			
A_3	A_2	A_1	A_0	E	F	Conditions	State
X	X	X	X	1	X	*Start*	*S_idle*
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		
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_F*
1	1	0	1	1	1		*S_idle*

Design Example #1 (Logic)

- Controller:
  - State Table:

Present-State Symbol	Present State P_1	P_0	Inputs Start	A_2	A_3	Next State N_1	N_0	Outputs 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

S_idle = $\overline{P_1} \cdot \overline{P_0}$

S_count = $\overline{P_1} \cdot P_0$

S_F = $P_1 \cdot P_0$

- Logic:

$N_1 =$ (Ps == S_1) & A_2 & A_3

$N_0 =$

$set\_E =$

$clr\_E =$ (Ps == S_1) · $\overline{A_2}$

$set\_F =$

$clr\_A\_F =$

$incr\_A =$

Design Example #1 (SV, Controller)

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 {S_idle, S_1, S_2 = 3} ps,ns;

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

asynchronous reset

negedge reset

```
   // next state logic
   always_comb
      case (ps)
         S_idle: ns = Start ? S_1 : S_idle;
         S_1:    ns = (A2 & A3) ? S_2 : S_1;
         S_2:    ns = S_idle;
      endcase

   // output assignments
   assign set_E =   (ps == S_1) & A2;
   assign clr_E =   (ps == S_1) & ~A2;
   assign set_F =   (ps == S_2);
   assign clr_A_F = (ps == S_idle) & Start;
   assign incr_A =  (ps == S_1);


endmodule  // controller
```

Design Example #1 (SV, Datapath)

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'b0;
            F <= 1'b0;
         end
      else if (set_F)  F <= 1'b1;
      else if (incr_A) A <= A + 4'h1;
   end  // always_ff

endmodule  // datapath
```

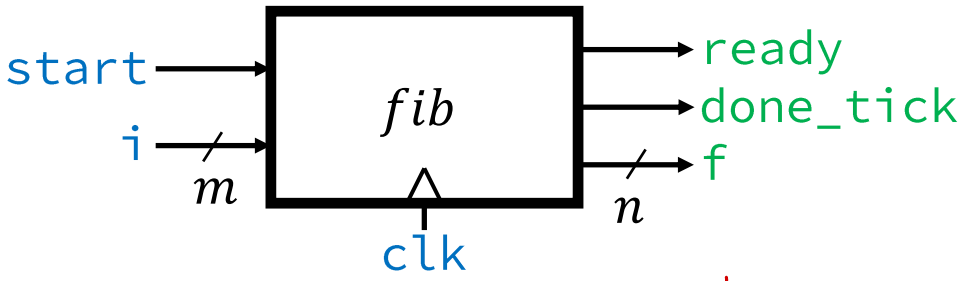
else if to avoid conflicts!

Design Example #1 (SV, Top-Level Design)

```
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
```

Design Example #2: Fibonacci

- Design a sequential Fibonacci number circuit with the following properties:



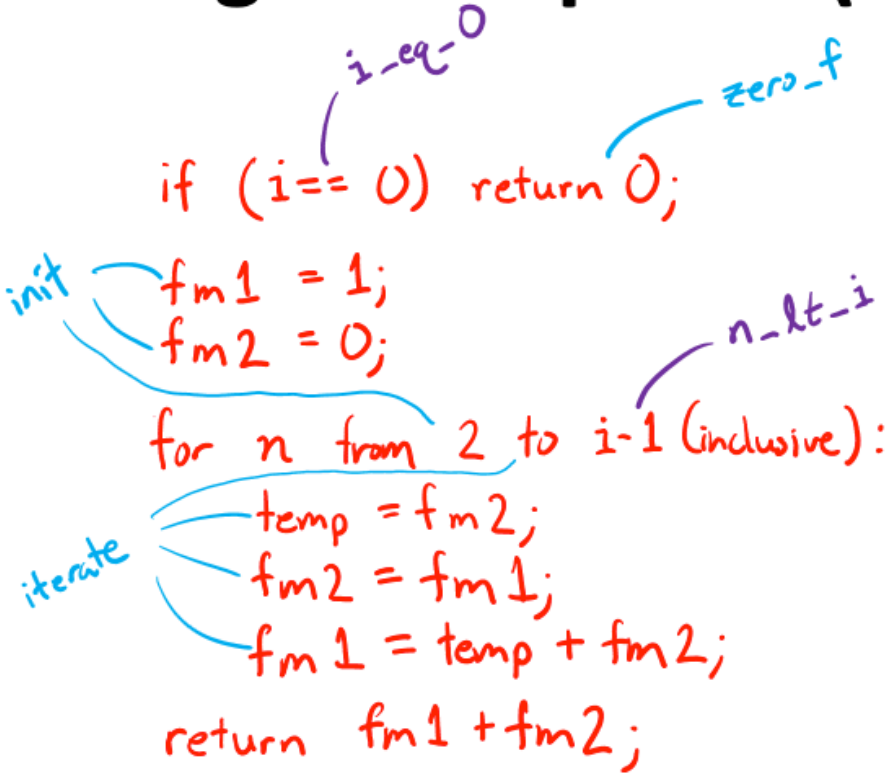
  - i is the desired sequence number
  - f is the computed Fibonacci number:

$$fib(i) = \begin{cases} 0, & i = 0 \\ 1, & i = 1 \\ fib(i-1) + fib(i-2), & i > 1 \end{cases}$$

i	fib
0	0
1	1
2	1
3	2
4	3
⋮	5
	8
	⋮

  - ready means the circuit is idle and ready for new input
  - start signals the beginning of a new computation
  - done_tick is asserted for 1 cycle when the computation is complete

Design Example #2 (Pseudocode)

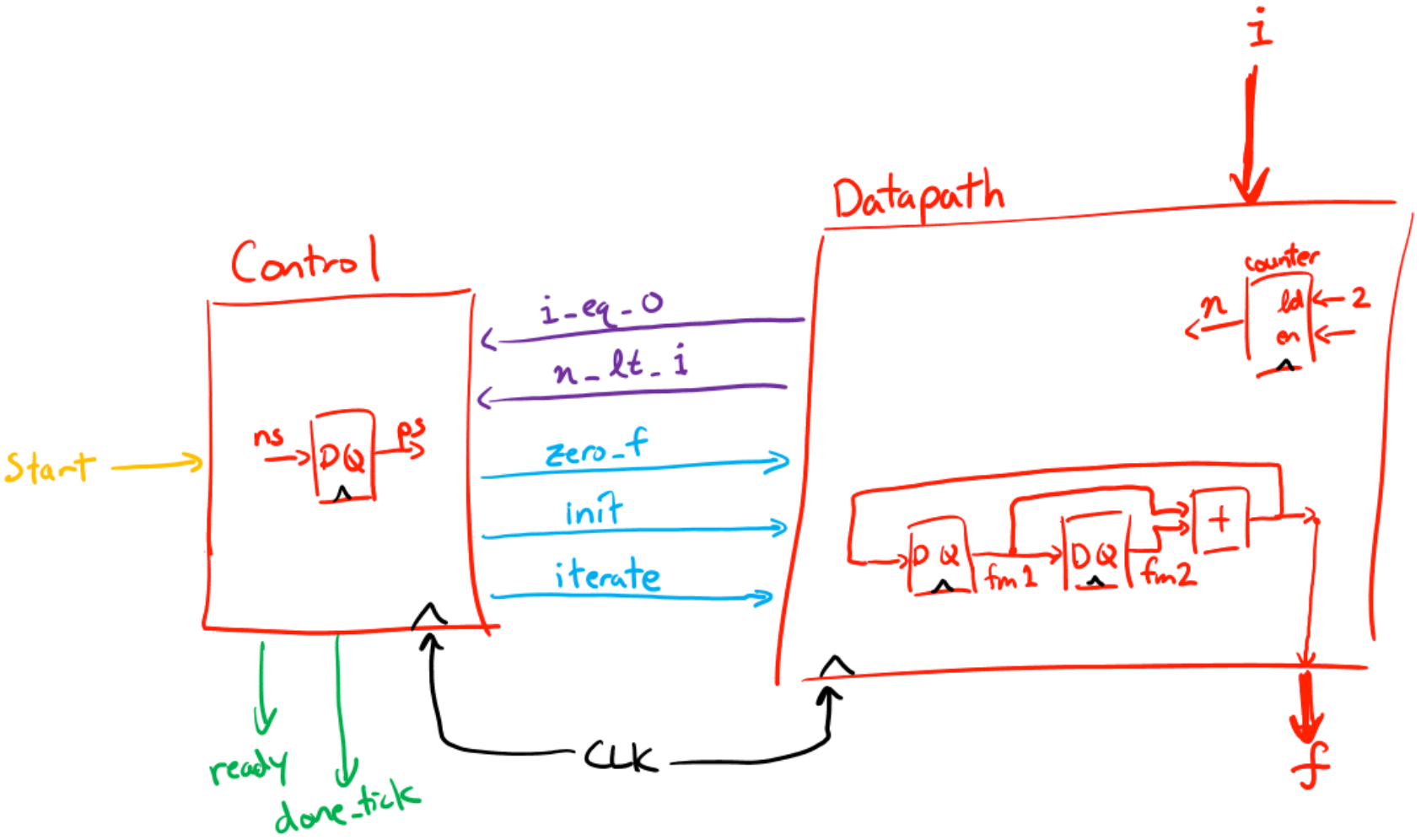


Some Variants:

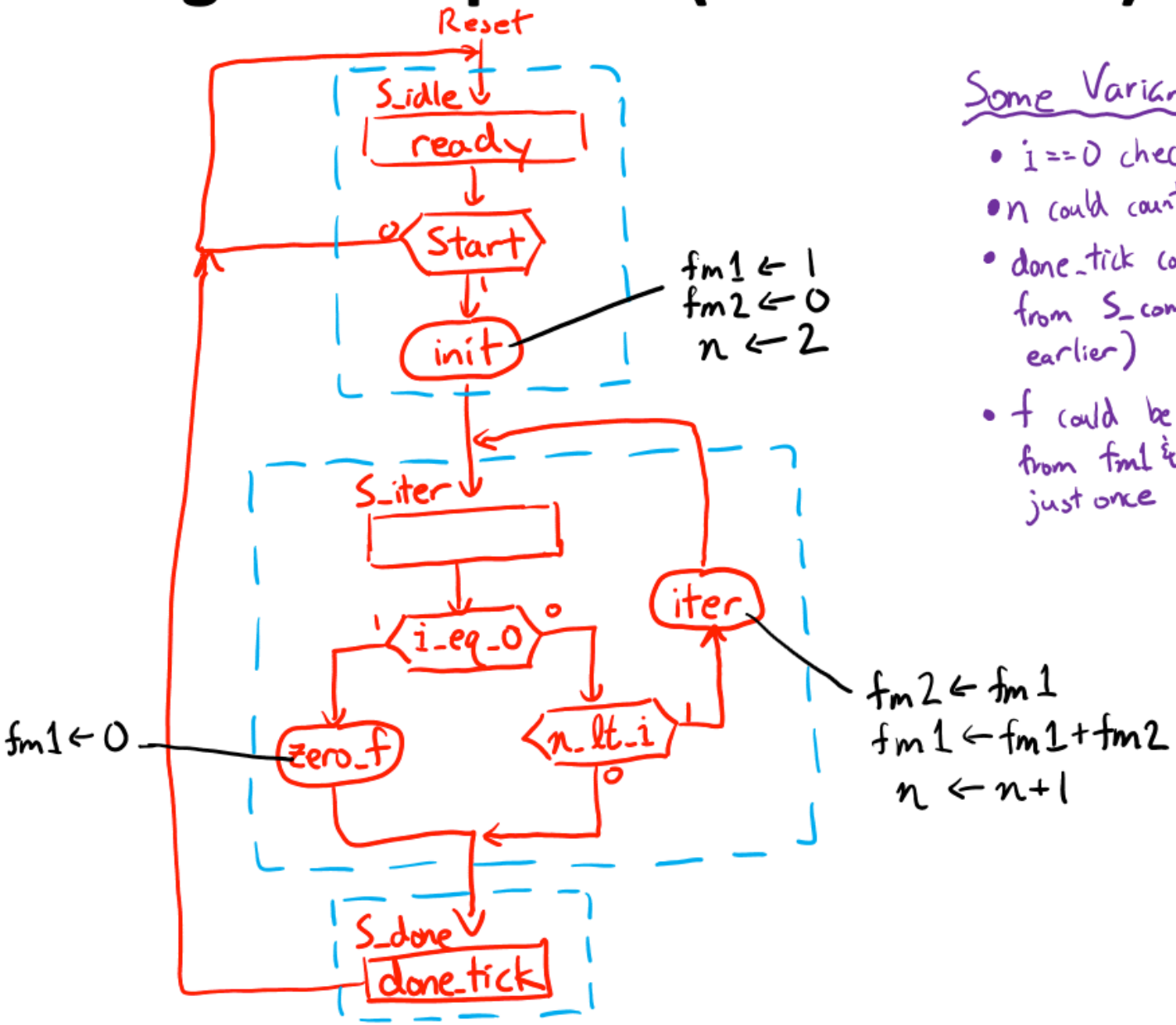
- could have explicitly shown i == 1 base case
- any loop bounds that execute i-2 times will work
- could have explicitly used a third f = fm1 + fm2 variable

- Pseudocode analysis:
  - Variables are part of **datapath**; assignments become RTL operations
  - Chunks of related actions should be triggered by **control signals**
  - Decision points become **status signals**

Design Example #2 (Control-Datapath)



Design Example #2 (ASMD Chart)



Some Variants:

- i==0 check could be in S_idle
- n could count down
- done_tick could be a Mealy output from S_comp (shows up 1 cycle earlier)
- f could be a separate register from fm1 & fm2 and be updated just once

Design Example #2 (SV)

```
fib_control:
   // port definitions
   // define state names and variables
   // controller logic w/synchronous reset
   // next state logic
   // output assignments

fib_datapath:
   // port definitions
   // datapath logic

fib:
   // port definitions
   // define status and control signals
   // instantiate control and datapath
```

Other Hardware Algorithms

- Sequential binary multiplier or divider
- Arithmetic mean

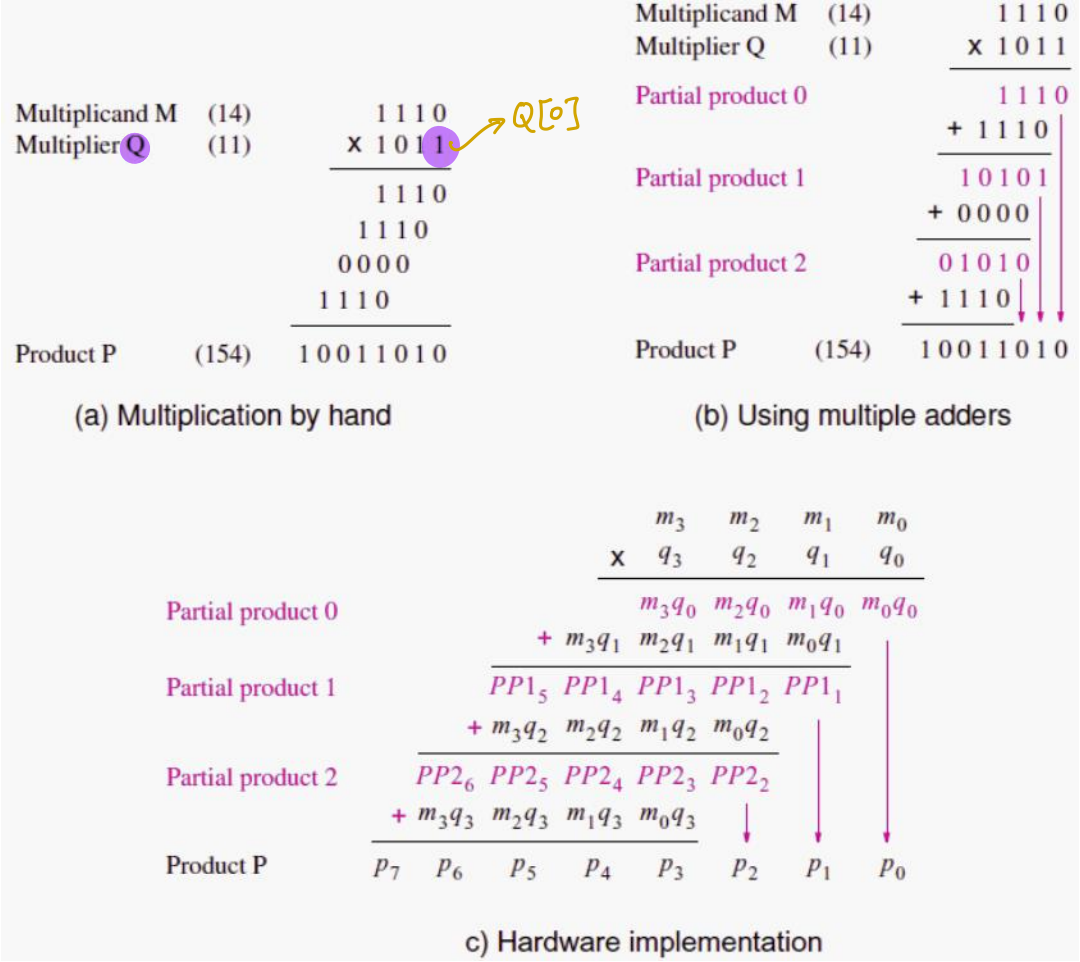
- Lab 4: Bit counting
- Lab 4: Binary search
- Lab 5: Bresenham's line

Hardware Acceleration

- ASMD as a design process can be used to implement software algorithms
- Custom hardware can accelerate operation:
  - Hardware can better exploit parallelism
  - Hardware can implement more specialized operations
  - Hardware can reduce "processor overhead" (*e.g.*, instruction fetch, decoding)
- "Hardware accelerators" are frequently used to complement processors to speed up common, computationally-intensive tasks
  - *e.g.*, encryption, machine vision, cryptocurrency mining

Binary Multiplication

- Multiplication of unsigned numbers



Parallel Binary Multiplier

- *Parallel* multipliers require a lot of hardware