FSM Modularity Pipelines

Stuff

- Pset 06 Due few minutes ago
- Pset 07 Out…Due Tuesday
- Lab 03 Due Tonight 10pm
- Lab 04 out tomorrow early.
  - Due next **Thursday** 5pm

Car Theft FSMs

- 2016, MA: ~7 million people, 6,600 car thefts for year
- 1975, MA: 5.8 million people, 91,000 car thefts for year (peak)

~5% of cars were stolen per year in MA

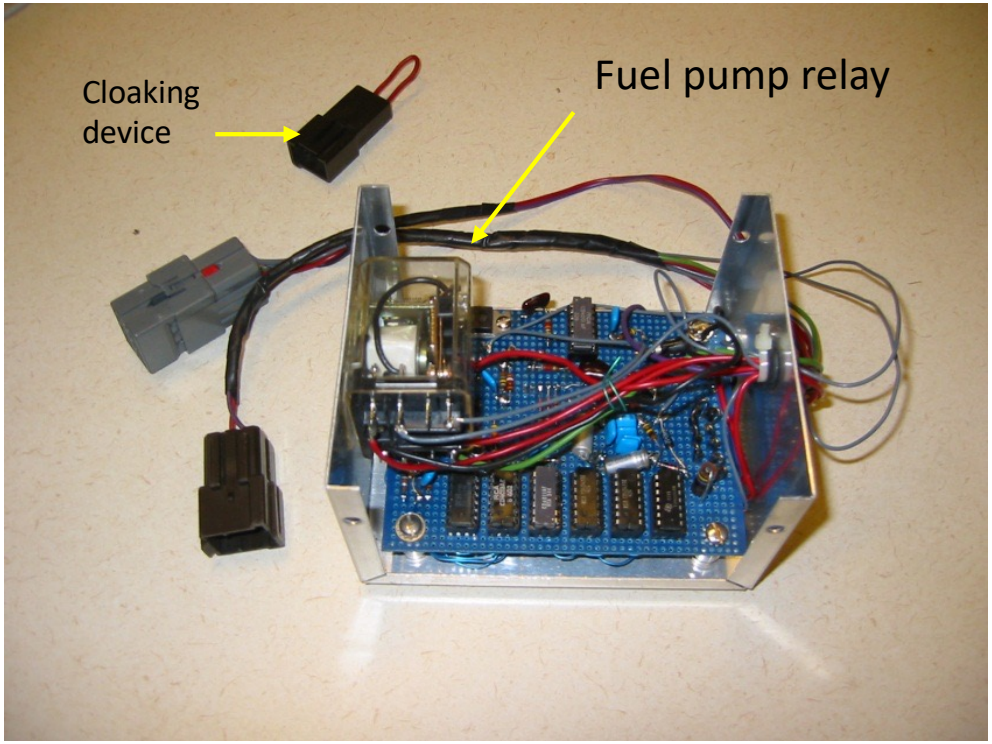
#1 state for car theft for 1965-1987

Massachusetts Population and Rate of Crime Rank Compared to other States

State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	1960	8	35	36	33	41	38	32	36	34	38	11
Massachusetts	1961	8	29	35	28	42	35	31	34	29	32	7
Massachusetts	1962	9	26	33	24	40	37	25	31	26	31	7
Massachusetts	1963	9	28	33	27	40	39	27	33	27	33	4
Massachusetts	1964	9	26	33	25	37	37	24	38	25	34	2
Massachusetts	1965	10	23	33	22	36	39	21	36	23	35	1
Massachusetts	1966	10	24	32	23	38	41	19	36	21	37	1
Massachusetts	1967	10	25	34	25	38	37	23	37	25	38	1
Massachusetts	1968	10	21	31	20	35	35	21	35	19	35	1
Massachusetts	1969	10	19	29	18	35	38	19	35	15	34	1
State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	1970	10	20	31	18	38	33	19	35	14	35	1
Massachusetts	1971	10	16	27	16	36	38	13	33	13	32	1
Massachusetts	1972	10	17	26	16	38	39	11	29	13	35	1
Massachusetts	1973	10	15	20	14	36	34	10	29	12	33	1
Massachusetts	1974	10	12	20	11	37	37	10	29	12	32	1
Massachusetts	1975	10	11	16	12	39	29	9	26	13	34	1
Massachusetts	1976	10	13	18	12	40	37	11	22	11	36	1
Massachusetts	1977	10	15	17	16	43	31	12	19	13	37	1
Massachusetts	1978	10	16	16	17	40	31	13	19	12	38	1
Massachusetts	1979	10	16	15	15	41	32	12	16	11	36	1
State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	1980	11	16	13	17	39	31	9	13	13	39	1
Massachusetts	1981	11	20	12	21	40	30	8	15	17	40	1
Massachusetts	1982	11	18	14	21	37	31	10	13	19	40	1
Massachusetts	1983	11	20	12	22	40	28	9	10	20	41	1
Massachusetts	1984	12	25	13	24	36	25	12	12	25	41	2
Massachusetts	1985	12	23	16	25	39	23	12	14	24	40	1
Massachusetts	1986	12	26	18	27	38	26	14	17	28	43	1
Massachusetts	1987	13	26	14	28	42	26	15	14	31	45	1
Massachusetts	1988	13	23	13	25	38	25	14	11	29	41	3

State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	2000	13	42	21	44	41	36	27	14	41	49	16
Massachusetts	2001	13	41	20	43	42	29	25	15	40	47	18
Massachusetts	2002	13	39	18	42	38	34	24	17	39	46	18
Massachusetts	2003	13	28	17	40	42	30	19	18	36	46	18
Massachusetts	2004	13	39	18	42	37	33	20	18	38	47	22
Massachusetts	2005	13	42	19	45	37	36	20	18	35	48	30
Massachusetts	2006	13	41	20	44	35	37	21	19	34	45	33
Massachusetts	2007	14	42	22	42	37	38	23	18	33	45	36
Massachusetts	2008	14	37	20	44	41	39	24	14	32	45	35
Massachusetts	2009	15	37	17	43	38	38	23	14	33	45	33
State	Year	Population	Index	Violent	Property	Murder	Rape	Robbery	Assault	Burglary	Larceny	Vehicle
Massachusetts	2010	14	32	13	38	31	35	18	11	29	41	34
Massachusetts	2011	14	36	14	42	37	37	20	12	32	43	34
Massachusetts	2012	14	40	19	44	46	39	20	14	33	48	39
Massachusetts	2013	14	36	16	45	43	16	18	15	35	46	39
Massachusetts	2014	14	39	18	46	43	35	20	14	38	47	40
Massachusetts	2015	15	42	18	47	45	38	27	14	42	48	42
Massachusetts	2016	15	45	23	47	47	42	26	20	45	49	44

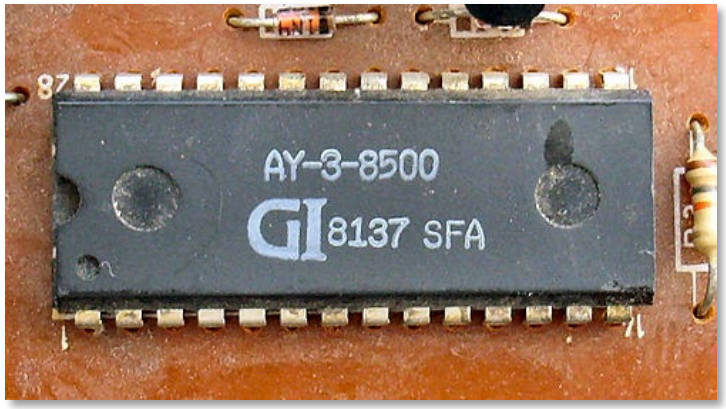
Car Alarm – CMOS Implementation



- Design Specs
  - Operating voltage 8-18VDC
  - Operating temp: -10C +65C
  - Attitude: sea level
  - Shock/Vibration
- Notes
  - Protected against 24V power surges
  - CMOS implementation
  - CMOS inputs protected against 200V noise spikes
  - On state DC current <10ma
  - Include T_PASSENGER_DELAY and Fuel Pump Disable
  - System disabled (cloaked) when being serviced.

Pong in History:

- http://www.pong-story.com/gi.htm



AY-3-8500 "Ball-and-Paddle" chip

https://commons.wikimedia.org/wiki/File:AY-3-8500.jpg

Tiger Electronics Games

https://oladaniel.com/pica-pic

Tiger Electronics Games

- Tiger Electronics had 100's of versions of these in the 1980s and 1990s
- Almost all of them were based on three or four common finite state machine game chips
- They'd slap a different LCD skin and game art onto the same chip and resell

Lab 04 (A)

- Build a Video Camera Pipeline:
  - Properly pipelined mirroring and scaling
  - Work with RGB and YCrCb threshold masking
  - Find Center-of-Mass of Channel
  - Use it to move a sprite around

- Demo…

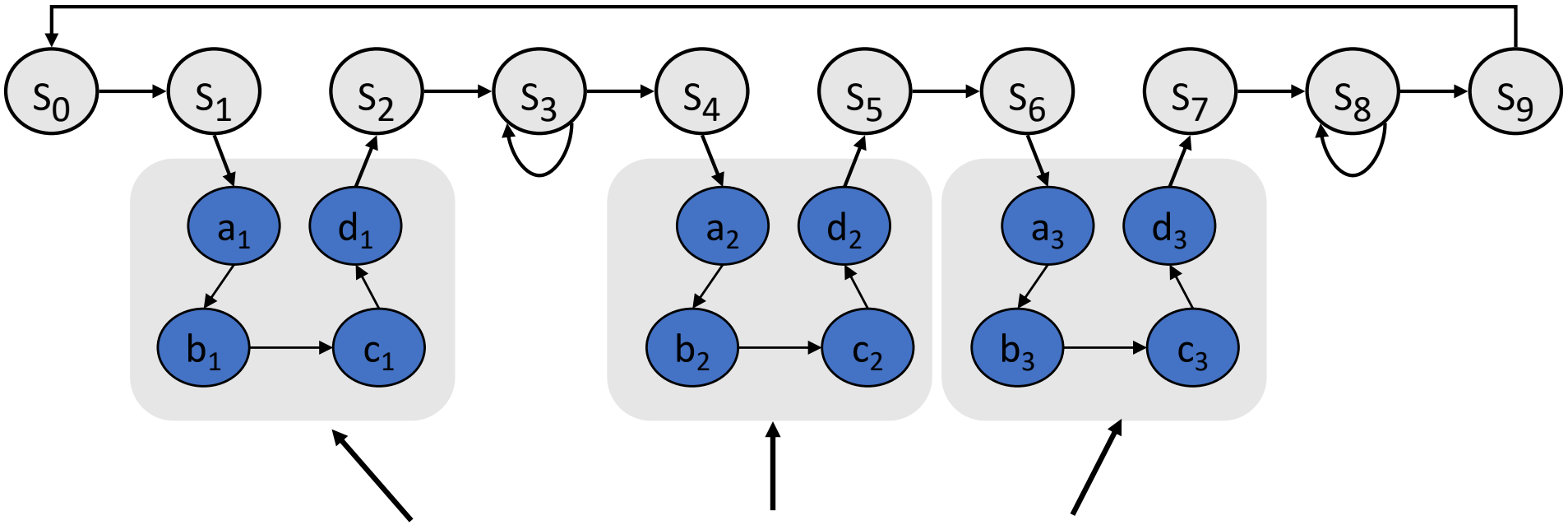
Stuff for Today

- FSM Modularity
- Pipelining I
- Clocks/Timing!

FSM Modularity

Toward FSM Modularity

- Consider the following abstract FSM:

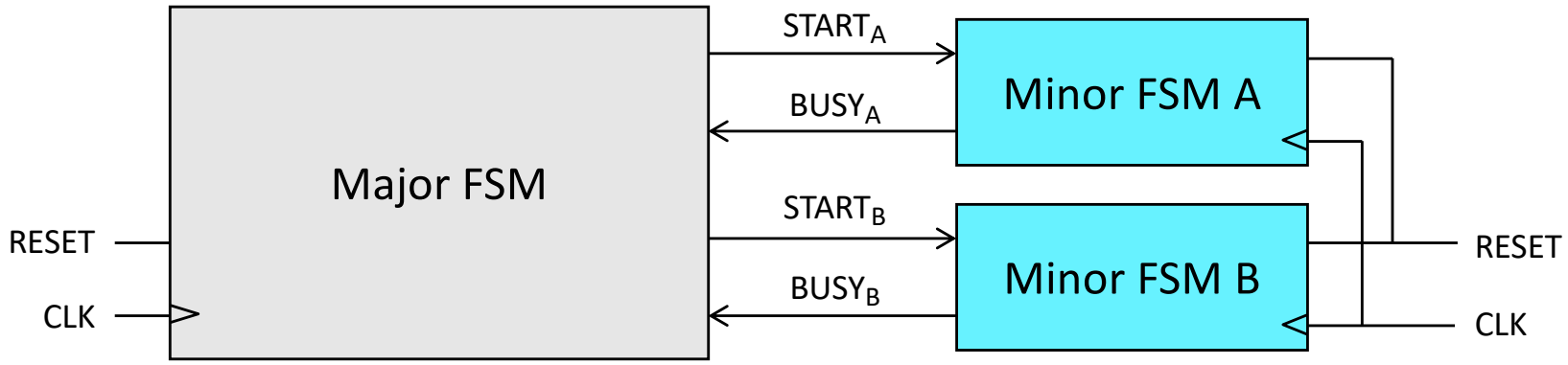


- Suppose that each set of states a_x...d_x is a "sub-FSM" that produces exactly the same outputs.
- Can we simplify the FSM by removing equivalent states?

  No! The outputs may be the same, but the next-state transitions are not.
- This situation closely resembles a procedure call or function call in software...how can we apply this concept to FSMs?

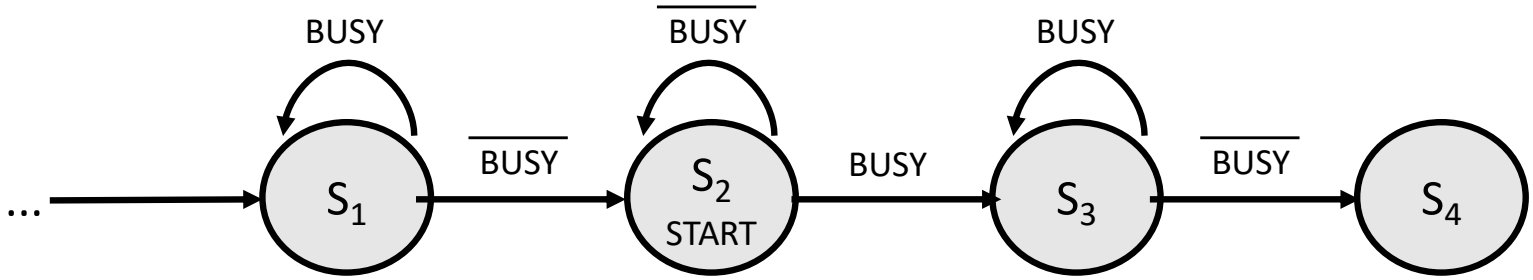
Acknowledgements: Rex Min

The Major/Minor FSM Abstraction



- Subtasks are encapsulated in minor FSMs with common reset and clock
- Simple communication abstraction:
  - START: tells the minor FSM to begin operation (the call)
  - BUSY: tells the major FSM whether the minor is done (the return)
- The major/minor abstraction is great for...
  - Modular designs (*always* a good thing)
  - Tasks that occur often but in different contexts
  - Tasks that require a variable/unknown period of time
  - Event-driven systems

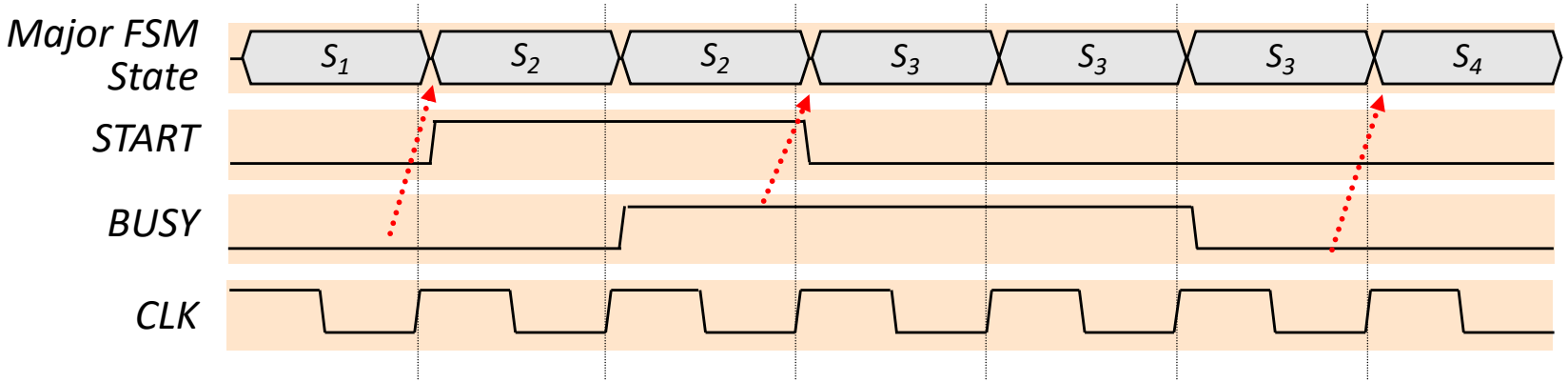
Inside the Major FSM



1. Wait until the minor FSM is ready

2. Trigger the minor FSM (and make sure it's started)

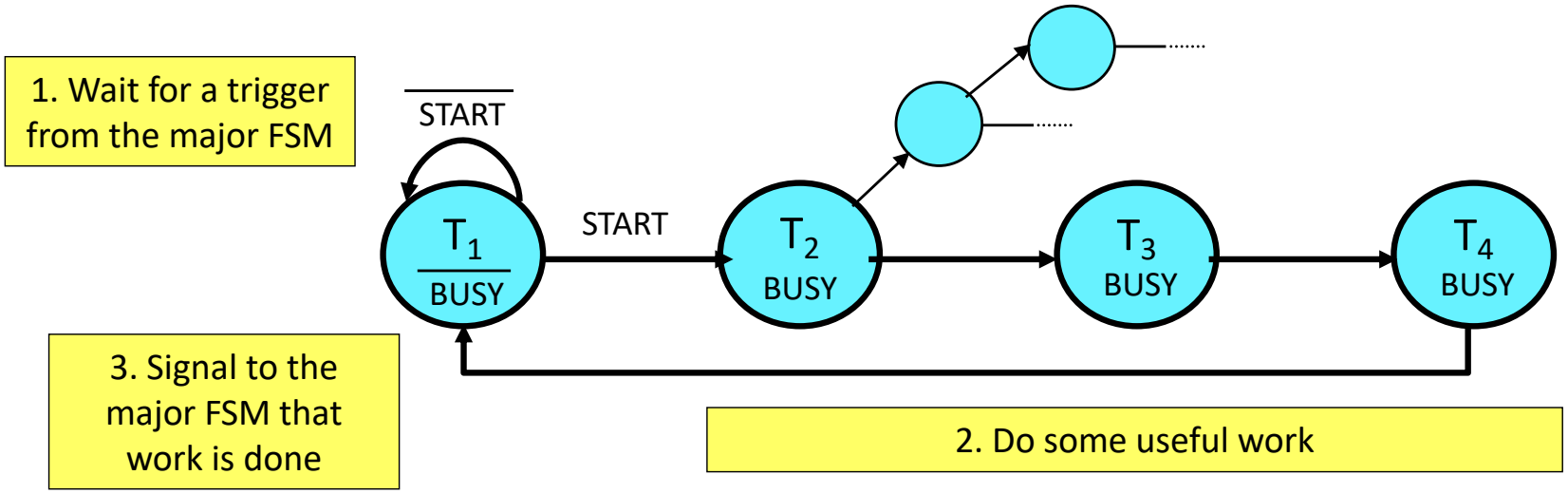
3. Wait until the minor FSM is done

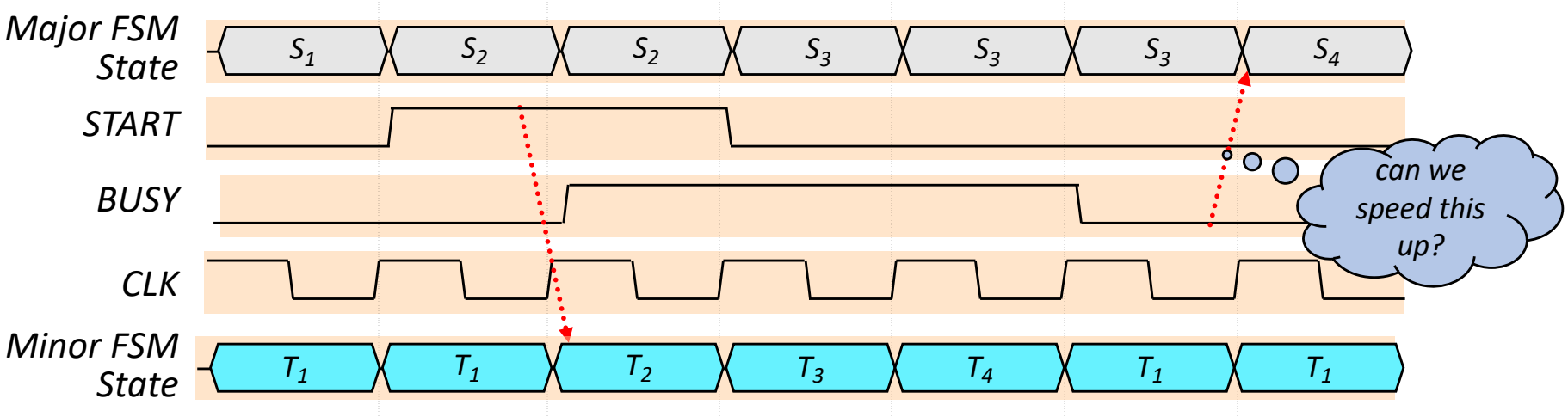


Variations:

- Usually don't need both Step 1 and Step 3
- One cycle "done" signal instead of multi-cycle "busy"

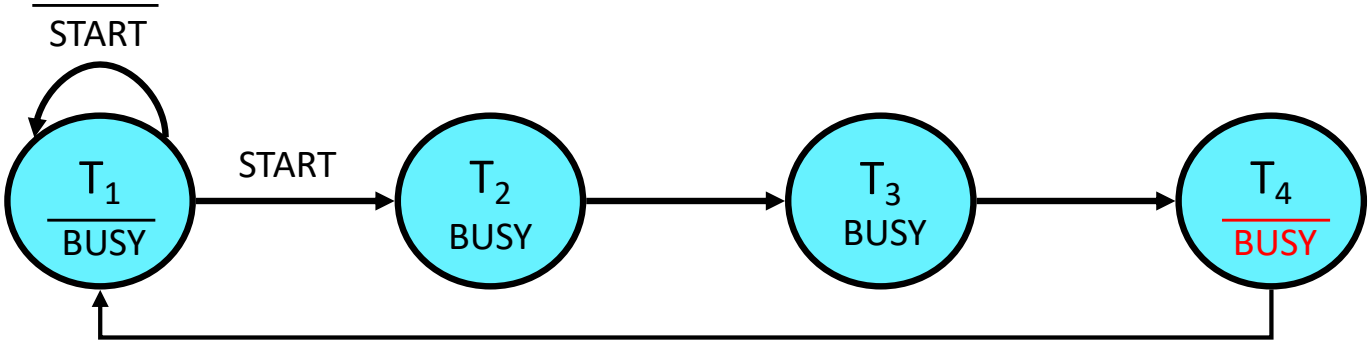
Inside the Minor FSM





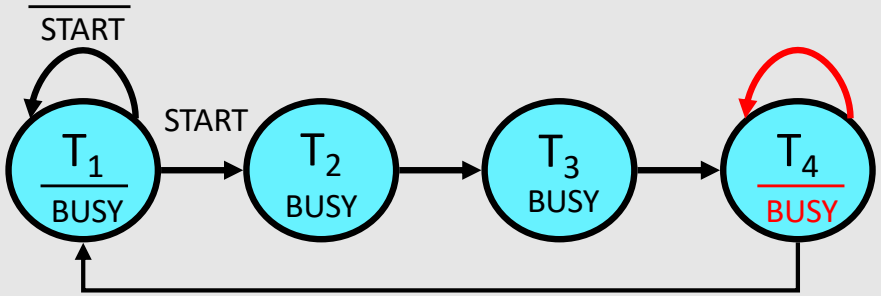
Optimizing the Minor FSM

Good idea: de-assert BUSY one cycle early



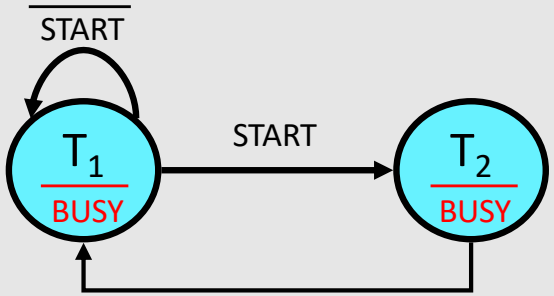
Bad idea #1:

T_4 may not immediately return to T_1

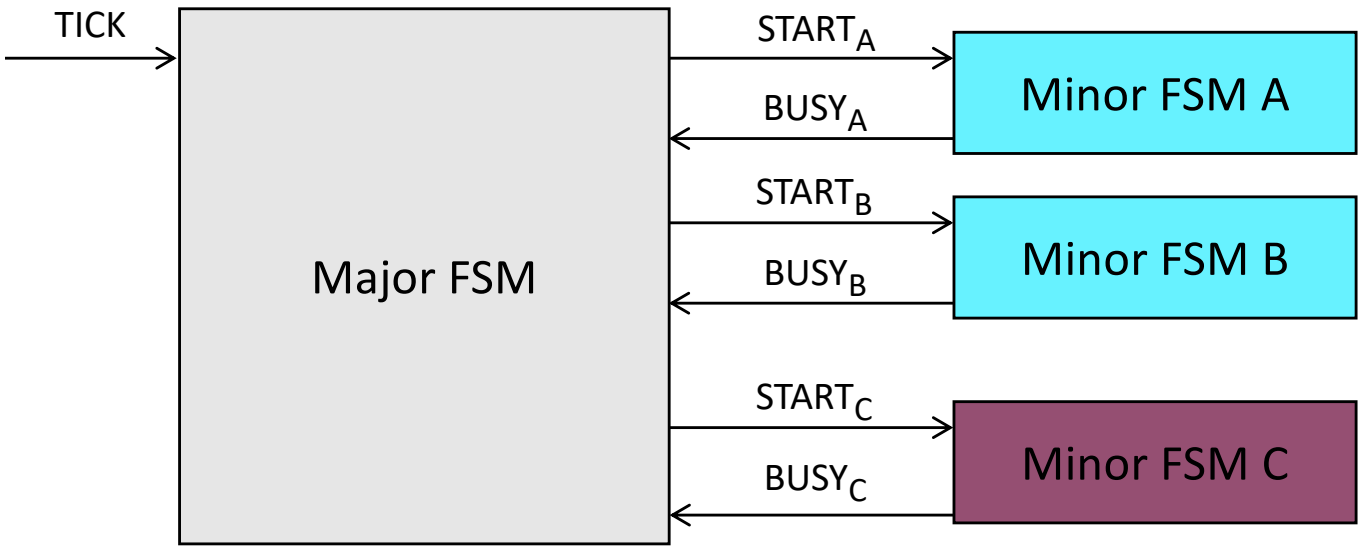


Bad idea #2:

BUSY never asserts!

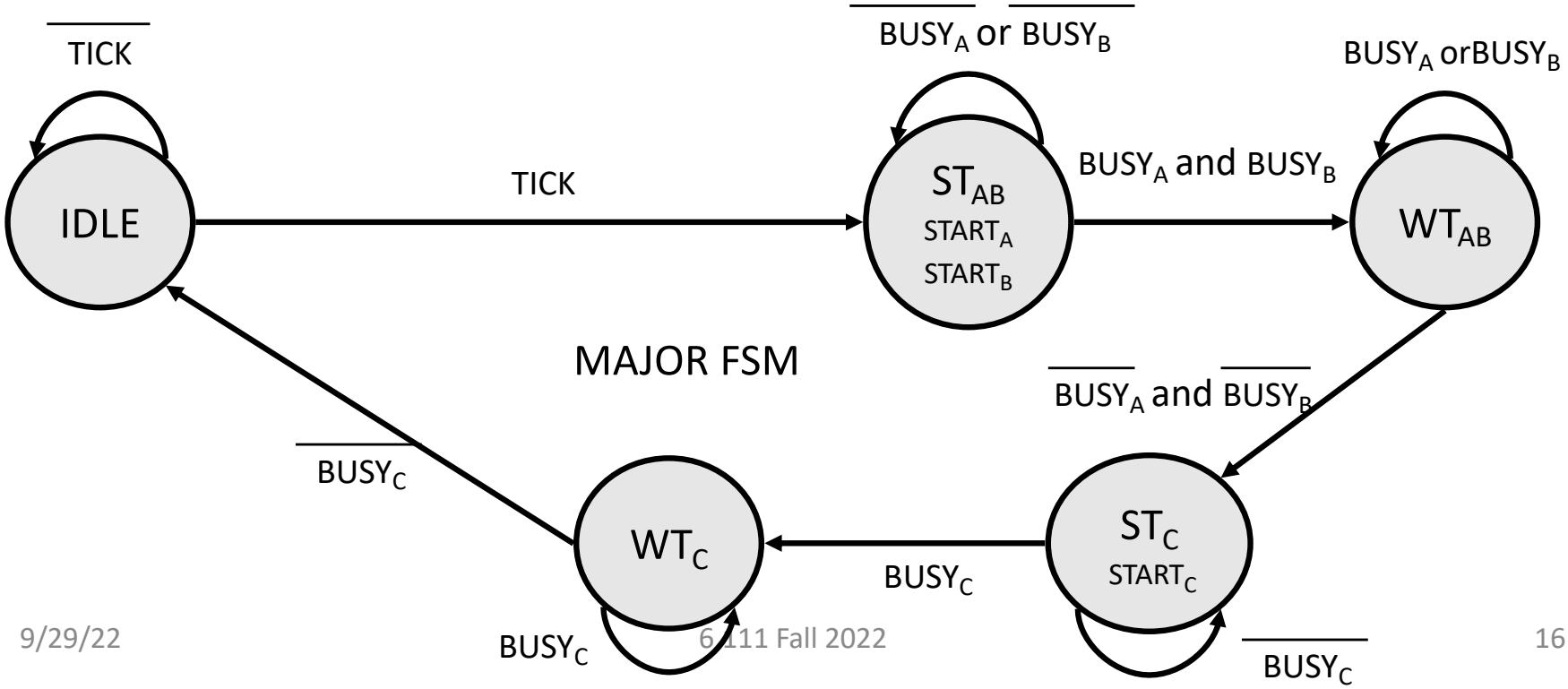


A Four-FSM Example

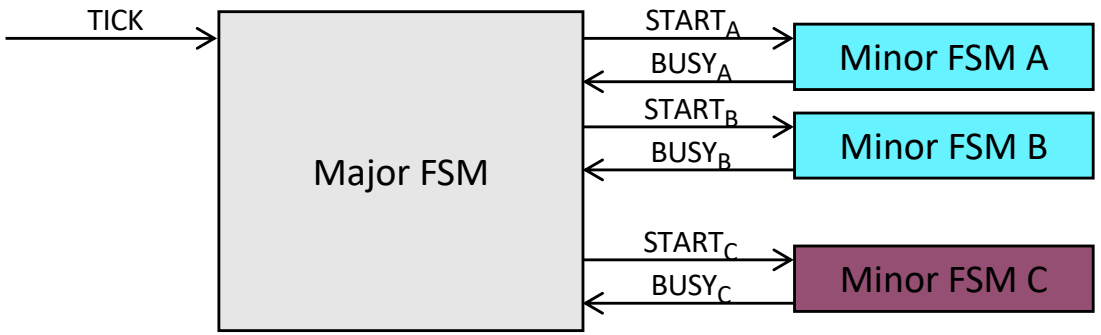


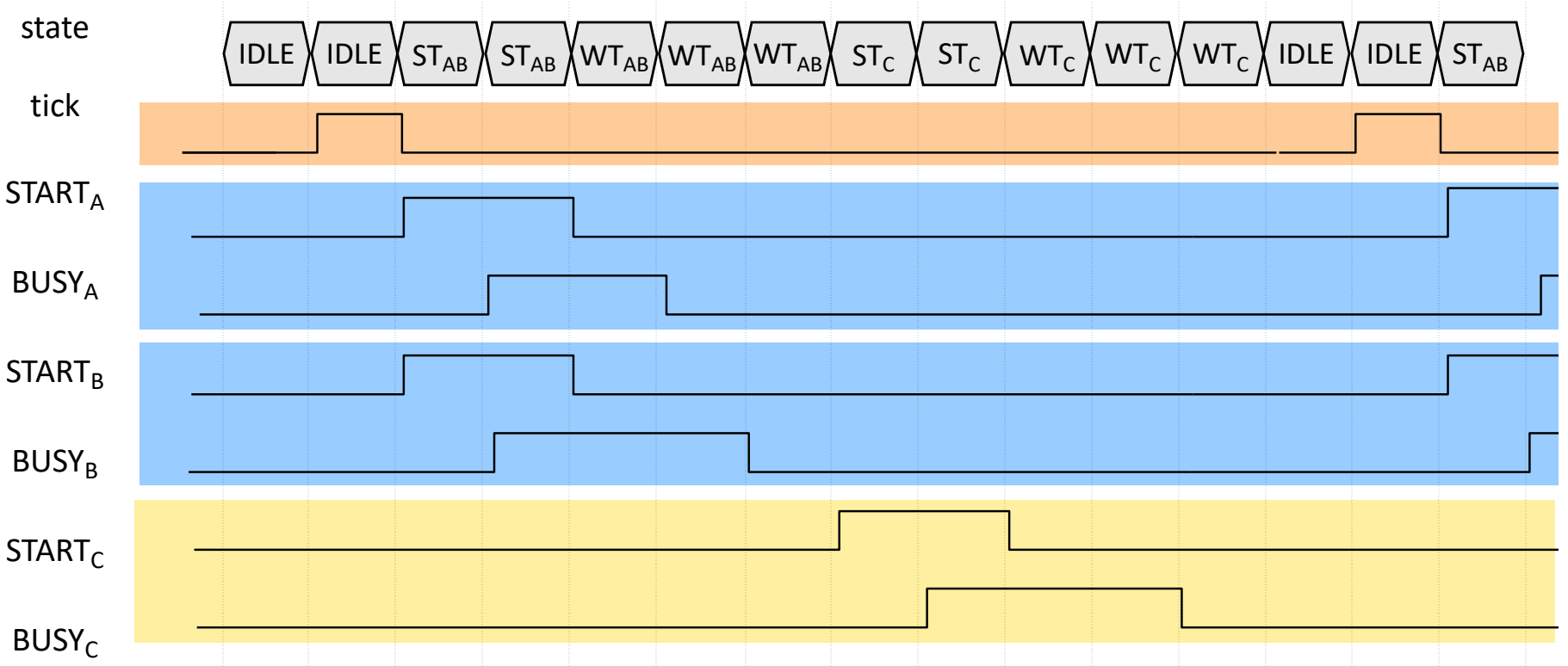
Operating Scenario:

- Major FSM is triggered by TICK
- Minors A and B are started simultaneously
- Minor C is started once both A and B complete
- TICKs arriving before the completion of C are ignored



Four-FSM Sample Waveform

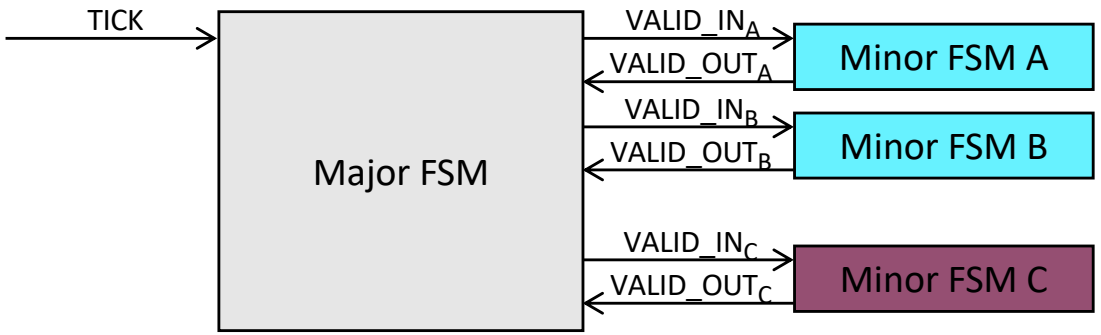


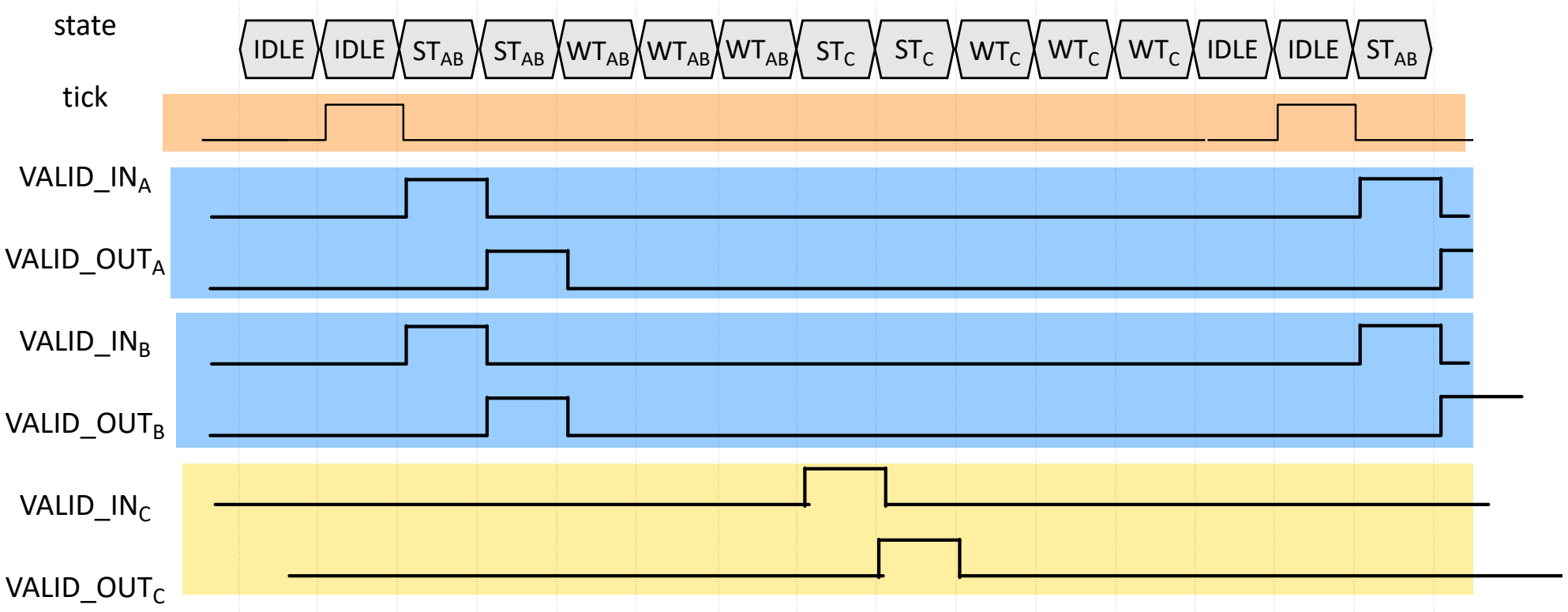


Alternative to Busy Signals

- As an alternative to busy signals sometimes just having a single-cycle “valid” signals is sufficient.
- If the downstream systems involved are stateful enough to be able to keep track of various system’s this can work
- Or you can do both

Alternative to Busy Signals





A Divider

- This is a Verilog FSM example of the algorithm on the previous page which will run an unknown number of times given a set of inputs
- This is how the functionality of a while loop could be developed in your modules
- Will not handle negative, or 0 or other things…

Code and testbench on the site's lecture page

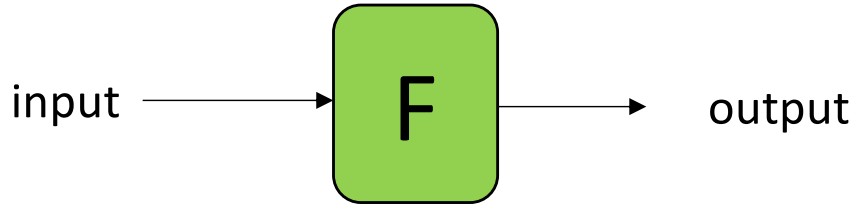
```
module divider (input wire clk_in,
                input wire rst_in,
                input wire[15:0] dividend_in,
                input wire[15:0] divisor_in,
                input wire data_valid_in,
                output logic[15:0] quotient_out,
                output logic[15:0] remainder_out,
                output logic data_valid_out,
                output logic error_out,
                output logic busy_out);
  localparam RESTING = 0;
  localparam DIVIDING = 1;
  logic [15:0] quotient, dividend, divisor;
  logic state;
  always_ff @(posedge clk_in)begin
    if (rst_in)begin
      quotient <= 16'b0;
      dividend <= 16'b0;
      divisor <= 16'b0;
      remainder_out <= 16'b0;
      busy_out <= 1'b0;
      error_out <= 1'b0;
      state <= RESTING;
      data_valid_out <= 1'b0;
    end else begin
      case (state)
        RESTING: begin
          if (data_valid_in)begin
            state <= DIVIDING;
            quotient <= 16'b0;
            dividend <= dividend_in;
            divisor <= divisor_in;
            busy_out <= 1'b1;
            error_out <= 1'b0;
          end
          data_valid_out <= 1'b0;
        end
        DIVIDING: begin
          if (dividend<=0)begin
            state <= RESTING; //similar to return statement
            remainder_out <= divisor;
            quotient_out <= 0;
            busy_out <= 1'b0; //tell outside world i'm done
            error_out <= 1'b0;
            data_valid_out <= 1'b1; //good stuff!
          end else if (divisor==0)begin
            state <= RESTING;
            remainder_out <= 0;
            quotient_out <= 0;
            busy_out <= 1'b0; //tell outside world i'm done
            error_out <= 1'b1; //ERROR
            data_valid_out <= 1'b1; //valid ERROR
          end else if (dividend < divisor) begin
            state <= RESTING;
            remainder_out <= dividend;
            quotient_out <= quotient;
            busy_out <= 1'b0;
            error_out <= 1'b0;
            data_valid_out <= 1'b1; //good stuff!
          end else begin
            //state staying in.
            quotient <= quotient + 1'b1;
            dividend <= dividend-divisor;
          end
        end
      endcase
    end
  end
endmodule
```

Center of Mass Calculation in Lab04A

- You will write a center-of-mass calculator that is best thought of as an FSM.
- For each frame:
  - Sum the x location and y location of every active pixel you come across
  - Keep track of how many pixels you've encountered
  - At end of frame (or beginning of next one, divide the two sums by the number of active pixels
  - This will give an average X,Y
- Division takes time! For lab04a you'll use a 32 bit variant of the division module we saw on Tuesday
- Need to create a major/minor FSM system

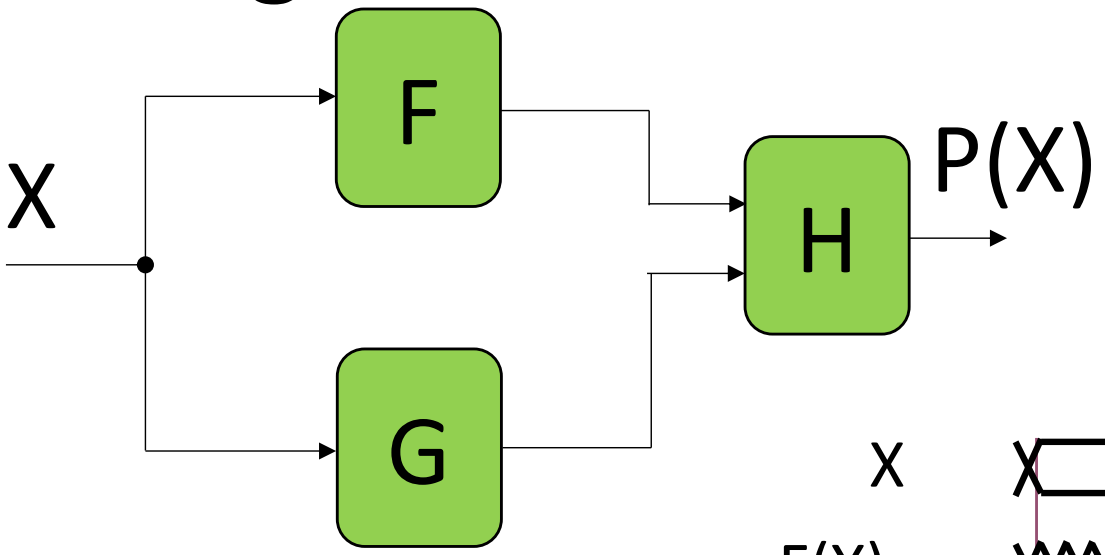
Pipelining

Performance Metrics



- Latency (L):
  - time between arrival of new input and generation of corresponding output.
  - For purely combinational circuits this is just t_{PD}.
- Throughput (T):
  - Rate at which new outputs appear.
  - For purely combinational circuits this is just $1/t_{PD}$ or 1/L.

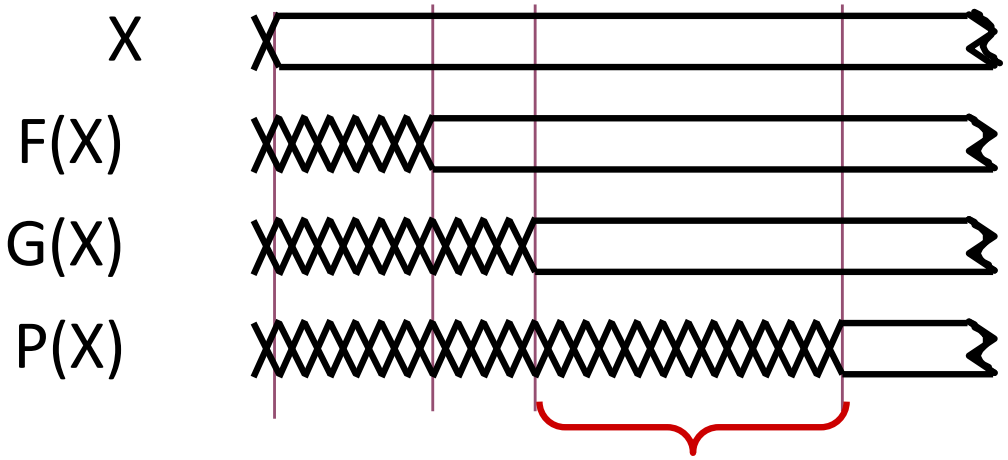
Performance of Combinational Logic



For combinational logic:

$L = t_{PD}$,
$T = 1/t_{PD}$.

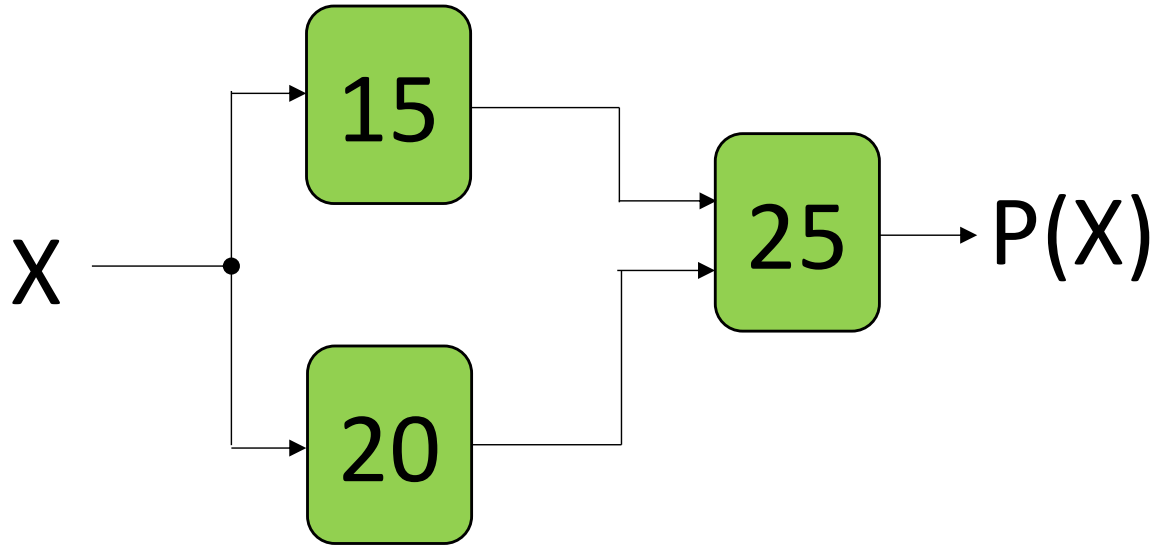
We can't get the answer faster, but are we making effective use of our hardware at all times?



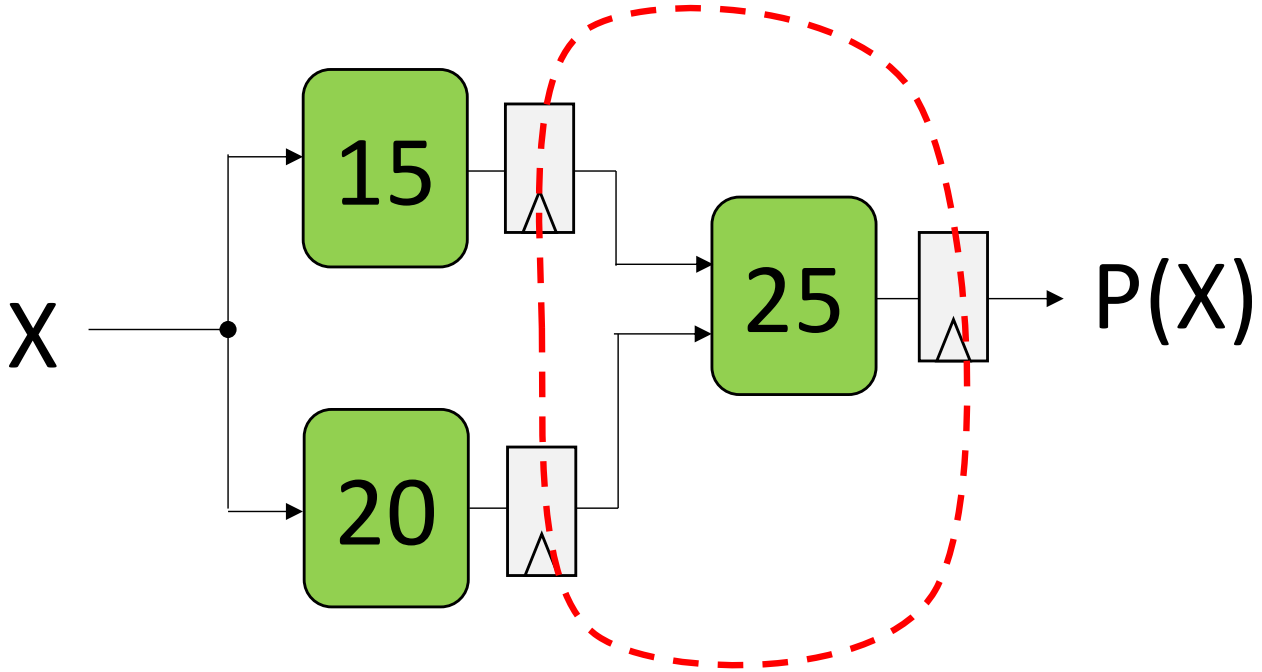
F & G are "idle", just holding their outputs stable while H performs its computation

Retiming: A useful transform

Propagation delays indicated by numbers:

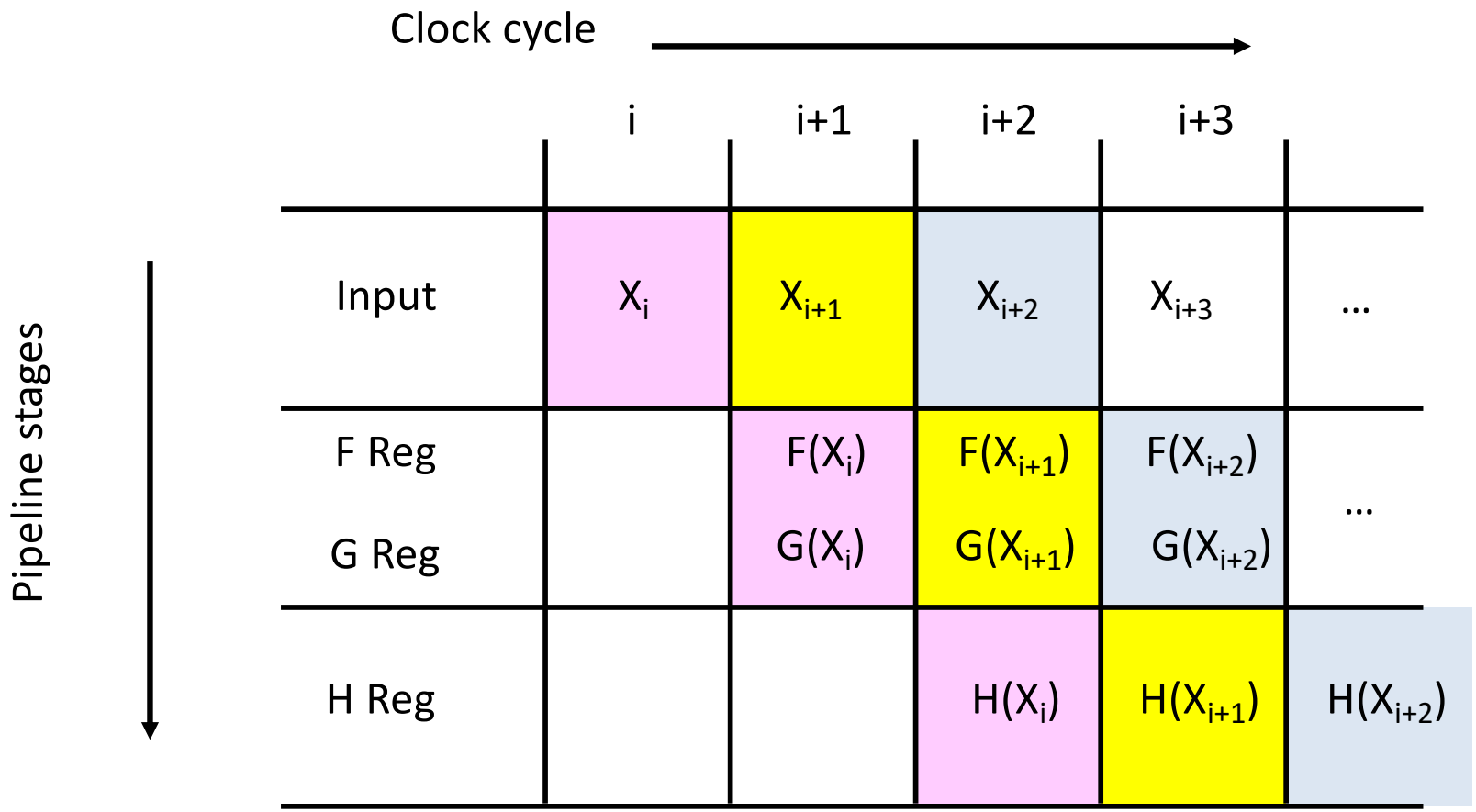


Retiming: A useful transform



Assuming ideal registers:
i.e., $t_{PD} = 0$, $t_{SETUP} = 0$

Pipeline Diagrams



The results associated with a particular set of input data moves diagonally through the diagram, progressing through one pipeline stage each clock cycle.

Pipeline Conventions

- a K-Stage Pipeline ("K-pipeline") is an acyclic circuit having exactly K registers on every path from an input to an output.
- a COMBINATIONAL CIRCUIT is thus a 0-stage pipeline.

Pipeline Conventions

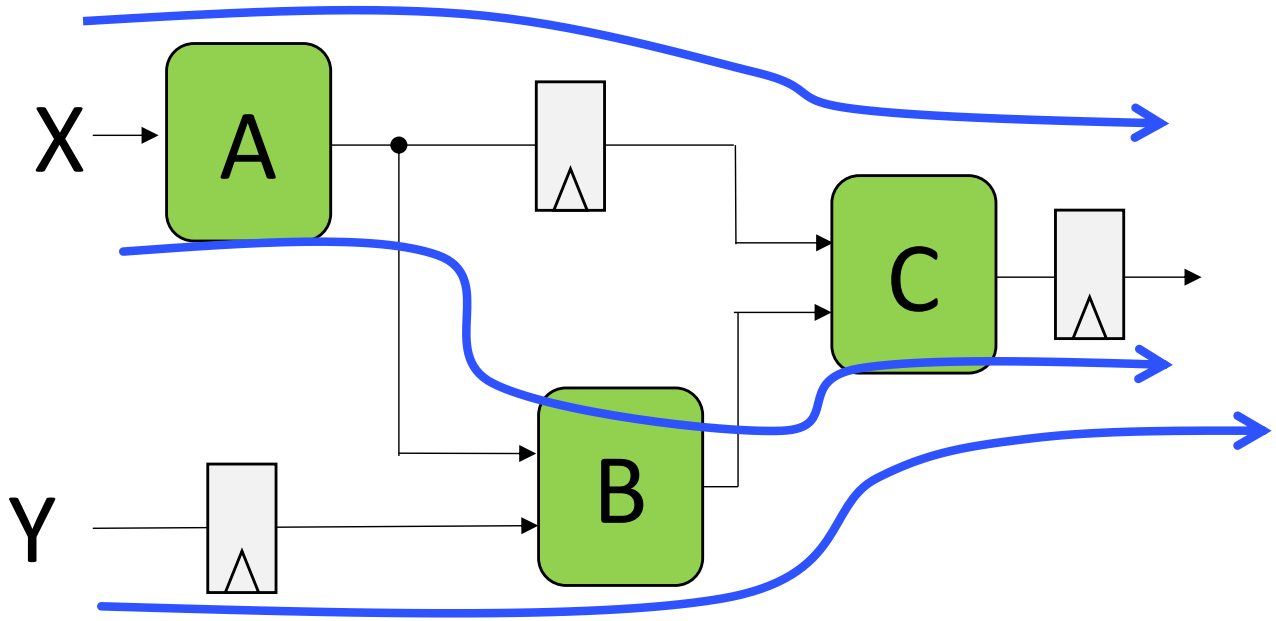
- CONVENTION:
  - Every pipeline stage, hence every K-Stage pipeline, has a register on its OUTPUT (not on its input).
- ALWAYS:
  - The CLOCK common to all registers must have a period sufficient to cover propagation over combinational paths PLUS (input) register t_{PD} PLUS (output) register t_{SETUP}.

$$t_{PD,reg1} + t_{PD,logic} + t_{SETUP,reg2} \leq t_{CLK}$$

The LATENCY of a K-pipeline is K times the period of the clock common to all registers.

The THROUGHPUT of a K-pipeline is the frequency of the clock.

ILL-formed Pipeline



For what value of K is the following circuit a K-Pipeline? ___none___

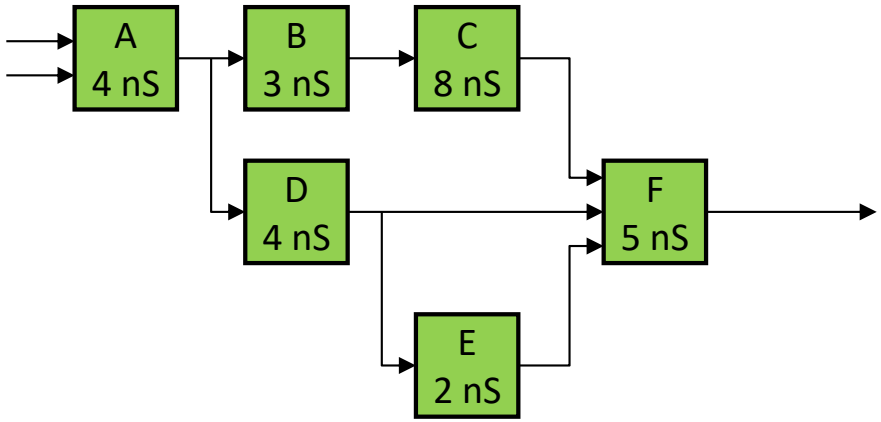
Problem:

Successive inputs get mixed: e.g., $B(A(X_{i+1}), Y_i)$. This happened because some paths from inputs to outputs have 2 registers, and some have only 1!

This CAN'T HAPPEN on a well-formed K pipeline!

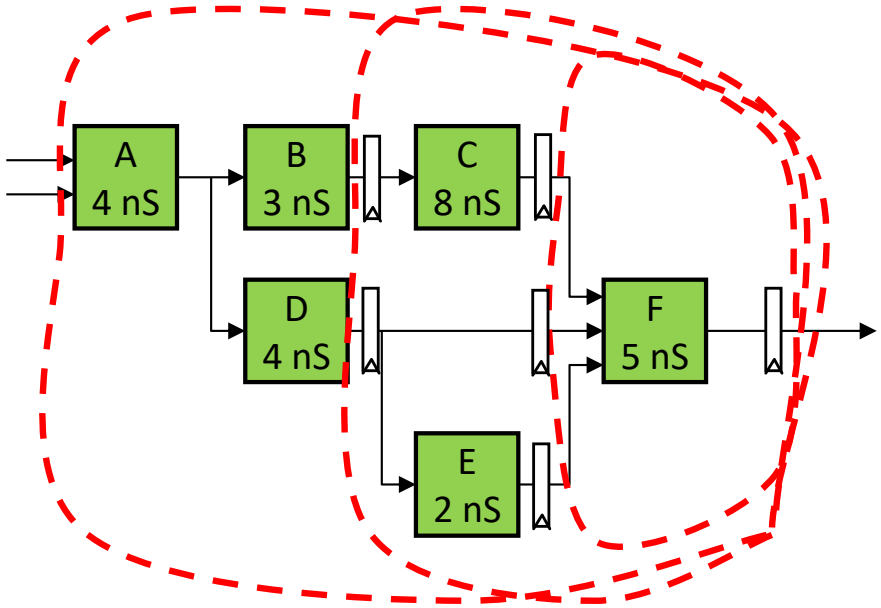
Pipelining

Let's say we want t_{clk} to be 8ns



Another Thing to Pipeline

Let's say we want t_{clk} to be 8ns



Step 1:

Add a register on the output.

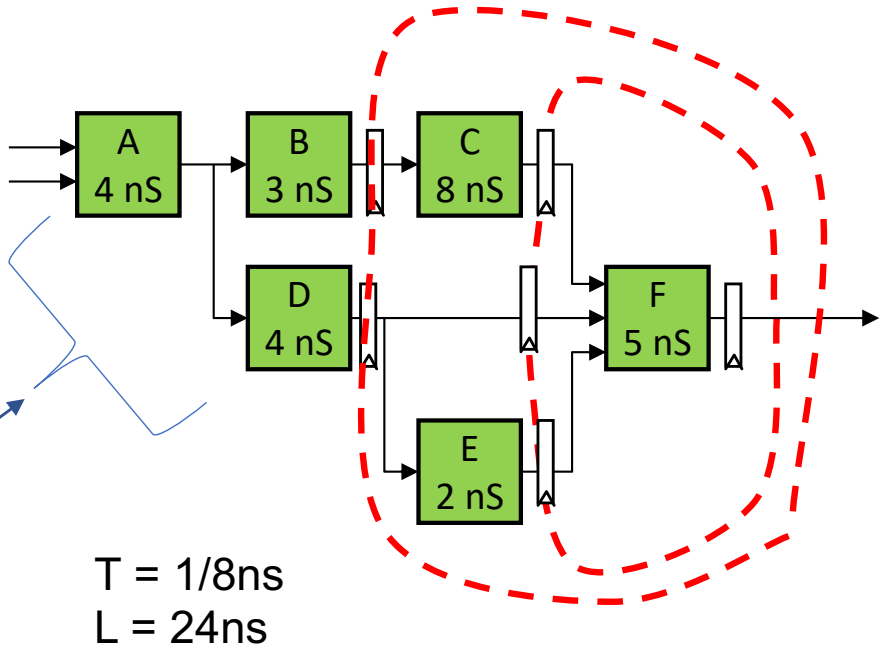
Step 2:

From register. Draw a contour backwards that includes as much of the circuit that will fit inside required period. Add registers

Repeat until satisfied with T. Look for redundant registers

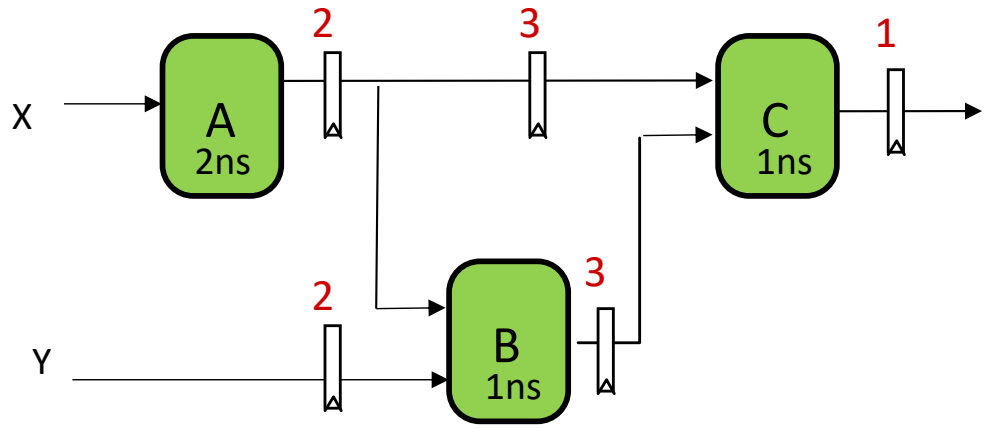
STRATEGY:

Focus your attention on placing pipelining registers around the slowest circuit elements (BOTTLENECKS).



Assuming this interfaces with other modules that have registered outputs the input will chain will be ok (<= 8ns)

Another Pipeline Example



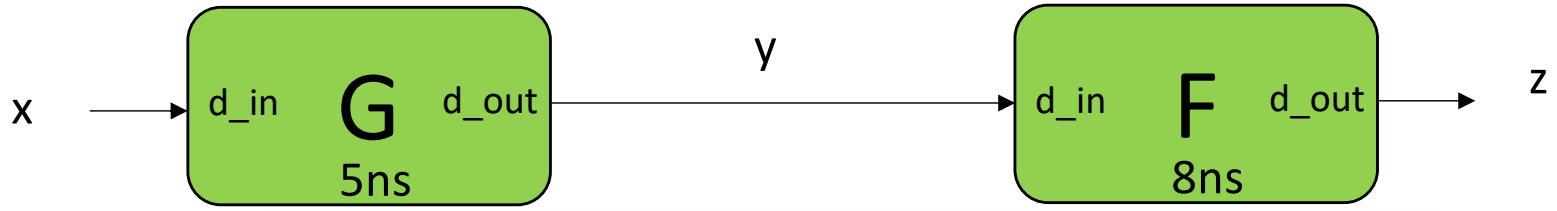
	LATENCY	THROUGHPUT
0-pipe:	4	1/4
1-pipe:	4	1/4
2-pipe:	4	1/2
3-pipe:	6	1/2

OBSERVATIONS:

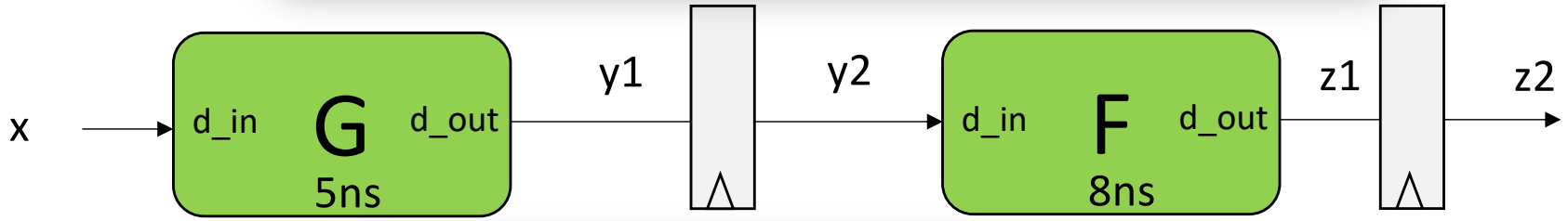
- 1-pipeline improves neither L or T.
- T improved by breaking long combinational paths, allowing faster clock.
- Too many stages cost L, don't improve T.
- Back-to-back registers are often required to keep pipeline well-formed.

Better throughput here means we can run at higher clock rate

Pipelining in Verilog



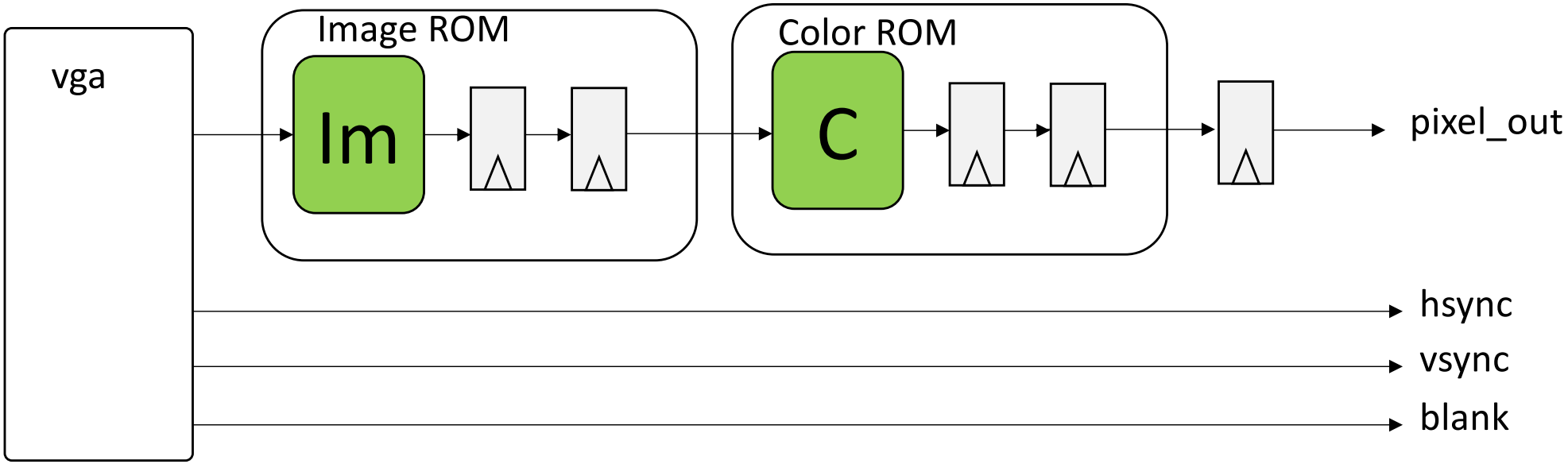

```
1
2  logic x,y,z;
3  //G and F are purely combinational modules.
4  G myg (.d_in(x), .d_out(y));
5  F myf (.d_in(y), .d_out(z));
```



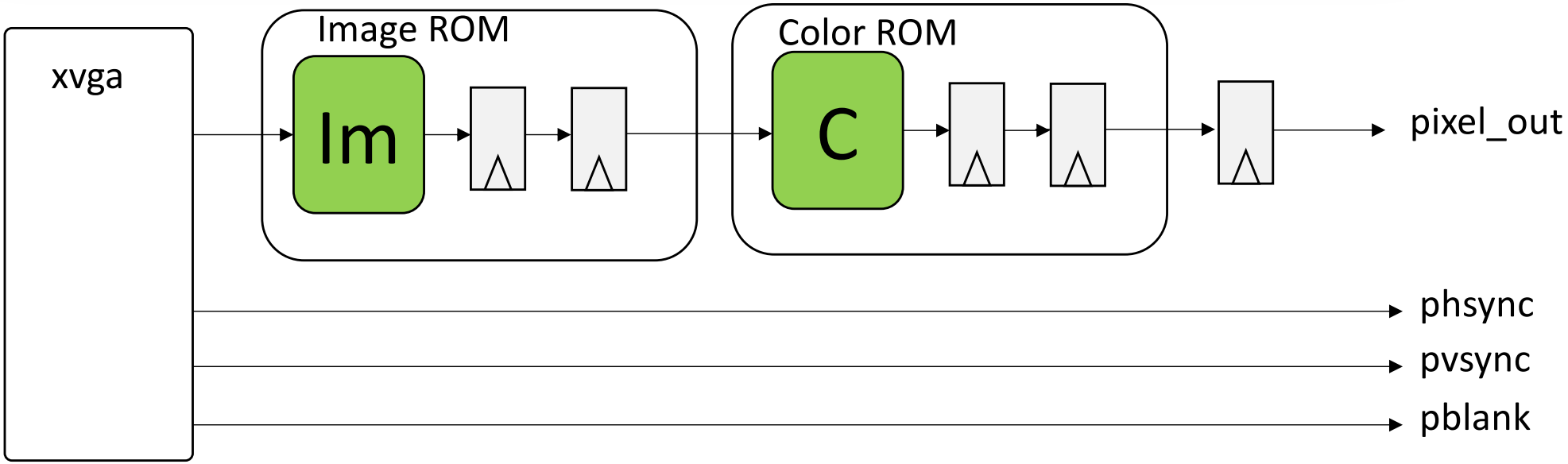

```
1
2  logic x,y1,y2,z1,z2;
3  //G and F are purely combinational modules.
4  G myg (.d_in(x), .d_out(y1));
5  F myf (.d_in(y2), .d_out(z1));
6
7  always_ff @(posedge clk)begin
8   y2 <= y1;
9   z2 <= z1;
10 end
11
```

(Two registers coming from delay in memory access/read)

- Monitor drawing based on vsync, hsync, blank,
- But what image rom is giving it is 5 clock cycles behind
- At start of Death Star nothing in the “pipeline” yet



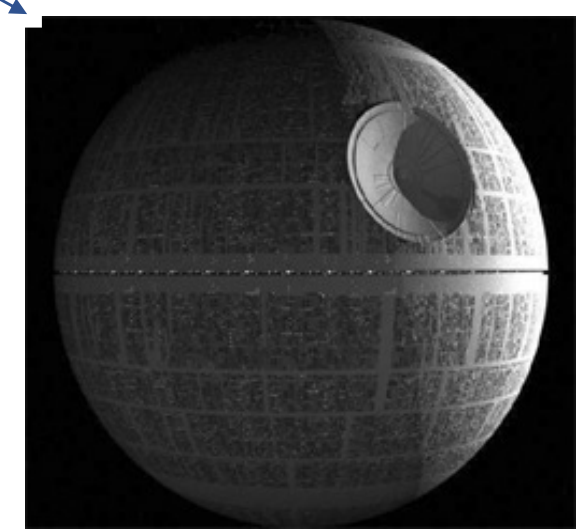
```
1
2  // calculate rom address and read the location
3  assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
4  image_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
5
6  red_coe rcm (.clka(pixel_clk_in), .addra(image_bits), .douta(red_mapped));
7
8  always_ff @(posedge pixel_clk) begin
9    if ((hcount_in >= x && hcount_in < (x_in+WIDTH)) &&
10          (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin
11      pixel_out <= {red_mapped[7:4], red_mapped[7:4], red_mapped[7:4]}; // greyscale
12   end else begin
13      pixel_out <= 0;
14   end
15 end
16
```



Pipelining is the issue!

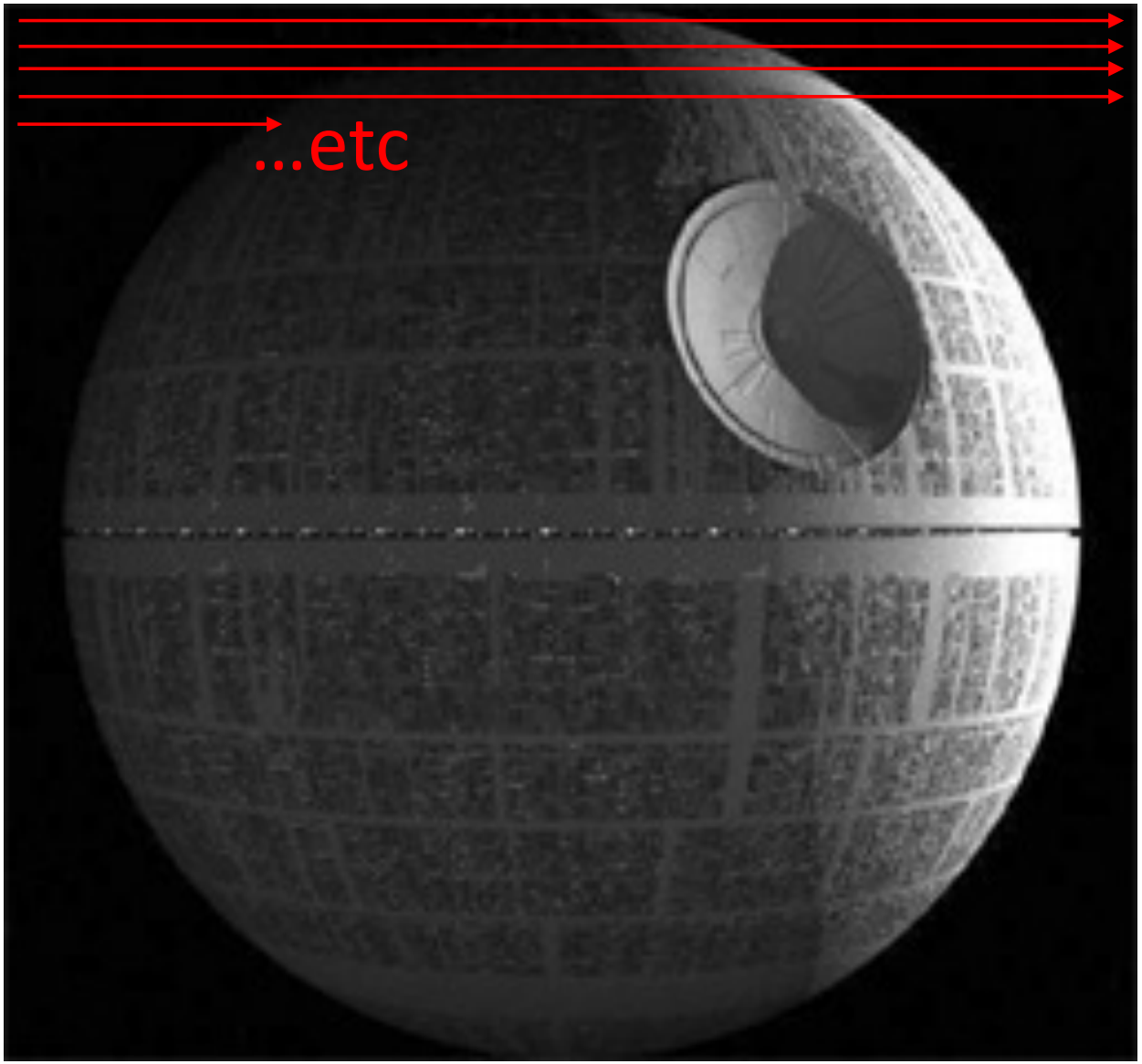
- Monitor drawing based on vsync, hsync, blank,
- But what image rom is giving it is 5 clock cycles behind
- At start of Death Star nothing in the “pipeline” yet

The white blip



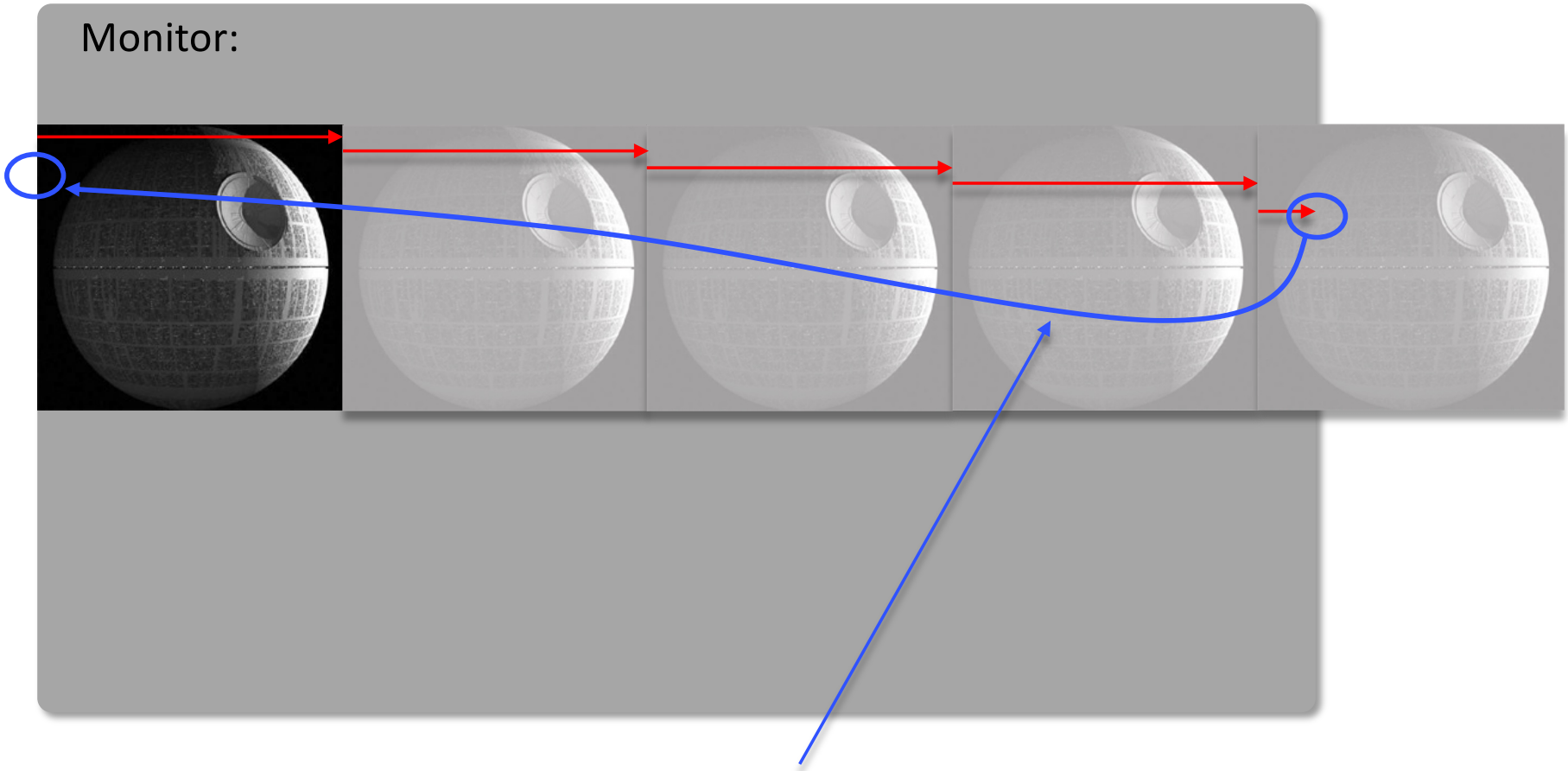
Also…

- Remember that image ROMS are really just 2D data stored in 1D

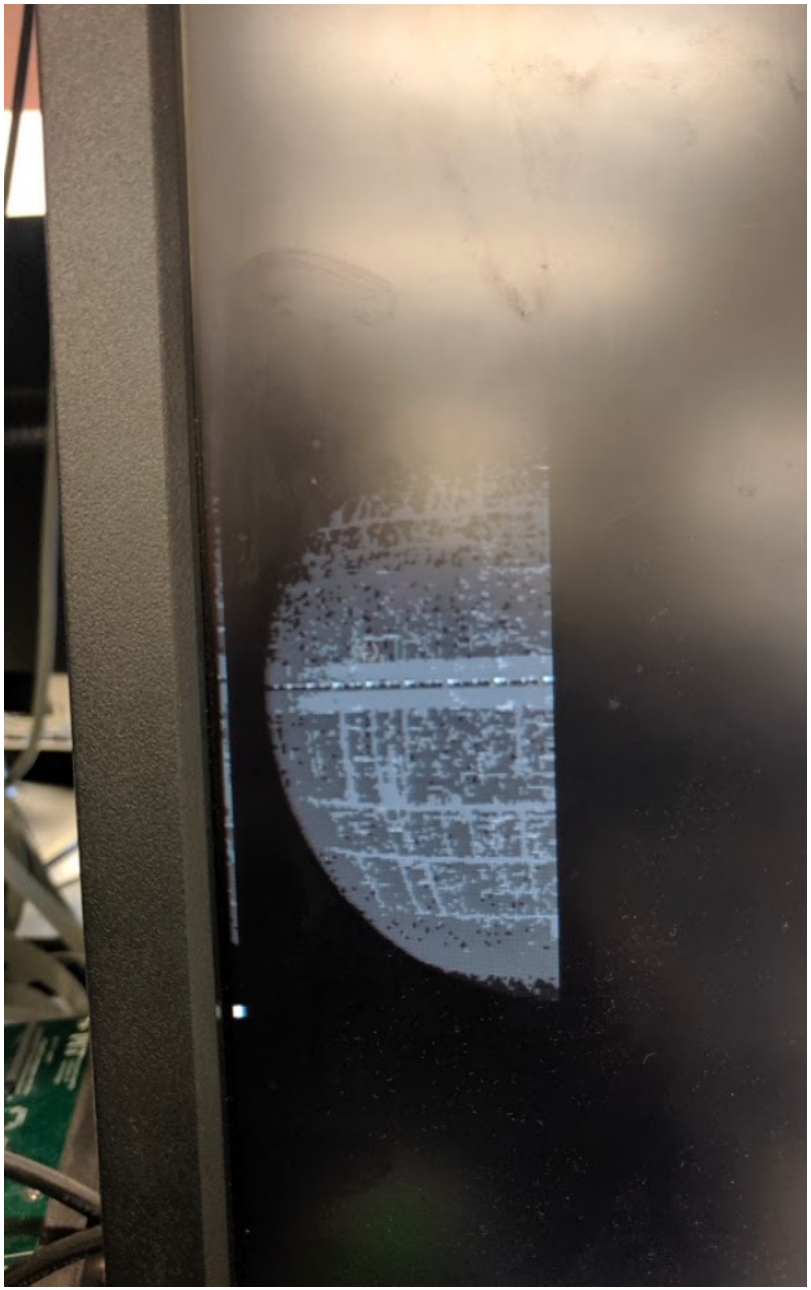


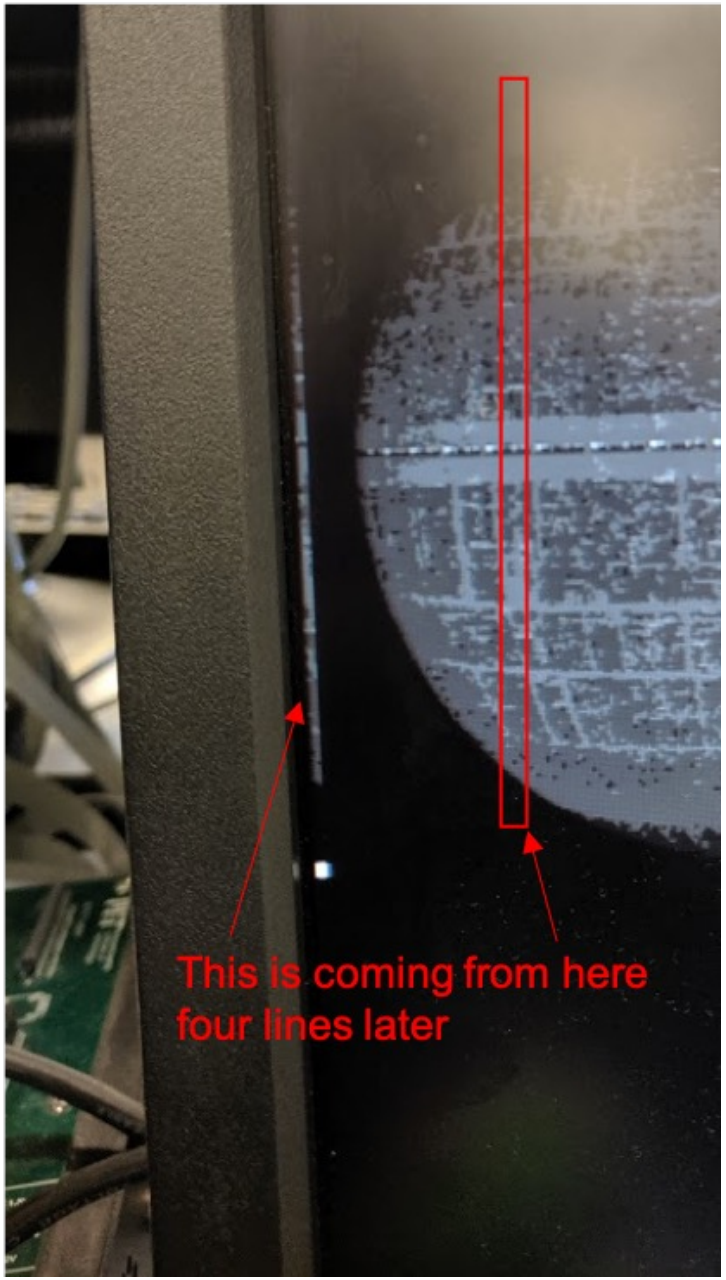
Also…

Keeps reading further into memory as draw pixel moves across screen:



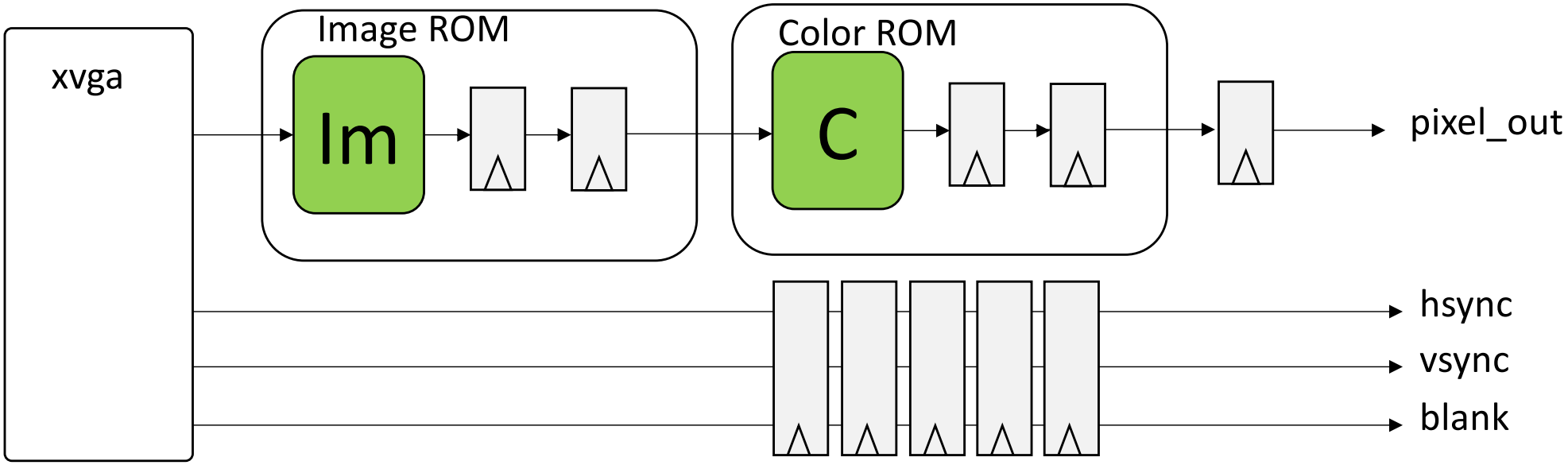
Move from here to here immediately…there's five pixels of data from here in the image already in the pipeline! Damnit!




This is coming from here
four lines later

How to Fix?

- Delay the other signals so everybody is the same



Turn the whole thing into a 5-stage pipeline!

Pipelining

- Pipeline in Verilog!
- Make sure other things are protected too!

```
logic hs_pip[4:0];
logic vs_pip[4:0];
logic b_pip[4:0];

always_ff@(posedge clk_in)begin
  hs_pip[0] <= hsync_in;
  vs_pip[0] <= vsync_in;
  b_pip[0] <= blank_in;
  for (int i=1; i<5; i = i+1)begin
    hs_pip[i] <= hs_pip[i-1];
    vs_pip[i] <= vs_pip[i-1];
    b_pip[i] <= b_pip[i-1];
  end
end
assign hsync_out = hs_pip[4];
assign vsync_out = vs_pip[4];
assign blank_out = b_pip[4];
```

DigiEyes

Live Pong