Welcome to The Hardware Lab!

Fall 2021
Final Exam Review

Prof. Chun-Yi Lee

Department of Computer Science
National Tsing Hua University

Agenda

- **Announcement**
- **Final Exam Review**

Today's class will help you:

1. Review the topics that we have discussed in the past few weeks

Announcements

- Lab 6
  - Lab 6 basic question demonstration on **12/2/2021 (Thu)**
  - Lab 6 advanced question demonstration on **12/16/2021 (Thu)**
  - **Please read the lab description and specification carefully**
  - **Compile your codes again before merging yours with your teammates**
  - **Please follow the template I/Os and submit your .v files**
- Final project
  - You can begin working on your final projects from now
  - Please avoid copying the works from past years

Final Project

- Use your FPGA to implement creative and interesting works

- Final project proposal due on **12/13/2021 (Mon)**
- Final project report due on **1/14/2022 (Fri)**

- Final exam
  - **12/9/2021 (Thu)**
  - 2 hours

Final Project Presentation

- Final project demonstration on **1/13/2022 (Thu)** and **1/14/2022 (Fri)**
  - Around 10 minutes per team

- **No restriction on your presentation style**
  - Be creative!
  - What is special and new in your project
  - Key features

- Order of presentation
  - We will randomly decide the order
  - Let us know if you have constraints

Final Project Report

- Final project report
  - Due on **1/14/2022, 23:59pm (Fri)**
  - Block diagrams and state transition diagrams
  - **Detailed explanation**

- Report contents
  - Introduction
  - Motivation
  - System specification
  - Experimental results
  - Conclusion
  - …And any other sections that you would like to include

Final Project Award

- **Category:** Difficulty and completeness
  - Graded by me and the TAs
    - Difficulty (**35%**)
    - Completeness (**30%**)
    - Source code coding style (**15%**) (Correctness, usage, comments, etc.)
    - Report (**20%**)
  - First place: **NTD $6,000**
  - Second place: **NTD $3,000**
  - Third place: **NTD $1,500**

- **Category:** Best creativity
  - Voted by everyone in the class
  - **NTD $1,000**

- **Category:** Deep learning models in FPGA
  - Extra **bonus points** and **cash rewards**. To be announced

Final Exam

- Dates and time
  - **12/9/2021 (Thu)**
  - **3:30pm ~ 5:30pm (2 hours)**

- Course contents
  - Verilog design questions
  - Logic design concepts
  - Lecture & lab contents

Agenda

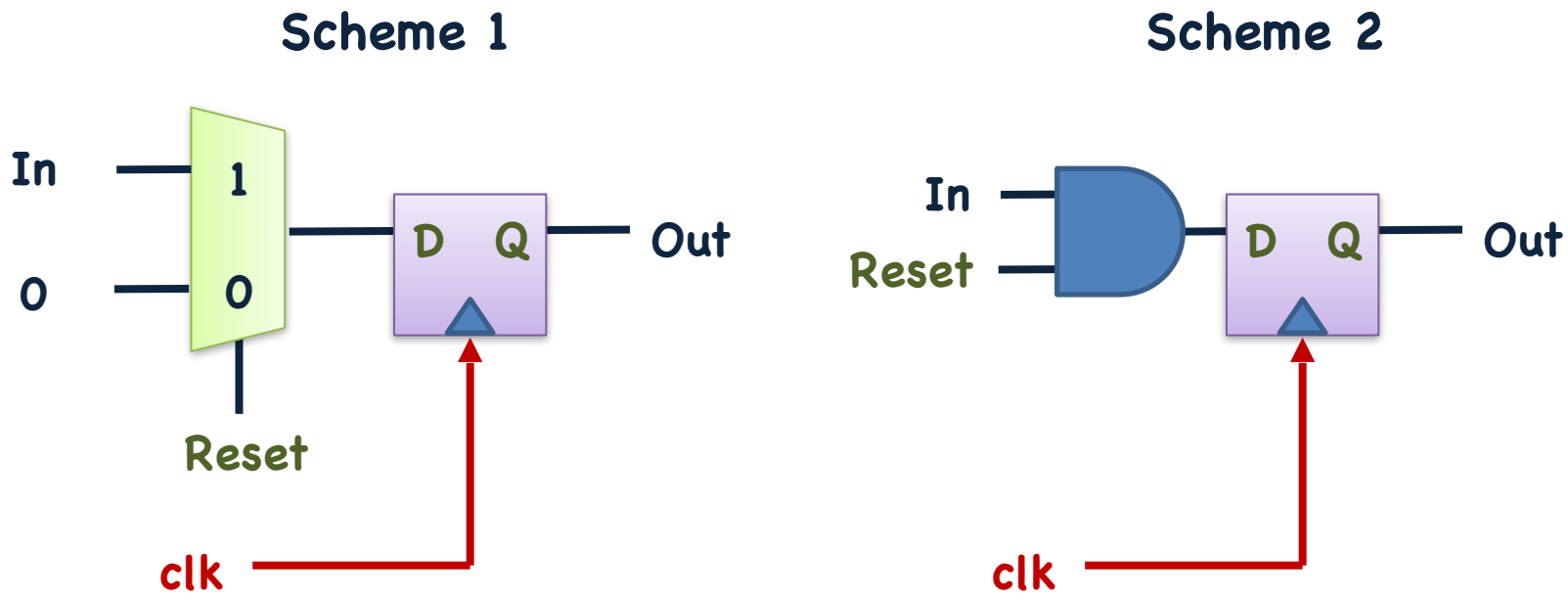
- Announcement
- Final Exam Review

Today's class will help you:

1. Review the topics that we have discussed in the past few weeks

Synchronous Reset

- Reset the value stored in a D Flip-Flop to 0
- Triggered by clock edges
- Assume that **Q = 1'b0** when Reset is set to **1'b0**

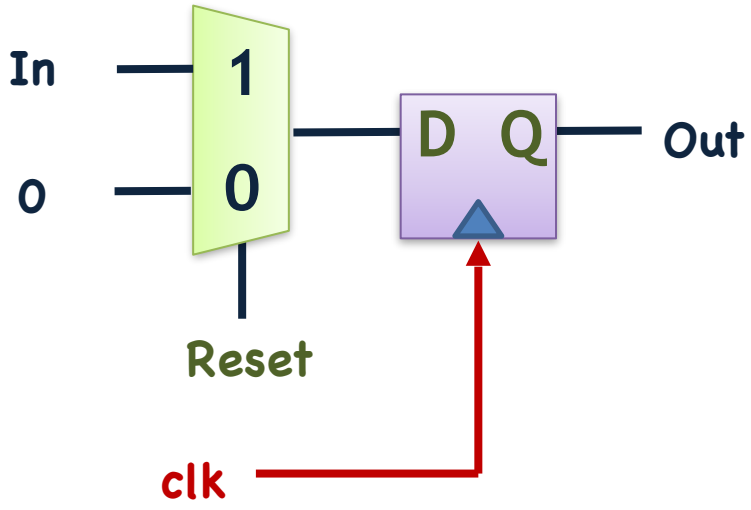


Synchronous Reset in Always Block

- **Prioritize** your reset signal over the rest of the inputs
- In your testbench, **initialize your reset signal first**
- You **shall not** use any initial block in your design modules

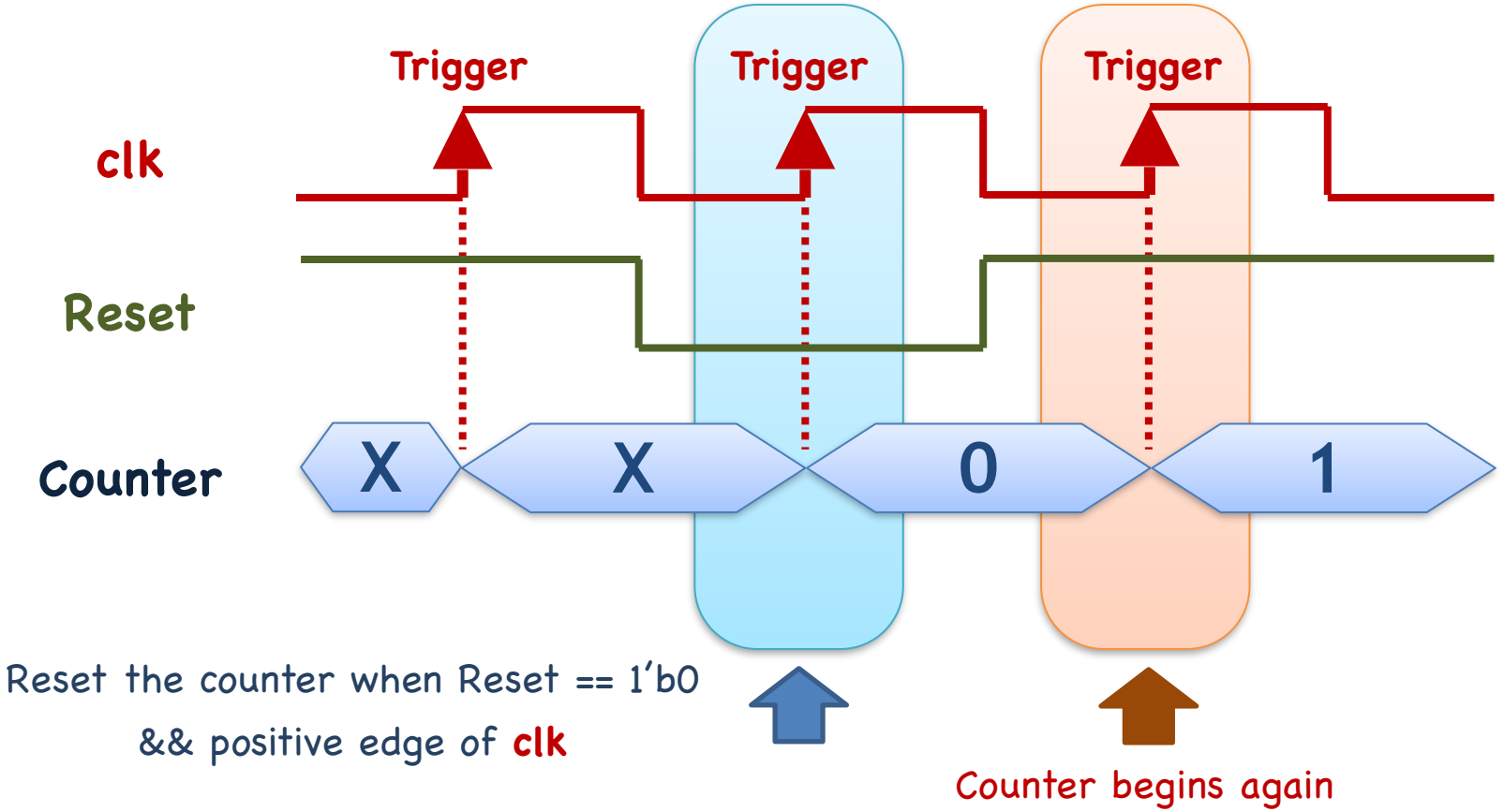
```
module My_Test_Flip_Flop (Out, In, clk, Reset);
  input    In, clk, Reset;
  output   Out;
  reg      Out;

always @ (posedge clk)
  begin
    if (Reset == 1'b0)      Out <= 1'b0;
    else                    Out <= In;
  end
endmodule
```

Declaration of Synchronous Behavior



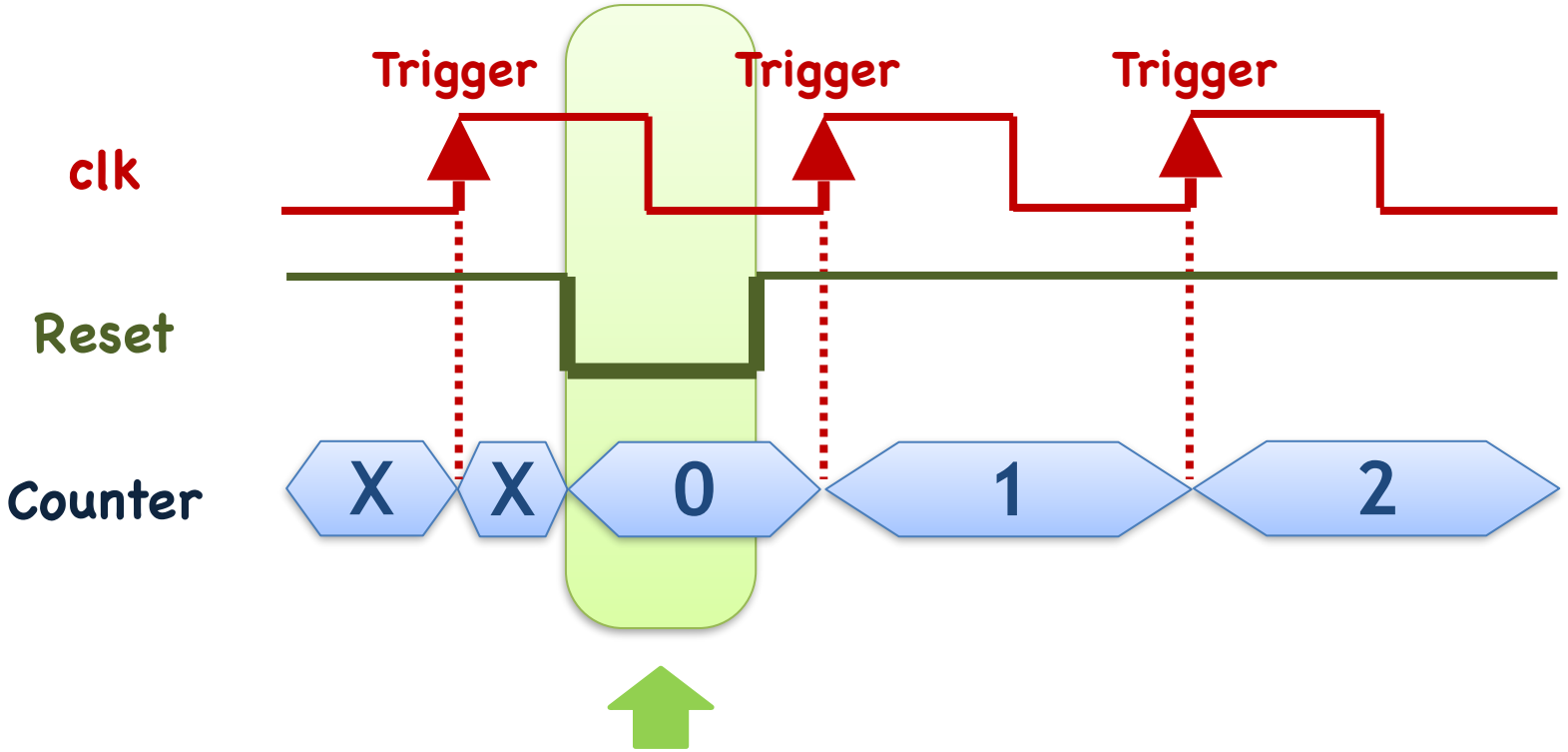
Synchronous Reset in Waveform

- Reset the flip-flops at **clock edges**



Asynchronous Reset in Waveform

- Reset the flip-flops **ANY TIME**
- Concept similar to **Latches**



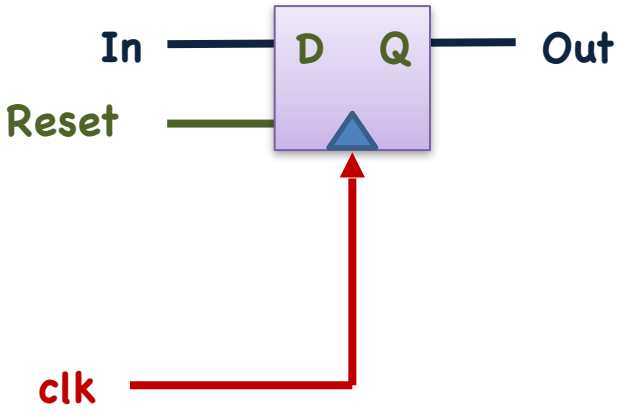
Asynchronous Reset in Always Block

- Add "**negedge Reset**" or "**posedge Reset**" in always statements
- Reset the flip-flops regardless of **clk**
- **Reset** signal gets the **highest priority**

```
module My_Test_Flip_Flop (Out, In, clk, Reset);
  input    In, clk, Reset;
  output   Out;
  reg      Out;

always @ (posedge clk or negedge Reset)
  begin
    if (Reset == 1'b0)        Out <= 1'b0;
    else                      Out <= In;
  end
endmodule
```

Declaration of Asynchronous Behavior

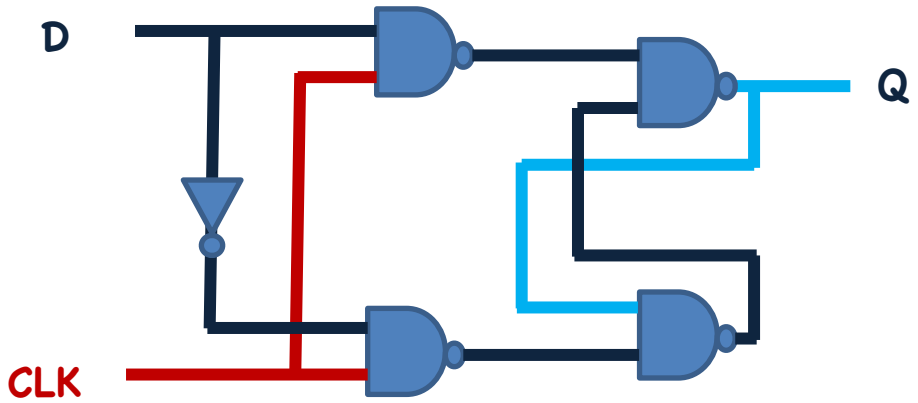
D flip-flop has an additional Reset input port



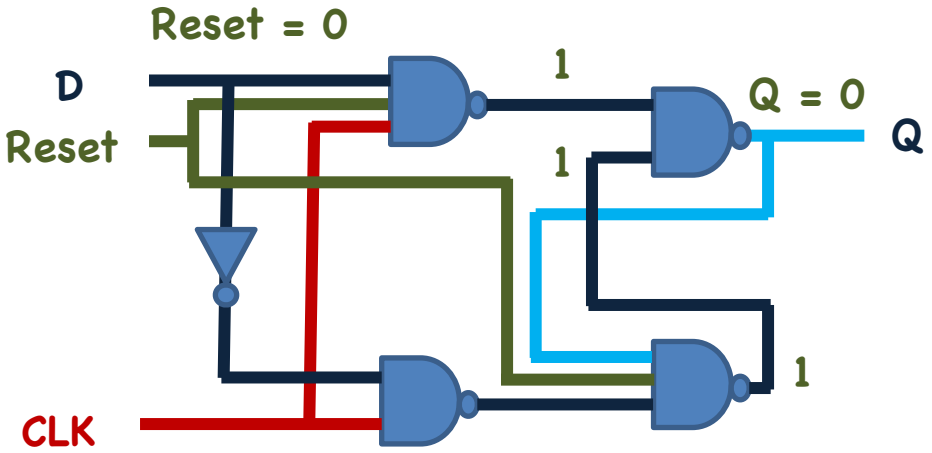
Out is reset to 0 when Reset goes down to zero

Asynchronous Reset in Latches

Latch in Lab 2

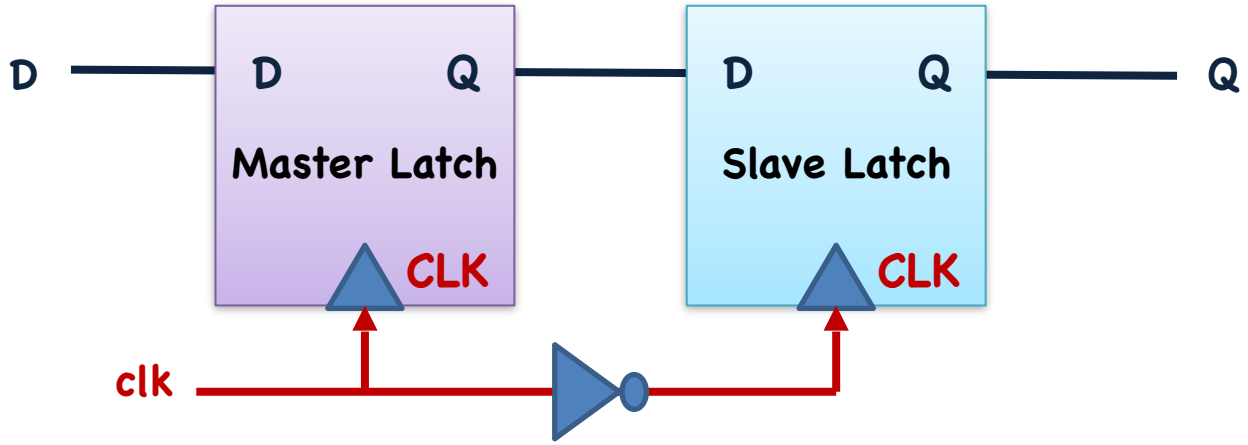


Latch with Asynchronous Reset

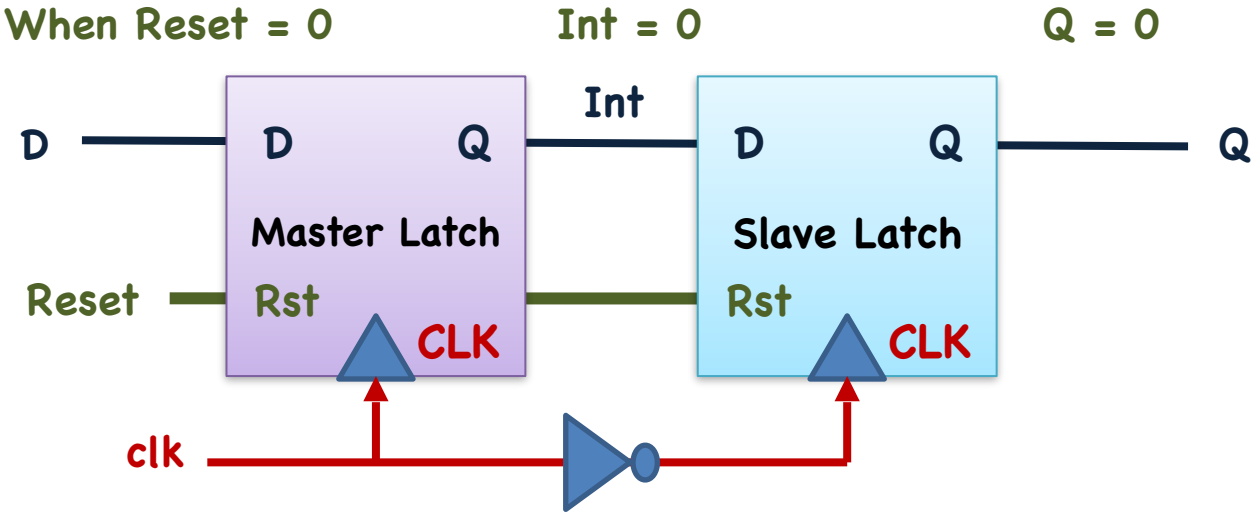


Asynchronous Reset in DFFs

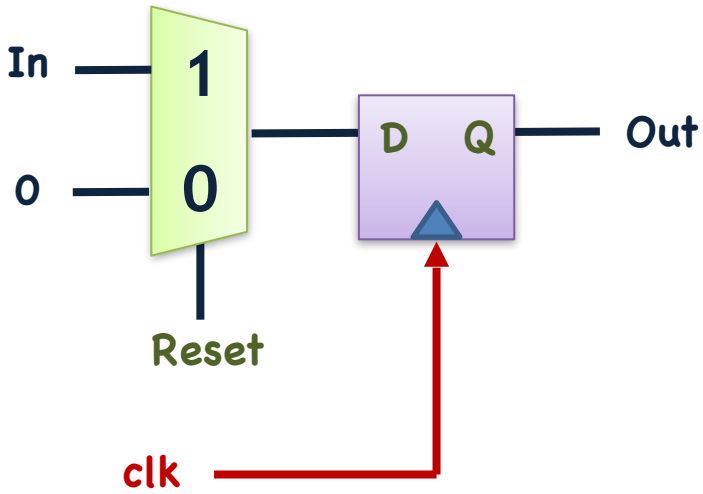
DFF in Lab 2 (Synchronous)

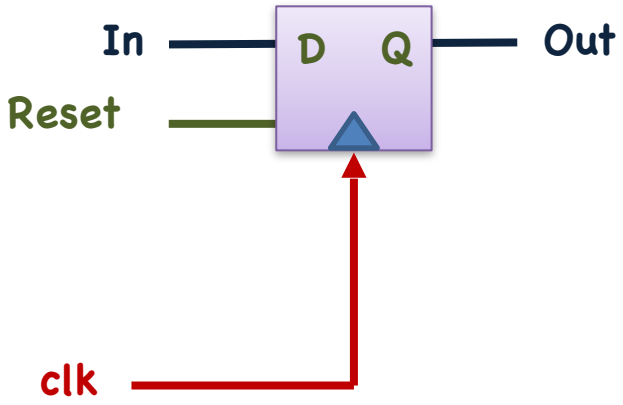


DFF with Asynchronous Reset



Synchronous vs. Asynchronous Reset



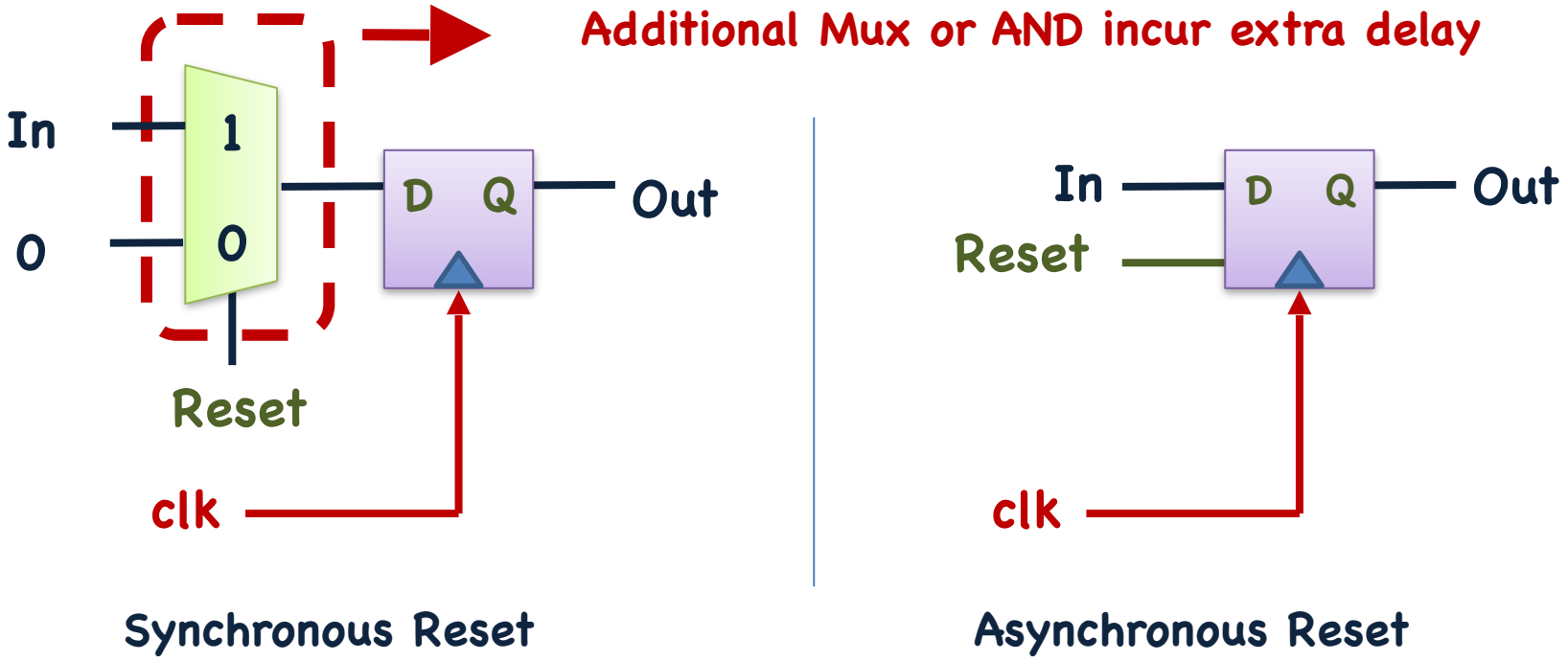


Synchronous Reset | **Asynchronous Reset**

	Synchronous Reset	Asynchronous Reset
Clock	Reset at clock edges	Regardless of clock
Always block	Always @ (posedge clk)	Always @ (posedge clk or **negedge Reset**)
Logic	Additional Mux or AND gate	Additional reset input port in DFF

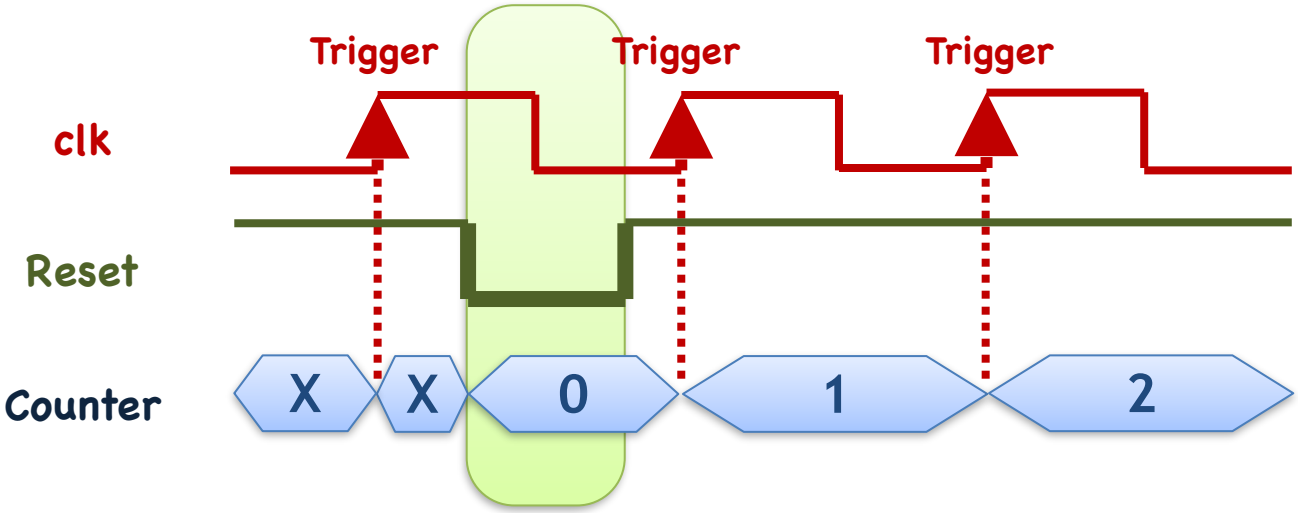
Advantages of Asynchronous Reset

- Fast
  - No additional Mux or AND
- Reset **ANYTIME**, no clock signal is required



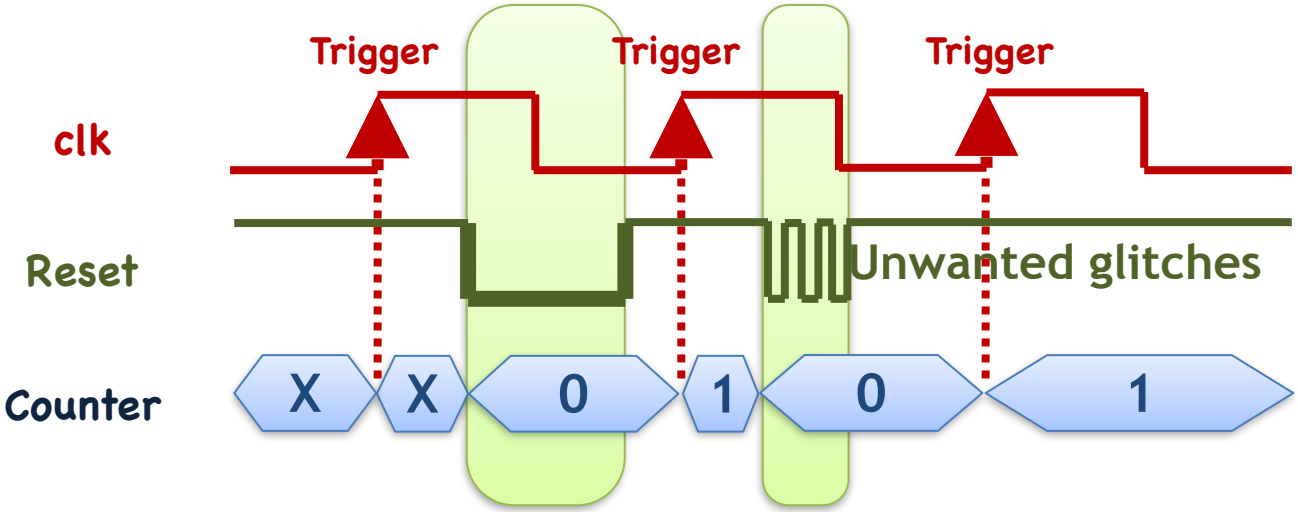
Disadvantages of Asynchronous Reset

Ideal



In reality

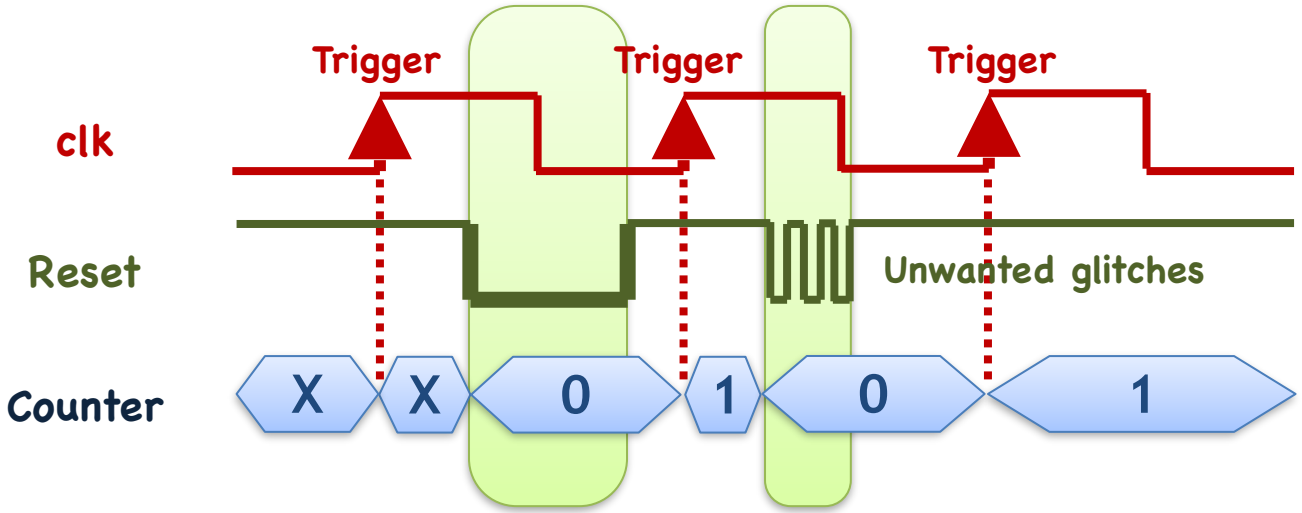
Asynchronous Reset is vulnerable to glitches



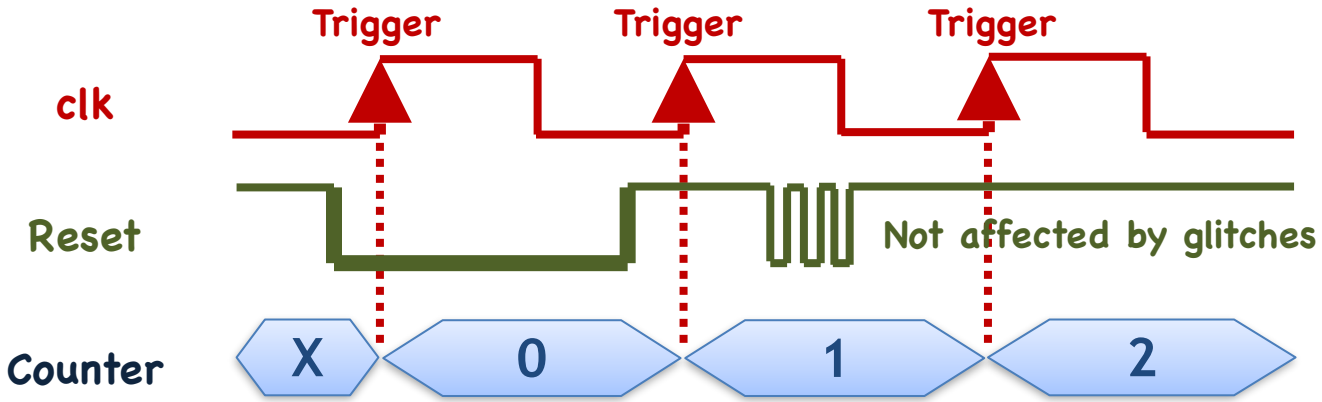
Advantages of Synchronous Reset

- Synchronous Reset is not affected by glitches
  - Unless glitches occur right at clock edges

Asynchronous Reset



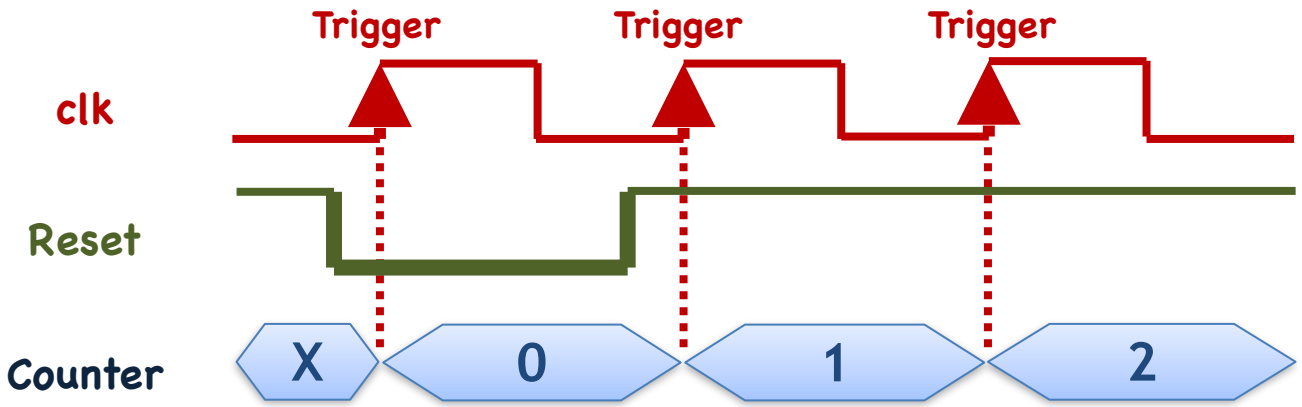
Synchronous Reset



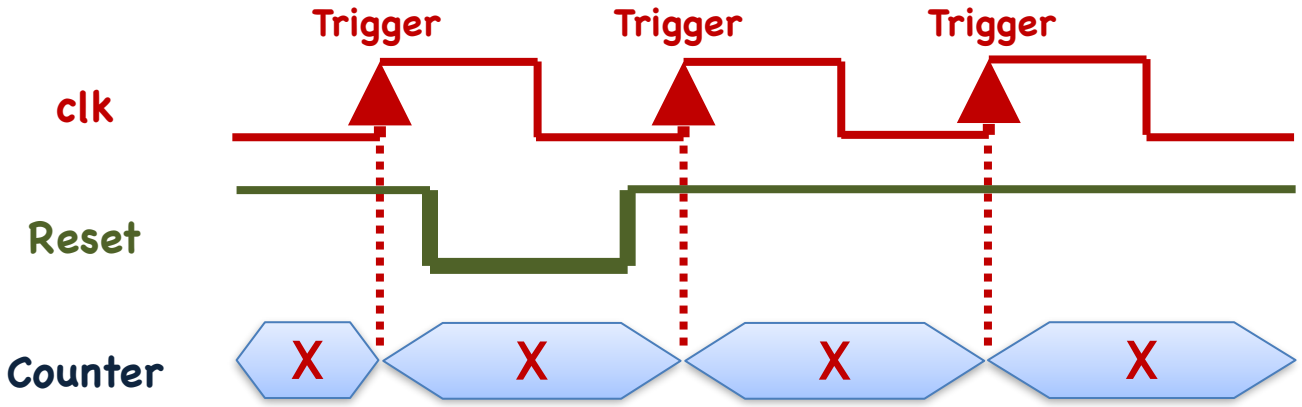
Disadvantages of Synchronous Reset

- Slower
- If Reset signal is too short, no DFFs are reset

Reset is long enough



Reset is too short



Summary of Pros and Cons

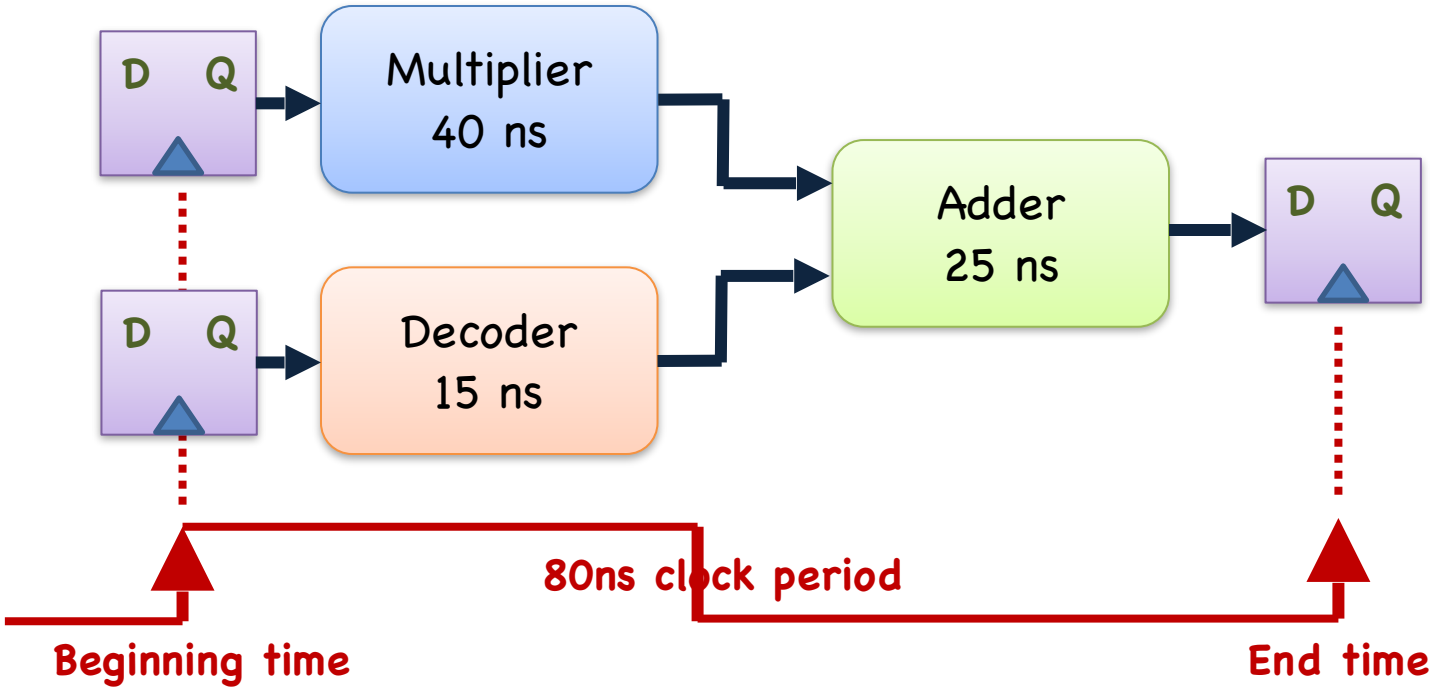
- Here we summarize the previous slides

Reset type	Advantages	Disadvantages
Synchronous	Not affected by glitches, Completely synchronous circuit	Slow, Reset cannot be too short
Asynchronous	Fast, Reset anytime	Vulnerable to glitches, Larger DFFs

Clock Signal for Logic Blocks

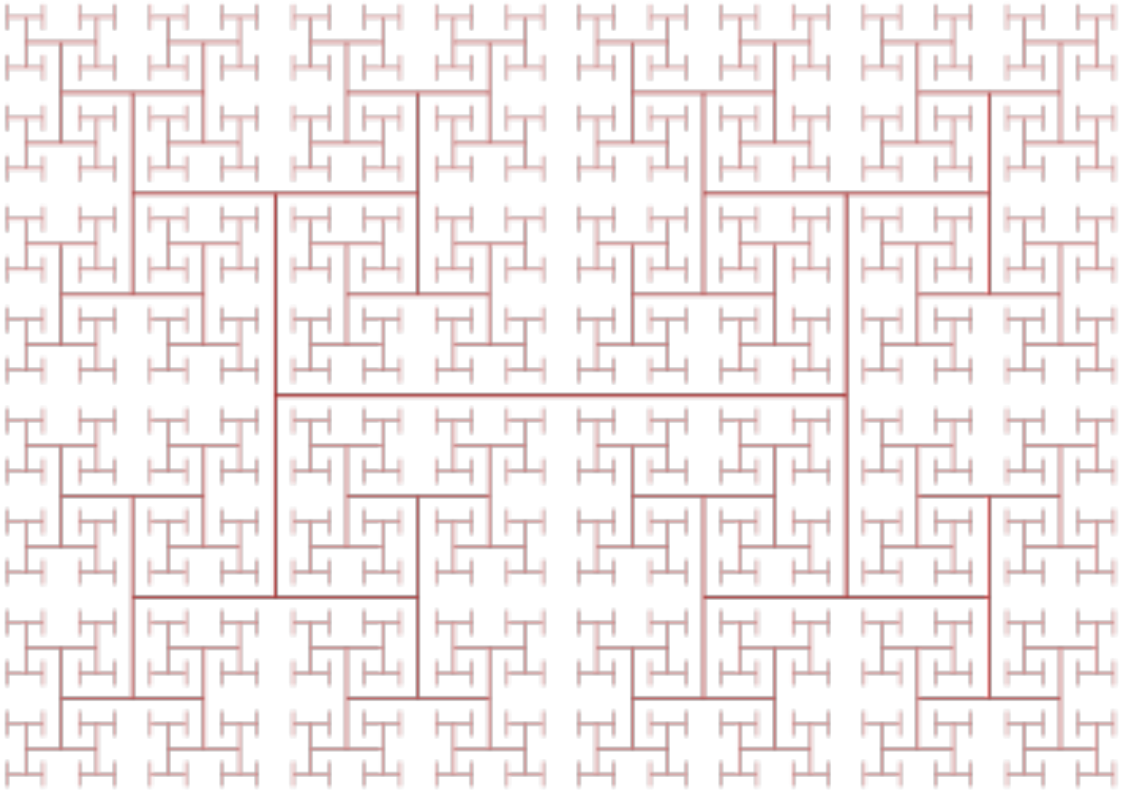
- Clock signals and DFFs are used to synchronize the beginning and end time of a computation

- Ensure the progresses of all the blocks
  - Similar to the concept of our beginning date and deadline of homework



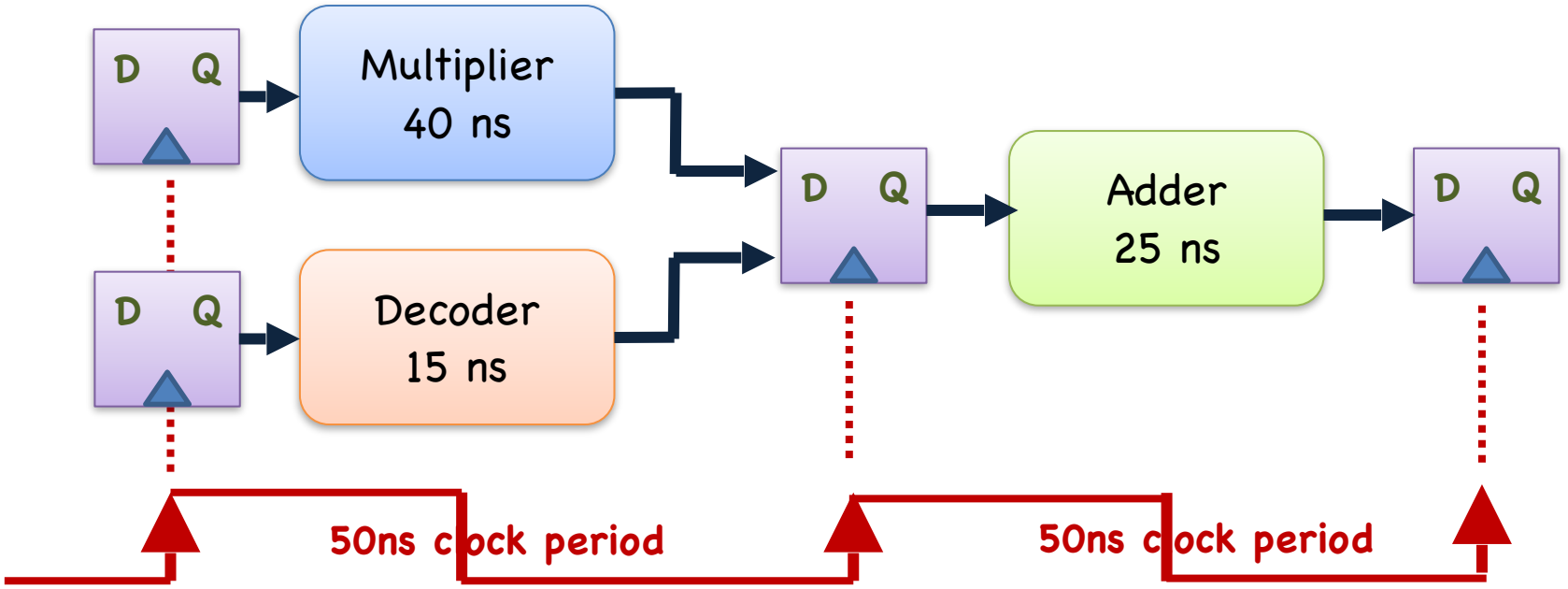
Clock Signal Distribution

- Clock signals are distributed all over the chip
- Different ways of distribution are possible



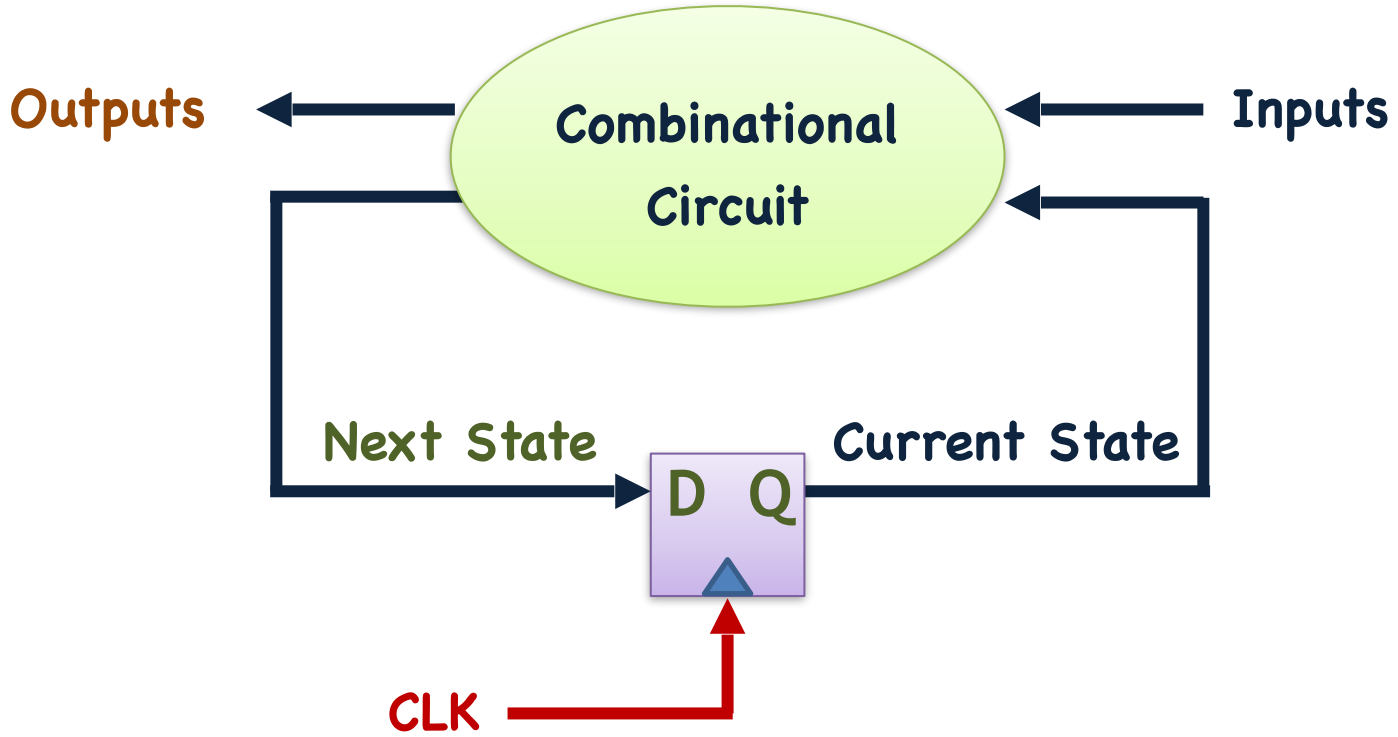
Deal with Faster Clock Requirements

- Divide your designs, and insert more pipeline stages
- Simply and redesign your logic
- Use faster devices



Finite State Machines

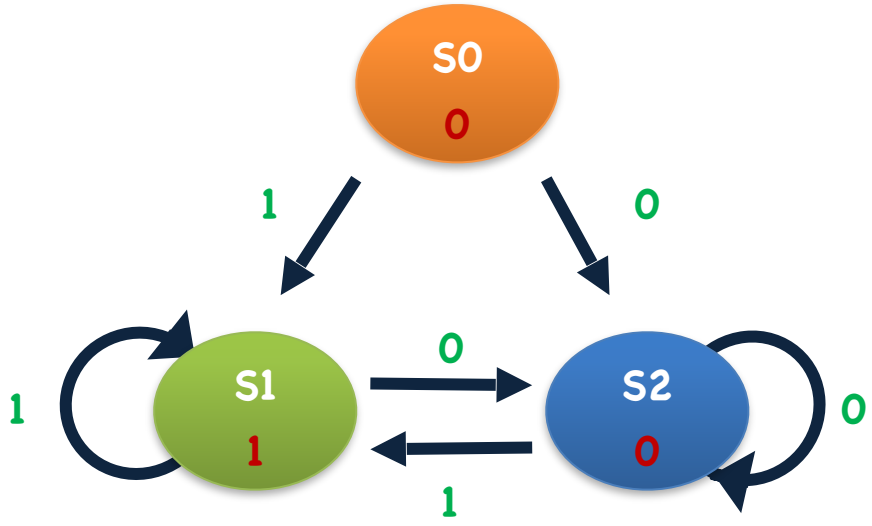
- Useful abstraction for **sequential circuits** with "**states**" of operation
- At **CLK** edge, combinational circuit computes **outputs** and **next state** as a function of **inputs** or **current states**



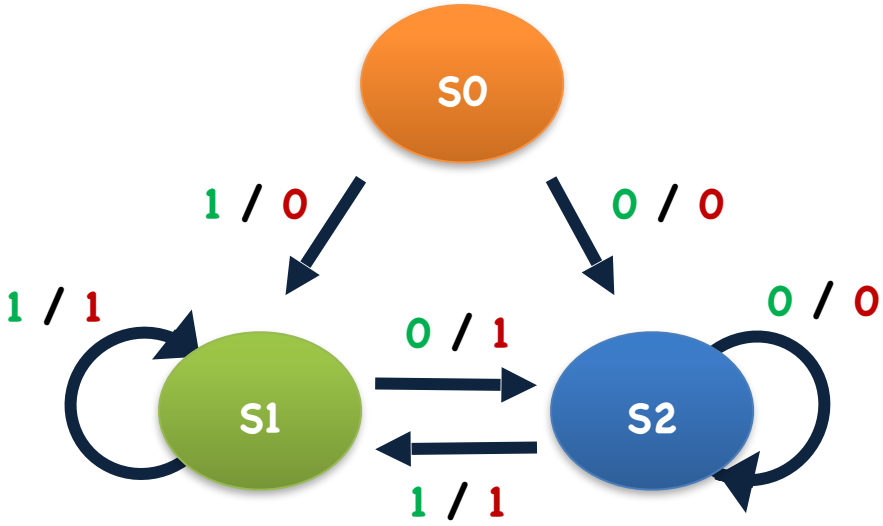
Moore vs. Mealy Machines

- **Moore machine:** **outputs** depend on **current states** only
- **Mealy machine:** **outputs** depend on **current states** and **inputs**

Moore machine

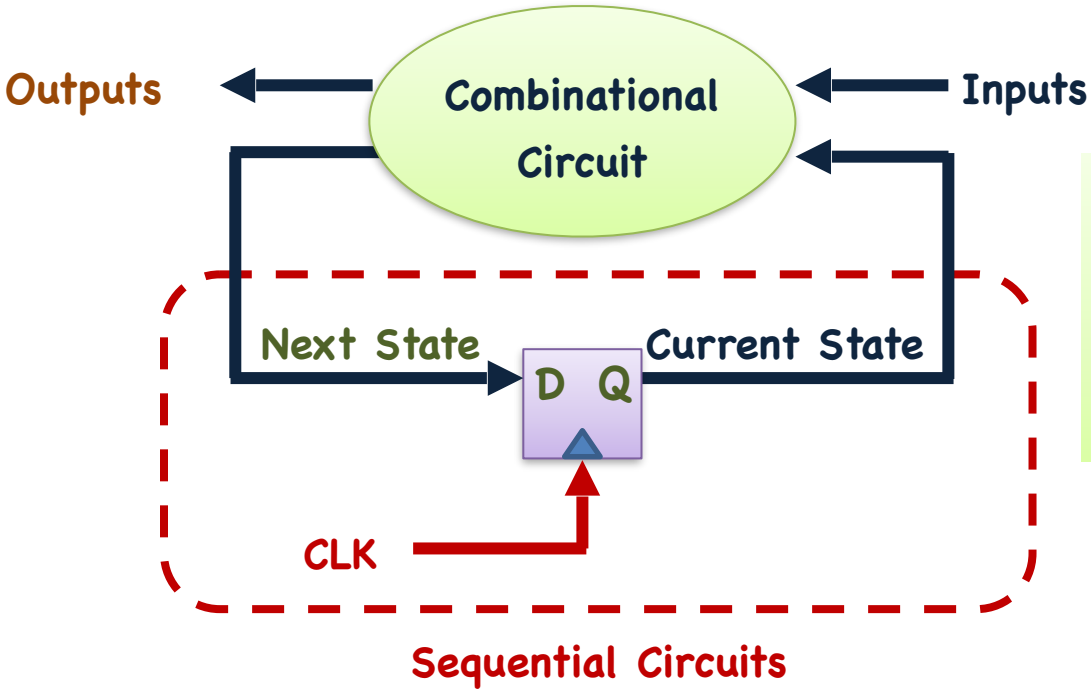


Mealy machine



Always Blocks for Sequential Circuits

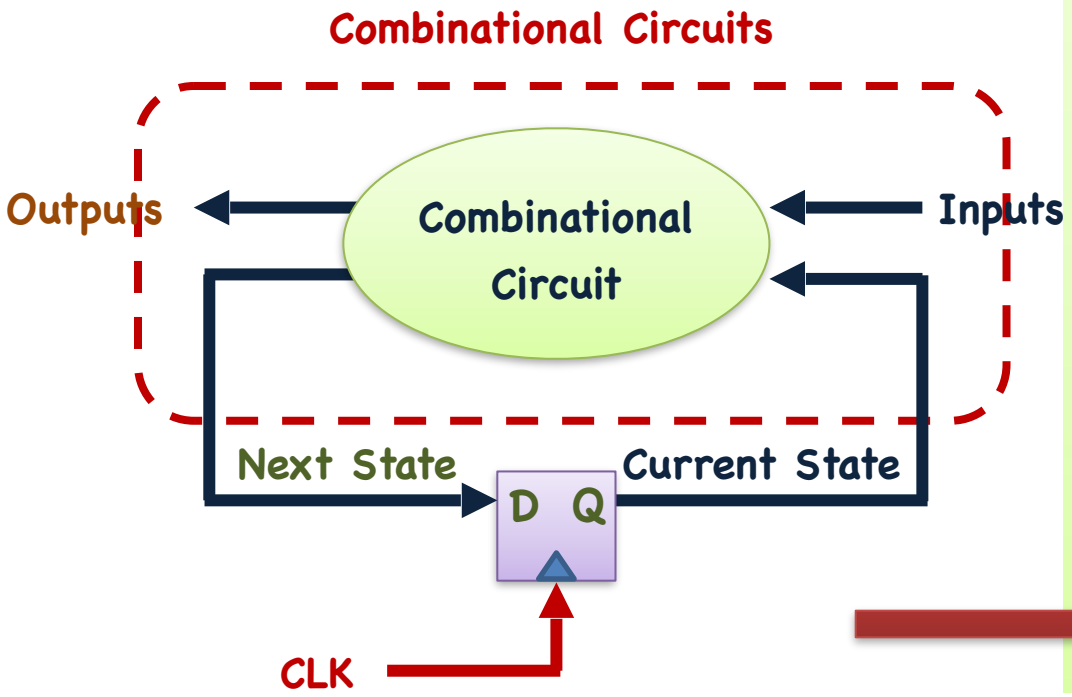
- Define your **Reset Conditions**
- Assign **Next_State** to **Current_State**
- State updates at the **next clock edge**
  - Similar to your counters




```
// Sequential circuit
always@(posedge CLK) begin
    if(!Reset)    Current_State <= S0;
    else          Current_State <= Next_State;
end
```

Always Blocks for Combinational Circuits

- Define two set of signals
  - Next_State
  - Outputs
- Use **Case** or **If-Else Statements**




```
// Combinational circuit
always@(State or Inputs) begin
        case (State)
            S0:  begin
                        Outputs = 1'b0;
                        if(Inputs == 1'b1)
                                Next_State = S1;
                        else
                                Next_State = S0;
                 end
            S1:  begin
                        Outputs = 1'b1;
                        if(Inputs == 1'b1)
                                Next_State = S0;
                        else
                                Next_State = S1;
                        . . .
```

Moore Machine Example

```
module mooreFSM(clk, Reset, in, out);
parameter S0 = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
input clk, Reset, in;
output out;
reg [1:0] Moore_state;
reg [1:0] next_state;

always @(posedge clk) begin
  if (!Reset)
    Moore_state <= S0;
  else
    Moore_state <= next_state;
end

always @(*) begin
  case (Moore_state)
    S0:
      if (in == 1'b1)
        next_state = S1;
      else
        next_state = S0;
    S1:
      if (in == 1'b0)
        next_state = S2;
      else
        next_state = S1;
    default:
      if (in == 1'b0)
        next_state = S0;
      else
        next_state = S1;
  endcase
end
assign out = (Moore_state == S2) ?1'b1 : 1'b0;
endmodule
```

Mealy Machine Example

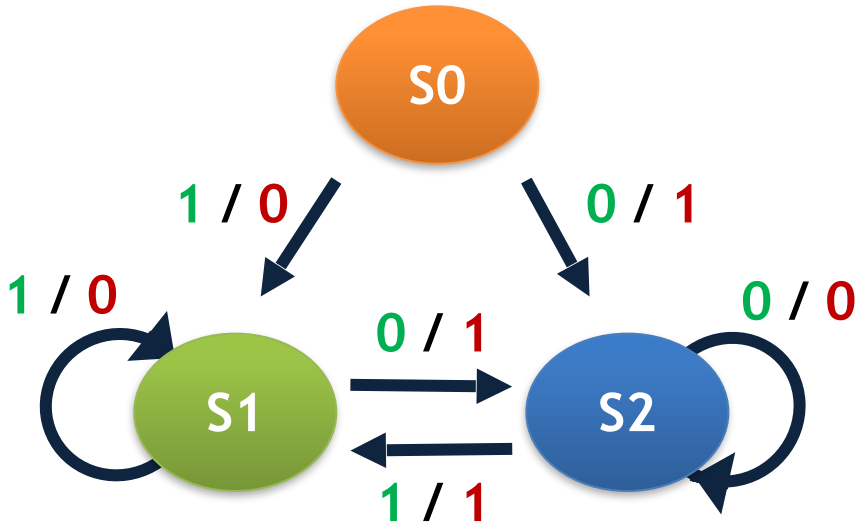
```
module mealyFSM(clk, Reset, in, out);
parameter S0 = 1'b0;
parameter S1 = 1'b1;

input clk, Reset, in;
output out;
reg out;
reg Mealy_state;
reg next_state;

always @(posedge clk) begin
  if (!Reset)
    Mealy_state <= S0;
  else
    Mealy_state <= next_state;
end

always @(*) begin
  case (Mealy_state)
```

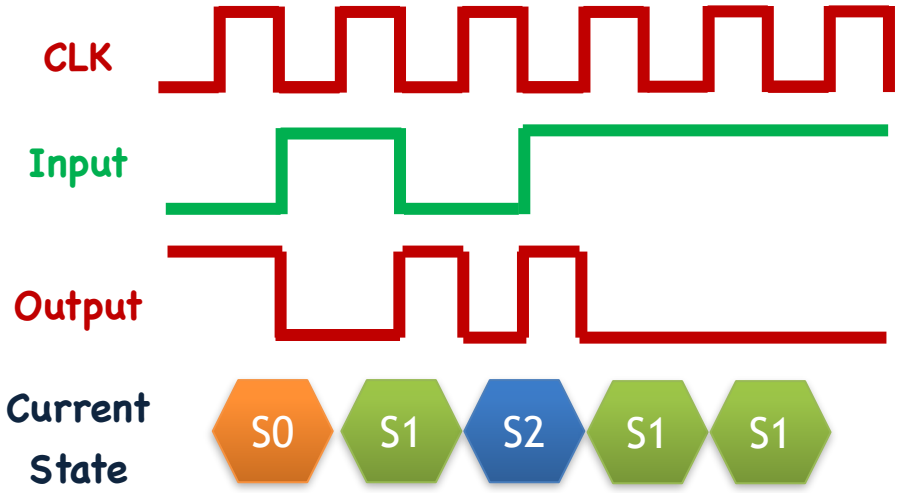
```
    S0:
      if (in == 1'b1) begin
        next_state = S1;
        out = 1'b0;
      end
      else begin
        next_state = S0;
        out = 1'b0;
      end
    default:
      if (in == 1'b0) begin
        next_state = S0;
        out = 1'b1;
      end
      else begin
        next_state = S1;
        out = 1'b0;
      end
  endcase
end
endmodule
```

Mealy Machine Input Changes

- Outputs depend on inputs and current state
- Inputs change may cause outputs change immediately
- State changes only at cock edges

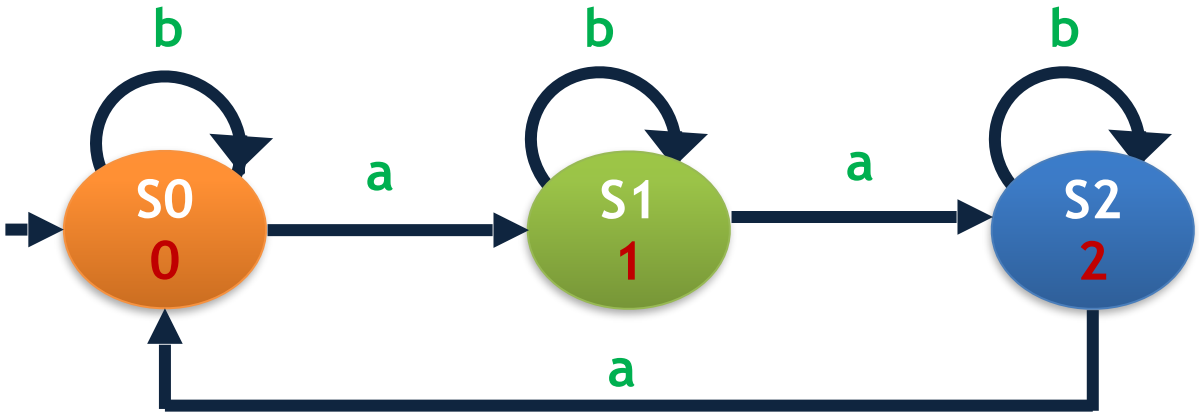
Mealy machine



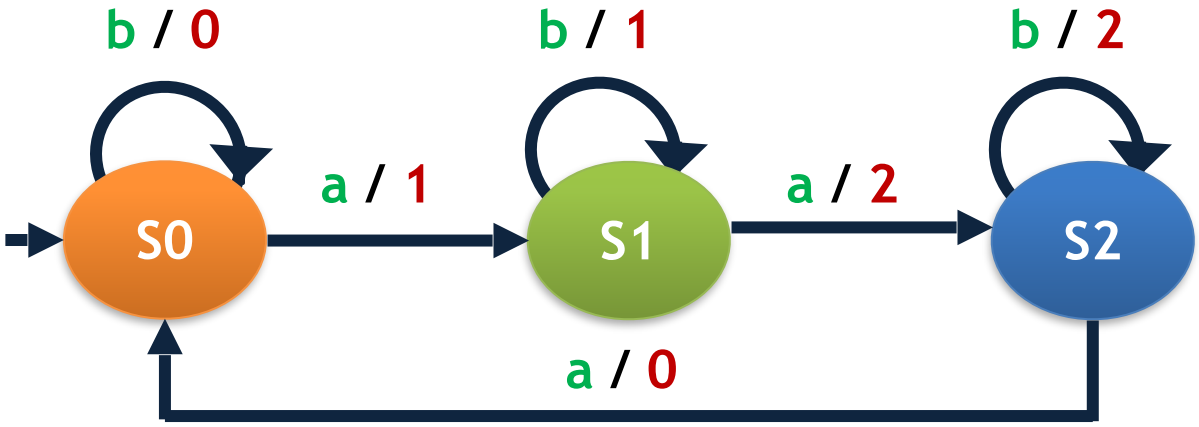


Moore to Mealy Conversion

Moore machine



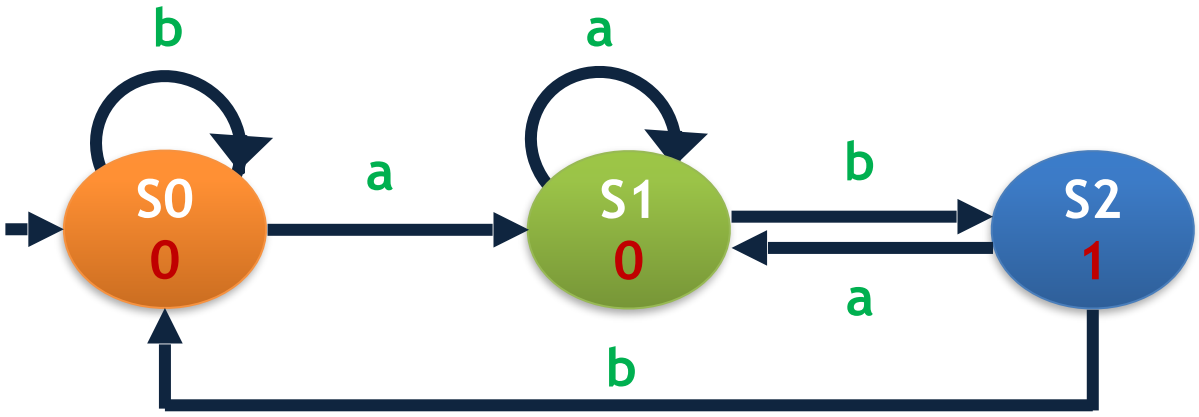
Mealy machine



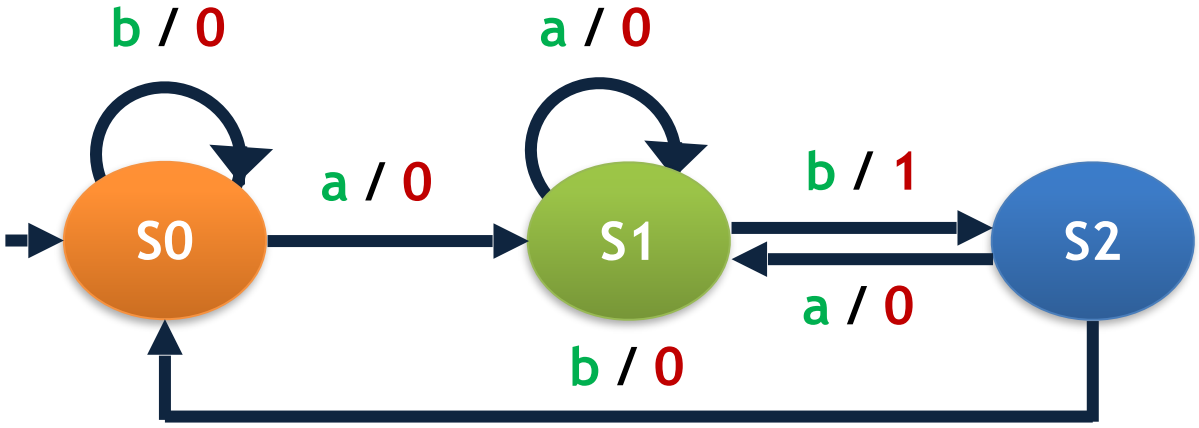
*Pictures revised from www.youtube.com/watch?v=vcNRb2cj1GI for education purpose only

Moore to Mealy Conversion

Moore machine



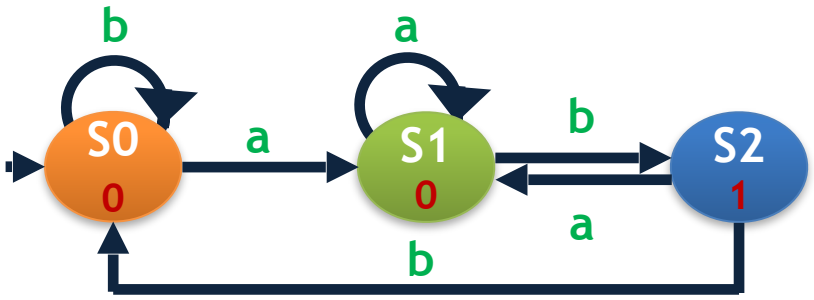
Mealy machine



*Pictures revised from www.youtube.com/watch?v=vcNRb2cj1GI for education purpose only

Moore to Mealy Conversion

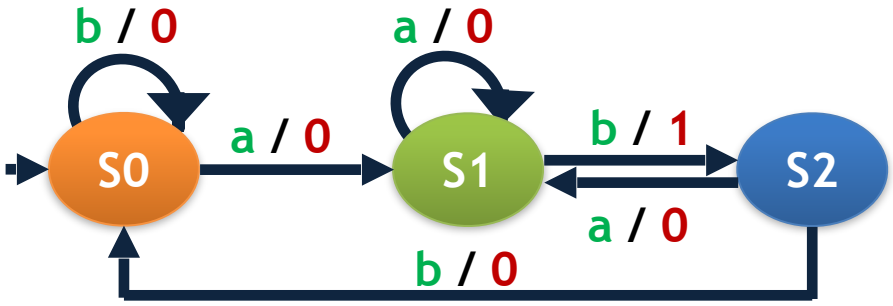
Moore machine



	a	b	output
S0	S1	S0	0
S1	S1	S2	0
S2	S1	S0	1

Next state

Mealy machine



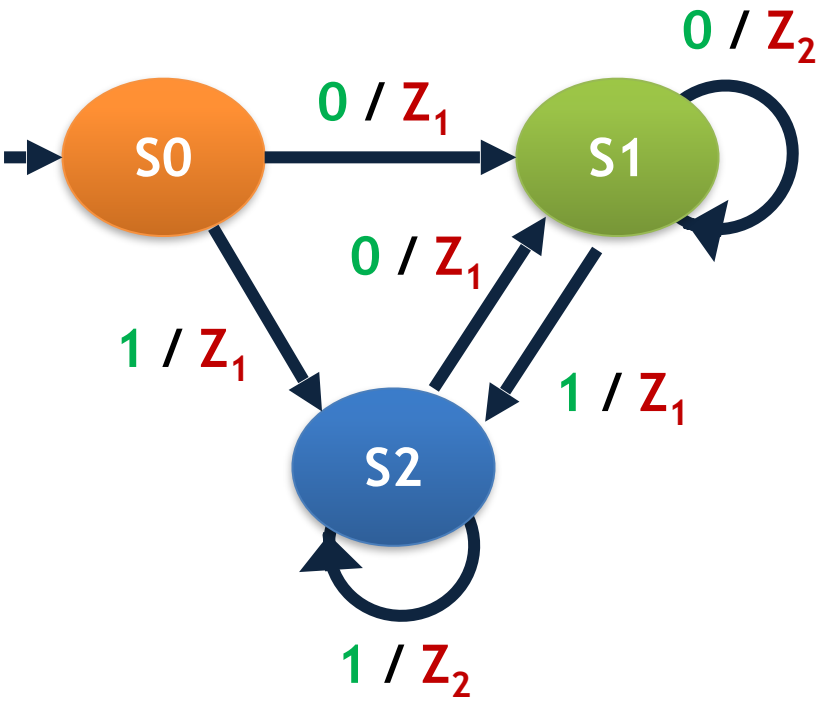
	a	b
S0	S1, 0	S0, 0
S1	S1, 0	S2, 1
S2	S1, 0	S0, 0

Next state, Output

*Pictures revised from www.youtube.com/watch?v=vcNRb2cj1GI for education purpose only

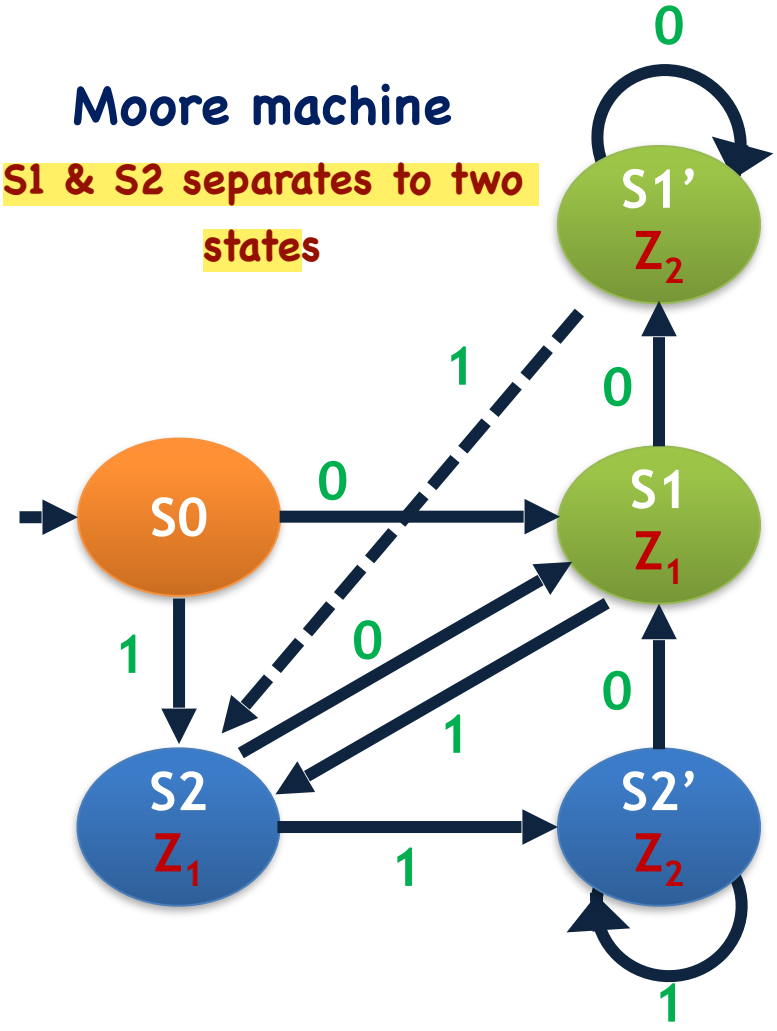
Mealy to Moore Conversion

Mealy machine



Moore machine

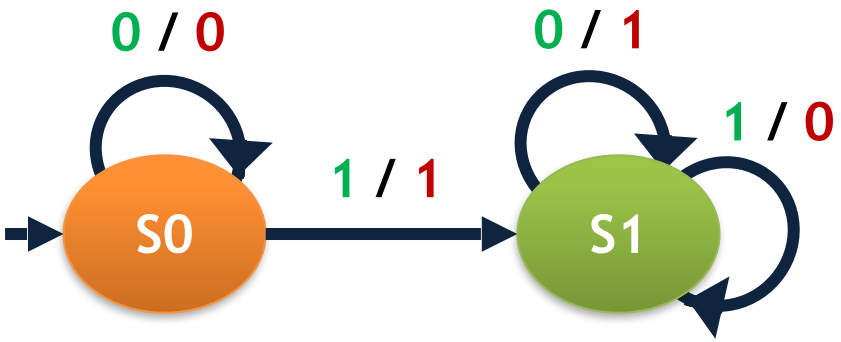
S1 & S2 separates to two states



*Pictures revised from www.youtube.com/watch?v=vcNRb2cj1GI for education purpose only

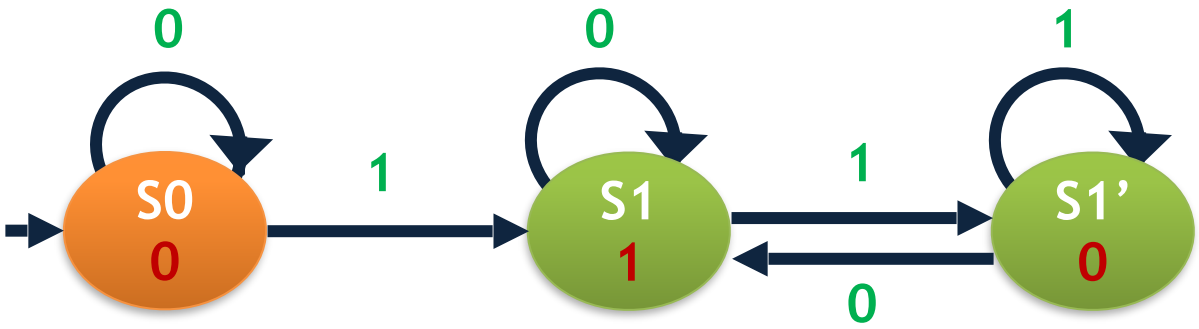
Mealy to Moore Conversion

Mealy machine



Moore machine

S1 separates to two states



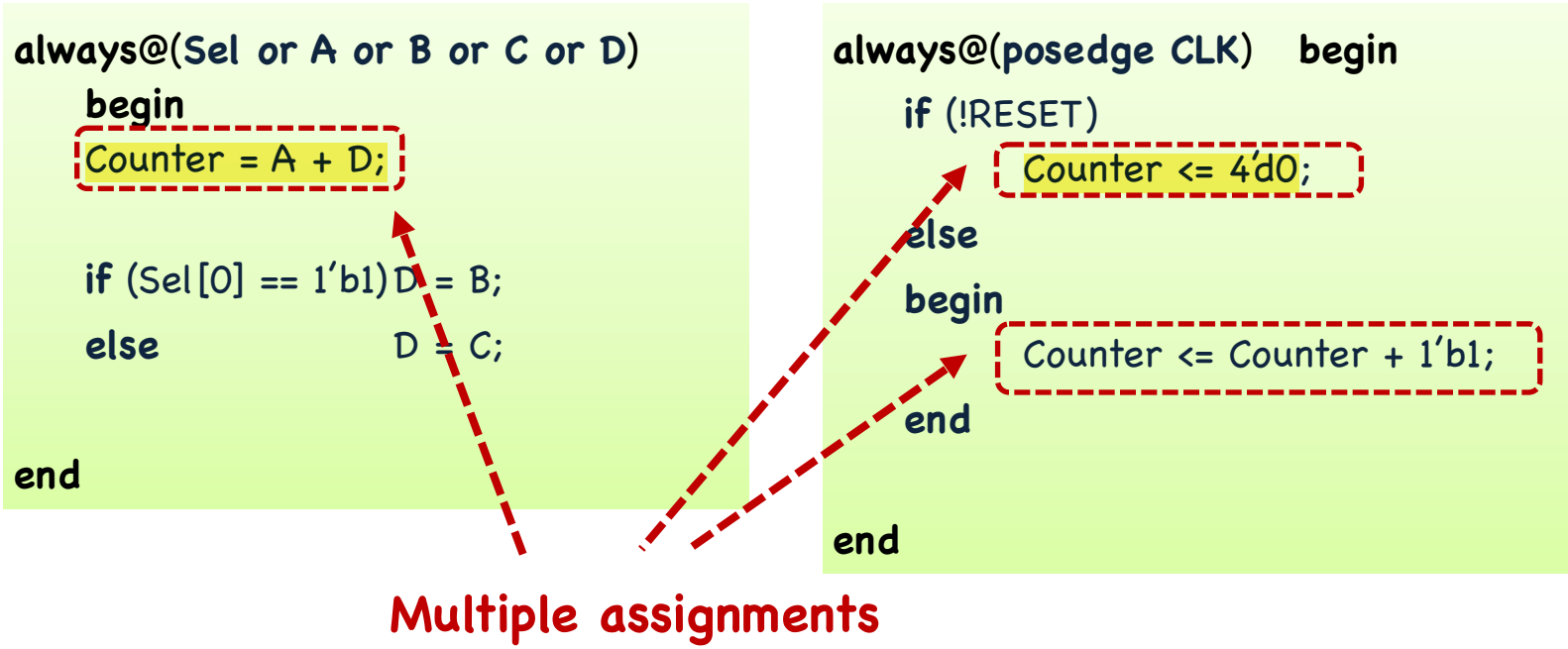
*Pictures revised from www.youtube.com/watch?v=vcNRb2cj1GI for education purpose only

State Changes in Conversion

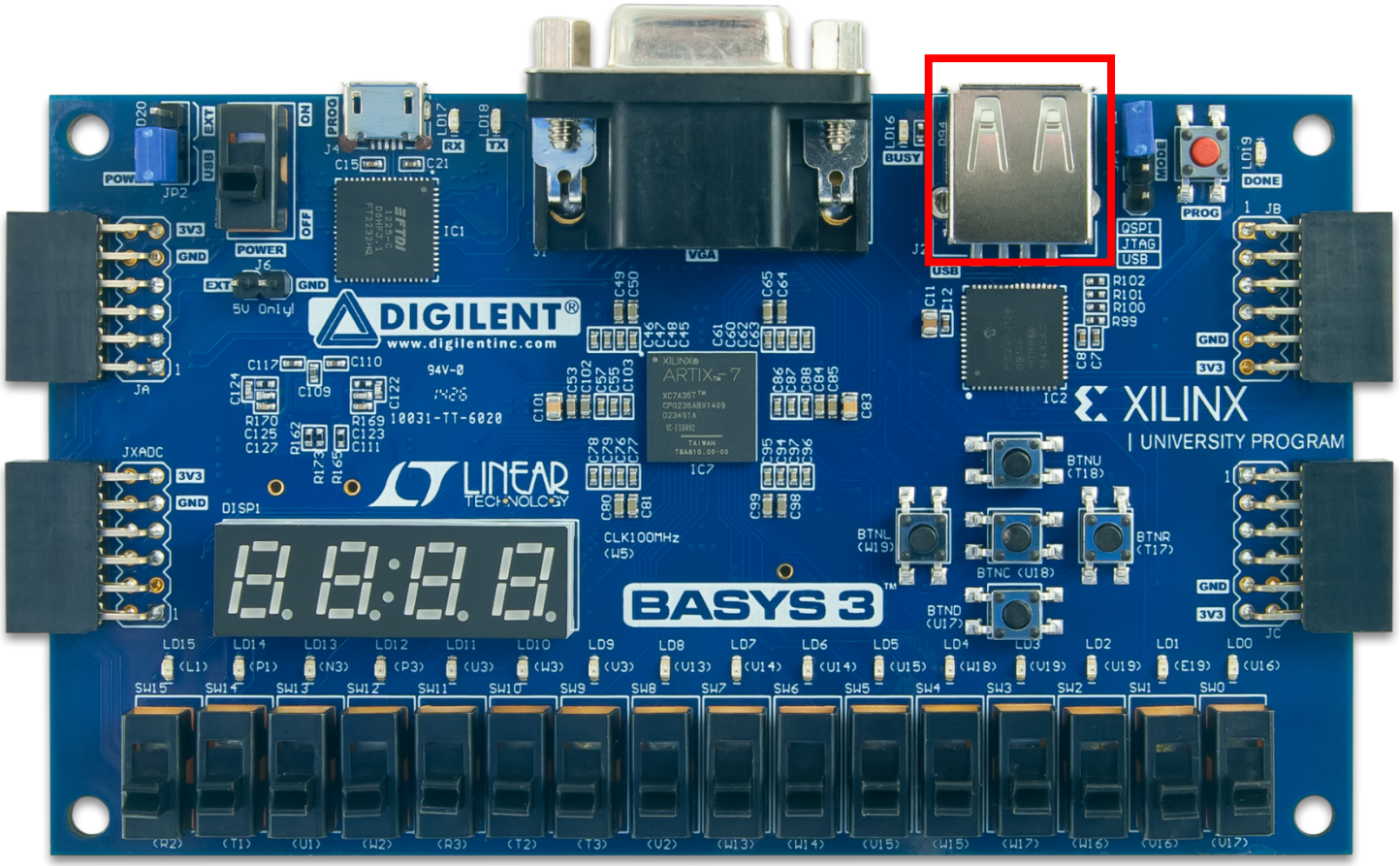
- Moore machine to Mealy machine
  - Number of states does not change
- Mealy machine to Moore machine
  - **May need to split the states**
  - Number of states increased
  - A Mealy machine with **N** states and **M** outputs may lead a Moore machine with a maximum of (**N x M**) states
  - In the previous example, **N** = 2 and **M** = 2

Why Can't I Configure FPGA...?

- Simulation passed, configuration failed
- Multiple assignments to **THE SAME** variable in **DIFFERENT** always blocks
- A variable can only be assigned in one single always block



USB HID Host (1/3)



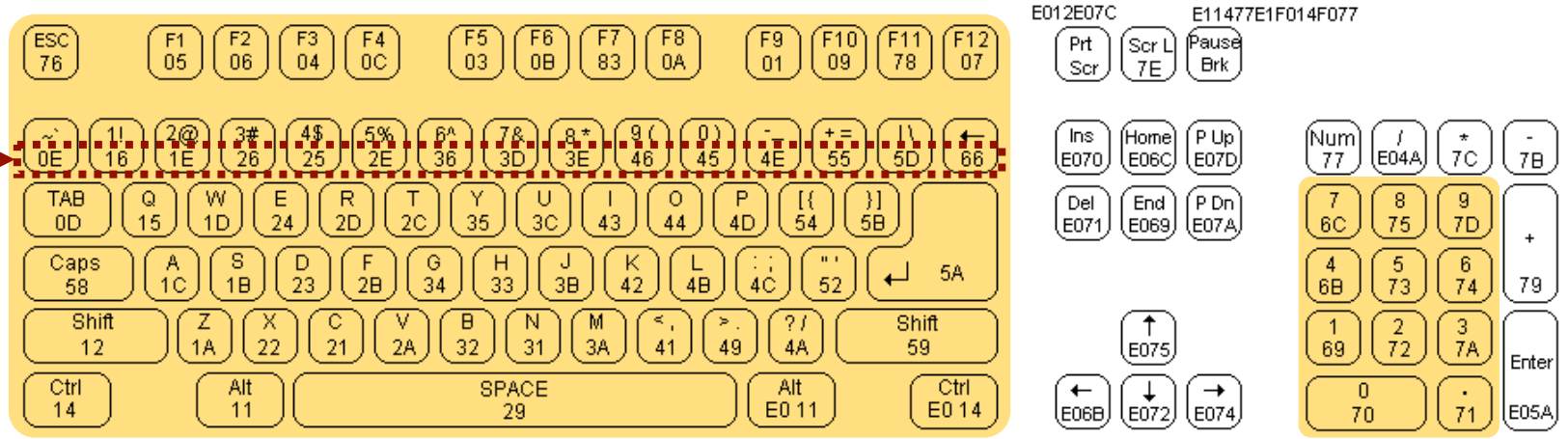
PS/2 Scancode

- Three types of scancodes
  - Make codes represent the key values
  - Break code represents the action of "release the key"
  - Extend code represent the **duplicate** of a key

Extend Code	Break Code	Make code
E0	F0	XX

(means "release")

Make code

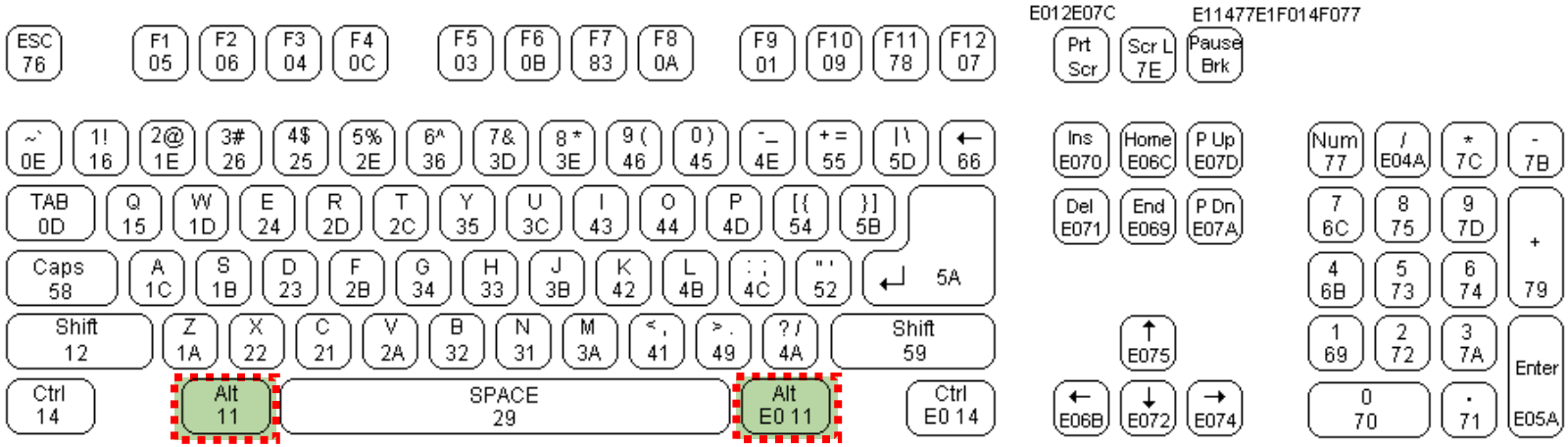


We only use the yellow parts of the keyboard

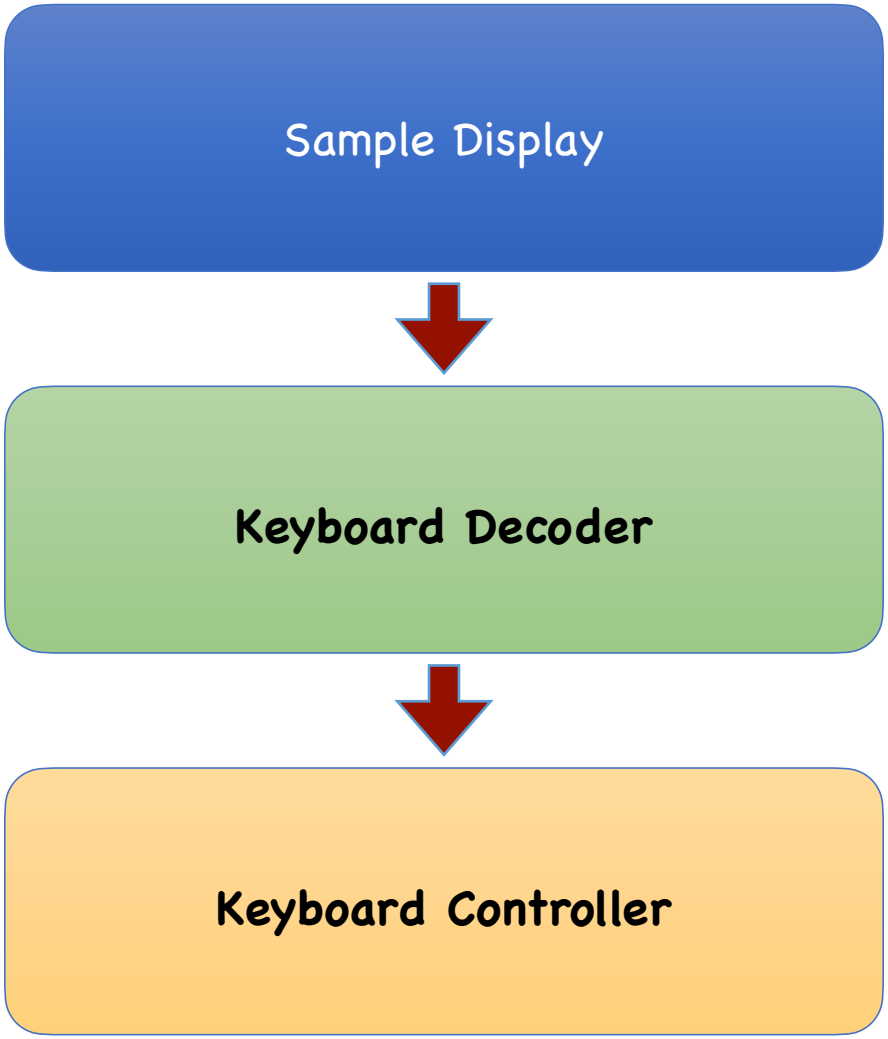
PS/2 Scancode (Example)

- Two "Alt" key are available on the keyboard
  - Additional keys are encoded as <**extend code**> + <**make code**>
- When releasing a key, a <**break code**> is inserted

L Alt press			11
L Alt release		F0	11
R Alt press	E0		11
R Alt release	E0	F0	11



Program Hierarchy



Verilog Module: KeyboardDecoder (1/5)

- In Keyboard Sample Code
  - KeyboardDecoder.v
  - SampleDisplay.v

- I/O for KeyboardDecoder

Output

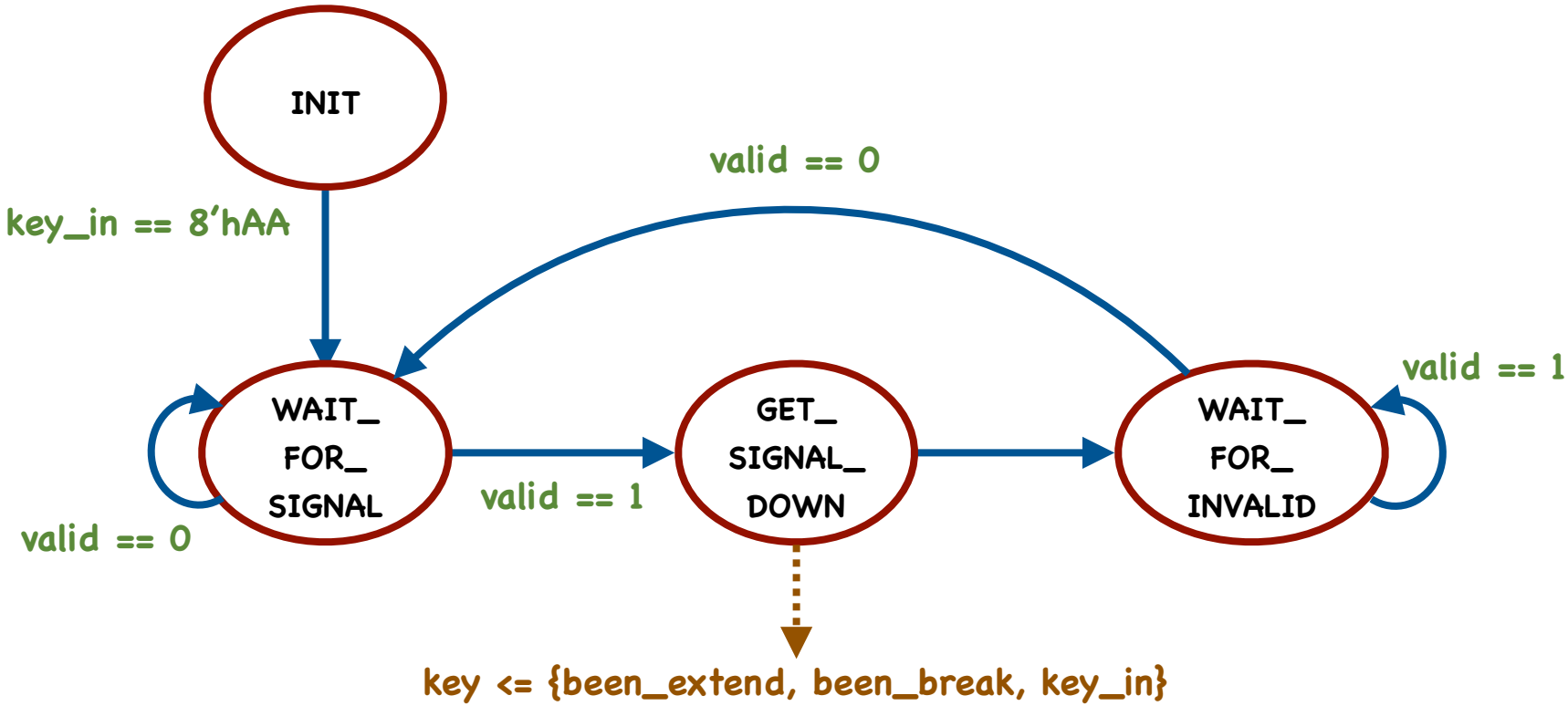
- key_down
- last_change
- Key_valid

Input

- PS2_CLK
- PS2_DATA
- rst
- clk

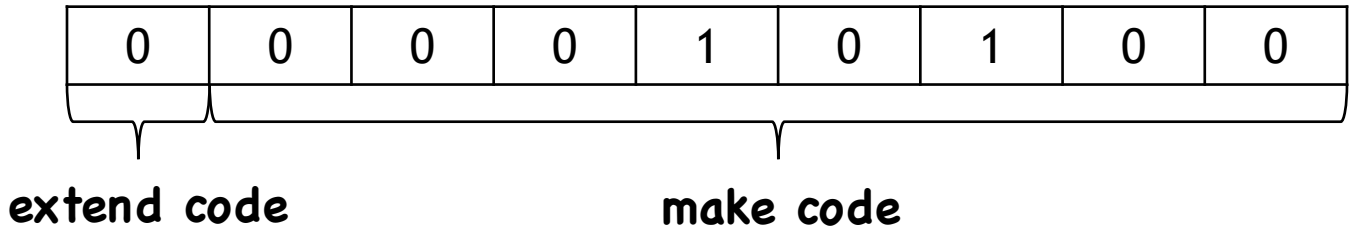
Verilog Module: KeyboardDecoder (2/5)

- Retrieves "**extend**", "**break**", and "**key**" from KeyboardCtrl
  - Gets value only when **valid = 1'b1**



Verilog Module: KeyboardDecoder (3/5)

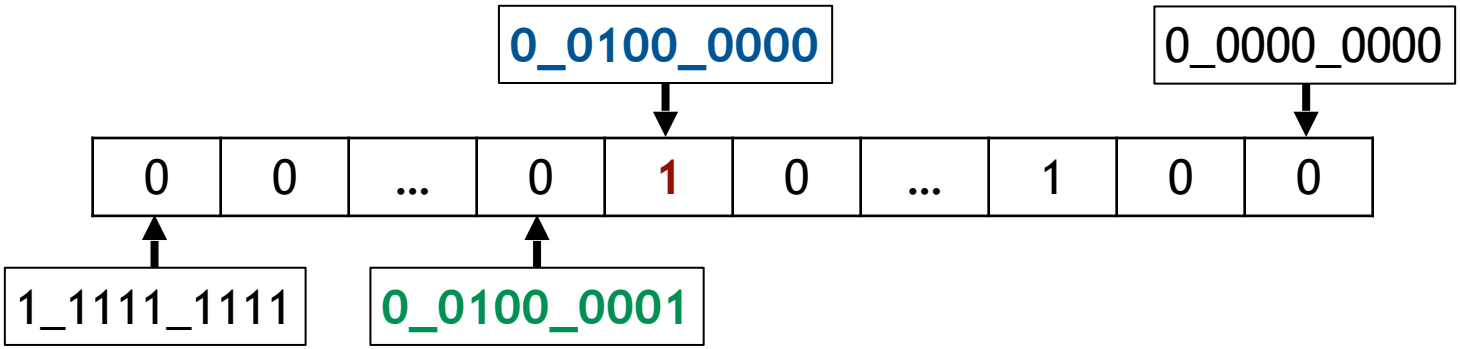
- **last_change**: 9 bits
  - Represent the key which has been pressed or released.



- **key_valid**: 1 bit
  - Should be active for one clock period (100MHz) when any key is pressed or released.

Verilog Module: KeyboardDecoder (4/5)

- Use an array "**key_down**" of 512 bits to record which key is currently being pressed
  - 1 means "key is pressed", while 0 means "key is released"



- The key indexed by "**0_0100_0000**" is pressed
- The key indexed by "**0_0100_0001**" is released

- Use "**key_decode**" (512 bits) to represent the key that was just pressed or released

Verilog Module: KeyboardDecoder (5/5)

- To represent that a key is pressed
  - key_down <= key_down | key_decode;

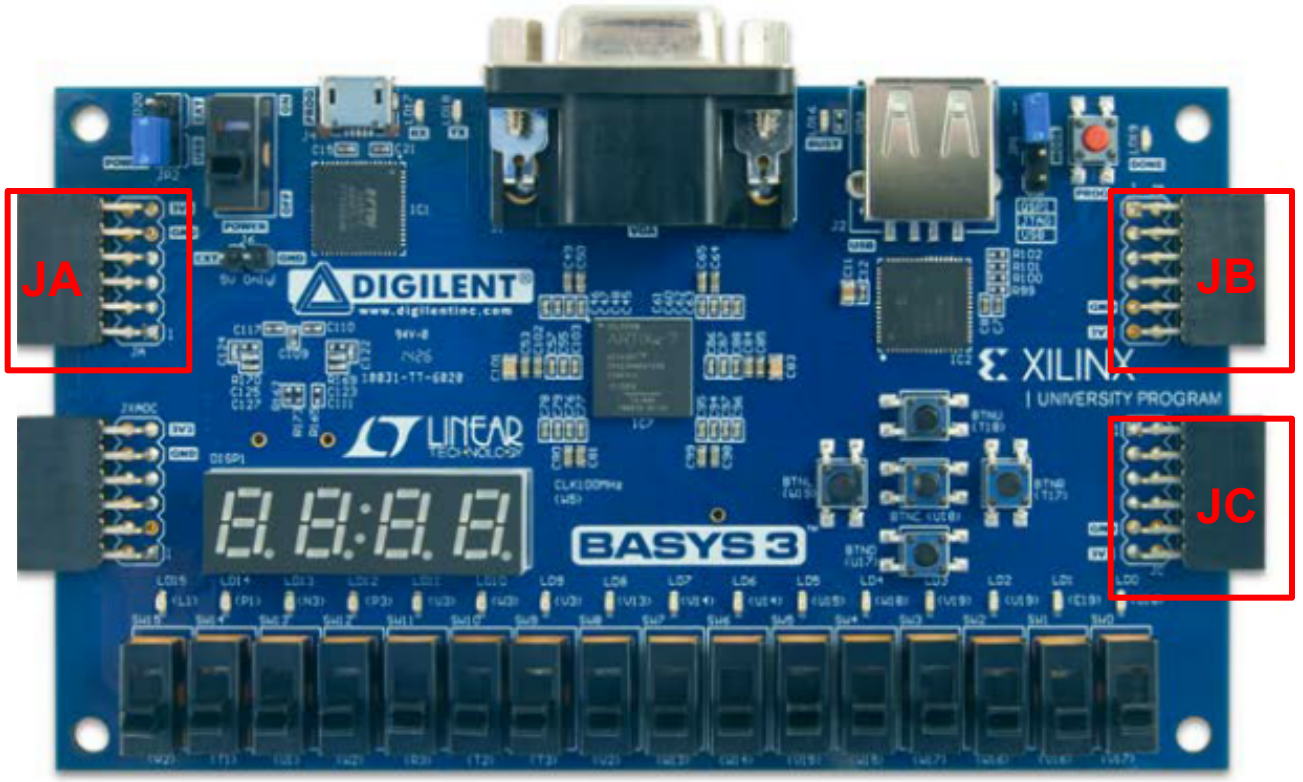
	0	1	1	0	1
or	0	0	0	1	0
	0	1	1	1	1

- To represent that a key is released
  - key_down <= key_down & (~key_decode)

	0	1	1	0	1
and	1	1	0	1	1
	0	1	0	0	1

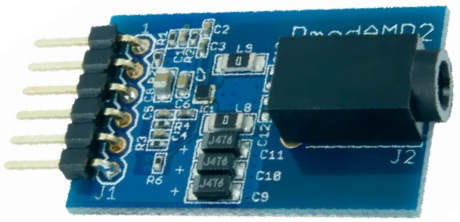
Pmod Connectors (1/2)

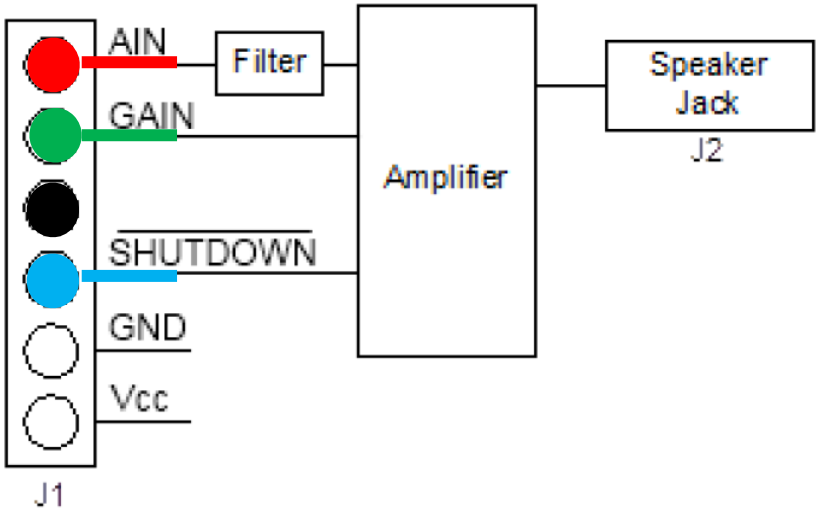
- Basys3 provides 3 Pmod connectors
  - JA, JB, and JC



Pmod Accessory Board : "PmodAMP2"

- **AIN (audio_in)**
  - < 16KHz
- **GAIN (default 1)**
  - 1'b1: 6dB
  - 1'b0: 12dB
- **SHUTDOWN (default 1)**
  - 1'b0: disable
  - 1'b1: enable





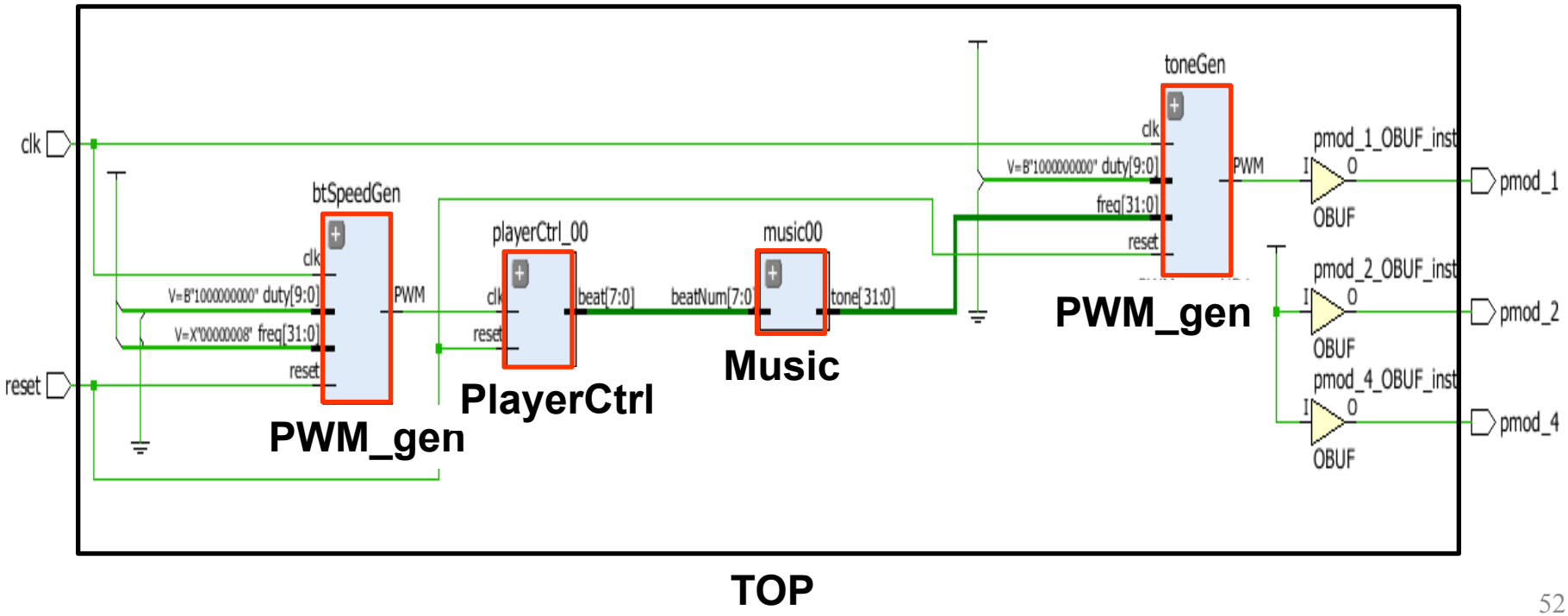
Ref. *"PmodAMP2 Reference Manual"*, revision: Aug 2, 2012, Digilent

Basic Concepts of Sound

- **Sound** that is perceptible by **human beings** has frequencies ranging from about **20** Hz to **20,000** Hz
- **Frequency**:
  - The **higher** (**lower**) the frequency, the **higher** (**lower**) the pitch
- **Duty cycle**:
  - The **evener** the duty cycle, the **better** the quality.

Demo 2: Block Diagram

- In the demo code, only **pin 1**, **pin 2**, and **pin 4** are used
  - **Pin 1**: AIN
  - **Pin 2**: GAIN
  - **Pin 4**: SHUTDOWN_N



Demo 2: Top Module

```
module TOP (
    input clk,
    input reset,
    output pmod_1,
    output pmod_2,
    output pmod_4
);
parameter BEAT_FREQ = 32'd8;     //one beat=0.125sec
parameter DUTY_BEST = 10'd512;   //duty cycle=50%

wire [31:0] freq;
wire [7:0] ibeatNum;
wire beatFreq;

assign pmod_2 = 1'd1;    //no gain(6dB)
assign pmod_4 = 1'd1;    //turn-on
```

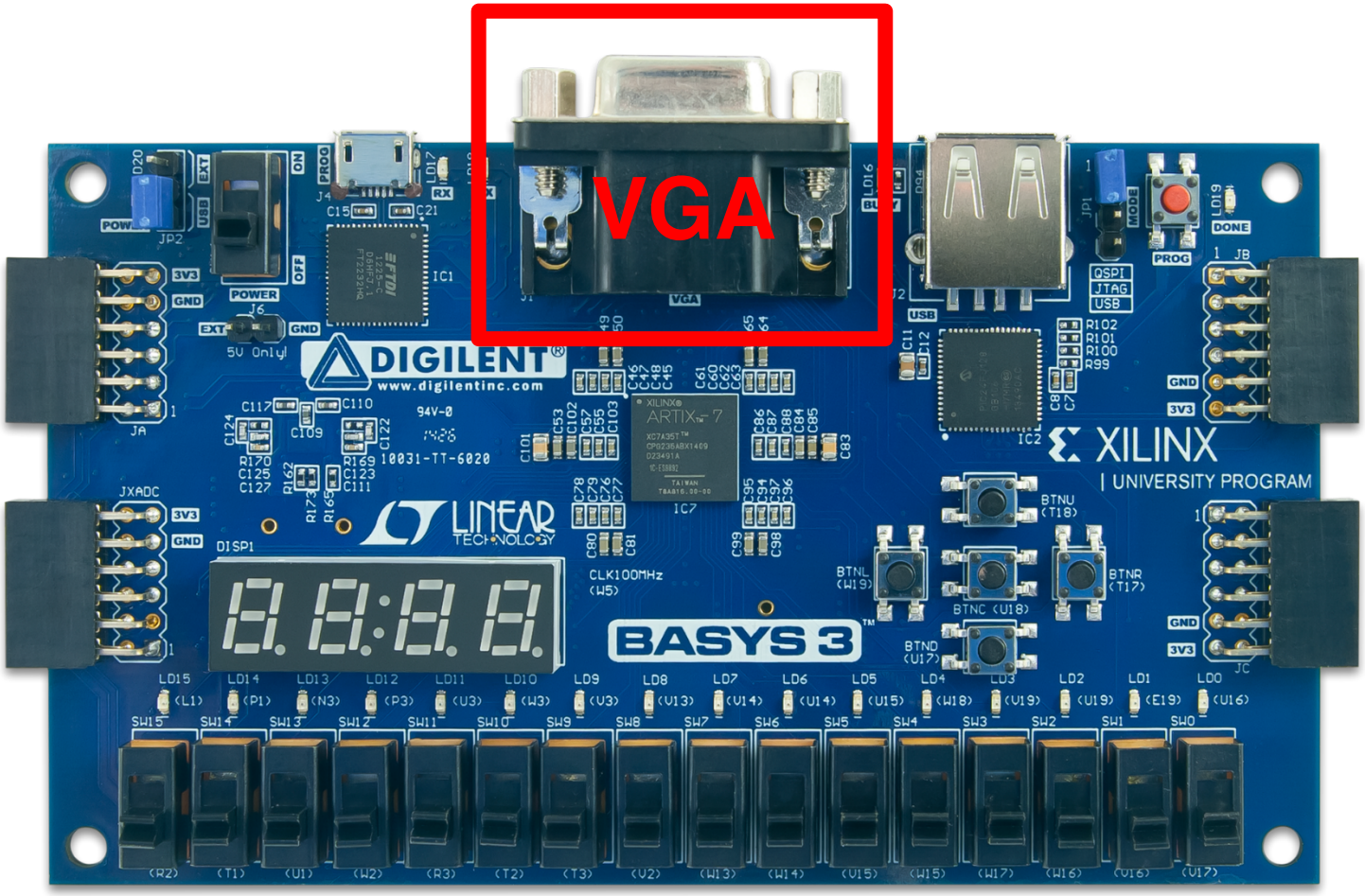
- Please note that this number represents the frequency
- 32'd8 -> 8Hz -> (period is 0.125 sec)

```
//Generate beat speed
PWM_gen btSpeedGen ( .clk(clk),
                     .reset(reset),
                     .freq(BEAT_FREQ),
                     .duty(DUTY_BEST),
                     .PWM(beatFreq)
);

//manipulate beat
PlayerCtrl playerCtrl_00 ( .clk(beatFreq),
                           .reset(reset),
                           .ibeat(ibeatNum)
);

//Generate variant freq. of tones
Music music00 ( .ibeatNum(ibeatNum),
                .tone(freq)
);

// Generate particular freq. signal
PWM_gen toneGen ( .clk(clk),
                  .reset(reset),
                  .freq(freq),
                  .duty(DUTY_BEST),
                  .PWM(pmod_1)
);
endmodule
```

- Four modules are used in the design
- You can change this module and use it in your final project

VGA Port On FPGA

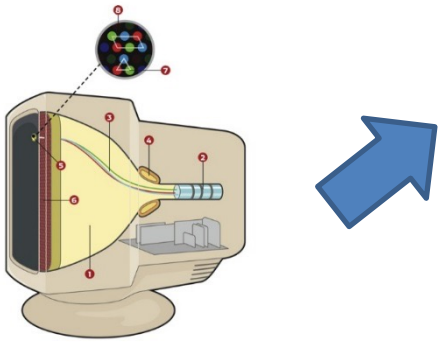
- A Video Graphic Array (VGA) port is available on FPGA

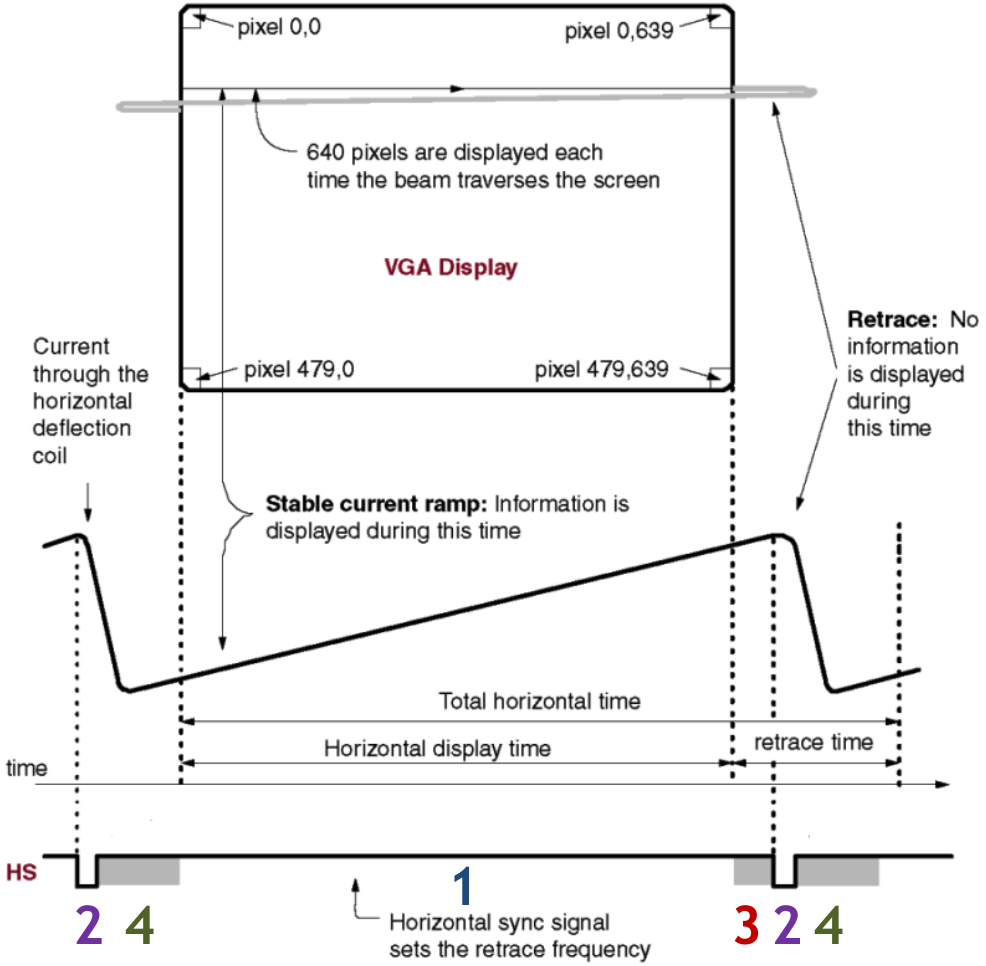


*Pictures cited from www.digilentinc.com for education purpose only

VGA Timing Diagram (1/3)

- Displays 640 x 480 pixels
- Scans the screen back and forth
- Four components (horizontal)
  - 1. Screen display time
  - 2. Sync pulse
  - 3. Front porch
  - 4. Back porch
- Retrace time = **2 + 3 + 4**
- Total display time = **1 + 2 + 3 + 4**

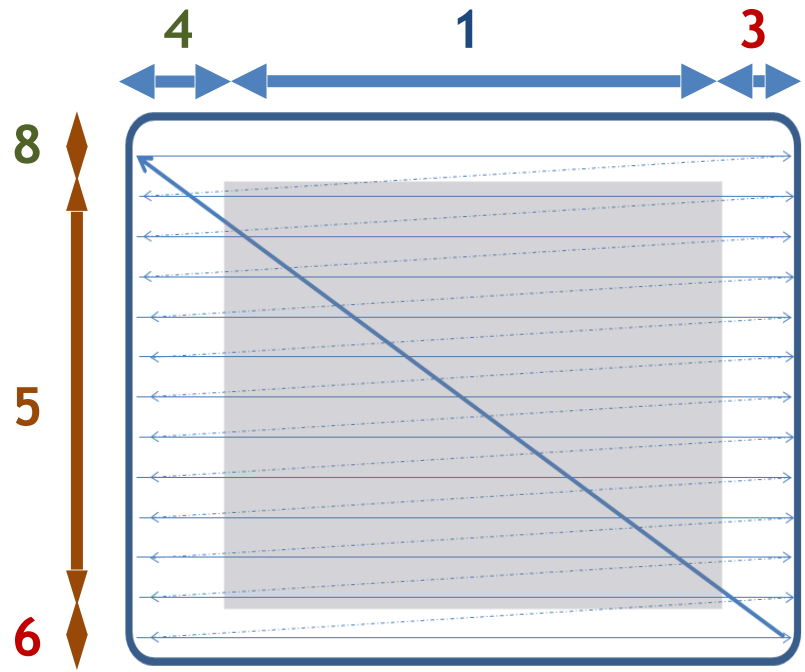


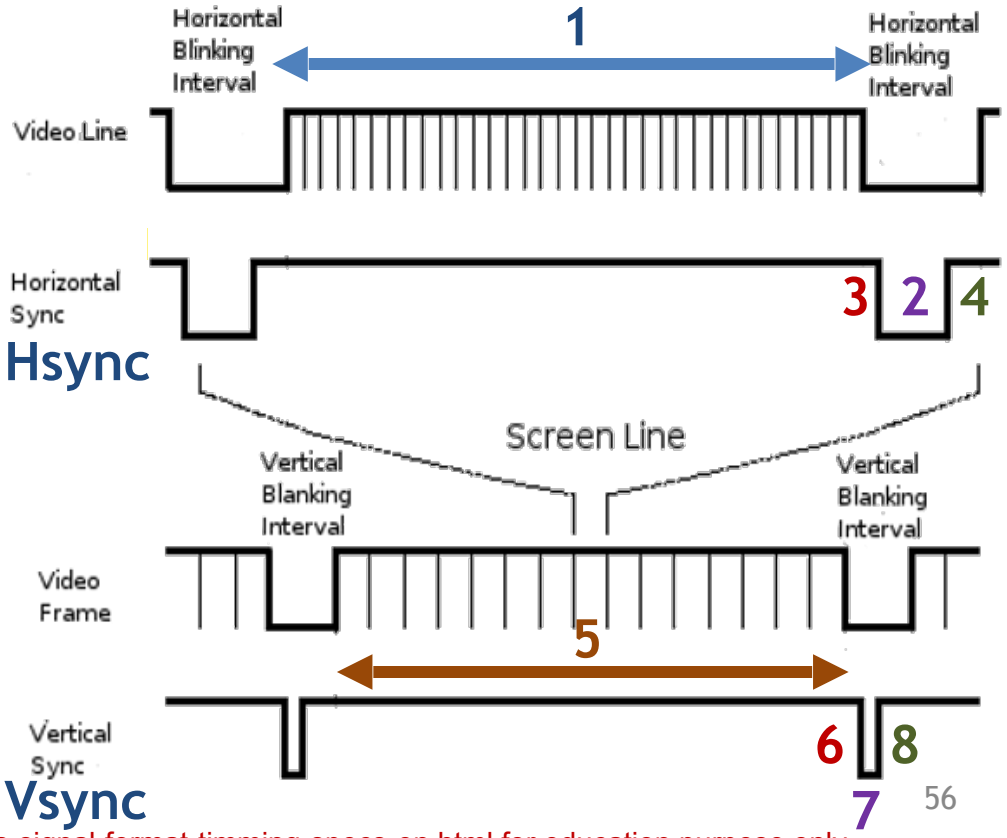


*Pictures cited from www.computershopper.com and www.cs.ucr.edu/~jtarango/cs122a_lab4.html for education purpose only

VGA Timing Diagram (2/3)

- Scans through each horizontal lines first
- Pull back the pointer to the beginning of the line at the end
- Two control signals
  - **Hsync (HS)**
  - **Vsync (VS)**

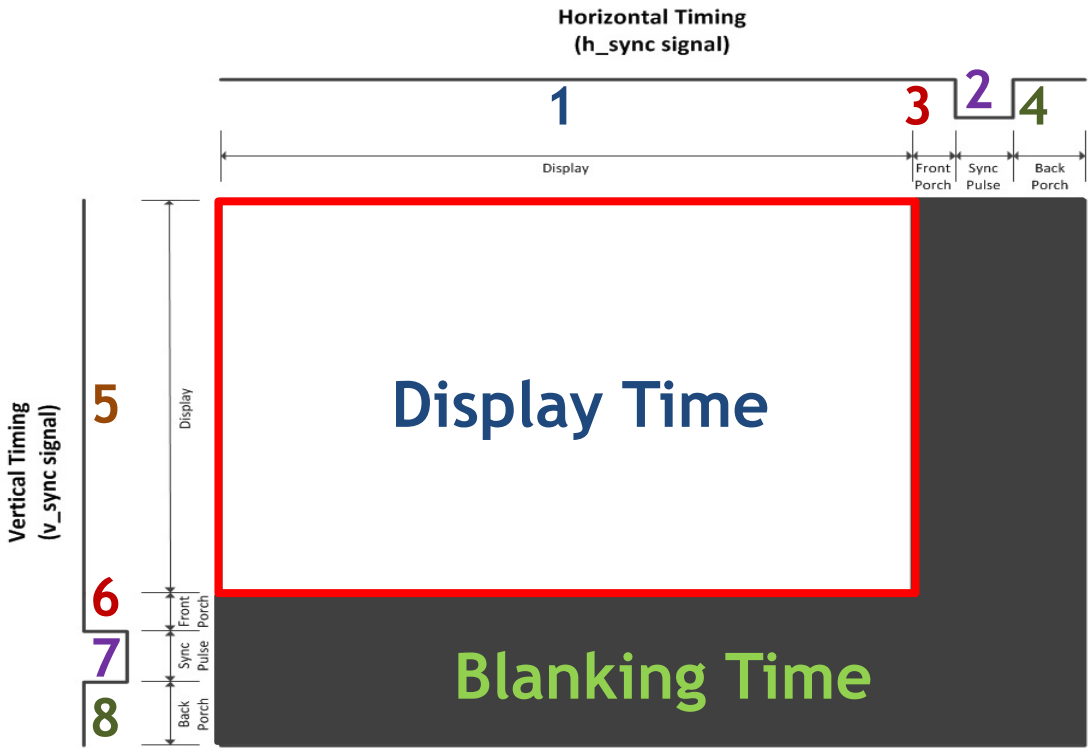




*Pictures cited from www.javiervalcarce.eu/html/vga-signal-format-timming-specs-en.html for education purpose only

VGA Timing Diagram (3/3)

- Signal timing for a 640-pixel by 480 rows display using a 25MHz pixel clock
- Higher resolution can be achieved via different timing spec



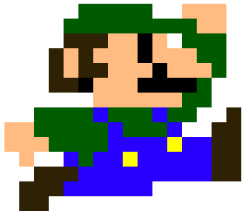
640 x 480 spec

Parameter	Ver. Sync		Hor. Sync	
	Lines	Time(ms)	Pixels	Time(μs)
Visible area	480	15.3	640	25
Front porch	10	0.3	16	0.64
Sync pulse	2	0.064	96	3.8
Back porch	33	1.05	48	1.9
Whole line	525	16.7	800	32

240p-480p-960p

*Pictures cited from emulation-general.wikia.com for education purpose only

Pixel Clock



- The clock frequency used to display one pixel on the screen
  - Use a clock divider to generate it

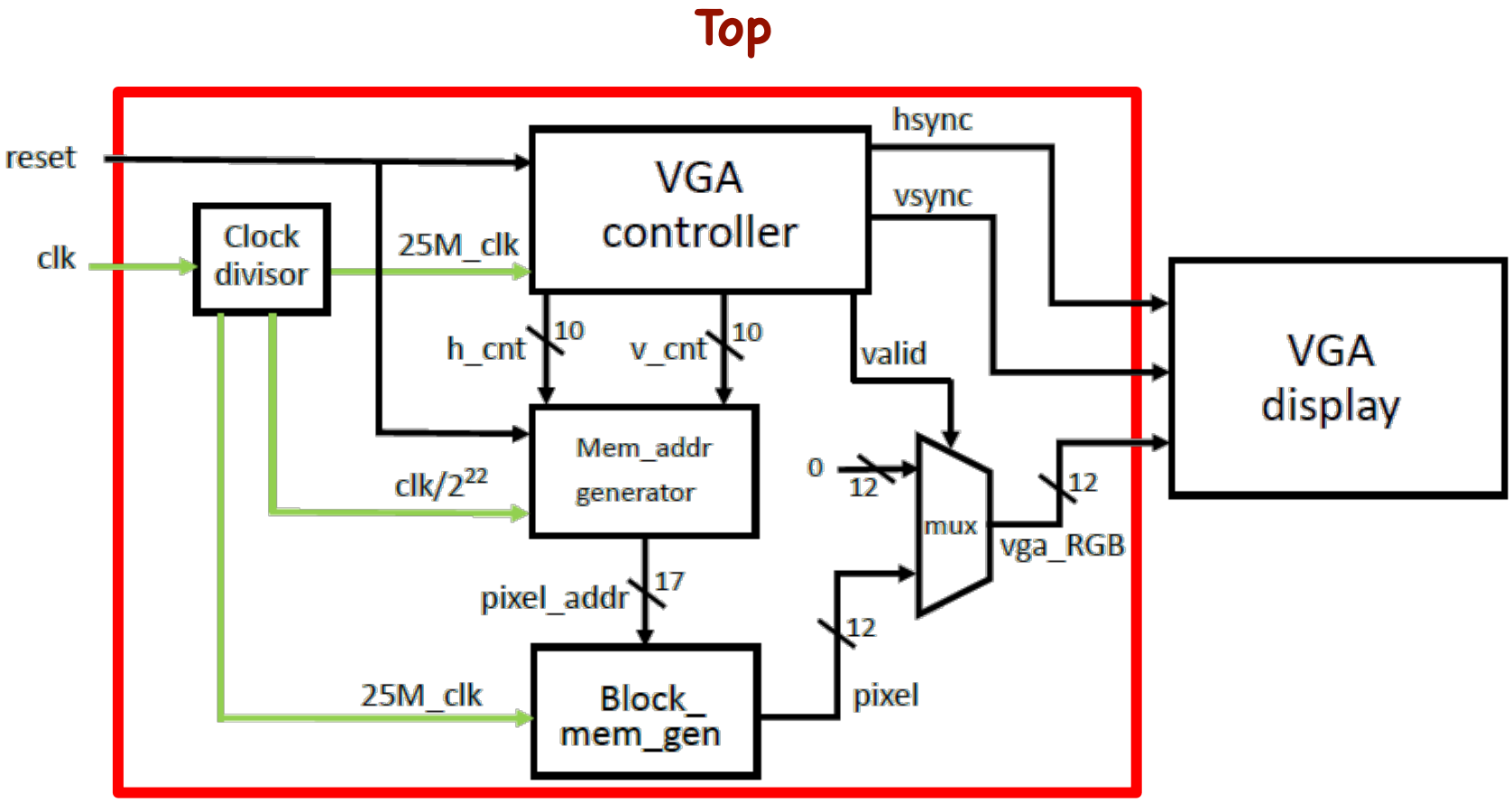
- Clock frequency depends on number of pixels and refresh rate
- For 640 x 480 pixels with 60Hz refresh rate, the clock frequency is: **800 x 525 x 60 (frame/sec) = 25M (pixel/sec)**

  - 800 (pixels/line) corresponds to **1 + 2 + 3 + 4**
  - 525 (lines/frame) corresponds **5 + 6 + 7 + 8**

- The 25MHz clock frequency can be generated by the 100MHz FPGA frequency (divide-by-four)

*Pictures cited from 14ipt.wikia.com for education purpose only

Demo 2: Block Diagram



Demo 2: Top Module

```
module top(
    input clk,
    input rst,
    output [3:0] vgaRed,
    output [3:0] vgaGreen,
    output [3:0] vgaBlue,
    output hsync,
    output vsync
     );

     wire [11:0] data;
     wire clk_25MHz;
     wire clk_22;
     wire [16:0] pixel_addr;
     wire [11:0] pixel;
     wire valid;
     wire [9:0] h_cnt; //640
     wire [9:0] v_cnt;  //480

assign {vgaRed, vgaGreen, vgaBlue}
       = (valid==1'b1) ? pixel:12'h0;
```

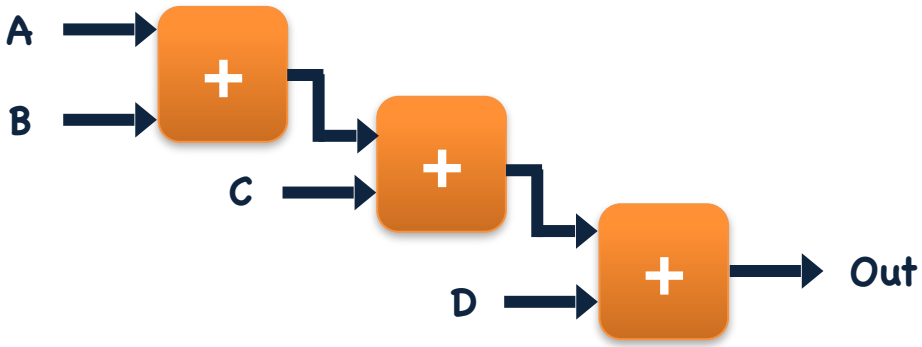
```
 clock_divisor clk_wiz_0_inst(
   .clk(clk),
   .clk1(clk_25MHz),
   .clk22(clk_22)
);

mem_addr_gen mem_addr_gen_inst(
.clk(clk_22),
.rst(rst),
.h_cnt(h_cnt),
.v_cnt(v_cnt),
.pixel_addr(pixel_addr)
);


blk_mem_gen_0 blk_mem_gen_0_inst(
   .clka(clk_25MHz),
   .wea(0),
   .addra(pixel_addr),
   .dina(data[11:0]),
   .douta(pixel)
);
```

```
vga_controller   vga_inst(
   .pclk(clk_25MHz),
   .reset(rst),
   .hsync(hsync),
   .vsync(vsync),
   .valid(valid),
   .h_cnt(h_cnt),
   .v_cnt(v_cnt)
);

endmodule
```

Operator Balancing

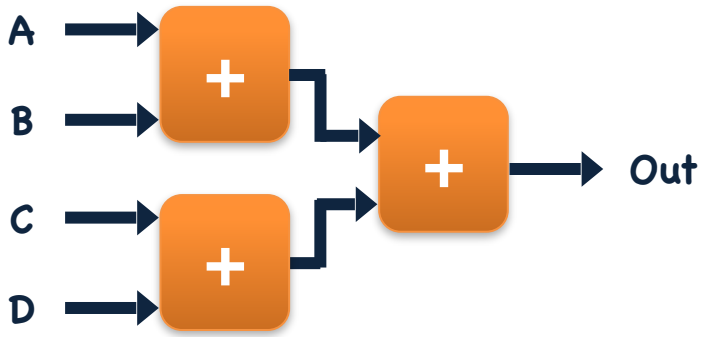
- Use parenthesis to guide synthesis

A + B + C + D



Longer path delay

(A + B) + (C + D)



Shorter path delay

Complete Your If Statements

- Avoid incomplete If statements
- Otherwise, you may get unwanted additional gates or latches

```
always@(A or B or C or D or E)
begin
    if (A < B)
        Out = C;
    else if (A > B)
        Out = D;
    else if (A == B)
        Out = E;
end
```

Incomplete If-Else Statement

```
always@(A or B or C or D or E)
begin
    if (A < B)
        Out = C;
    else if (A > B)
        Out = D;
    else
        Out = E;
end
```

Complete If-Else Statement

Complete Your Case Statements

- Completely specify all conditions in **case** and **if statements**
- Completely specify all outputs in every **case** and **if statements**

```
always@(Sel)
begin
        case (Sel)
              2'b00:     A = 1'b1;
              2'b01:     A = 1'b0;
              2'b10:     B = 1'b1;
     endcase
end
```

Missing cases and outputs

```
always@(Sel)
begin
        case (Sel)
              2'b00:    begin
                          A = 1'b1;
                          B = 1'b0;
                        end
             default: begin
                          A = 1'b0;
                          B = 1'b1;
                      end
     endcase
end
```

Thank you for your attention!

*Pier view taken at Stockholm, Sweden.
This picture is taken by Chun-Yi Lee himself, who is also a fan of photography