[T]EE2026 DIGITAL DESIGN

LAB4: SEQUENTIAL CIRCUITS IN VERILOG (2/2)

Gu Jing
E4-03-10
eleguji@nus.edu.sg

REVISION FOR LAB 3

clock divider

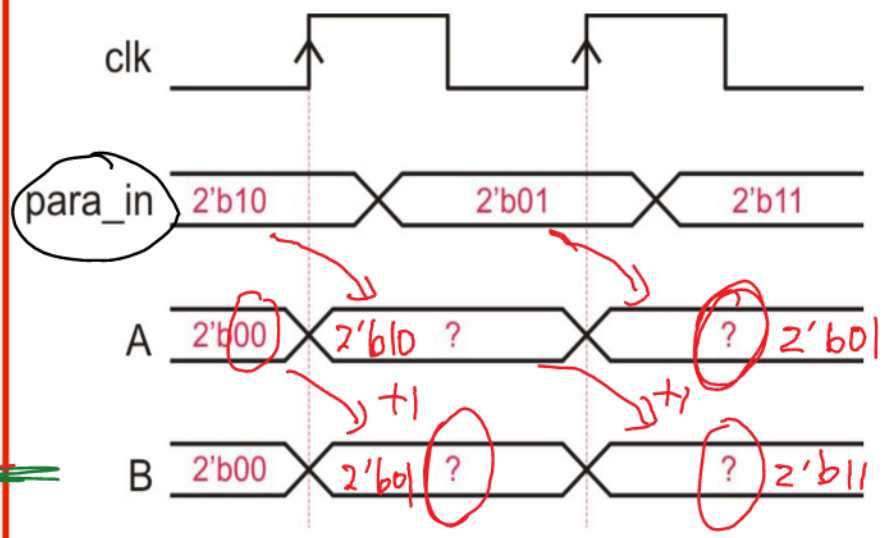
Sequential Circuits :

- Outputs depend on the current AND past values of inputs or outputs
- Sequential logic: always @ (posedge/negedge CLOCK) begin.... end
- Within sequential block, use non-blocking assignment (<=) to make sure values are updated in the next clock cycle

```
module example_for_non_blocking (
input clk,
input [1:0] para_in,
output reg [1:0] A= 2'b00,
output reg [1:0] B= 2'b00
);

    always @ (posedge clk) begin
      A <= para_in;
      B <= A + 1;
      .....
    end

endmodule
```

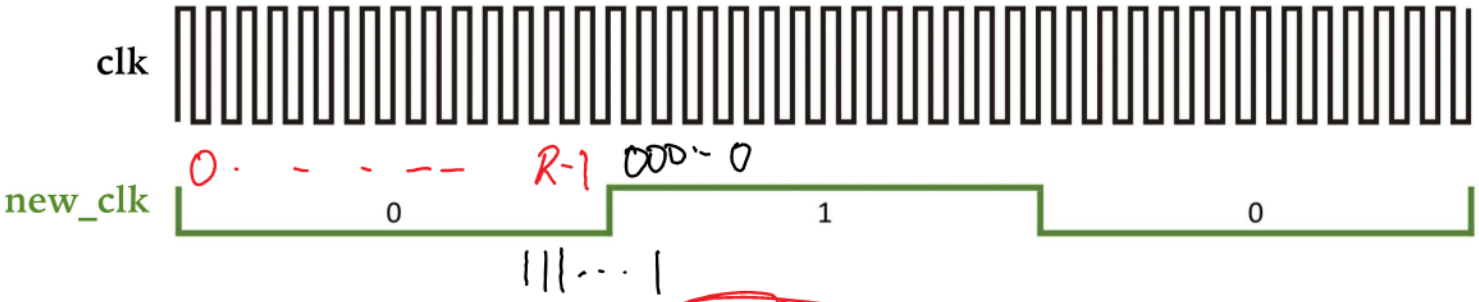
para_in

B is +1 A.



REVISION FOR LAB 3

Clock Divider to achieve f_{new} :



$$\frac{f_{new}}{f_0} = \frac{1}{2^{n+1}}$$

n = 23, 5

24

An internal n-bit counter: allow us to count how many cycles of clk will cause new_clk to flip.

always @ ↑
counter <= counter + 1 ;
newclk <= (counter==0) ~ newclk : newclk;

OUTLINE FOR LAB 4

Precise Frequency Generation (continue from lab 3)

- Generate an Accurate Frequency from 100MHz

Single Pulse Circuit & Counter

- D-Flip-Flop (DFF)

DFF → applications

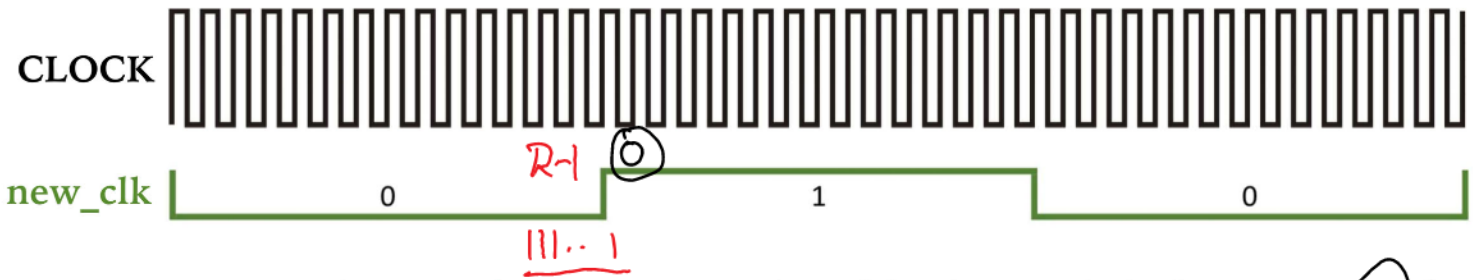
- Single Pulse Circuit by two DFF (for simulation)
- Single Pulse and Counter (for simulation)

- Single Pulse Debounced Circuit (for Hardware)
- Single Pulse Debounced Circuit and Counter (for Hardware)

PRECISE CLOCK GENERATION

PRECISE CLOCK GENERATION

Clock Divider to achieve exactly 1Hz from 100MHz :



Step 1: Determine exactly how many cycles of fundamental clock CLOCK, R, should cause new_clk to flip?

$$R = \frac{\frac{T_{new}}{2}}{T_0} = \frac{f_0}{2 f_{new}}$$

count 0 → R-1

Step 2: create a counter with Maximum bit number (32 bits) that to be able to count up to (R-1), an exact cycle number:

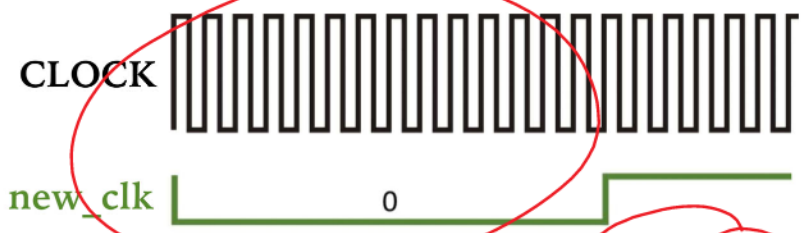
```
reg [31:0] counter = 32'b0; // counter counting from 0 to R-1
```

Step 3: create the sequential logic when there is a CLOCK:

```
always @ begin
    counter <= (counter==R-1) ? 0 : counter + 1;
    new_clk <= (counter==0) ? ~new_clk : new_clk;
end
```

0 ~ R-1

PRECISE CLOCK GENERATION



20MHz

Design File

```
module my_precise_clk(
//100MHz Fundamental Clock
input CLOCK,
//Precisely no. of count R to flip new_clk
//Pre-calculated by flip_count=(f_100M)/(2*f_new)-1
input [31:0] flip_count,
//New clock with frequency f_new
output reg new_clk = 0
);
  //Define a counter with largest size 32bits
  reg [31:0] count = 32'b0;
  //Sequential logic when there is a fundamental CLOCK
  always @ (posedge CLOCK) begin
    count <= (count == flip_count)? 0 : count+1;
    new_clk <= (count == flip_count)? ~new_clk : new_clk;
  end
endmodule
```

Simulation

```
module test_my_precise_clk( );
  reg CLOCK; //Inputs and Outputs of DUT
  reg [31:0]flip_count;
  wire new_clk;
  my_precise_clk DUT (CLOCK,
                      flip_count,
                      new_clk); //Instantiation

  initial begin //Stimuli
  CLOCK = 0; //Provide an initial value for the inputs
  R = 32'd4; //To generate 10MHz slow clock, R=100/(2*10)-1
  end

  always begin
    #5 CLOCK = ~CLOCK;  // Generating a 100MHz CLOCK
  end
endmodule
```

```
// Structural modeling to creat f_new
my_precise_clk dev(CLOCK,32'd    ,new_clk);
```

$R = \frac{f_0}{2f_{new}} = \frac{100\,MHz}{2 \times 4} = 12.5\,M = 12500000$

12499999

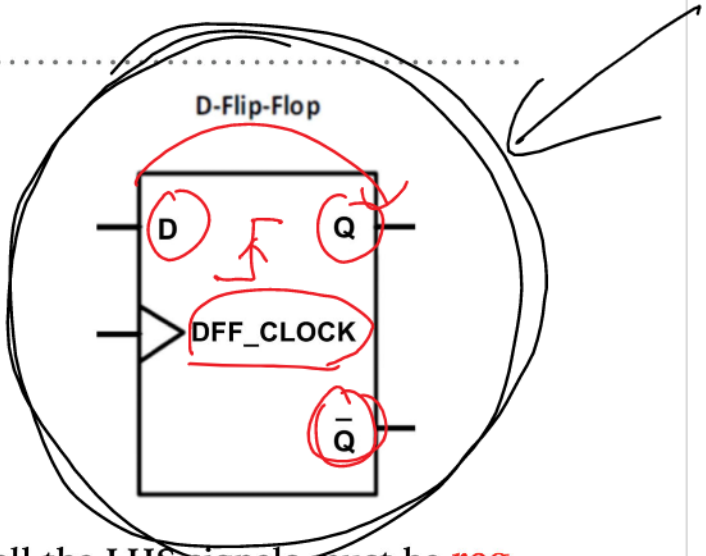
Note: DO NOT run simulate for low frequencies (may take infinite amount of time!)

D-FLIP-FLOP

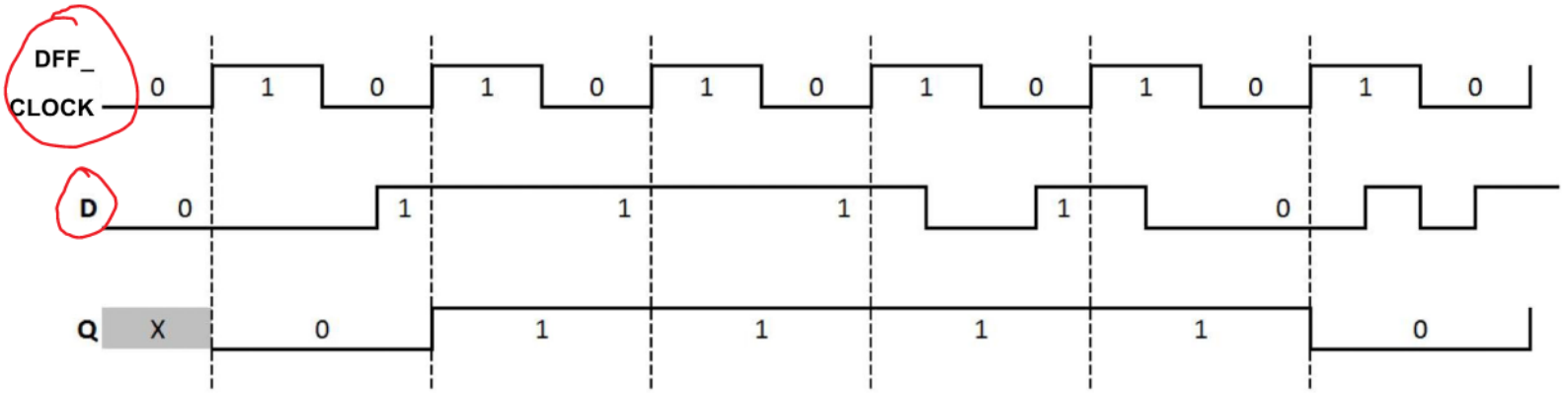
D-FLIP-FLOP: SINGLE DFF

- D-Flip Flop (DFF): "hold the input"

```
module my_dff(input DFF_CLOCK, D, output reg Q = 0);

    always @ (posedge DFF_CLOCK) begin
        Q <= D;
    end

endmodule
```



For sequential circuit, inside "always @ .... end" section, all the LHS signals must be reg



Observation of Q: "hold" the input D even there is a short duration of "bouncing".

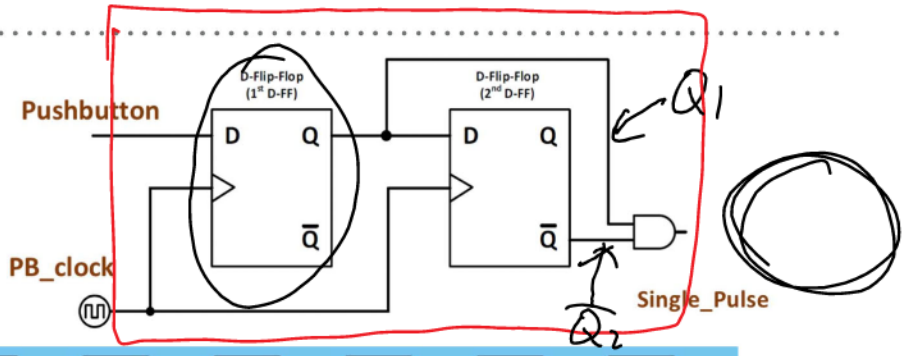
SINGLE PULSE CIRCUIT AND COUNTER

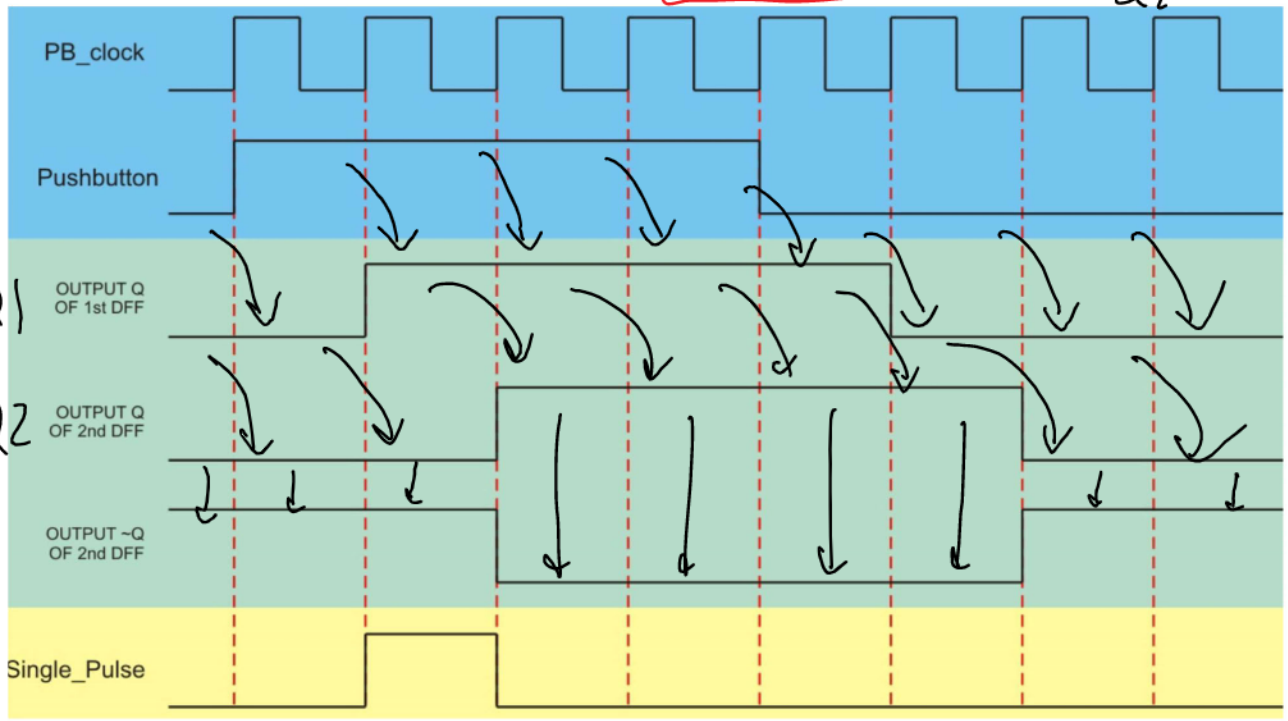
FOR SOFTWARE SIMULATION

SINGLE PULSE CIRCUIT

- Two DFFs in series:

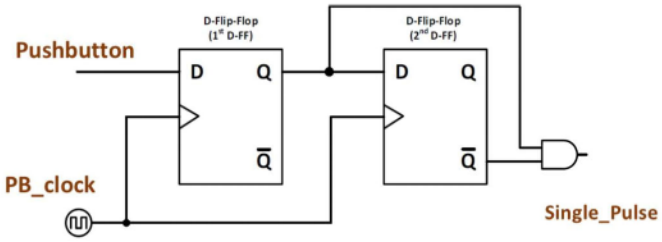
  create single pulse from pushbutton, which lasts for only 1 clock cycle.





SINGLE PULSE CIRCUIT

- Design file: using structural modeling to create single_pulse_circuit.v
- Create a corresponding simulation file to test the output



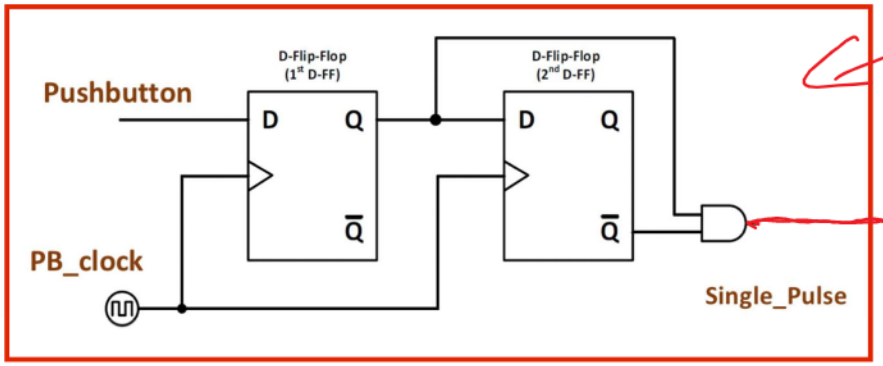

```v
module my_dff(
    input CLOCK,
    input D,
    output reg Q
    );

    always @ (posedge CLOCK) begin
        Q <= D;
    end
endmodule
```

```v
module single_pulse_circuit(
    input clock,
    input PB,
    output single_pulse_out
    );

    wire Q1;
    wire Q2;
    // instantiate two DFF
    my_dff DFF1 (clock,PB,Q1);
    my_dff DFF2 (clock,Q1,Q2);

    assign single_pulse_out = Q1 & (~Q2);

endmodule
```

```v
module sim_single_pulse_circuit(

    );

    // 1. inputs and outputs
    reg sim_clock;
    reg sim_PB;
    wire sim_pb_pulse_out;

    // 2. instantiate the design module you want to test
    single_pulse_circuit DUT (sim_clock, sim_PB, sim_pb_pulse_out);

    // 3. describe your inputs
    initial begin
        sim_clock = 0;
        sim_PB = 0; #15;
        sim_PB = 1; #100;
        sim_PB = 0; #15;
    end


    always begin
        sim_clock = ~sim_clock; #10;
    end


endmodule
```

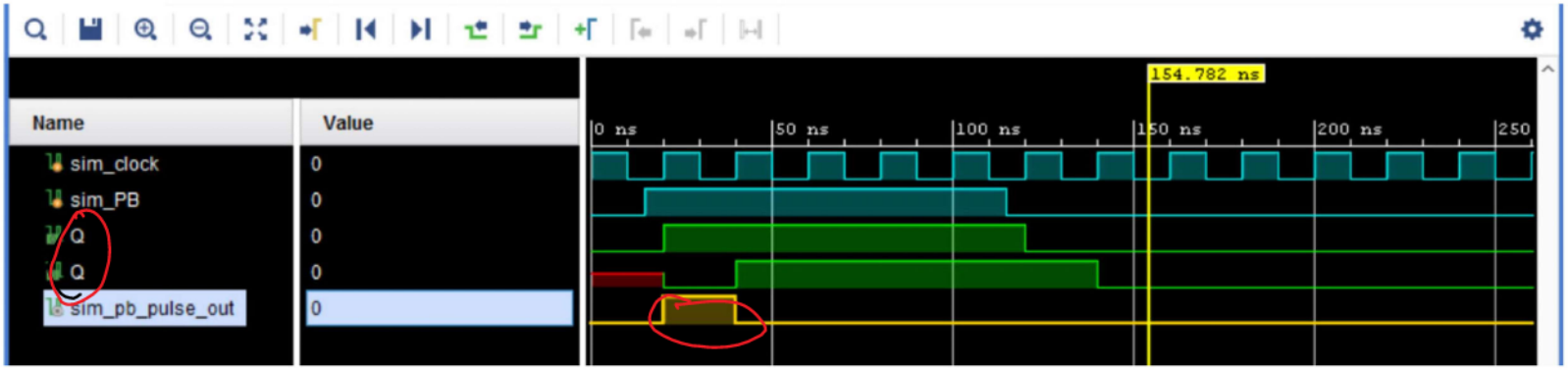
SINGLE PULSE CIRCUIT

- Design file: using structural modeling to create single_pulse_circuit.v
- Create a corresponding simulation file to test the output



count the single pulse

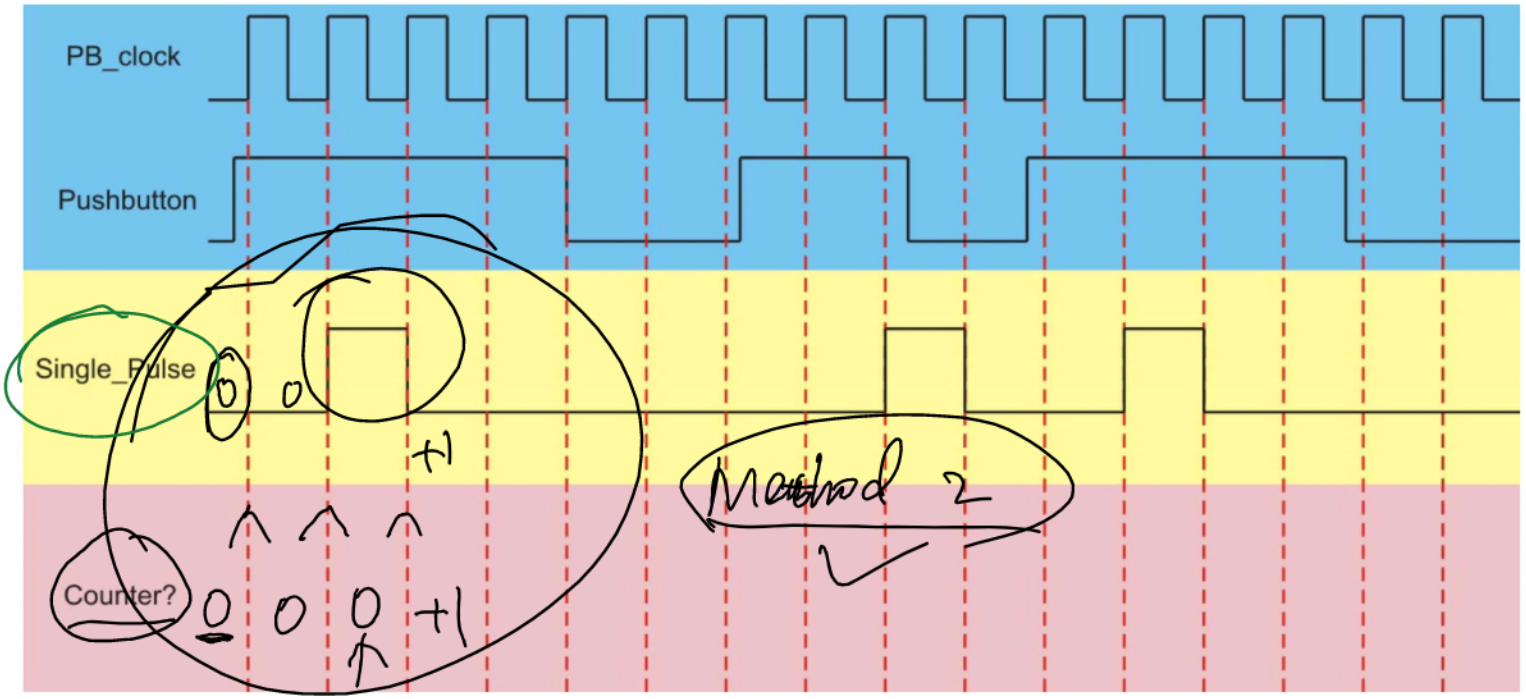
Counter



SINGLE PULSE CIRCUIT FOR COUNTER

Simulation Practice: 8-bit Counter Counting number of Presses of Pushbutton

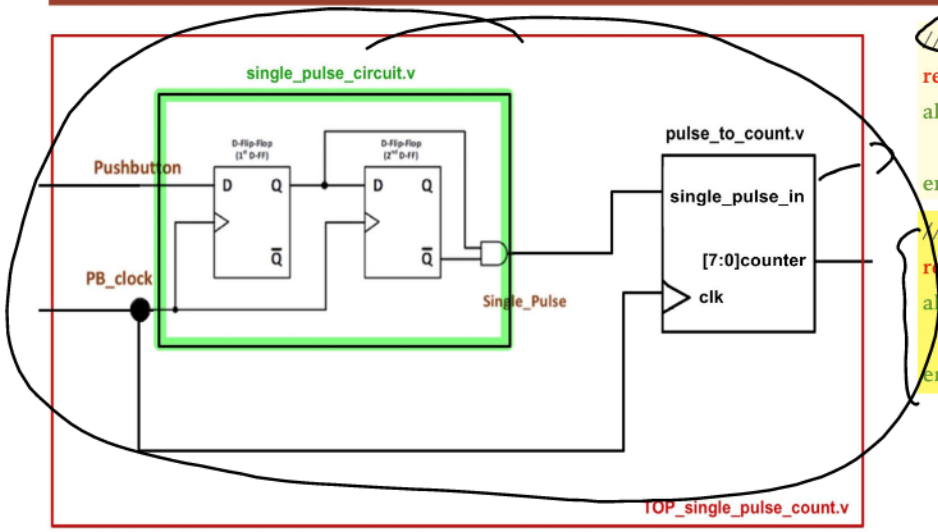
- Increase a counter when there is a button press



*** Clock used for single_pulse_circuit and pulse_to_count must be the same clock! ***

SINGLE PULSE CIRCUIT FOR COUNTER

Simulation Practice: 8-bit Counter Counting number of Presses of Pushbutton




```
// Method 1 for Count
reg [7:0] counter = 8'b0;
always @ (posedge single_pulse_in) begin
    counter <= counter + 1;
end
```

```
// Method 2 for Count (**SUGGESTED**)
reg [7:0] counter = 8'b0;
always @ (posedge clk) begin
    counter <= single_pulse_in? counter + 1 : counter;
end
```

should be a proper clock

```
module pulse_to_count(
    input clk,
    input single_pulse_in,
    output reg [7:0] counter = 8'b0
    );

    always @ (posedge clk) begin
        if (single_pulse_in == 1)
        begin
        counter <= counter + 1;
        end
    end
endmodule
```

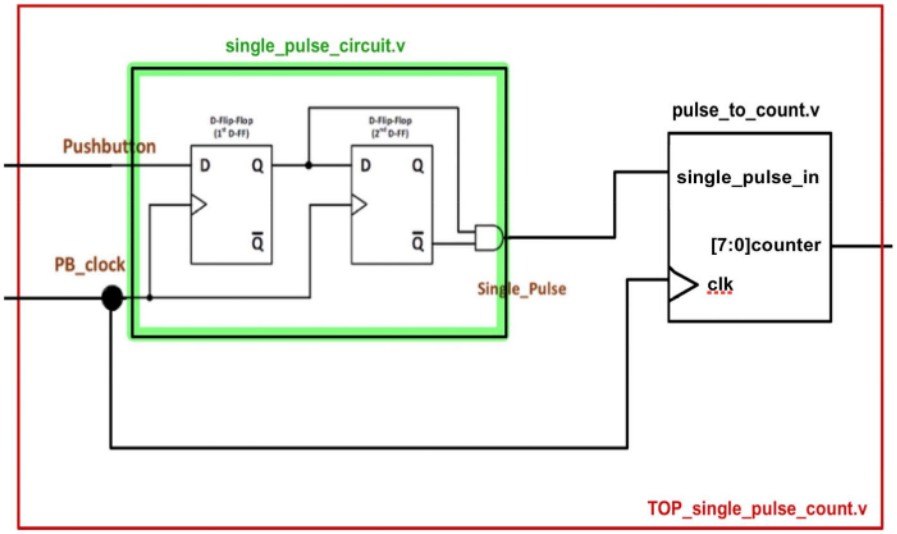
```
module TOP_single_pulse_count(
    input PB,
    input CLOCK,
    output [7:0] COUNT
    );

    wire single_pulse;

    // COMPONENTS INSTANTIATION
    single_pulse_circuit DUT1 (CLOCK,PB,single_pulse);
    pulse_to_count DUT2 (CLOCK,single_pulse, COUNT);

endmodule
```

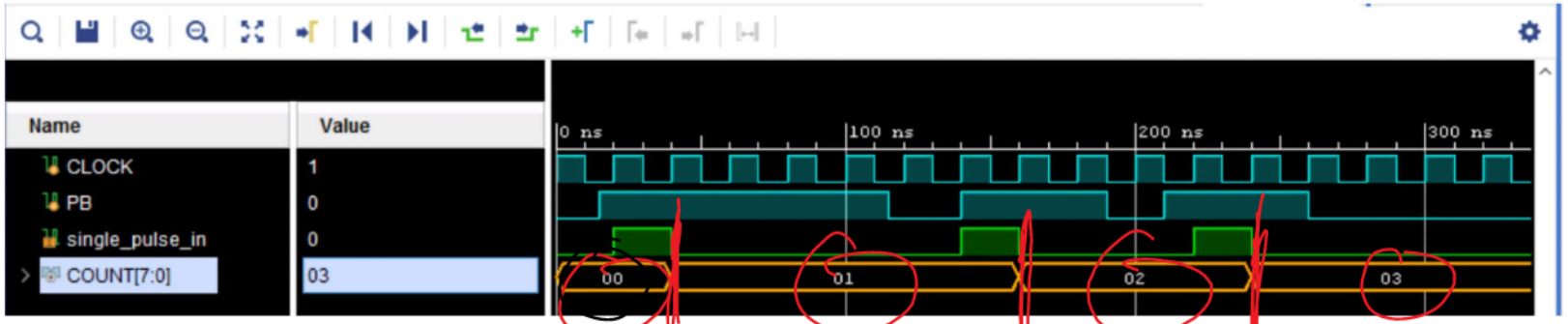
```
module sim_TOP(

    );
    // 1. inputs and outputs
        reg PB;
        reg CLOCK;
        wire [7:0] COUNT;

        // instantiation
        TOP_single_pulse_count DUT (PB,CLOCK,COUNT);

        // 3. describe your inputs
        initial begin
            CLOCK = 0;
            PB = 0; #15;
            PB = 1; #100;
            PB = 0; #15;

            PB = 0; #10;
            PB = 1; #50;
            PB = 0; #10;

            PB = 0; #10;
            PB = 1; #50;
            PB = 0; #10;
        end

        always begin
            CLOCK = ~CLOCK; #10;
        end

endmodule
```

*** Clock used for single_pulse_circuit and pulse_to_count must be the same clock! ***

SINGLE PULSE CIRCUIT FOR COUNTER

Simulation Practice: 8-bit Counter Counting number of Presses of Pushbutton





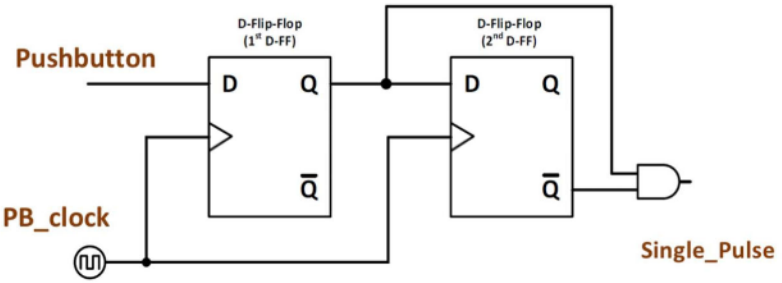
*** Clock used for single_pulse_circuit and pulse_to_count must be the same clock ***

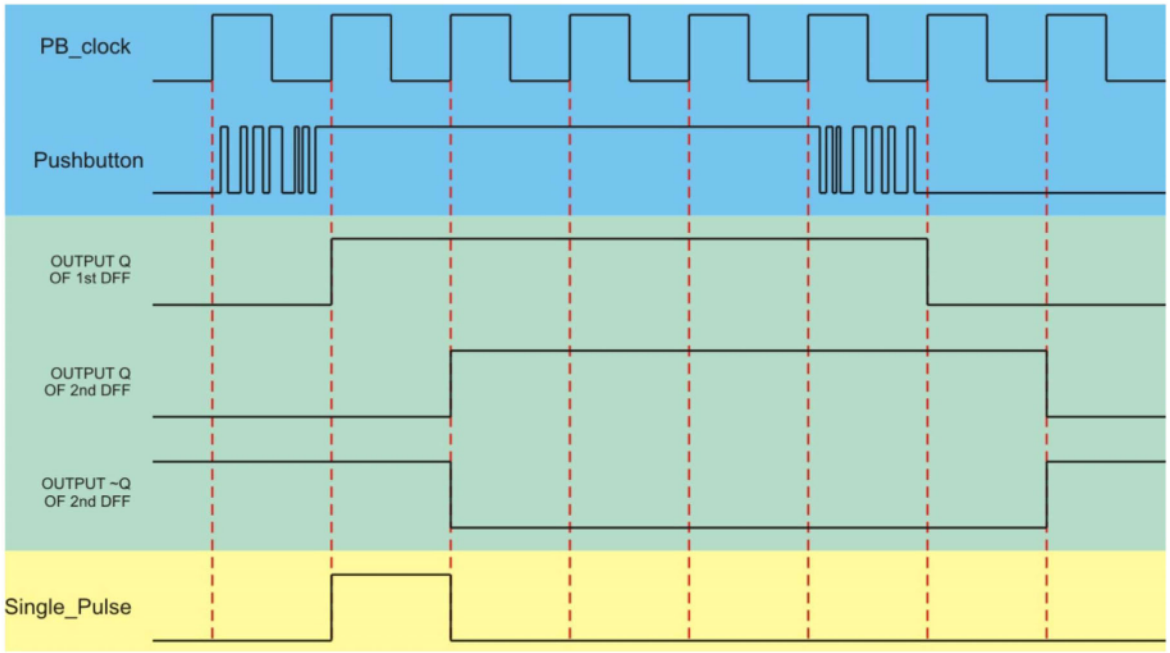
SINGLE PULSE CIRCUIT AND COUNTER

FOR HARDWARE IMPLEMENTATION

SINGLE PULSE CIRCUIT FOR REAL CASE (HARDWARE)

In the real case, pushbutton takes some time to stabilize.



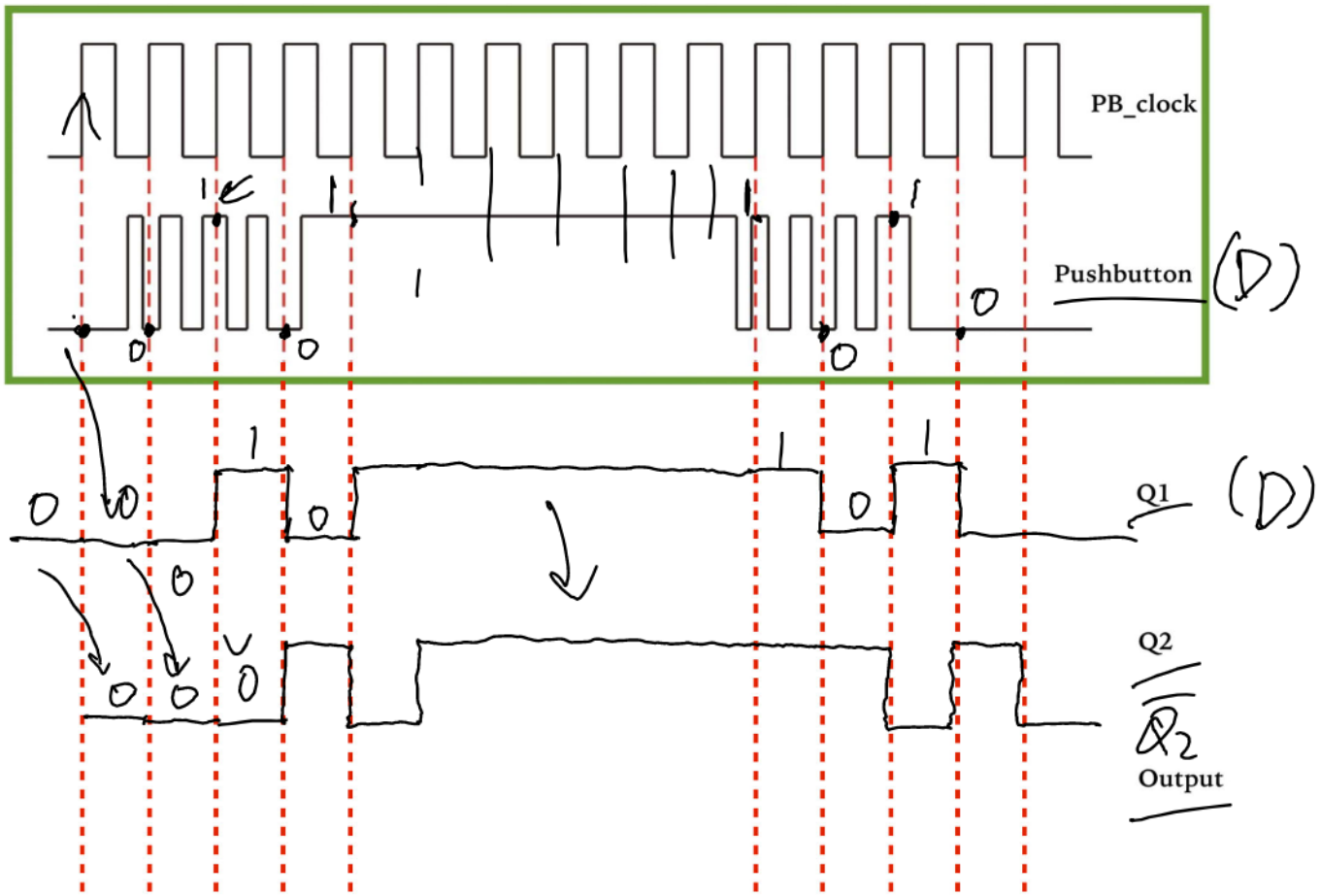


*** PB_clock needs to be much slower than the Pushbutton's bouncing ***

SINGLE PULSE CIRCUIT FOR REAL CASE (HARDWARE)

Reason why PB_clock needs to be much slower than the Pushbutton's bouncing:

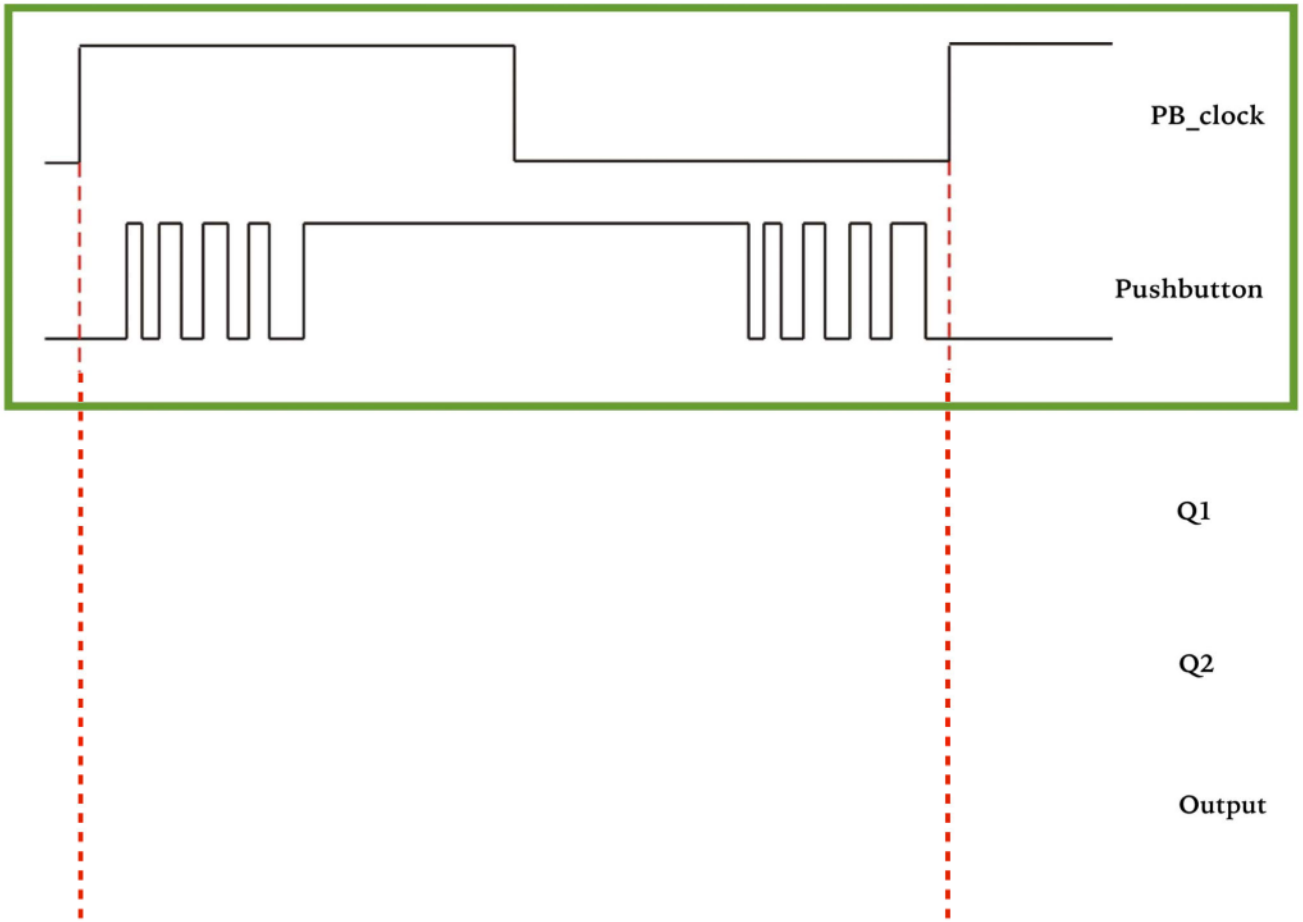
If the PB_clock is too fast:



SINGLE PULSE CIRCUIT FOR REAL CASE (HARDWARE)

Reason why PB_clock needs to be much slower than the Pushbutton's bouncing:

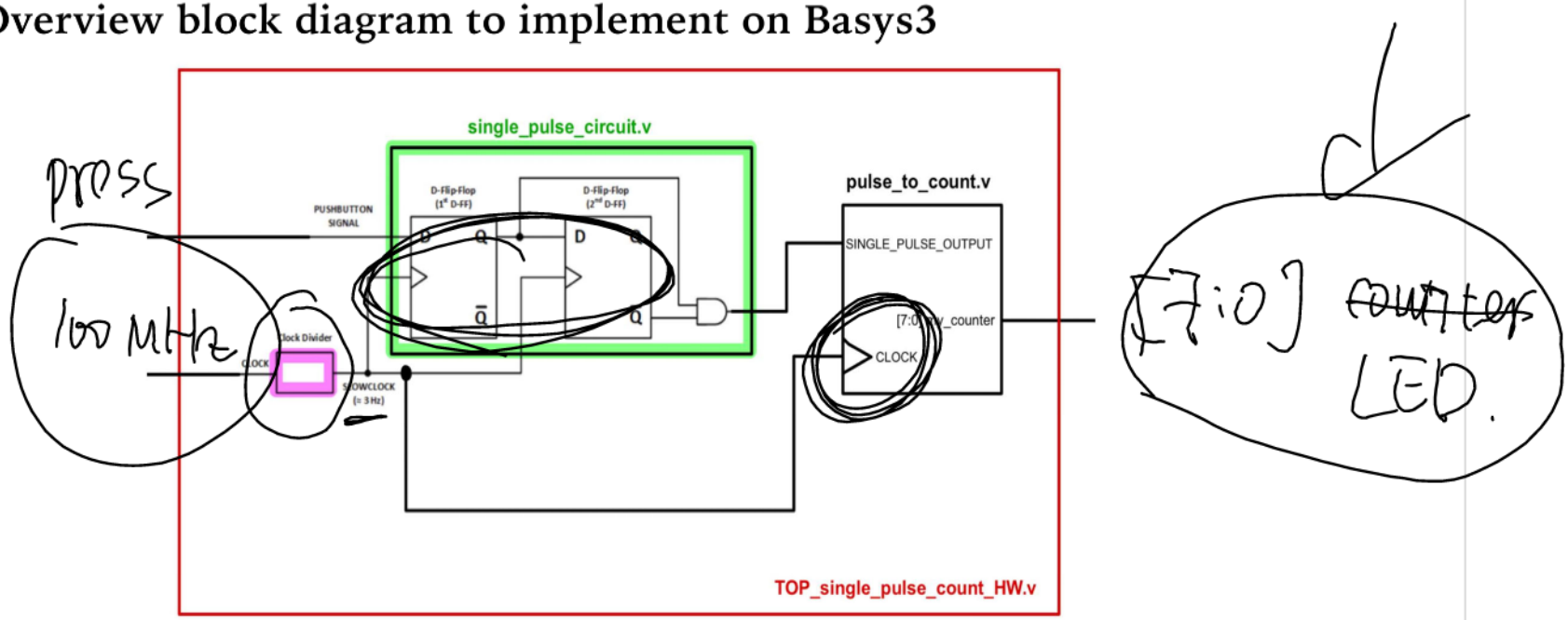
If the PB_clock is too slow:



SINGLE PULSE CIRCUIT FOR COUNTER (REAL CASE / HARDWARE)

Hardware Practice 1: [7:0]LED as a Counter Counting number of Presses of BTNC

- Overview block diagram to implement on Basys3



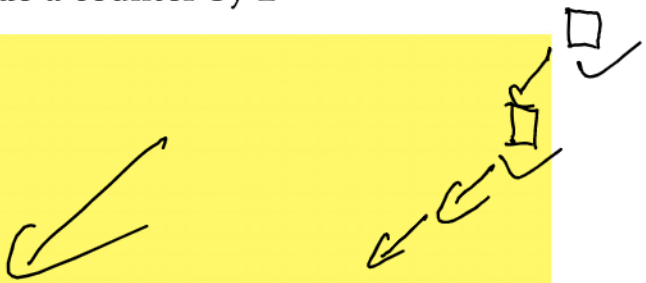
- Clock for counter must be exactly the same clock that was used to create the single pulse output.
- SLOWCLOCK: determines the "sensitivity" of Pushbutton (try 3Hz, 12Hz and 25MHz).

SINGLE PULSE CIRCUIT FOR COUNTER (REAL CASE / HARDWARE)

Hardware Practice 2: LED shifts to left (between LED0 to LED15) when BTNC is pressed

- Shifter operator can be used to multiply / devide a counter by 2^n

```
// Application of Shift Operator

reg [15:0] counter = 16'b0;

always @ (posedge SLOWCLOCK) begin

// double the counter, because shift to left by 1 position
counter <= PB_pulse? (counter <<1) : counter;

// counter value increase by 4 times, because shift to left by 2 positions
counter <= PB_pulse? (counter <<2) : counter;

// counter value is half-ed, because shift to right by 1 position
counter <= PB_pulse? (counter >>1) : counter;
end
```



SINGLE PULSE CIRCUIT FOR COUNTER (REAL CASE / HARDWARE)

➤ List of operators:

Bitwise Operators		
~	~m	invert each bit of m
&	m & n	AND each bit of m with each bit of n
|	m | n	OR each bit of m with each bit of n
^	m ^ n	exclusive-OR each bit of m with n
~^ *or* ^~	m ~^ n	exclusive-NOR each bit of m with n
<<	m << n	shift m left n-times and fill with zeros
>>	m >> n	shift m right n-times and fill with zeros
Unary Reduction Operators		
&	&m	AND all bits in m together (1-bit result)
~&	~&m	NAND all bits in m together (1-bit result)
|	|m	OR all bits in m together (1-bit result)
~|	~|m	NOR all bits in m together (1-bit result)
^	^m	exclusive-OR all bits in m (1-bit result)
~^ *or* ^~	~^m	exclusive-NOR all bits in m (1-bit result)
Logical Operators		
!	!m	is m not true? (1-bit True/False result)
&&	m && n	are both m and n true? (1-bit True/False result)
||	m || n	are either m or n true? (1-bit True/False result)
Equality and Relational Operators (return X if an operand has X or Z)		
==	m == n	is m equal to n? (1-bit True/False result)
!=	m != n	is m not equal to n? (1-bit True/False result)
<	m < n	is m less than n? (1-bit True/False result)
>	m > n	is m greater than n? (1-bit True/False result)
<=	m <= n	is m less than or equal to n? (1-bit True/False result)
>=	m >= n	is m greater than or equal to n? (1-bit True/False result)

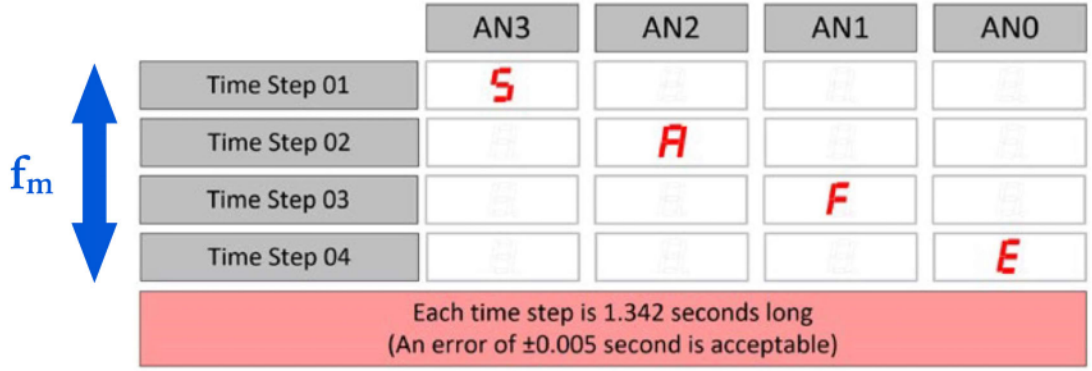
Identity Operators (compare logic values 0, 1, X, and Z)		
===	m === n	is m identical to n? (1-bit True/False results)
!==	m !== n	is m not identical to n? (1-bit True/False result)
Miscellaneous Operators		
? :	sel?m:n	conditional operator; if sel is true, return m: else return n
{}	{m,n}	concatenate m to n, creating a larger vector
{{}}	{n{ }}	replicate inner concatenation n-times
–>	–> m	trigger an event on an event data type
Arithmetic Operators		
+	m + n	add n to m
–	m – n	subtract n from m
–	–m	negate m (2's complement)
*	m * n	multiply m by n
/	m / n	divide m by n
%	m % n	modulus of m / n
**	m ** n	m to the power n (new in Verilog-2001)
<<<	m <<< n	shift m left n-times, filling with 0 (new in Verilog-2001)
>>>	m >>> n	shift m right n-times; fill with value of sign bit if expression is signed, otherwise fill with 0 (Verilog-2001)

ASSIGNMENT 4

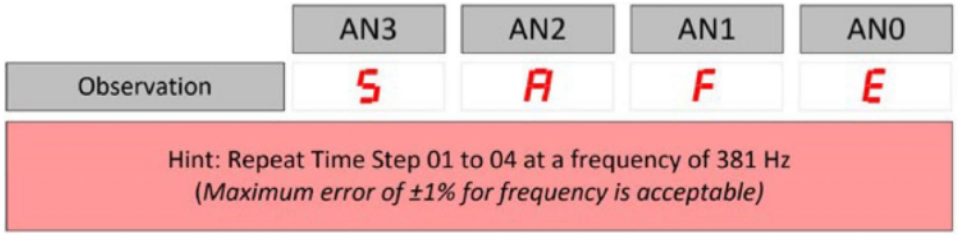
ASSIGNMENT 4: DISPLAY LOA

Sub-Task A: Four 7-SEG Multiplexing with Different f_m

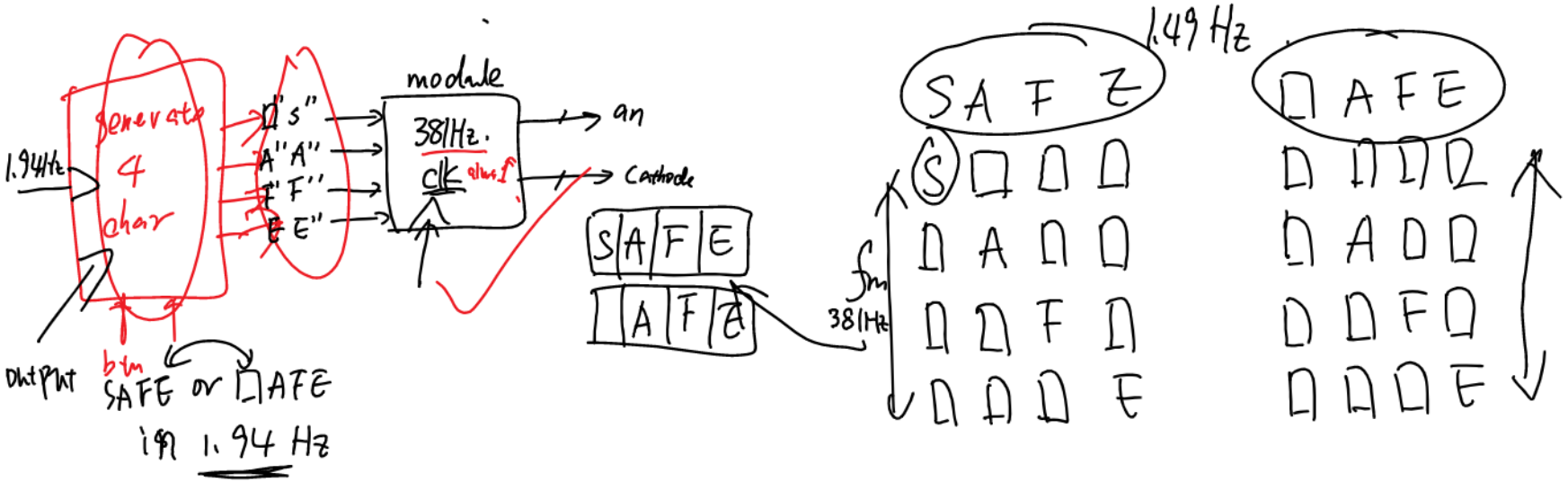
- Display specific character on specific 7-SEG at specific time step
- 'Multiplexing' among the steps 01 to 04, with frequency f_{m_slow} and f_{m_fast}
- Observation: when f_m is large, it seems displaying 4 different characters on four 7-SEG statically



when f_m is fast, e.g. 381Hz



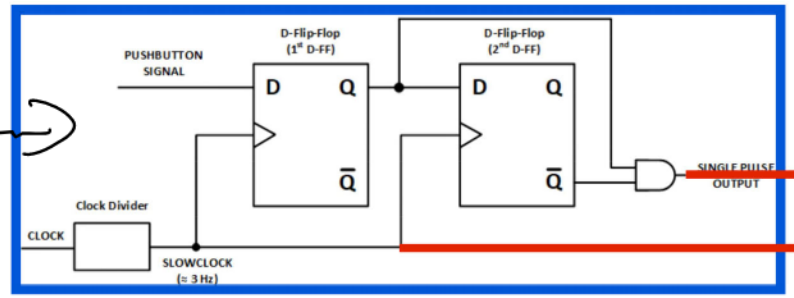
ASSIGNMENT 4:

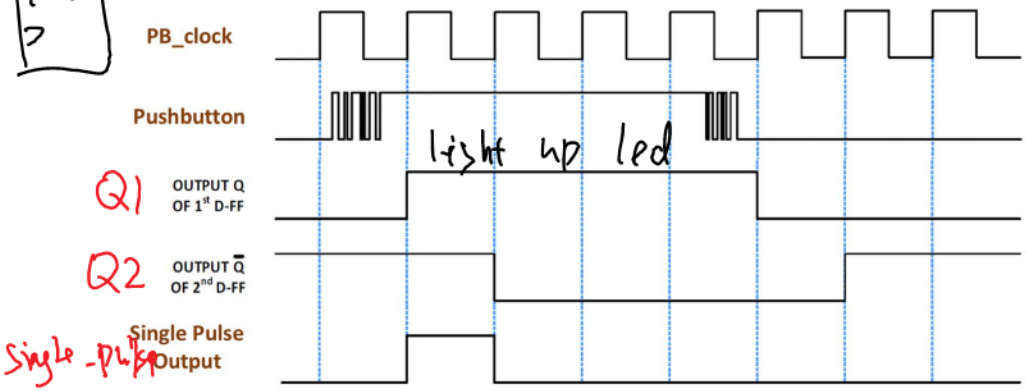


ASSIGNMENT 4: DURING LOA AND QUAR MODE

Sub-Task A & B: Changing 7-Segments Display using Pushbuttons

- Pushbuttons have to be debounced first
- Sense the buttons by using the same PB_clock, to guarantee that ~~each~~ press will be only recognized as 1 press.





Module to sense the button

Similar as "button_to_count.v" created during the lab

ASSIGNMENT 4: DURING LOA AND QUAR MODE

Sub-Task A: Changing 7-Segments Display using Pushbuttons

- To sense if BTNC is held for LEDs to light up from LD0 to LD15, better not create sequential logic based on the edges of BTNC, but still the edge of Clock.

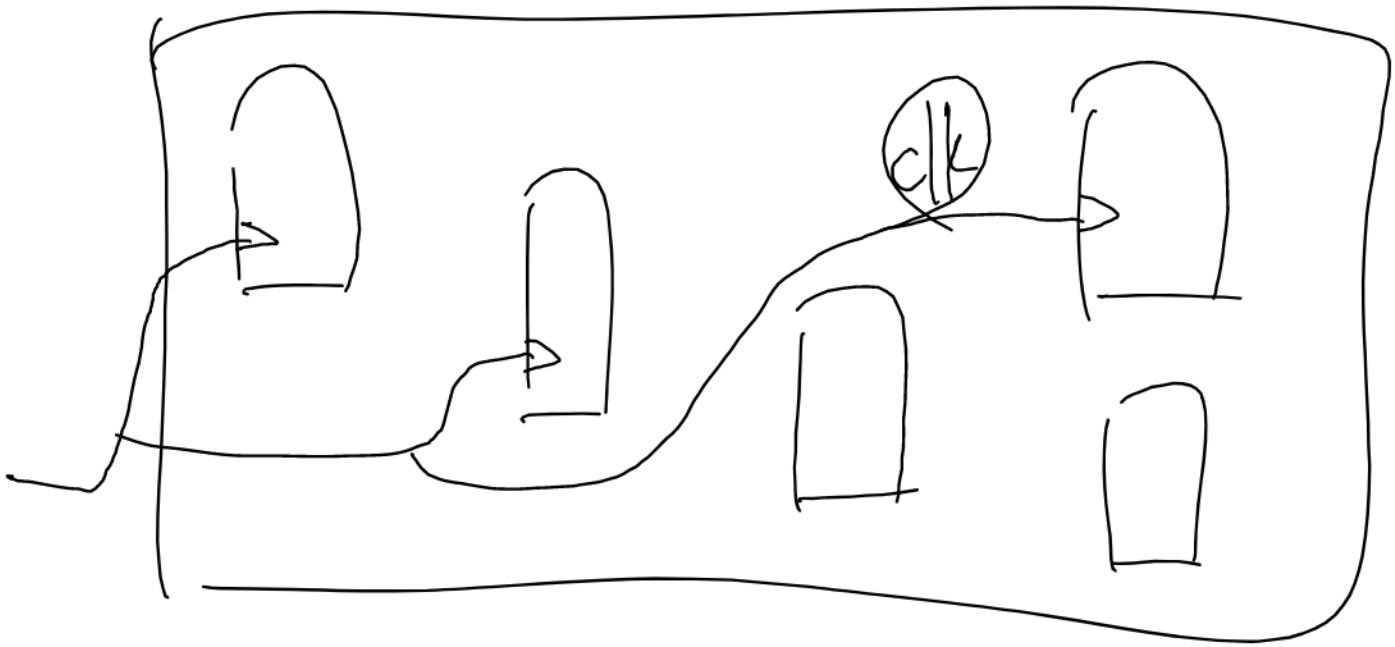
```
// Method 1 for Count
reg [7:0] counter = 8'b0;
always @ (posedge single_pulse_in) begin
  counter <= counter + 1;
end
```

```
// Method 2 for Count (**Much Better Solution**)
reg [7:0] counter = 8'b0;
always @ (posedge clk) begin
  counter <= single_pulse_in? counter + 1 : counter;
end
```

*** Some general rules for Verilog programming:

1. In sequential modules always @(posedge/negedge xxx) begin .... end, xxx should be a proper clock signals to prevent hardware limitations.
2. In one system, try to use the same clock for all sequential modules, so that hardware synchronization can be well retained.

ASSIGNMENT 4:



END

[T]EE2026 Wiki: tiny.cc/ee2026wiki

Email: eleguji@nus.edu.sg