# EE354L

## Divider design

**1. Objective**: To introduce to students

-- RTL coding style for state machine and datapath coding
-- Testbench with a "task"
-- debouncing mechanical Push Buttons and generating DPB, SCEN, MCEN, CCEN
-- Single-stepping and Multi-stepping using the push-button debouncing unit

**2. Files provided**:

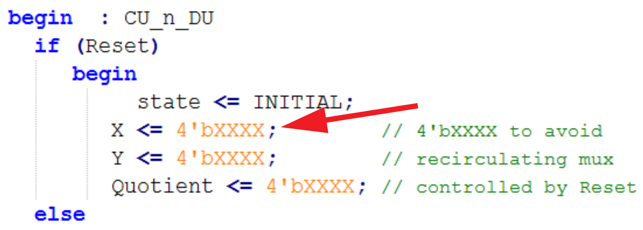
A zip file, `ee354_single_step_synthesis_N4.zip`, is provided containing source files for three sample synthesis designs in three folders. *Please read the notes at the top of each file to get to know important aspects of the design.*

1. `ee354_divider_simple`
2. `ee354_divider_with_debounce`
3. `ee354_divider_with_single-step`

Also another zip file, `ee354_debounce_simulate.zip`, is provided to simulate the debounce_DPB_SCEN_CCEN_MCEN state machine.

A short description of each of the above 3 designs follows.

Extract from divider_combined_cu_dpu.v

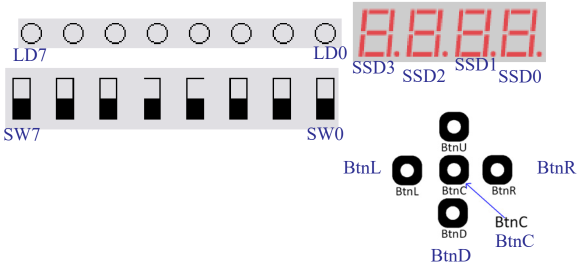


```
begin  : CU_n_DU
  if (Reset)
    begin
         state <= INITIAL;
       X <= 4'bXXXX;              // 4'bXXXX to avoid
       Y <= 4'bXXXX;              // recirculating mux
       Quotient <= 4'bXXXX;  // controlled by Reset
  else
```

**3. ee354_divider_simple:**

Points to note:

The datapath elements shall be inferred by the synthesis tool. So, we do not code OFL explicitly. See the diagram on the next page.



The datapath and the control unit can be combined in one **`case`** statement under clock as shown in `divider_combined_cu_dpu.v`. Notice the lines on the side which avoid unnecessary recirculating muxes.

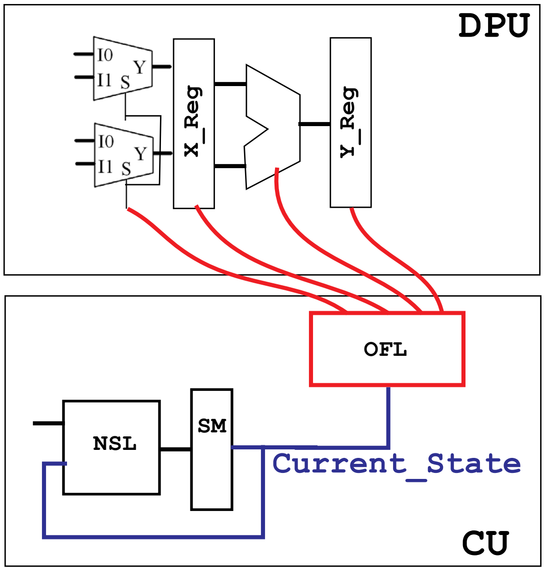
We have also provided another file:

`divider_separate_cu_dpu.v`.

In a big design with 20 states and code running over several pages, readability suffers a lot, if state transitions associated with a particular state are coded in one page in the CU (Control Unit) case statement, and the data transformations for the same state are coded in another page in the DPU (Datapath Unit) case statement. So, we do not recommend industry practice developed for technician engineers. More on this is discussed in the EE254L_RTL_coding_style_verilog.pdf .

## Traditional division between DPU and CU

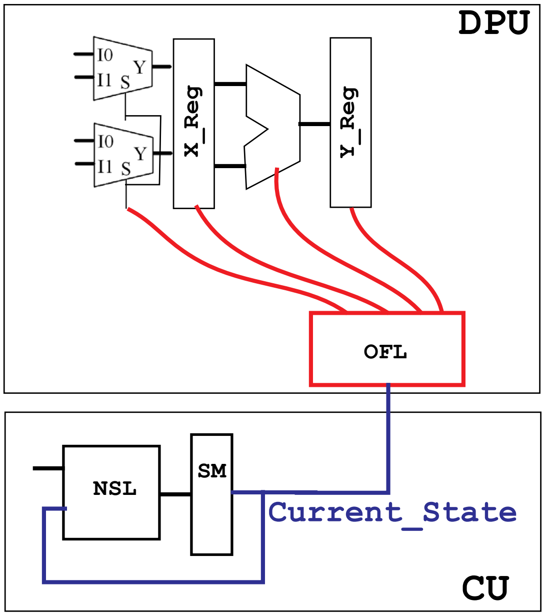
OFL (combinational logic) is in the CU.



## Division between DPU and CU for HDL coding

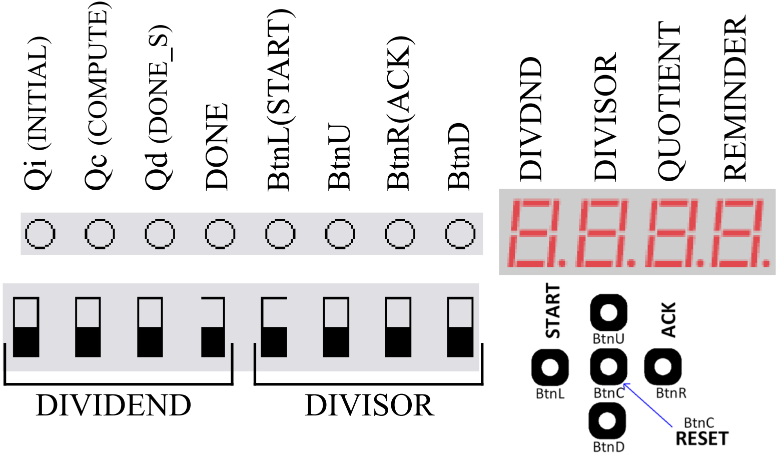
OFL (combinational logic) is moved to DPU. It is NOT coded explicitly.

The OFL is implicit in the DPU's RTL in the CASE statement.



**ee354_divider_simple**

Go through the files and download the provided bit file and test.



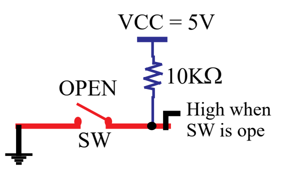
**Questions for the ee354_divider_simple design**:

A. What happens if you divide by zero? Is the behavior of the quotient digit display on SSD1 different if you attempt to divide 3 by 0 vs. if you attempt to divide F by 0. How about 0 divided by 0?

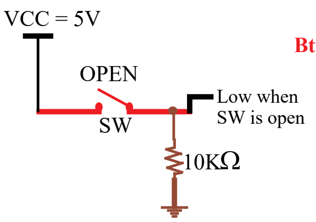
B. If you improve the divider design to move from compute state to done state if X is equal or less than Y (instead of the current X less than Y), will the above behavior change? Does your answer to Q#1 above change?

C. Why does the behavior of the next design (**ee354_divider_with_debounce**) appear to be quite different from this design for division by zero? Is it just appearance only or is it really different? Note: Look at the rate at which sysclk runs in both designs

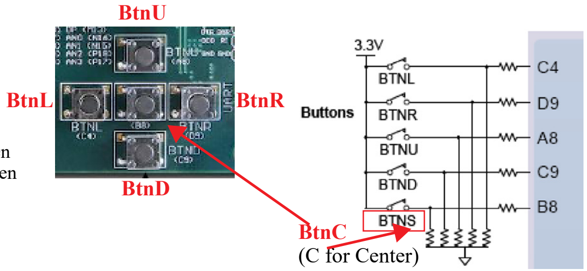
## 4. Bouncing of mechanical Switches and Push Buttons:



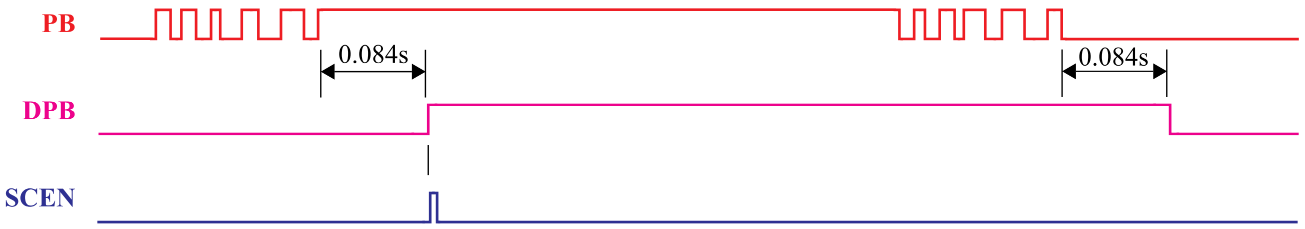
Works for TTL and CMOS logics



Works for CMOS but not TTL logic

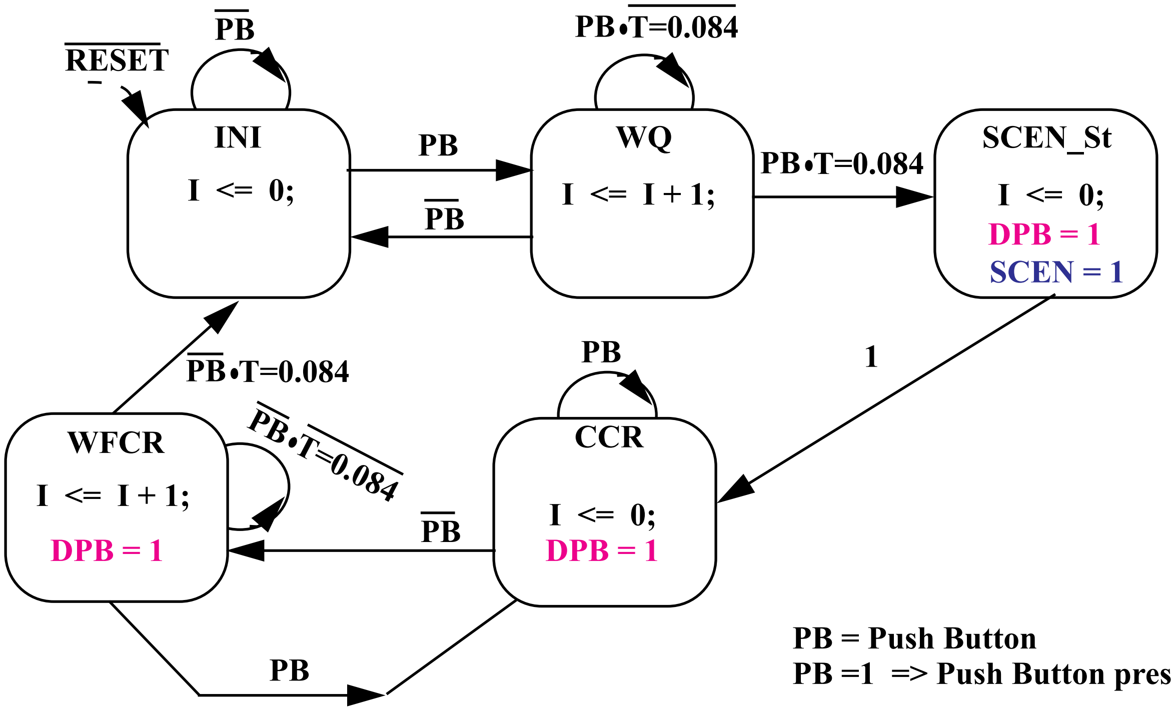


Buttons on Nexys-4:
When pressed, they produce high.



## 5. Debouncing State Machines:

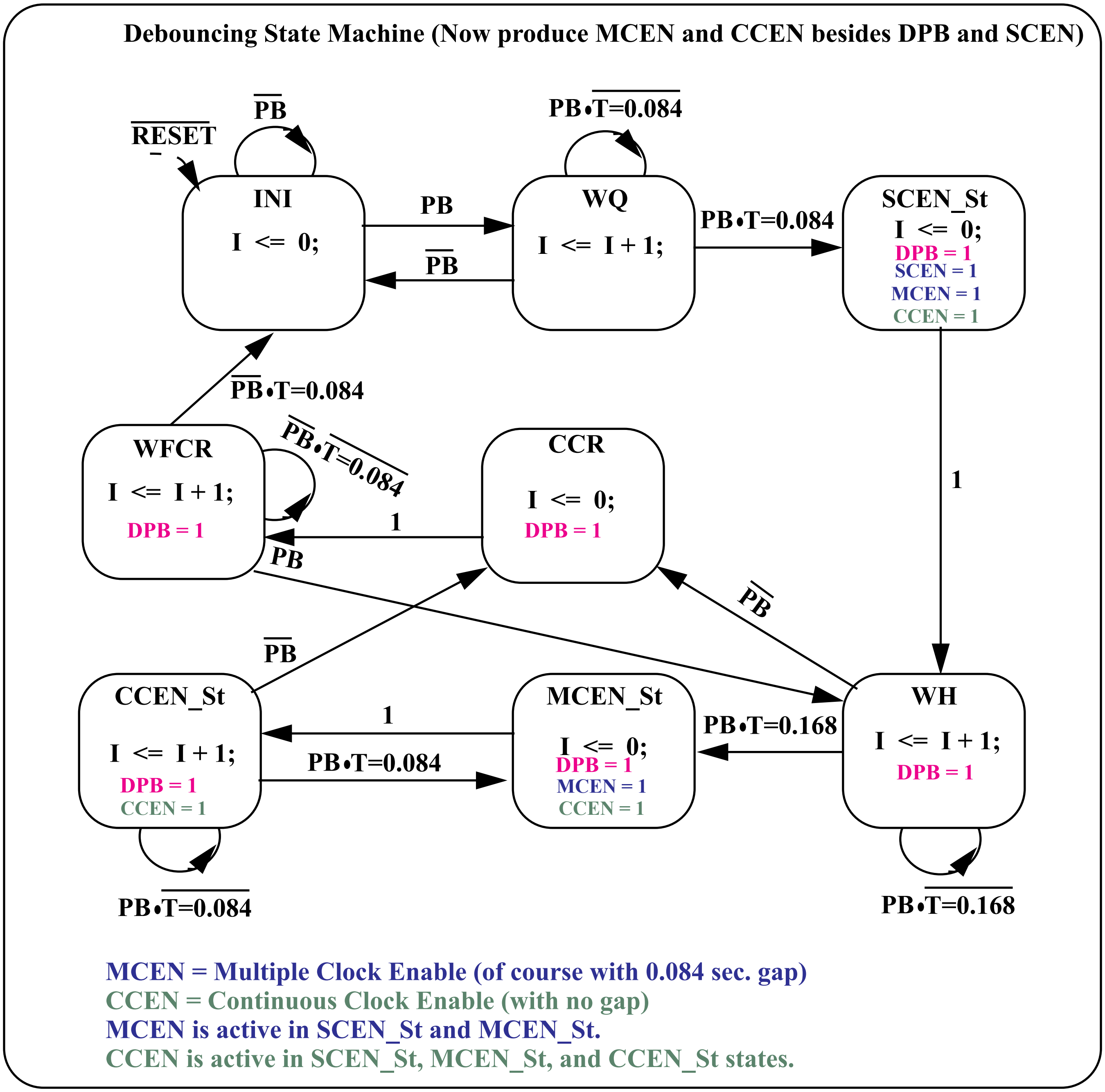
**Debouncing State Machine (To start with just produce DPB and SCEN)**



**PB = Push Button**
**PB =1 => Push Button pressed**

**WQ = Wait for a Quarter Second (actually 0.084ms)**
**SCEN = Single Clock Enable**
(enable the RTL transfer operation and/or state transfer operation for one clock of the 100 MHz system clock)
**CCR - Clear Counter**
**WFCR = Wait For Complete Release**
**DPB = 1 in all states except for INI and WQ states.**
**SCEN = 1 in SCEN_St only. Hence SCEN is a single-clock wide pulse.**

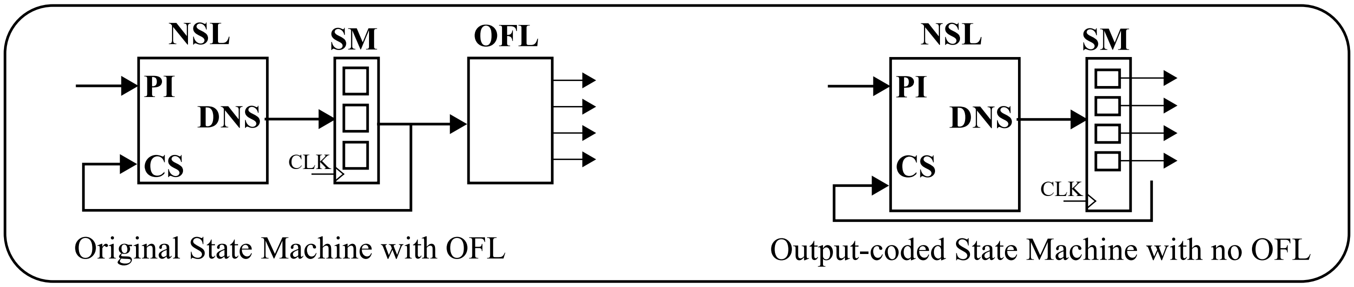
Debouncing State Machine (Now produce MCEN and CCEN besides DPB and SCEN)



MCEN = Multiple Clock Enable (of course with 0.084 sec. gap)
CCEN = Continuous Clock Enable (with no gap)
MCEN is active in SCEN_St and MCEN_St.
CCEN is active in SCEN_St, MCEN_St, and CCEN_St states.

**6. How to produce glitch-free outputs from a state machine**:

Earlier, in class, we showed how easily glitches are produced by a combinational logic such as a mux or an equality checker. If we can avoid the OFL (Output Function Logic) in a Moore kind of state machine by cleverly coding symbolic states using output coding, then the output control signals come out of state flip-flops and they will be glitch free!

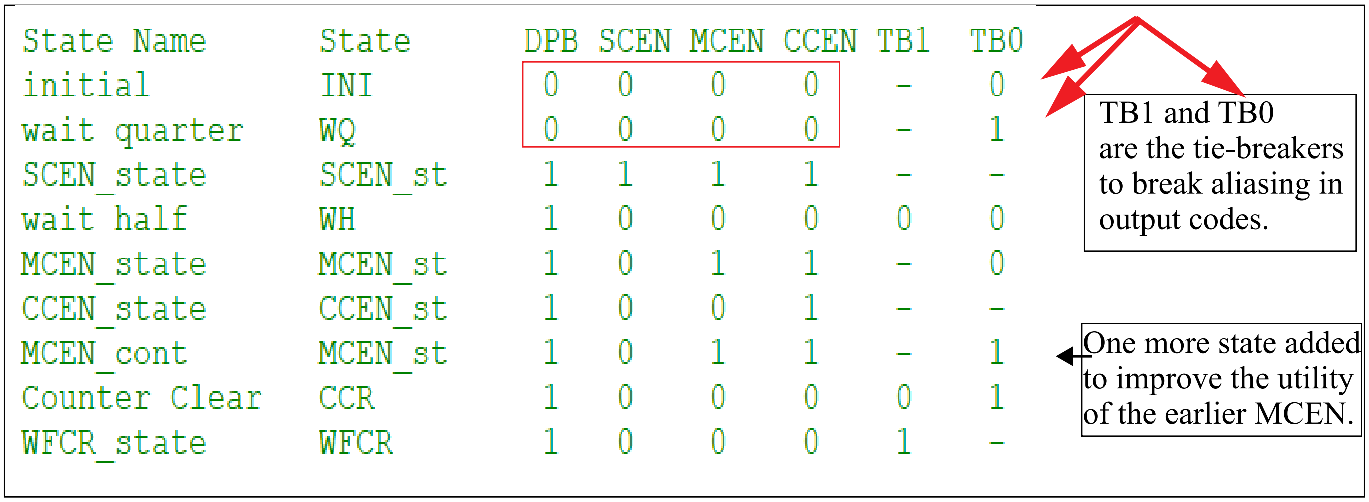


Original State Machine with OFL

Output-coded State Machine with no OFL

**7. ee354_divider_with_debounce**:

Let us go through the debouncer design, ee201_debounce_DPB_SCEN_CCEN_MCEN.v. It debounces a given push button and produces 4 outputs: DPB, SCEN, CCEN, MCEN.
Output coding (for the states in the state machine) is used to produce glitch free outputs.

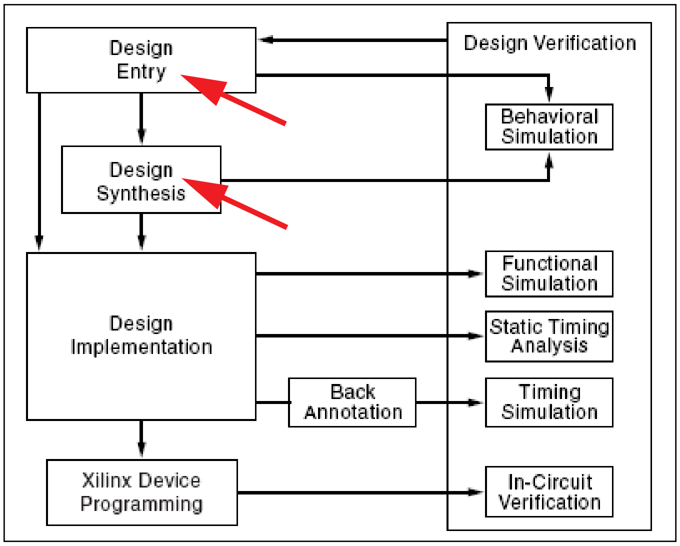


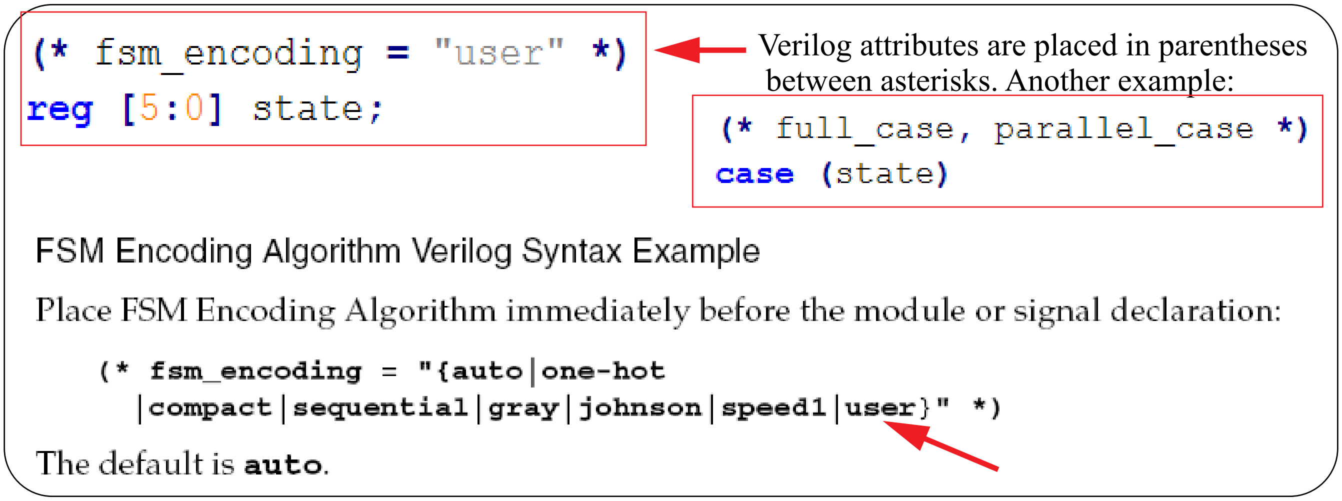
| State Name | State | DPB | SCEN | MCEN | CCEN | TB1 | TB0 |
|---|---|---|---|---|---|---|---|
| initial | INI | 0 | 0 | 0 | 0 | - | 0 |
| wait quarter | WQ | 0 | 0 | 0 | 0 | - | 1 |
| SCEN_state | SCEN_st | 1 | 1 | 1 | 1 | - | - |
| wait half | WH | 1 | 0 | 0 | 0 | 0 | 0 |
| MCEN_state | MCEN_st | 1 | 0 | 1 | 1 | - | 0 |
| CCEN_state | CCEN_st | 1 | 0 | 0 | 1 | - | - |
| MCEN_cont | MCEN_st | 1 | 0 | 1 | 1 | - | 1 |
| Counter Clear | CCR | 1 | 0 | 0 | 0 | 0 | 1 |
| WFCR_state | WFCR | 1 | 0 | 0 | 0 | 1 | - |

TB1 and TB0 are the tie-breakers to break aliasing in output codes.

One more state added to improve the utility of the earlier MCEN.

It appears that the current Xilinx Synthesis tool, Vivado, does not allow user defined FSM encoding to specify the above user defined  In the earlier Xilinx tool set called ISE, it was possible to set FSM Encoding option under
ISE => Synthesis XST => Properties => HDL options => FSM Encoding Algorithm = User.
But this will apply to the entire design!





```
(* fsm_encoding = "user" *)
reg [5:0] state;
```

Verilog attributes are placed in parentheses between asterisks. Another example:

```
(* full_case, parallel_case *)
case (state)
```

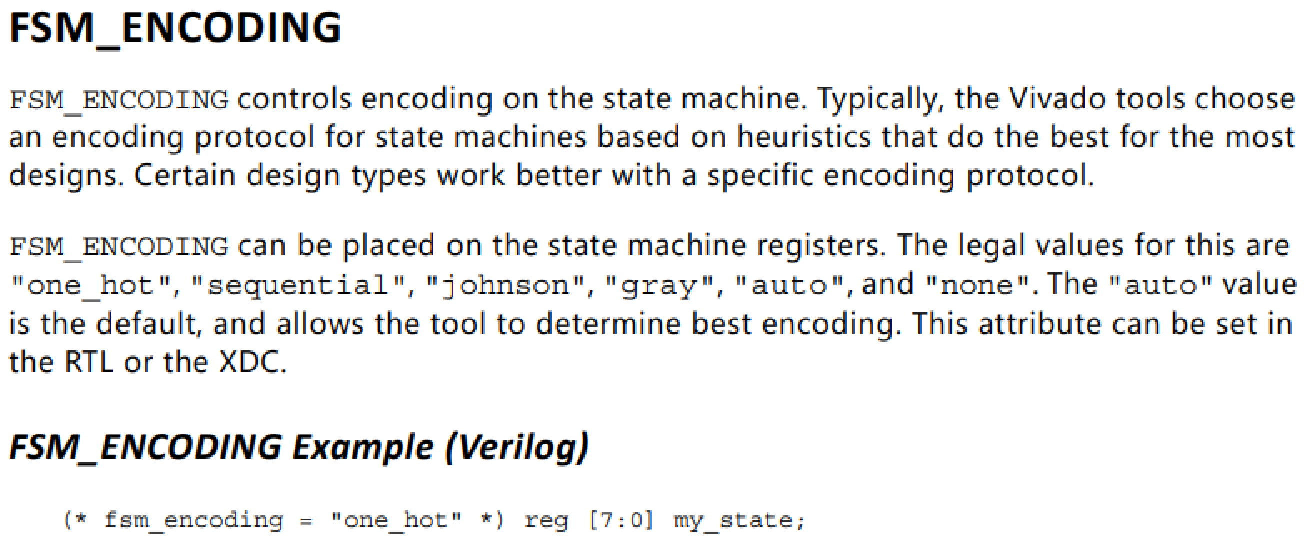
FSM Encoding Algorithm Verilog Syntax Example

Place FSM Encoding Algorithm immediately before the module or signal declaration:

```
(* fsm_encoding = "{auto|one-hot
  |compact|sequential|gray|johnson|speed1|user}" *)
```

The default is **auto**.

The following is an extract from the Vivado synthesis user guide UG901 (xilinx2019_2/ug901-vivado-synthesis.pdf).



## FSM_ENCODING

FSM_ENCODING controls encoding on the state machine. Typically, the Vivado tools choose an encoding protocol for state machines based on heuristics that do the best for the most designs. Certain design types work better with a specific encoding protocol.

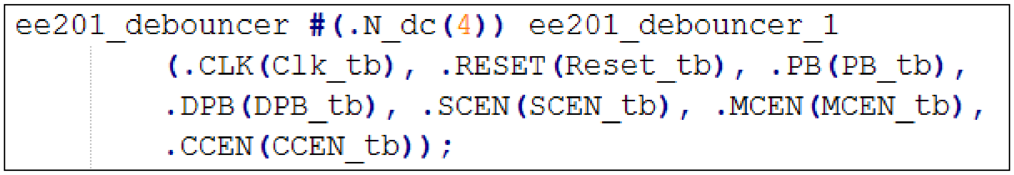
FSM_ENCODING can be placed on the state machine registers. The legal values for this are "one_hot", "sequential", "johnson", "gray", "auto", and "none". The "auto" value is the default, and allows the tool to determine best encoding. This attribute can be set in the RTL or the XDC.

### *FSM_ENCODING Example (Verilog)*

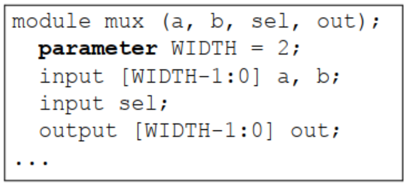
```
(* fsm_encoding = "one_hot" *) reg [7:0] my_state;
```

## Behavioral simulation of the debouncer

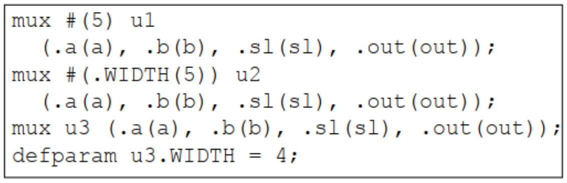
Read the code (ee201_debounce_DPB_SCEN_CCEN_MCEN.v) and complete the state diagram on the next to next page. Simulate it using ee201_debounce_DPB_SCEN_CCEN_MCEN_tb.v for 9 us.



```
ee201_debouncer #(.N_dc(4)) ee201_debouncer_1
        (.CLK(Clk_tb), .RESET(Reset_tb), .PB(PB_tb),
        .DPB(DPB_tb), .SCEN(SCEN_tb), .MCEN(MCEN_tb),
        .CCEN(CCEN_tb));
```



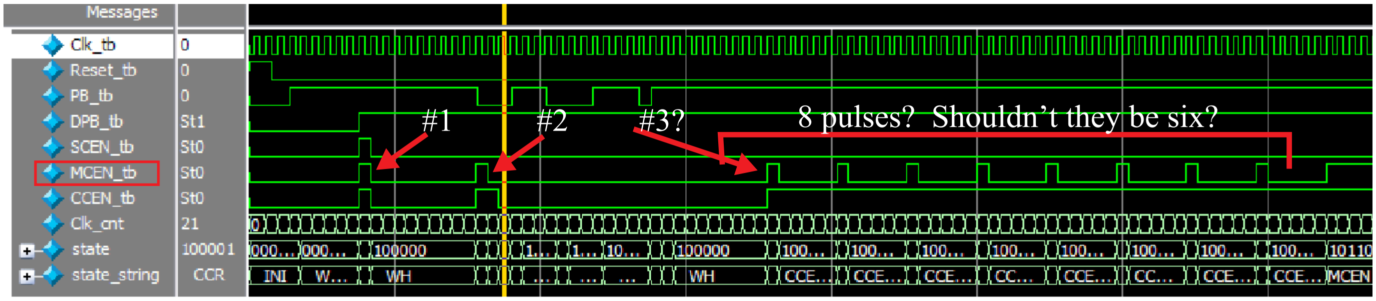
```
module mux (a, b, sel, out);
  parameter WIDTH = 2;
  input [WIDTH-1:0] a, b;
  input sel;
  output [WIDTH-1:0] out;
...
```



```
mux #(5) u1
   (.a(a), .b(b), .sl(sl), .out(out));
mux #(.WIDTH(5)) u2
   (.a(a), .b(b), .sl(sl), .out(out));
mux u3 (.a(a), .b(b), .sl(sl), .out(out));
defparam u3.WIDTH = 4;
```

Notice that, the testbench has instantiated the UUT with the instance label ee201_debouncer_1. Also notice how the N_dc parameter value of 4 is specified in the instantiation. (Refer to pages 95 and 111 of the guide). A simple (rather construed) example of the necessity of the SCEN pulse of debouncer is as follows.

Suppose, we are running short of the buttons on the board and we wish to use a single button (BtnL) both as a START button and an ACK button, Then DPB pulse does not help as our divider is running at full speed (100MHz) and one operation of the BtnL (say 0.2 sec) will be considered as several thousands of these START and ACK operations. So when you let the BtnL go, you can not tell whether the state machine is waiting in the Initial state or Done state! But with SCEN, only one-clock wide pulse per operation is applied to the circuitry! So, when the state machine is waiting in the Initial state and when you press the BtnL, the SCEN pulse produced is treated as a START signal. Similarly, when the state machine is waiting in the Done state and when you press the BtnL, the SCEN pulse produced is treated as a ACK signal.

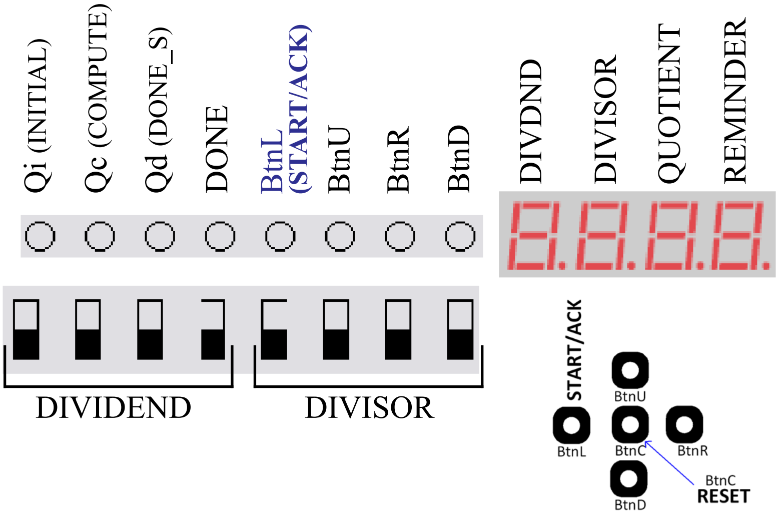


Explain why you see 10 MCEN pulses in the above waveform. When you simulate, you can check if we went to the CCR (Counter Clear) state for any reason. Do you clear just the counter I or also the MC (MCEN Count) in the CCR state?

**ee354_divider_with_debounce**

Go through the files and download the provided bit file and test.

Note that, unlike in the earlier design, (**ee354_divider_simple**), we run the core divider in this design at the **full speed of 100Mhz**.



**Questions on the debouncer and the divider with debouncer**:

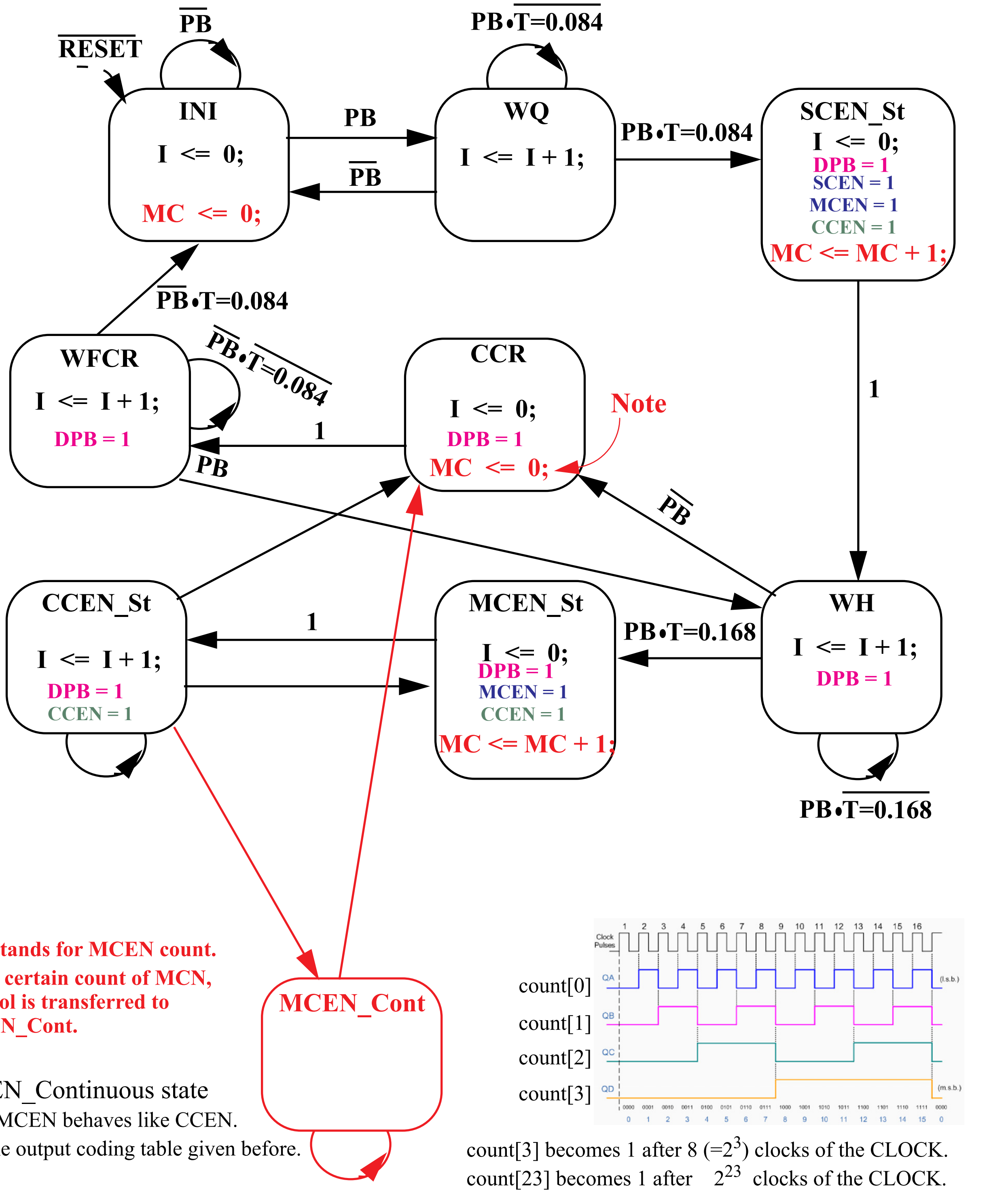
1. Briefly explain why the N_dc parameter was changed to 4 during simulation (from the actual value of 25 for synthesis and implementation). Use words such as “inefficient”, “wasteful”, “readability of waveform”, etc.

2. When you simulate, zoom into the area of above waveform extract and arrive at your answer for the above question in the waveform extract  (why do we see 8 more pulses on MCEN after already seeing two pulses.

3. Did we use the DPB (Debounced Push-Button) pulse or SCEN (Single-Clock enable) pulse to act as the Start signal as well as the Acknowledge signal?  Could we have used anyone of them?

## Complete the Debouncing State Machine with the added state MCEN_cont

**Complete the missing state transition conditions and also any RTL in the state MCEN_Cont**



**MC stands for MCEN count. After certain count of MCN, control is transferred to MCEN_Cont.**

MCEN_Continuous state

Here MCEN behaves like CCEN.

See the output coding table given before.

count[3] becomes 1 after 8 (=$2^3$) clocks of the CLOCK.

count[23] becomes 1 after $2^{23}$ clocks of the CLOCK.

$2^{23}$ clocks each of 10 ns make 0.084 sec. Hence T1 = 0.084 sec

$2^{24}$ clocks each of 10 ns make 0.168 sec. Hence T2 = 0.168 sec

**Nexys4 board clock is at 100MHz.**

100MHz frequency corresponds to 10ns clock period.

**Names of the students submitting:**

**1.**

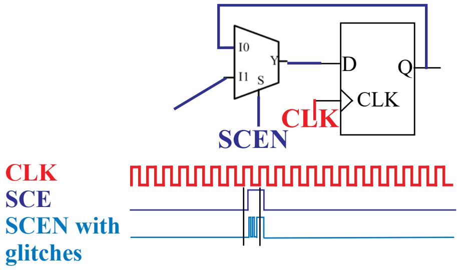
**2.**

## 8. Single-stepping:

Single-stepping and break-point setting are used in software or hardware debugging. Here we wish to show a hardware debugging mechanism involving single-stepping and multi-stepping, which will lead to setting break points. This will be useful particularly when you are interfacing your design with an external system which can not be simulated and proven in simulation. Also sometimes there will be simulation/synthesis mismatches and this helps in debugging in those situations. In later labs, we will also show you chipscope to gather hardware signal activity at full speed. Chipscope is essentially a logic analyzer placed inside the FPGA chip to sample and gather signals and show them to us on the PC monitor as waveforms or state listings.

Let us first talk about single-stepping. Intuitively, the most common idea is to apply one clock pulse at a time whenever the single-step PB is pressed. One can think of using a clean (glitch-free) pulse such as DPB as the clock to the system. However the problem in FPGA is to put this derived clock on global routing resources in FPGA. if interested, read more from the following:
**7-series FPGA Clocking Distribution** page 8/18 in the ds180_7Series_Overview.pdf

If we can not use the global routing resources for our DPB, then this DPB reaches different registers in our design at different times and the relative skew (difference in the arrival times of these clock pulses) causes the circuit to fail. For example, consider a simple right-shift register, with progressively delayed clock sent to the right-side flip-flops. Multi-stepping occurs and the shift register fails. Hence, we designed a better way to implement single-stepping. We do not use DPB or SCEN as "the clock" but we use SCEN as the clock enable. SCEN stands for Single Clock Enable and it is nominally equal in width to a single clock cycle. Since it is the clock enable and controls the data-recirculating mux, even if SCEN has some glitches, they do not hurt the circuit operation. The glitches are in the beginning of the clock and die down by the end of the clock. It is the responsibility of the STA (Static Timing Analyzer, which is part of any synthesis tool) to make sure that the glitches die down before the arrival of the next clock-edge. So, if the circuit passed timing-design, we can be assured that the glitches do not hurt our circuit.



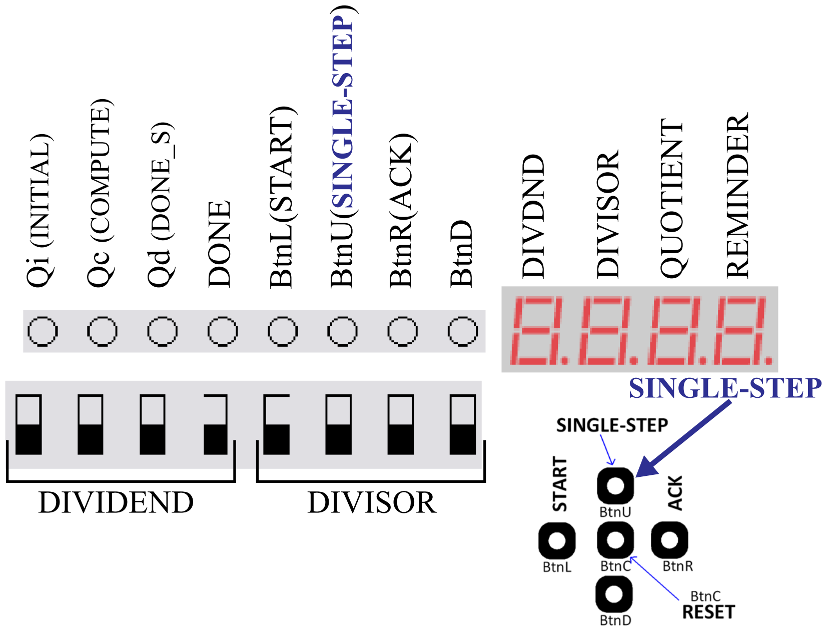
Single-stepping is not a complete solution for debugging as very often, we need thousands or millions of clocks before the suspected malfunctioning part of the circuit behavior can be encountered. For example, a real-time clock (a wall-clock) may misbehave at the roll-over from 23:59:59 to 00:00:00. So, it is a good idea to produce MCEN and CCEN.  We can easily modify the above state diagram to terminate the CCEN or MCEN to force the debounce state machine go back to initial state under any break-point condition (such as time = 23:59:59).

## 9. ee354_divider_with_single_step

**ee354_divider_with_single_step**

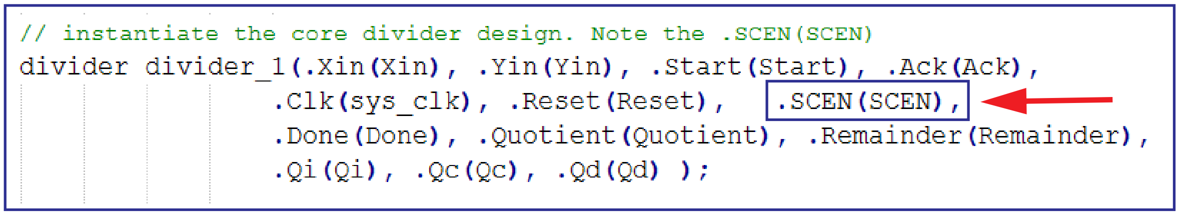
Go through the files and download the provided bit file and test.

Note that, unlike in the first design, (**ee354_divider_simple**), we run the core divider in this design at the **full speed of 100MHz**.



Here, in the compute state, we single-step the division operation using the SCEN produced out of BtnU. Notice the following aspects of the  design.

A. The divider and the divider instantiation have a new port pin called SCEN for the top-level design to generate and pass SCEN pulses (Single-Clock-wide clock enable pulses) (more accurately data-enable pulses as the clock itself is not inhibited).



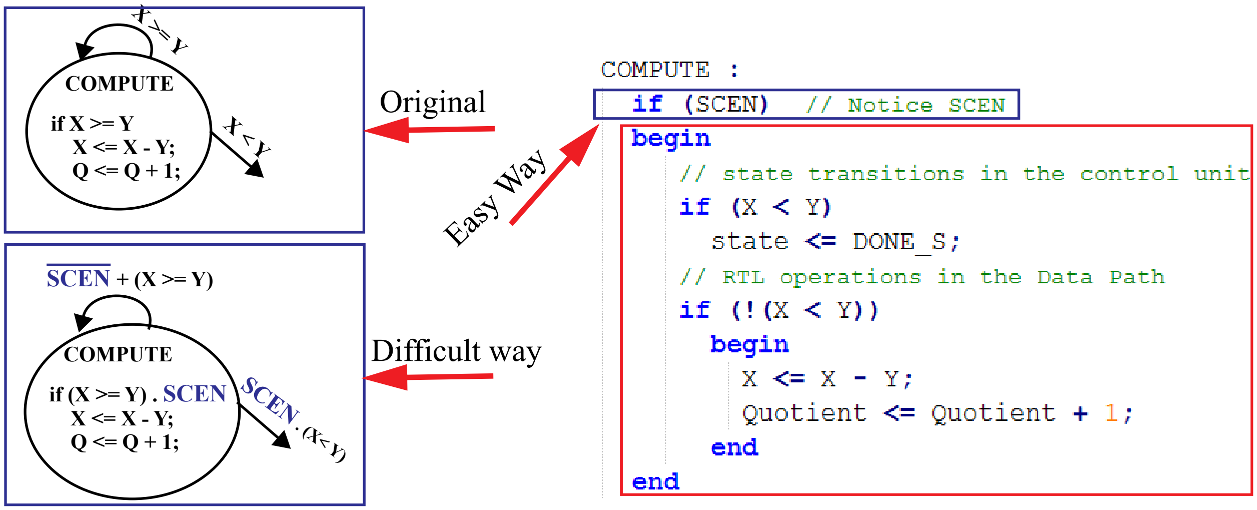
```
// instantiate the core divider design. Note the .SCEN(SCEN)
divider divider_1(.Xin(Xin), .Yin(Yin), .Start(Start), .Ack(Ack),
                  .Clk(sys_clk), .Reset(Reset), .SCEN(SCEN),
                  .Done(Done), .Quotient(Quotient), .Remainder(Remainder),
                  .Qi(Qi), .Qc(Qc), .Qd(Qd) );
```

B. Single-Step Control can easily be exercised on selected states such as the compute state in the divider as shown below. The "if (SCEN)" clause before "begin" ensures that

(i) all state transformations from the COMPUTE state  and

(ii) all data transformations with-in the compute state,

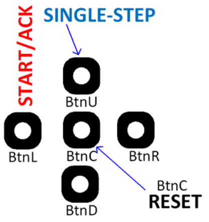
are under the control of SCEN. We do not have to rewrite the state diagram as shown below.



```
COMPUTE :
  if (SCEN)   // Notice SCEN
  begin
     // state transitions in the control unit
     if (X < Y)
       state <= DONE_S;
     // RTL operations in the Data Path
     if (!(X < Y))
       begin
         X <= X - Y;
         Quotient <= Quotient + 1;
       end
  end
```

Questions on **ee354_divider_with_single_step**:

A. Is it possible to use SCEN to control one state (or a few states), MCEN to control another state, and further CCEN to control yet another state? When we say "control a state" here, we mean control the RTL operations in the state and also the state-transitions going away from the state (excluding looping-around state transitions). If we are not going away from the state (because of absence of the SCEN pulse) then we will remain in the state, whether originally there is a loop-around state-transition or not.



B. Can we choose to place **all three states** of the divider design under single-stepping control and *simultaneously* combine Start and Ack under one button (say BtnL)?
Is this just not possible or it works if we produce a BtnL_ SCEN and use it as START as well as ACK, or ...?
Can you press two buttons exactly at the same time to 10ns or 5ns accuracy? Even if you press at the same time to that accuracy, can you guarantee that they bounce for the same length of time and the two instances of the debouncing state machine would produce their respective SCEN pulses at the same time?

C. We took time to design output-coded state machine with no OFL at all, there by avoiding any glitches in the SCEN, MCEN, etc. Are glitches really harmful in our design or we have just shown a way to produce glitch-free outputs?

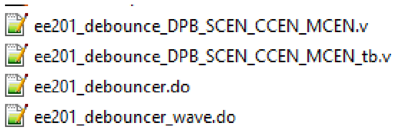
## 10. Task to be performed

1. Download the .zip file provided to you into your C:\xilinx_projects\ directory and extract files to form C:\Xilinx_projects\ee354_single_step_N4 directory with 3 sub-folders:

```
1. ee354_divider_simple
2. ee354_divider_with_debounce
3. ee354_divider_with_single-step
```

All the three folders have Verilog HDL source files, .xdc Xilinx Design Constraint file, a .bit file (with TAs_ prefix) of the completed design .
After reading the code, you can download the .bit file to the Nexys-4 board and operate the divider.
The bit files provided to you have a "TAs_" prefix so that you do not overwrite when you compile the sample designs to get practice in forming a Xilinx Vivado project and implementing the same.

2. Download the zip file, `ee354_debounce_simulate.zip`, and simulate the debounce_DPB_SCEN_CCEN_MCEN state machine using the provided testbench and do file. Files in the .zip file are



ee201_debounce_DPB_SCEN_CCEN_MCEN.v
ee201_debounce_DPB_SCEN_CCEN_MCEN_tb.v
ee201_debouncer.do
ee201_debouncer_wave.do

3. When you are done, please submit your lab report to your TA with your answers to questions posted under the three designs.

## 11. Celebrate your success!!!

Don't forget this step!