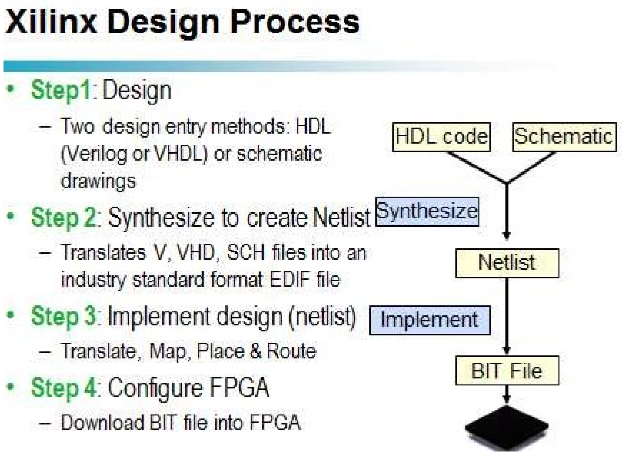
Timing Analysis and Timing Constraints

1. Synopsis:

The objective of this lab is to make you familiar with three critical reports produced by the Xilinx Vivado during your design synthesis and implementation. The lab introduces you to timing constraints and uses a division-by-subtraction example to illustrate “packing” multiple subtractions per clock to utilize the clock period fully and to reduce the number of clocks needed for the complete operation. We also make the timing **tool fail** by having too many subtractions per clock and learn to read the timing reports.
An easy to read XST reference (though for ISE, and not for Vivado) is: http://www2.units.it/marsi/elettronica2/lucidi/XSTsynthesis.ppt

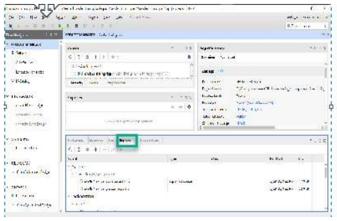
2. Introduction

Translating Verilog code to a configuration bit stream is a three-step process in the Xilinx Vivado. (1) Synthesis using the **Xilinx Synthesis Tool (XST)** is the first step (2) Implementation (3) Generation of the bit stream. Xilinx Vivado generates several reports during these operations to help us understand how the tool inferred (understood) and implemented our design. Knowing how to parse these reports for critical information is a vital part of learning the Xilinx Vivado toolset. The next two sections discuss the information reported in two of these reports.


Xilinx Design Process

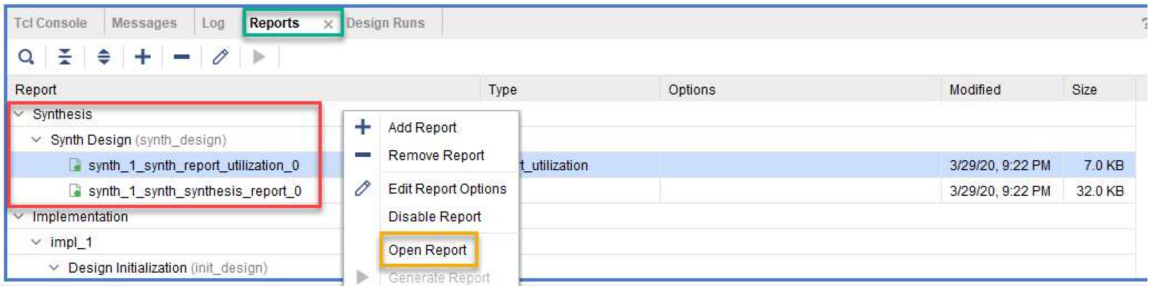
- Step1: Design
  - Two design entry methods: HDL (Verilog or VHDL) or schematic drawings
- Step 2: Synthesize to create Netlist
  - Translates V, VHD, SCH files into an industry standard format EDIF file
- Step 3: Implement design (netlist)
  - Translate, Map, Place & Route
- Step 4: Configure FPGA
  - Download BIT file into FPGA

3. Reading Synthesis (and implementation) Reports:



XST translates behavioral Verilog code to logic components during the first step. Once it completes, it produces a detailed report that you can view. Here, I have opened a completed project for Part 1 and went to the console area (bottom right corner) and clicked on the Reports Tab.

You can right click on any of the reports with the green dot and select “Open Report” as shown below. The report appears in the top right pane.



Report	Type	Options	Modified	Size
Synthesis				
Synth Design (synth_design)				
synth_1_synth_report_utilization_0	..._utilization		3/29/20, 9:22 PM	7.0 KB
synth_1_synth_synthesis_report_0			3/29/20, 9:22 PM	32.0 KB
Implementation				
impl_1				
Design Initialization (init_design)				

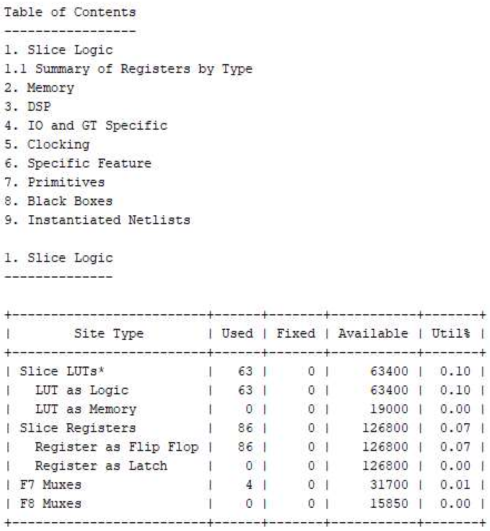
Look at the top right corner of the report window. You can either maximize the window here itself or un-dock (float) it □ ⧉ and then maximize it.

3.1Utilization_synth.rpt

synth_1_synth_report_utilization_0 - synth_1

divider_timing_part1/divider_timing_part1.runs/synth_1/divider_timing_top_utilization_synth.rpt

XST translates behavioral Verilog or VHDL code to logic components during the first step. This section lists the logic components (or "macros") that XST inferred from your code. This report will indicate problems in your design, such as extra flip-flops or latches inferred because of bad coding. The screen-shot on the side is an excerpt from this section after synthesis of part 1 of Timing lab on ARTIX-7 FPGA on our Nexys-4 board (FPGA: xc7a100t-csg324-1). As we can see, our small design occupied about 0.1% of this big FPGA. On this very FPGA, we implement a 4-core processor (with each core running 4 threads) in our EE560 course.


```
Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   63 |     0 |     63400 |  0.10 |
|   LUT as Logic          |   63 |     0 |     63400 |  0.10 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |   86 |     0 |    126800 |  0.07 |
|   Register as Flip Flop |   86 |     0 |    126800 |  0.07 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    4 |     0 |     31700 |  0.01 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

3.2 Synthesis report

synth_1_synth_synthesis_report_0 - synth_1

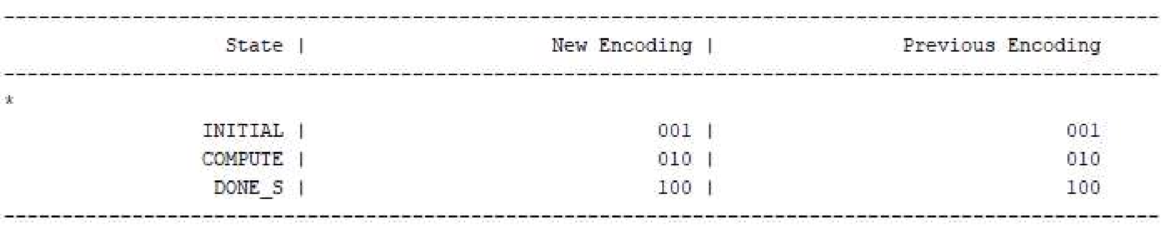
This reports various (potential) problems in our design. Example unconnected outputs. Undriven inputs. Latch inference, for example, usually occurs because of bad coding rather than intentional design to use latches. It lists signals driven with constants to remind us whether it is necessary to have such signals or whether you really meant to do so.

Also nets which are driven by more than one source are listed to let us review if that's what we meant to do to create a shared tri-state bus or a open-collector bus.

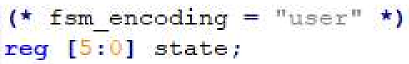
Some examples from our design are listed below.

```
WARNING: [Synth 8-3331] design divider_timing_top has unconnected port Ld15
WARNING: [Synth 8-3917] design divider_timing_top has port Dp driven by constant 0
```

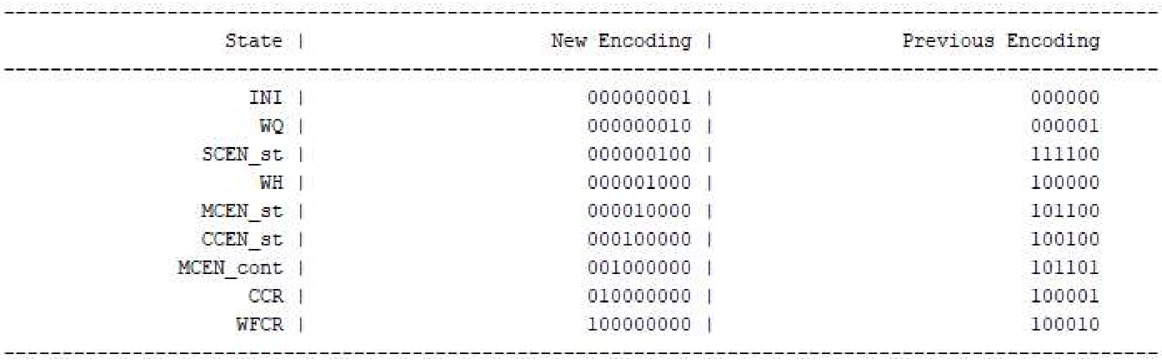
FSM (Finite State Machine) coding used (i.e. whether the state memory FFs were chosen based on encoded state assignment method or one-hot method, etc.) is revealed in this report.


```
---------------------------------------------------------------------------------------------------
                  State |                     New Encoding |                 Previous Encoding
---------------------------------------------------------------------------------------------------
*
                INITIAL |                              001 |                               001
                COMPUTE |                              010 |                               010
                 DONE_S |                              100 |                               100
---------------------------------------------------------------------------------------------------
```

We wanted output-coding for the debouncing state machine and have written the following special attribute.


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

This works in the earlier Xilinx tool called ISE, but in Vivado, this “user” option for coding the FSM state is ignored. It has provided one-hot coded FSM.



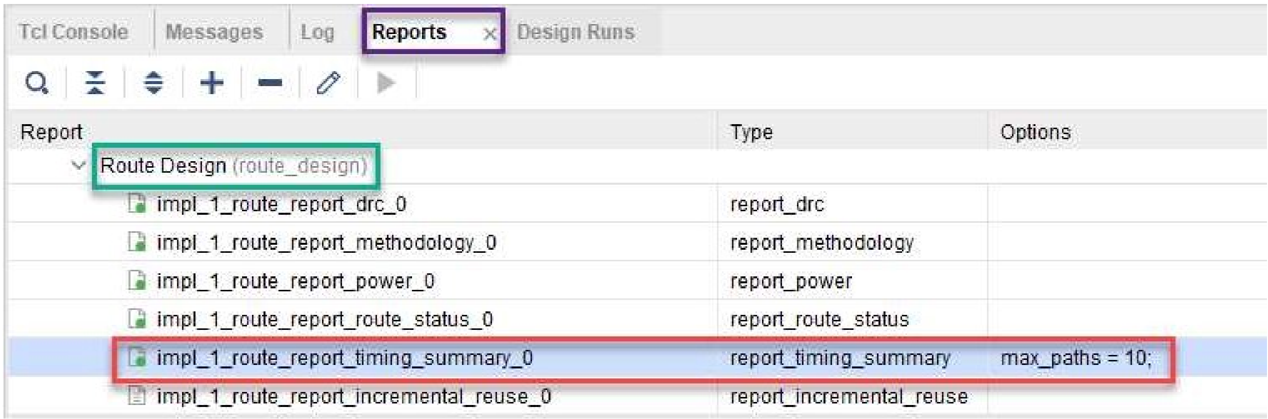
```
---------------------------------------------------------------------------------------------------
                   State |                     New Encoding |                Previous Encoding
---------------------------------------------------------------------------------------------------
                     INI |                        000000001 |                           000000
                      WQ |                        000000010 |                           000001
                 SCEN_st |                        000000100 |                           111100
                      WH |                        000001000 |                           100000
                 MCEN_st |                        000010000 |                           101100
                 CCEN_st |                        000100000 |                           100100
               MCEN_cont |                        001000000 |                           101101
                     CCR |                        010000000 |                           100001
                    WFCR |                        100000000 |                           100010
---------------------------------------------------------------------------------------------------
```

3.3 Report timing summary

impl_1_route_report_timing_summary_0 - impl_1

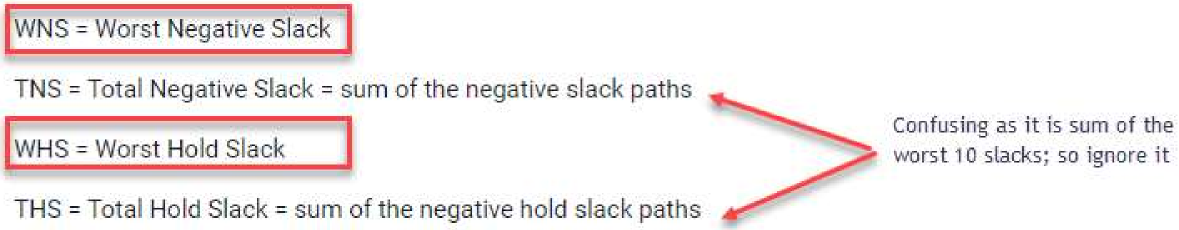
divider_timing_part1/divider_timing_part1.runs/impl_1/divider_timing_top_timing_summary_routed.rpt

Timing Reports are generated in various phases of the tool flow, but the most accurate report is the one produced after place and route.



Report	Type	Options
Route Design (route_design)		
impl_1_route_report_drc_0	report_drc	
impl_1_route_report_methodology_0	report_methodology	
impl_1_route_report_power_0	report_power	
impl_1_route_report_route_status_0	report_route_status	
impl_1_route_report_timing_summary_0	report_timing_summary	max_paths = 10;
impl_1_route_report_incremental_reuse_0	report_incremental_reuse	

Place-and-Route is the final step before the tool generates a configuration file (.bit file) for the FPGA. In this step the Xilinx tool maps the circuit to physical locations in the FPGA and creates the signal routes that connect various logic elements. Routes contribute almost half of the latency in the circuit. So only after Place-and-Route is complete the tools can compute the precise delay of each path. The portion of the Place-and-Route report explaining the timing constraints is shown below. This shows the WNS (Worst Negative slack or Worst Case Setup time Slack). WNS is the difference between the clock period and the delay between a pair of registers. A positive worst case setup time slack means the constraint is met and a negative slack means that the longest path has a path delay longer than the clock period of the circuit.



WNS = Worst Negative Slack

TNS = Total Negative Slack = sum of the negative slack paths

WHS = Worst Hold Slack

THS = Total Hold Slack = sum of the negative hold slack paths

Confusing as it is sum of the worst 10 slacks; so ignore it

By default a few (10) worst paths are reported. The longest path delay determines the maximum frequency at which the design can operate. The report contains more details about the timing of these paths. A timing path originates at a given starting point (usually the Q output of a source FF, but more accurately the starting point is the clock trigger at the clock input to the source FF, so that we can take into account the tffpd of the source FF), goes through some combinational logic, and then to a certain terminal point (usually the D input to the destination FF).

Xilinx user guide “UG906 Design Analysis and Closure Techniques” is a good reference for the teaching team and the EE560 students. The following figures are extracted from this guide. These figures (from pages 219, 220, and 223) help us in reading and interpreting the Timing report generated by XST.
I have annotated these figures with the numbers from the timing report produced in part 3 of our lab, so that we can relate the report with the concepts depicted in these figures.

Figure 174: **Timing Paths Example**

Annotations from part3 with 8 subtractions in one clock!

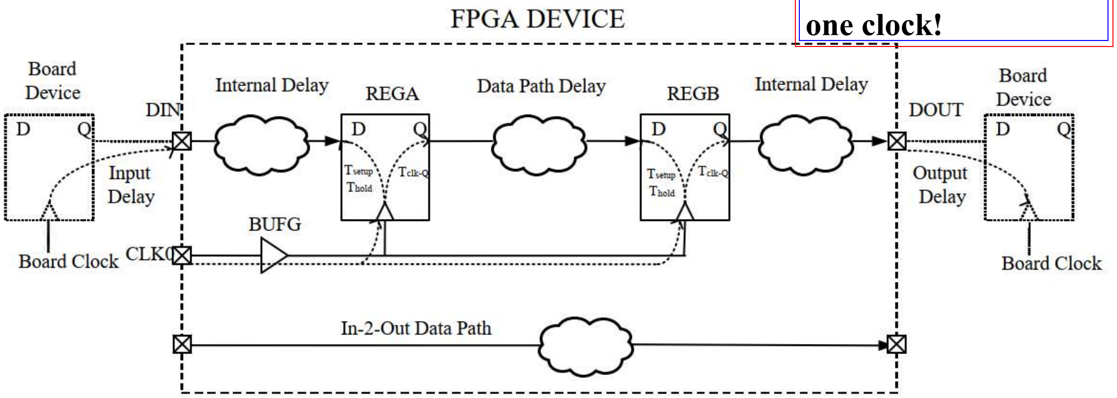
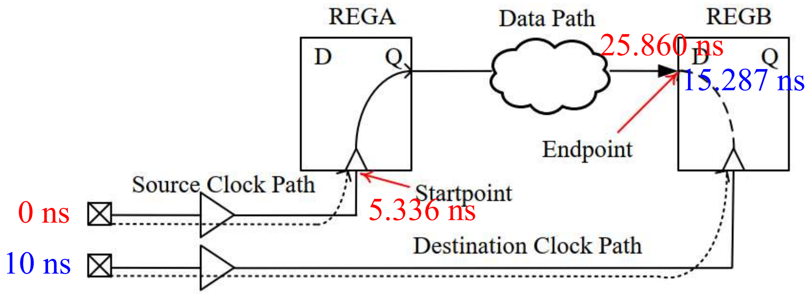


Figure 175: **Typical Timing Path**

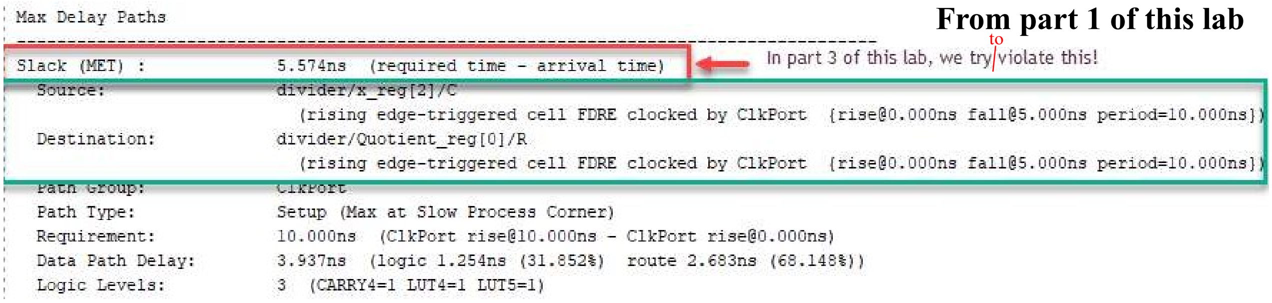


Note: Negative setup time specification is nonintuitive but possible! Based on the relative path delays of the clock and D-input of the FF. Some of the Xilinx FFs have negative setup time spec.!

Data Required Time (setup) **15.287 ns**	=	**capture edge time** 10 ns **+ destination clock path delay** +5.292 ns **- clock uncertainty** -0.035 ns **- setup time** **- (-0.031 ns)**
Data Arrival Time (setup) **25.860 ns**	=	launch edge time 0 ns + source clock path delay +5.336 ns + datapath delay +20.324 ns
Slack (setup) **-10.572 ns**	=	Data Required Time - Data Arrival Time **15.287 ns - 25.860 ns**

Slack is negative, hence setup time check failed (violated).

In the timing report, XST provides ten worst path information by default, so that we do not reduce the delay of one path of the design only to have another path slowed to the point where it becomes the longest path. Following is an excerpt from the timing report that details the delay of a single path. While we can easily identify the source and destination in this path, the intermediate signal names are obfuscated.



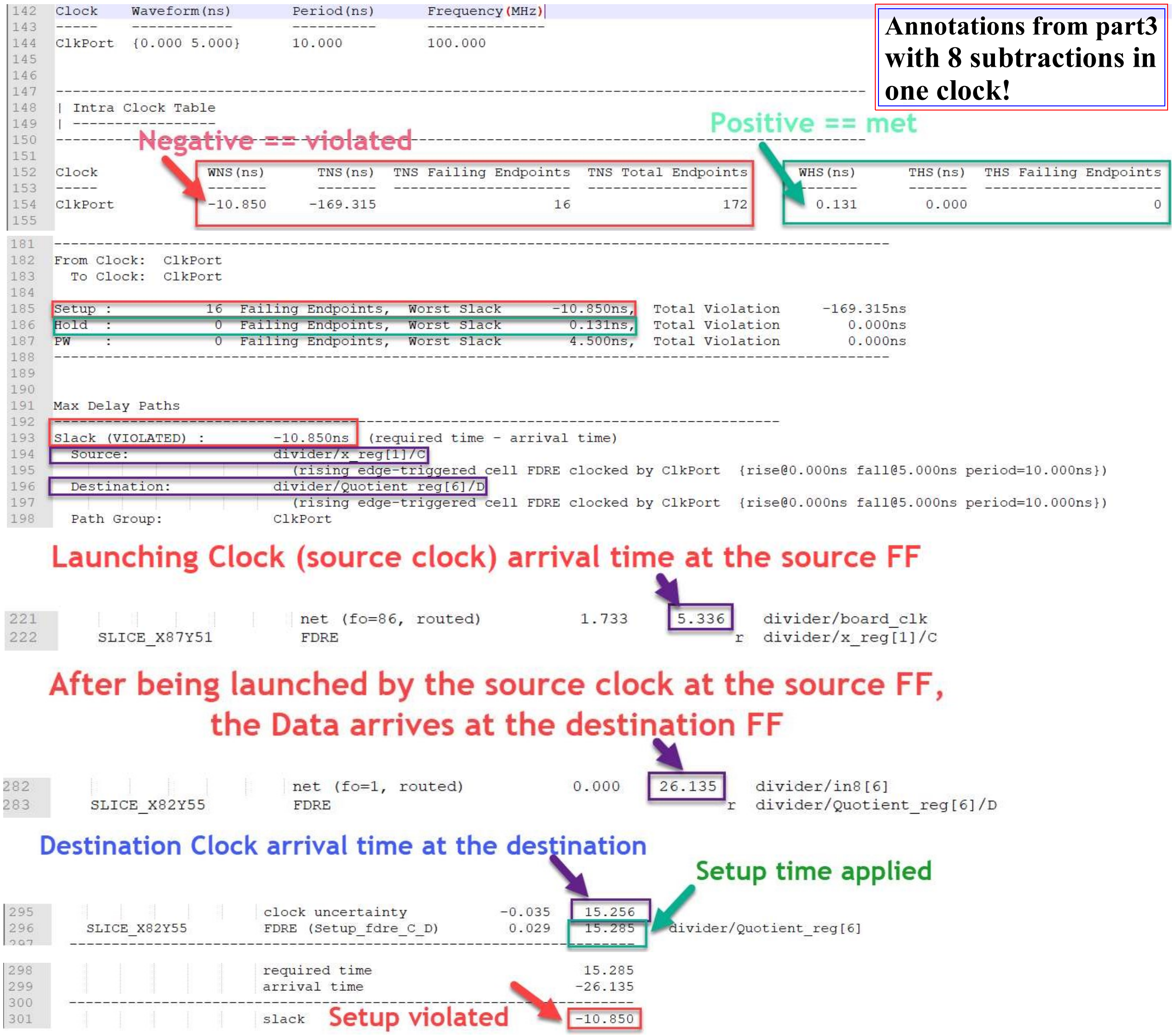
From part 1 of this lab

In part 3 of this lab, we try to violate this!

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             5.574ns  (required time - arrival time)
  Source:                 divider/x_reg[2]/C
                            (rising edge-triggered cell FDRE clocked by ClkPort  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            divider/Quotient_reg[0]/R
                            (rising edge-triggered cell FDRE clocked by ClkPort  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             ClkPort
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (ClkPort rise@10.000ns - ClkPort rise@0.000ns)
  Data Path Delay:        3.937ns  (logic 1.254ns (31.852%)  route 2.683ns (68.148%))
  Logic Levels:           3  (CARRY4=1 LUT4=1 LUT5=1)
```

Excerpts from the timing report for part 3 for the worst WNS. Your report values may be different. Synthesis results may vary from run to run as optimization is somewhat *heuristic* and *non-deterministic!* You need to look for similar lines in your report.

<u>Your TA may require you to demonstrate that you located such similar lines!</u>



Annotations from part3 with 8 subtractions in one clock!

```
Clock     Waveform(ns)       Period(ns)      Frequency(MHz)
-----     ------------       ----------      --------------
ClkPort   {0.000 5.000}      10.000          100.000


------------------------------------------------------------------------------------------
| Intra Clock Table
| -----------------
------------------------------------------------------------------------------------------

Clock             WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints
-----             -------      -------  ---------------------  -------------------      -------      -------  ---------------------
ClkPort           -10.850     -169.315                     16                  172        0.131        0.000                      0
```

Negative == violated

Positive == met

```
------------------------------------------------------------------------------------------
From Clock:  ClkPort
  To Clock:  ClkPort

Setup :           16  Failing Endpoints,  Worst Slack      -10.850ns,  Total Violation     -169.315ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.131ns,  Total Violation        0.000ns
PW    :            0  Failing Endpoints,  Worst Slack        4.500ns,  Total Violation        0.000ns
------------------------------------------------------------------------------------------


Max Delay Paths
--------------------------------------------------------------------------------------
Slack (VIOLATED) :        -10.850ns  (required time - arrival time)
  Source:                 divider/x_reg[1]/C
                            (rising edge-triggered cell FDRE clocked by ClkPort  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            divider/Quotient_reg[6]/D
                            (rising edge-triggered cell FDRE clocked by ClkPort  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             ClkPort
```

Launching Clock (source clock) arrival time at the source FF

```
                                 net (fo=86, routed)        1.733     5.336    divider/board_clk
    SLICE_X87Y51                 FDRE                                       r  divider/x_reg[1]/C
```

After being launched by the source clock at the source FF, the Data arrives at the destination FF

```
                                 net (fo=1, routed)         0.000    26.135    divider/in8[6]
    SLICE_X82Y55                 FDRE                                       r  divider/Quotient_reg[6]/D
```

Destination Clock arrival time at the destination

Setup time applied

```
                           clock uncertainty           -0.035    15.256
    SLICE_X82Y55           FDRE (Setup_fdre_C_D)        0.029    15.285    divider/Quotient_reg[6]
  -------------------------------------------------------------------

                           required time                         15.285
                           arrival time                         -26.135
  -------------------------------------------------------------------
                           slack                                -10.850
```

Setup violated

4. Applying constraints

Timing constraints are instructions that the designer gives to the Xilinx tool about the speed at which the designer wants to run the design. The Xilinx tool uses these instructions to construct an implementation that meets the timing constraints. Remember that the tool reports failures in the Place-and-Route report by indicating a negative slack if the constraint is not met. Then you can either modify the constraint or the design based on your system objective. Timing constraints are specified in the Xilinx Design Constraints (.xdc) file. This is the same used to specify pin location constraints. A few more constraints are discussed in this lab: clock period constraint and false-path constraint.

4.1 Clock period constraint:

This constraint tells the tool the frequency at which you want to run the design. The tool tries to ensure that all combinational paths between registers have delays shorter than this clock period so that the design will work reliably at that frequency. The syntax for this constraint is:

```
create_clock -add -name <net_name> -period <clock_period> [get_ports <net_name>]
```

where net name is the name of the clock signal (e.g., clk, sys clk, or ClkPort as in our designs) and clock period is the time period of the clock in nanoseconds. In addition, the duty cycle and phase shift of the clock can be specified with the waveform option. waveform is the list of rising edge and falling edge absolute times, in nanoseconds, within the clock period. Unless specified otherwise, the duty cycle defaults to 50% and the phase shift to 0ns. For example,

```
create_clock -add -name ClkPort -period 10.00 [get_ports ClkPort]
```

creates a 10ns clock with 50% duty cycle and no phase shift.

```
create_clock -add -name ClkPort -period 10.00 -waveform {2 8} [get_ports ClkPort]
```

creates a 10ns clock 80% duty cycle and a 2 ns rising edge phase shift.

4.2 False path constraint:

False path constraints instructs the tool to ignore “false paths”, which includes (i) paths across different clock domains, and (ii) paths connected to peripheral pins (such as switches SWs and LEDs). To instruct the tool to ignore input signals such as switches and buttons, the syntax is:

```
set_false_path -through [get_nets <net_name>]
set_false_path -through [get_nets {Sw0}]
```

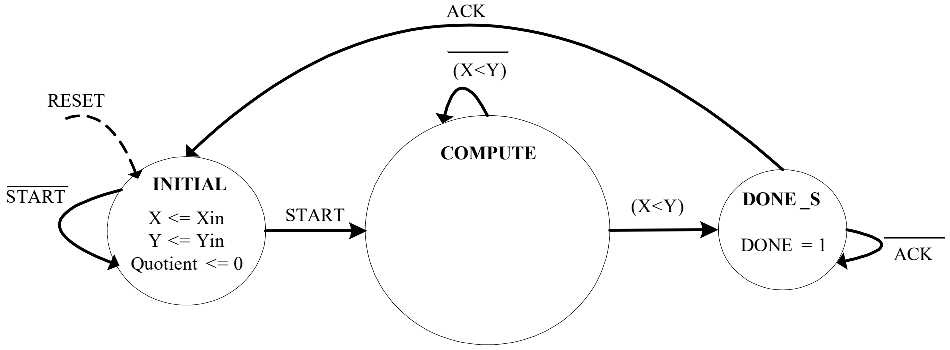
You would declare all inputs (switches and buttons) as belonging to a *false path* using the above syntax. The tool ignores any timing paths involving any such signals.

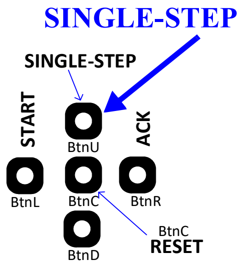
To instruct the tool to ignore timing check associated with output signals such as LEDs and SSDs:

```
set_false_path -from [all_registers -edge_triggered] -to [all_fanout -end-
points_only -flat -from [get_nets <net_name>]]
set_false_path -from [all_registers -edge_triggered] -to [all_fanout -end-
points_only -flat -from [get_nets {Ld0}]]
```

5. Description of the Circuit

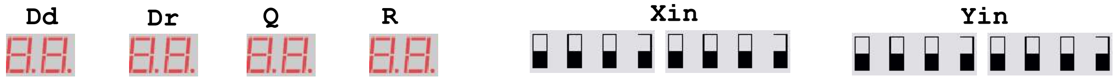
The design that will be used in this lab is a division-by-subtraction design. You will not need to modify the state diagram below, except for adding RTL in COMPUTE state. You will experiment with the number of subtraction operations performed in each clock cycle when the state machine is in the `COMPUTE` state.





While (X <= Y) is a better exit condition to the DONE_S state compared to (X < Y) stated in the above design, if we change the number of subtractions per clock to 3, we may want to make the exit condition as (X <= 3Y). This leads to other problems. 3Y may not fit in the 8-bit binary number. So, for this lab, let us use the exit condition as (X < Y). Saving of one clock is not the focus of this lab. In EE560, we will talk about a multi-cycle timing constraint and we will discuss more about this state transition condition.

We are learning about how to constrain the design in timing and how to read the reports generated to see if it has met or violated the specified timing. Here we are doing an 8-bit division and displaying the dividend (Xin), deviser (Yin), Quotient (Q), and the Remainder (R) on the 8 SSDs.



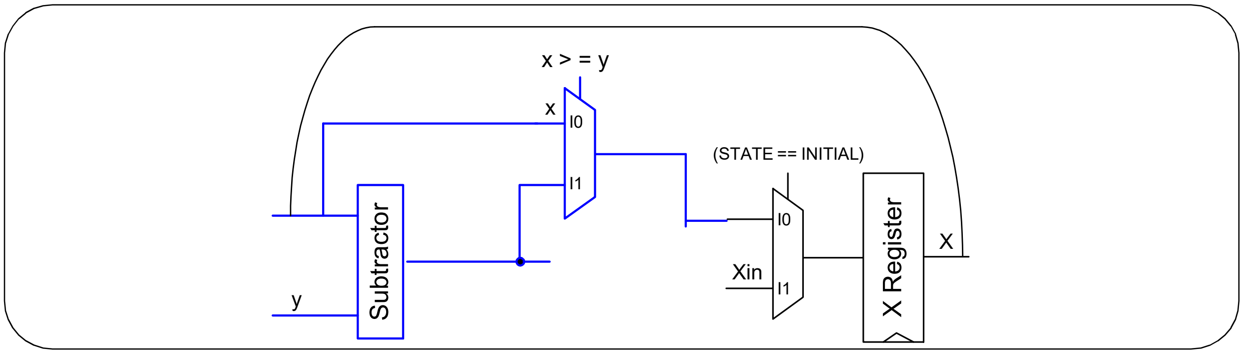
We are using the single-stepping divider design from a previous lab except that it became an 8-bit divider now.

5.1 Part 1

In **part 1** you perform only one subtraction per clock. The complete control unit and data path code for this design (part 1) is given below. This code shows the simplest implementation with only one subtraction performed during each cycle in the `COMPUTE` state (highlighted code).

```
always @(posedge Clk, posedge Reset)

  begin  : CU_n_DU
    if (Reset)
       begin
        state <= INITIAL;
        x <= 8'bXXXX_XXXX;
        y <= 8'bXXXX_XXXX;
        Quotient <= 8'bXXXX_XXXX;
       end
    else
       begin
         (* full_case, parallel_case *)
         case (state)
           INITIAL:
             begin
               // state transitions in the Control Unit
               if (Start)
                 state <= COMPUTE;
               // RTL operations in the Data Path Unit
                 x <= Xin;
                 y <= Yin;
                 Quotient <= 0;
             end

           COMPUTE:
             begin
               if (x < y)
                 state <= DONE_S;

               if (x >= y))
                 begin
                   x <= x - y;
                   Quotient <= Quotient + 1;
                 end
             end

           DONE_S:
             begin
               if (Ack)
                 state <= INITIAL;
             end
      endcase
    end
  end
```

The following diagram graphically represents the data path components for X register.



5.2 Part 2

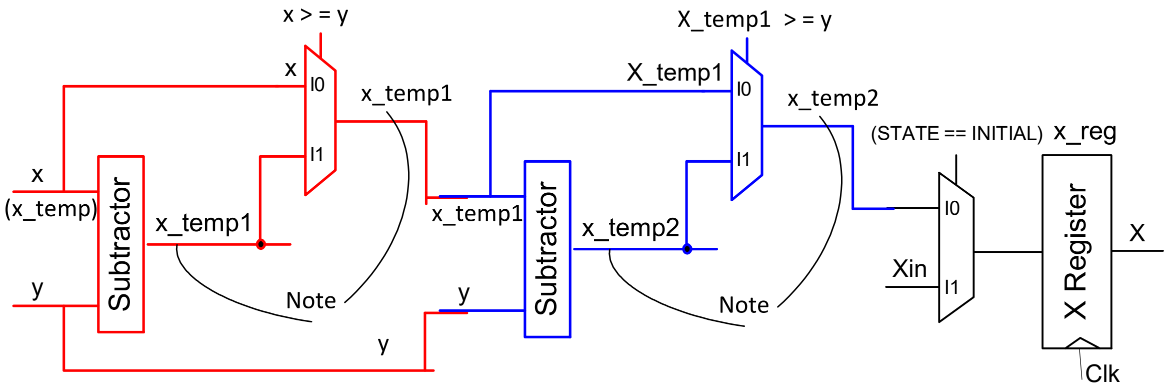
In **part 2** you will modify the data path to perform more than one subtraction (in a cascaded fashion). The following code implements a chain of two subtracters using temporary variables (x_temp, x_temp1, x_temp2,etc.). It is best to declare these temporary variables locally in the named procedural block (begin .... end block). The name of the block can be any name.

```
begin: The_Compute_Block
         // local variable declarations
            reg [7:0] x_temp, x_temp1, x_temp2, Quo_temp, Quo_temp1, Quo_temp2;
```

```
x_temp  = x; // gather x into x_temp
Quo_temp  = Quotient; // gather Quotient into Quo_temp;

x_temp1 = x_temp;
Quo_temp1 = Quo_temp;

if (x_temp >= y)
   begin
     x_temp1 = x_temp - y;
     Quo_temp1 = Quo_temp + 1;
   end

x_temp2 = x_temp1;
Quo_temp2 = Quo_temp1;
if (x_temp1 >= y)
   begin
     x_temp2 = x_temp1 - y;
     Quo_temp2 = Quo_temp1 + 1;
   end

// final gathering of the computed combinational outputs into registers
x <= x_temp2;
Quotient <= Quo_temp2;
```

two subtractions are combined into one clock.

The above code produces the following logic for the X register. Each additional subtraction adds one 8-bit subtracter and one 8-bit wide 2-to-1 multiplexer to the X-path in the DPU.



Note: When designing, in your schematic, you can not use a label (ex: x_temp) for the input as well as for the output of a mux. But it makes sense in the sequential code in your Verilog always block.

As shown below we can just use one `x_temp` variable instead of the `x_temp, x_temp1, x_temp2`. Similarly, we can use one `Quo_temp` instead of the three `Quo_temp, Quo_temp1, Quo_temp2`.

Notice the *blocking assignments* for each *intermediate combinational* output.

```
x_temp   = x; // gather x into x_temp
Quo_temp  = Quotient; // gather Quotient into Quo_temp;


if (x_temp >= y)
   begin
     x_temp = x_temp - y;
     Quo_temp = Quo_temp + 1;
   end


if (x_temp >= y)
   begin
     x_temp = x_temp - y;
     Quo_temp = Quo_temp + 1;
   end
// final gathering of the computed combinational outputs into registers
x <= x_temp;
Quotient <= Quo_temp;
```

two subtractions were combined into one clock.

5.3 Part 3

In **part 3** we experiment with a ***for-loop*** to construct the multiple subtractions block. The method used in part 2 is error-prone and laborious if you were to do more subtractions (more than 2 subtractions) in a clock. It can be replaced by a for-loop that implicitly performs the same function. The synthesis tool "unrolls" the for loop and generates logic that performs as many subtractions as you specify with the loop bounds. By adjusting the loop bounds you can control the number of subtraction operations performed in each cycle.
The following code implements **two** subtractions per cycle. Notice **`(I = 0; I <= 1; I = I+1)`**
The for loop is unrolled by both, the simulation tool and the synthesis tool.

It leads to the same hardware as shown in the figures in part 2 above.

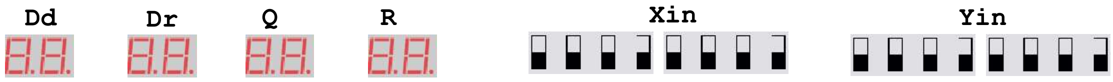
```
reg [7:0] x_temp, Quo_temp;
integer I;

x_temp = x;
Quo_temp = Quotient;

for (I = 0; I <= 1; I = I+1)
 begin
      if (x_temp >= y)
       begin
           x_temp = x_temp - y;
           Quo_temp = Quo_temp + 1;
       end
 end

x <= x_temp;
Quotient <= Quo_temp;
```

Note that the "for" loop executes in *zero-time* in simulation when the time is frozen.
The above for loop does not take two clocks. The "for" loop is only a convenience mechanism to describe the datapath here.

5.4 Top Design

BtnC is connected to reset, Xin is SW15-SW8, and Yin is SW7-SW0. BtnL generates the START signal and BtnR generates ACK. It is a single-stepping design like in the earlier lab with
BtnU acting as the single stepping button.

The 8-bit Xin is displayed on {SSD7, SSD6} pair. The 8-bit Yin is displayed on {SSD5, SSD4} pair.
The 8-bit computed quotient is displayed on {SSD3, SSD2} pair.
The 8-bit computed remainder is displayed on {SSD1, SSD0} pair.



6. Procedure:

6.1 Download the Xilinx Vivado project zip file `ee354l_divider_timing.zip` from Blackboard. Extract the zipped project into the projects folder (`C:\Xilinx_Projects\`). Open the project file `ee354l_divider_timing\workspace\divider_timing_part1\divider_timing_part1.xpr` in the Xilinx Vivado Project Navigator. Note that all core files (`divider_timing_part1.v`, `divider_timing_part2.v`, `divider_timing_part3.v`) use the same module name (`divider_timing`). Hence you can make a copy of `ee354l_divider_timing\workspace\divider_timing_part`**1** as `ee354l_divider_timing\workspace\divider_timing_part`**2**. In this new directory, you can replace one core file with another core file and synthesize the same TOP (`divider_timing_top_with_single_step.v`) with the new core file.

File	**Description**
divider_timing_part1.xpr divider_timing_part2.xpr divider_timing_part3.xpr	Xilinx Project Files.
divider_timing_top_with_single_step.v	This top file is complete and is common to all three parts.
divider_timing_part1.v	This Verilog code for Part 1 of this lab is complete. The code implements the data path executing one subtraction per cycle. You do NOT have to modify this code in Part 1.
divider_timing_part2.v	Modify `divider_timing_part1.v` to create this Verilog code for Part 2 of this lab. Section of code to implement data path with two subtractions per cycle was given in this handout. You will create and modify this file as explained in Procedure Part 2.
divider_timing_part3.v	Modify `divider_timing_part1.v` to create this Verilog code for Part 3 of this lab. Section of code with a "for" loop to implement data path with two subtractions per cycle was given in this handout. You will create and modify this file as explained in Procedure Part 3. Instead of writing in-line code for each subtraction as in Part 2, a for-loop is specified in Part 3. By changing the bounds of this for-loop you can easily perform as many subtractions as you want within one clock period. To save time, just do **4** subtractions per clock for passing the timing check and **8** subtractions per clock for causing timing violations.
ee201_debounce_DPB_SCEN_CCEN_MCEN.v	To support single-stepping. Generates SCEN.
ee354_divider_top.xdc	This is the Xilinx Design Constraint File that you are required to read and appreciate. In addition to pin location constraints, the file has timing constraints. The same .xdc file is used for all three designs.

6.2 Open the top file `divider_timing_top_with_single_step.v` and understand its function. Recall that *all* constraints -- pin location and timing constraints -- are applied to signals in the top level module (the .xdc file is associated with the top module). So you must be familiar with the signals (and their names) in the top file and the function they perform in order to apply timing constraints.

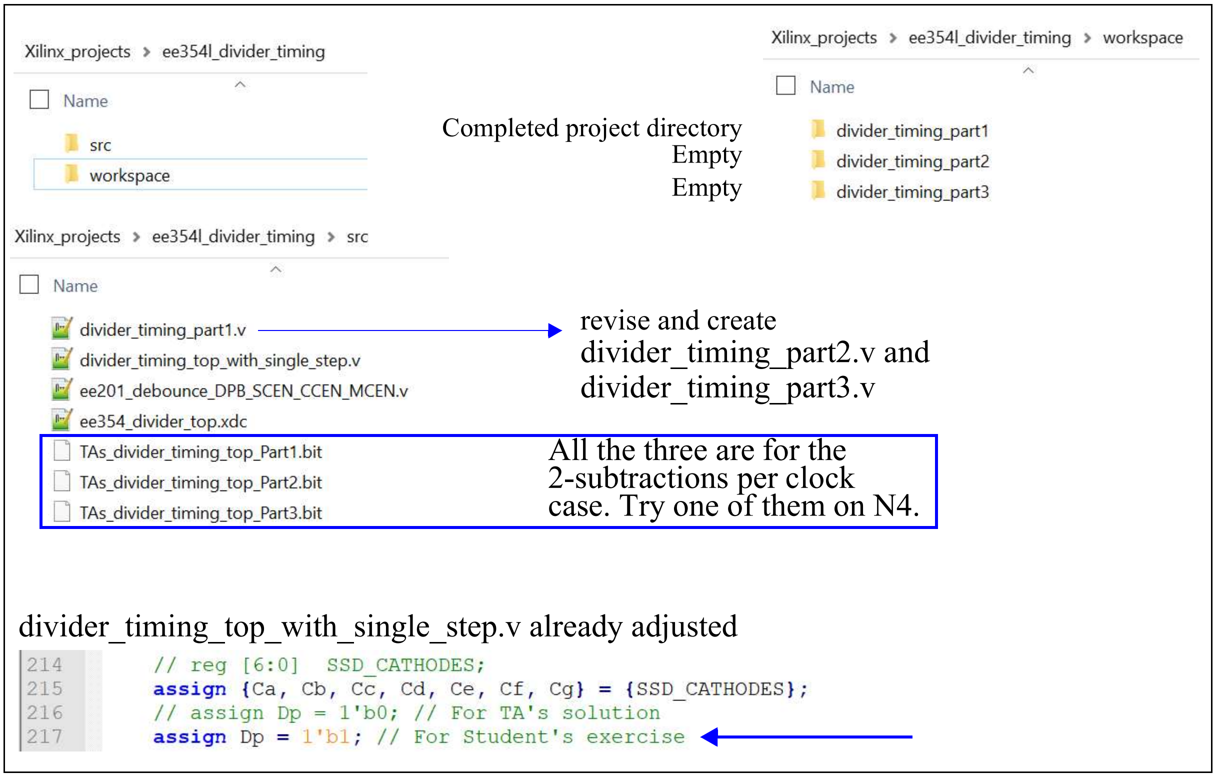
6.3 Part 1 design: To save time, we have provided a completely synthesized project folder so that you do not need to synthesize. Please open the project in Vivado and read the three reports stated in section 3. Show to your TA to prove that you understood how to open and read the reports.
Verify the operation on Nexys 4 board and show to your TA.

6.4 Create part 2 and part 3 designs.

6.4.1 Confirm that Part 2 design passes timing and the design works well on the N4 board. Demonstrate to your TA.

6.4.2 In Part 3, originally we wanted students to increase the number of subtractions in the "for" loop from 2 to until they violate the timing constraint. But to save time, now we are asking students to synthesize the two cases, (a) a four-subtractions per clock case (which succeeds timing design) and (b) a eight-subtractions per clock case (which fails to pass timing design).
Demonstrate the successful design on board and also by reading the timing report and confirming that it passed the timing. Read timing report of the failed design and show to your TA that you noticed the lines similar to the lines in section 3.3 of this handout.

6.5 Directories and files in the .zip file provided



Xilinx_projects > ee354l_divider_timing

Name
src
workspace

Xilinx_projects > ee354l_divider_timing > workspace

Name	
divider_timing_part1	Completed project directory
divider_timing_part2	Empty
divider_timing_part3	Empty

Xilinx_projects > ee354l_divider_timing > src

Name
divider_timing_part1.v
divider_timing_top_with_single_step.v
ee201_debounce_DPB_SCEN_CCEN_MCEN.v
ee354_divider_top.xdc
TAs_divider_timing_top_Part1.bit
TAs_divider_timing_top_Part2.bit
TAs_divider_timing_top_Part3.bit

revise and create divider_timing_part2.v and divider_timing_part3.v

All the three are for the 2-subtractions per clock case. Try one of them on N4.

divider_timing_top_with_single_step.v already adjusted

```
        // reg [6:0]  SSD_CATHODES;
        assign {Ca, Cb, Cc, Cd, Ce, Cf, Cg} = {SSD_CATHODES};
        // assign Dp = 1'b0; // For TA's solution
        assign Dp = 1'b1; // For Student's exercise
```

7. Lab Report:

Name:__________________________ Date: __________________________
Lab Session: ____________________ TA's Signature: ___________________

For TAs:

Comments:

Q 7. 1: Consider 13 divided by 3.
How many cycles in the `COMPUTE` state (include the *failed* subtraction clock, but do not count clocks in `INITIAL` and `DONE_S` states) does the given state machine take to perform this subtraction using the data path in Part 1 (one subtraction per clock)?

Number of Cycles ______

Q 7. 2: Assuming you **have two subtracters** (i.e., the data path given in Part 2) how many cycles (in the `COMPUTE` state) will it take to do 13_{decimal} divided by 3_{decimal}? Complete the table below (add more lines, if necessary) with values at the beginning of the clock.

```
State: Initial Xin = 13 Yin = 3  Quotient = 0

State: Compute X =   13 Y = 3    Quotient = 0 Remainder = 13

State:         X =      Y =      Quotient =   Remainder =

State:         X =      Y =      Quotient =   Remainder =

State:

State:

State:
```

Q 7. 3: What would happen if we give `Xin = 6`$_{\text{decimal}}$ and `Yin = 12`$_{\text{decimal}}$ to the **two subtractions** per clock design. Write the list of states visited and the final values of Quotient and Remainder using the format of Q 7.2.

```
State: Initial    Xin =  6  Yin = 12      Quotient = 0 Remainder = 6

State: Compute    X =    6    Y = 12      Quotient = 0 Remainder = 6

State:            X =         Y =         Quotient =   Remainder =

State:

State:
```

Q 7. 4: Assuming you have **four subtracters** (like in the first part of Part 3) how many cycles (in the COMPUTE state) will it take to divide 30 **Hex** by 04 **Hex**? Complete the table below (add more lines, if necessary) with values in **Hex** at the beginning of the clock.

State: Initial Xin = 30 Yin = 04 Quotient = 00 Remainder = 30

State: Compute X = 30 Y = 04 Quotient = 00 Remainder = 30

State: X = Y = Quotient = Remainder =

State: X = Y = Quotient = Remainder =

State:

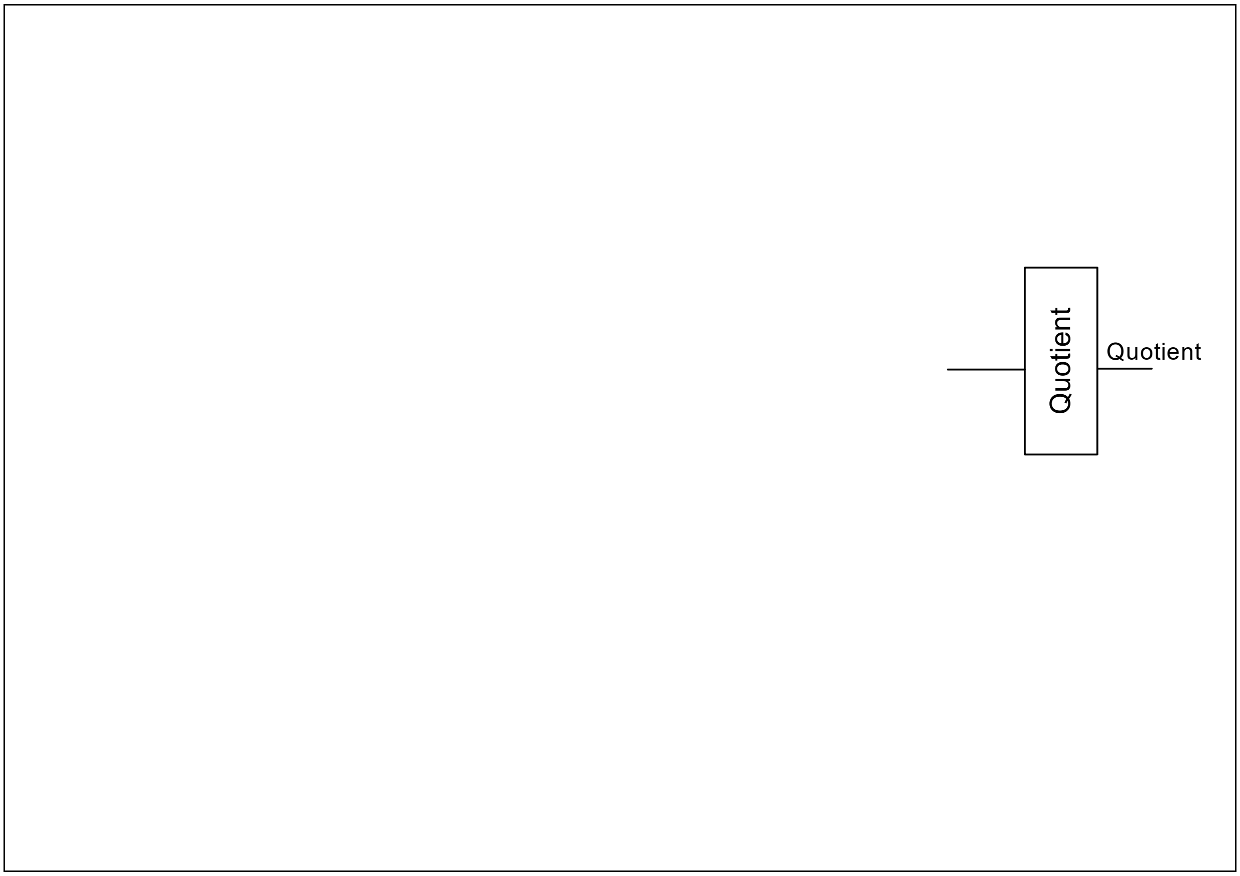
State:

State:

State:

State:

Q 7. 5: Complete the block-level diagram for the Quotient register. Assume two cascaded incrementers. Refer to similar figure on previous pages for the register X.


Quotient
Quotient