EE201L
Divider design

Objective: To introduce to students
-- RTL coding style for state machine and datapath coding
-- Testbench example
-- Simple TOP design making use of I/O resources on Nexys-2 board
-- UCF file example
-- Introduce Epp protocol
-- Exploit the I/O resources in Adept 2.0 I/O Expansion

References (for the TAs, not for students):

1. **Nexys-2 board** reference manual (Nexys2_rm.pdf) and schematic

http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,789&Prod=NEXYS2
http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_rm.pdf
http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_sch.pdf

2. **Epp protocol**
First 4 pages of the **Digilent Parallel Interface Model Reference Manual**
http://www.digilentinc.com/Data/Products/ADEPT/DpimRef%20programmers%20manual.pdf
Also see http://www.beyondlogic.org/epp/epp.htm

Files provided:
A zip file is provided containing source files for four sample designs in four folders. *Please read the notes at the top of each file to get to know important aspects of the design to note.*

1. ee201_divider_simple
2. ee201_divider_with_debounce
3. ee201_divider_with_single-step
4. ee201_divider_with_VIO_multi_step

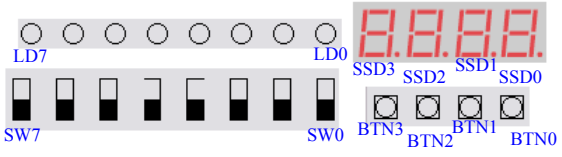
A short description of each of the above 4 designs follows.

3. ee201_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.

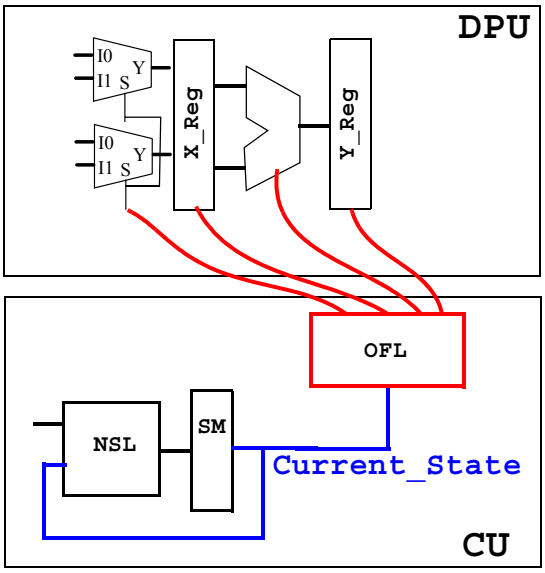
Extract from divider_combined_cu_dpu.v

```
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
```



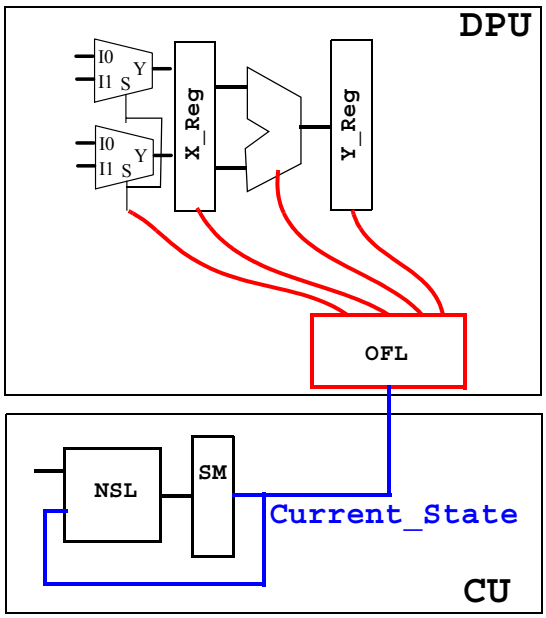
Traditional division between DPU and CU

OFL (combinational logic) is in the CU.



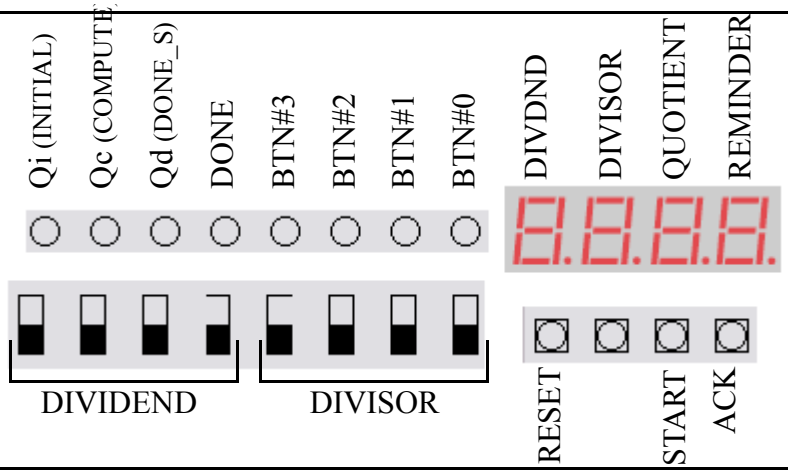
Division between DPU and CU for HDL coding

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.



ee201_divider_simple

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



Questions for the ee201_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 (**ee201_divider_with_debounce**) appear to be quite different from this design? Is it just appearance only or is it really different?

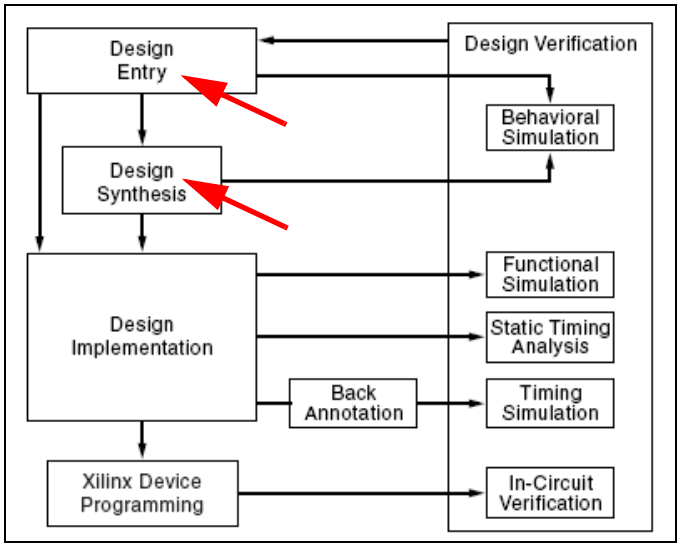
4. ee201_divider_with_debounce:

First a debouncer design (ee201_debounce_DPB_SCEN_CCEN_MCEN.v) is presented to debounce a given push button and produce 4 outputs: DPB, SCEN, CCEN, MCEN.
Output coding (for the states in the state machine) is used for glitch free outputs.

State Name	State	DPB	SCEN	MCEN	CCEN	TB1	TB0
initial	INI	0	0	0	0	0	0
wait quarter	WQ	0	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.

ISE => Help => Software Manuals => Click on Design Synthesis in the diagram (copy shown on the side) => XST User guide => Search for FSM Encoding



As shown here, we used verilog attributes to enforce our output coding. Through these attributes, we are informing the tool-vendor (Xilinx here) that we want the tool to honor and retain our user encoding.

It is 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:

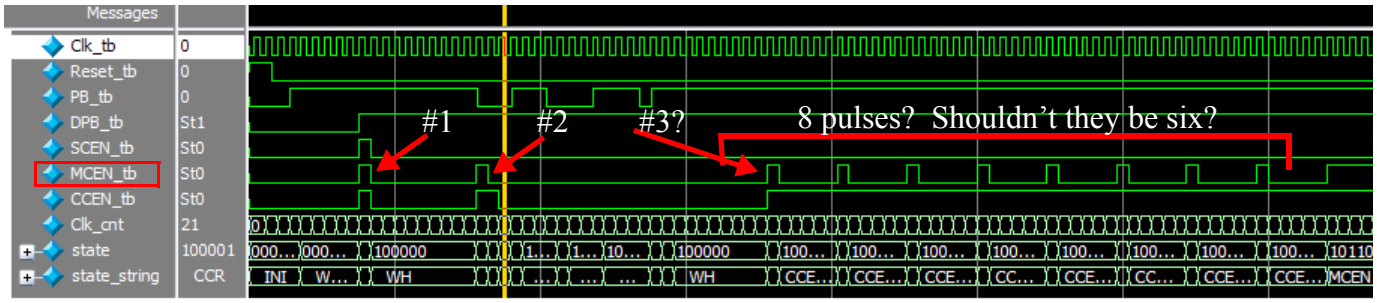
```
(* fsm_encoding = "{auto|one-hot
  |compact|sequential|gray|johnson|speed1|user}" *)
```

The default is **auto**.

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

Notice that, the testbench has instantiated the UUT with N_dc of 4 in the generic map

```
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));
```



Questions on the debouncer:

1. Briefly explain why the N_dc parameter was changed to 4 during simulation (from the actual value of 24 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.

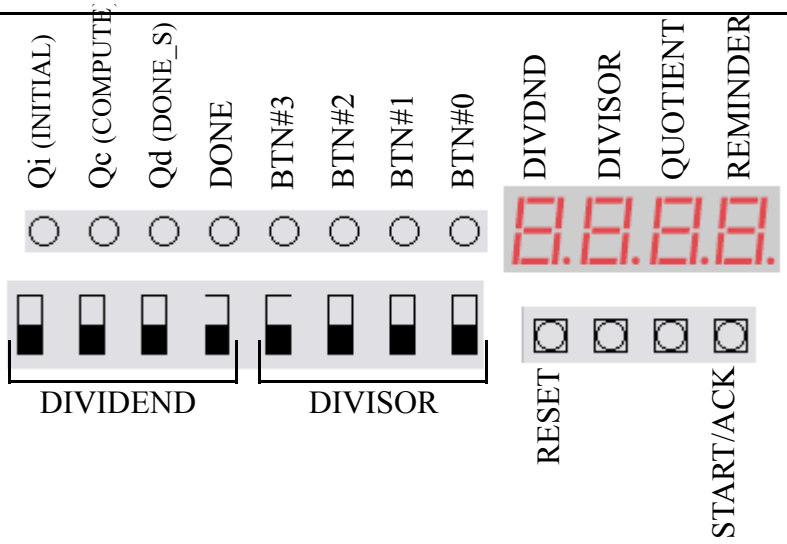
3. We took time to design output-coded state machine with no OFL at all, there by avoiding any glitches in the SCEN, MCEN, etc. Are glitches really harmful in our design or we have just shown a way to produce glitch-free outputs?

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

ee201_divider_with_debounce

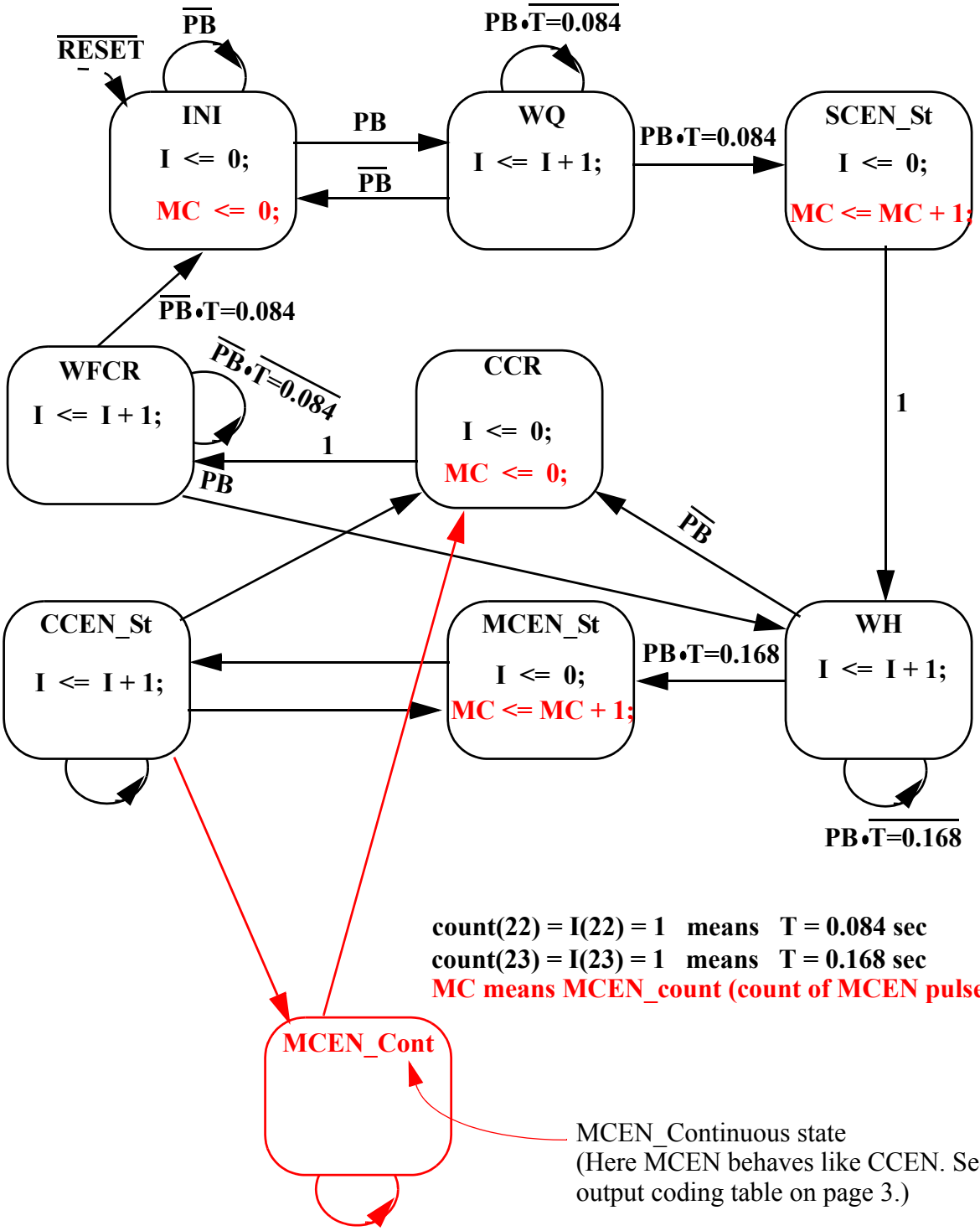
Go through the files and download the provided bit file and test.

Note that, unlike in the earlier design, (**ee201_divider_simple**), we run the core divider in this design at the **full speed of 50Mhz**.



Debouncing State Machine

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



count(22) = I(22) = 1 means T = 0.084 sec
count(23) = I(23) = 1 means T = 0.168 sec
MC means MCEN_count (count of MCEN pulses)

MCEN_Continuous state
(Here MCEN behaves like CCEN. See the output coding table on page 3.)

Names of the students submitting:

1.

2.

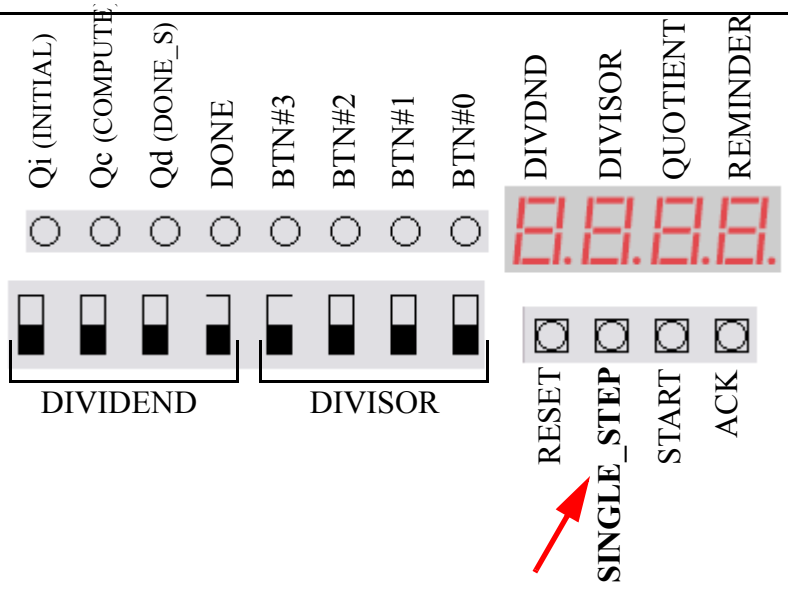
5. **ee201_divider_with_single_step**

Here, in the compute state, we single-step the division operation using the SCEN produced out of Btn2

ee201_divider_with_single_step

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

Note that, unlike in the first design, (**ee201_divider_simple**), we run the core divider in this design at the **full speed of 50Mhz**.



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).

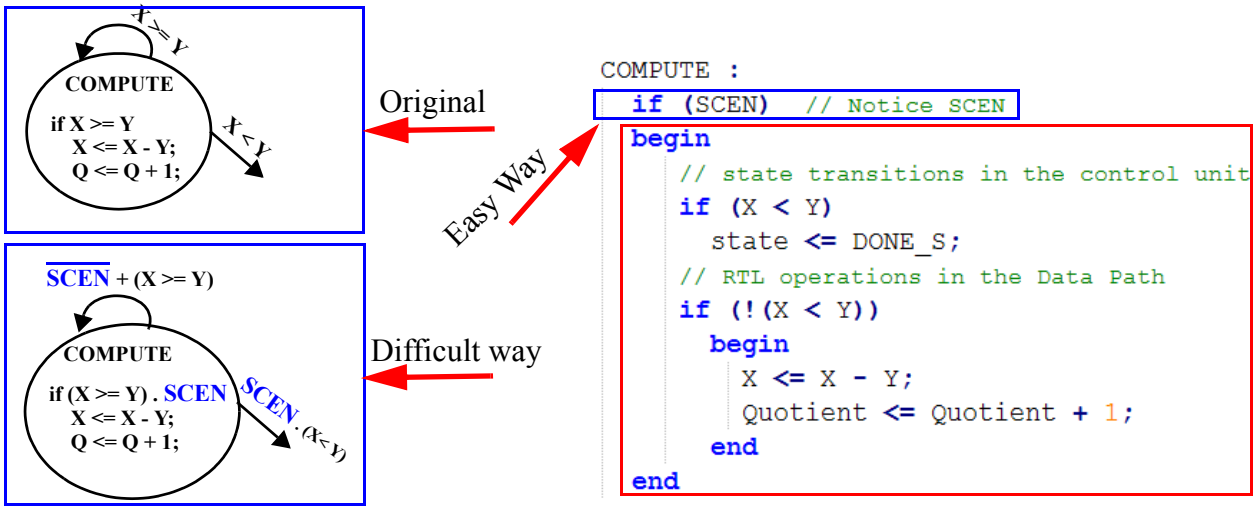
```
// instantiate the core divider design. Note the .SCEN(SCEN)
divider divider_1(.Xin(Xin), .Yin(Yin), .Start(Start), .Ack(Ack),
                  .Clk(sys_clk), .Reset(Reset),   .SCEN(SCEN),
                  .Done(Done), .Quotient(Quotient), .Remainder(Remainder),
                  .Qi(Qi), .Qc(Qc),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

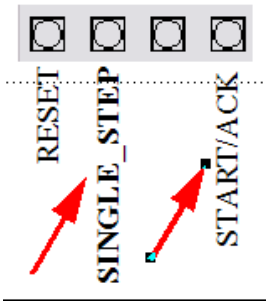


Questions on **ee201_divider_with_single_step**:

A. Is it possible to use SCEN to control one state (or a few states), MCEN to control another state (or a separate bunch of states) and further CCEN to control yet another state?

B. If we are simulating an external event such as a sensor embedded under a road for traffic-light control, we can produce a Btn1_SCEN pulse (SCEN pulse produced by pressing Btn1), can you single-step such a system using another button (say Btn2) to produce Btn2_SCEN? Do you see any problem (operational or logical or ...)?

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 Btn0)?
Is this just not possible or it works if we produce a Btn0_ SCEN and use it as START as well as ACK, or ...?



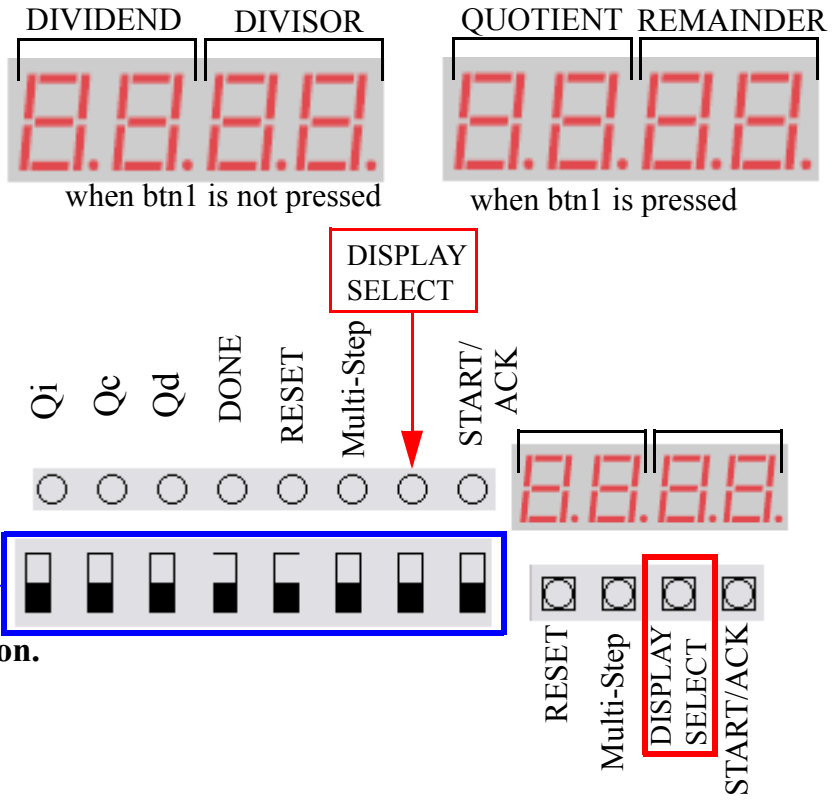
6. ee201_divider_with_VIO_multi_step and the Epp Interface

Here we are interfacing to the virtual I/O in Adept 2.0. The file, **IOExpansion.vhd**, provided by Digilent, implements the Epp slave-side address and data registers in FPGA. We translated the same to Verilog. The file is called **IOExpansion.v**. Note that now, the UCF file needs to have pins associated with Epp to talk to the Cypress USB interface chip. Please refer to the Adept User's manual on your PC (Start => All Programs => Digilent => Adept => Adept User's manual.

ee201_divider_with_VIO_multi_step

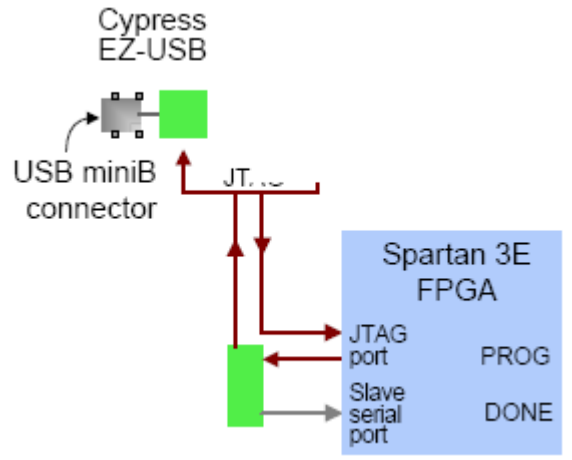
Multiplexing the two displays on the four 7-seg displays using Btn1.

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



The switches on the board are not used here.
8-bit dividend and 8-bit divisor are set using the 16 switches on Adept IOExpansion.

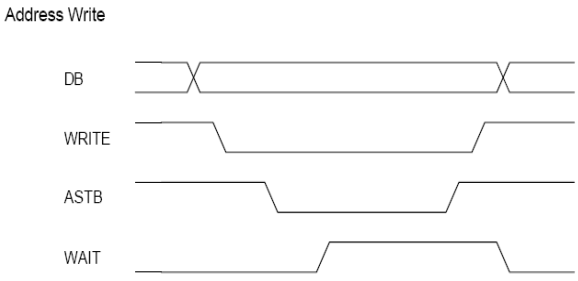
).

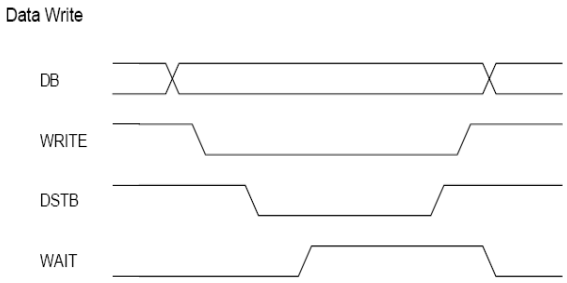


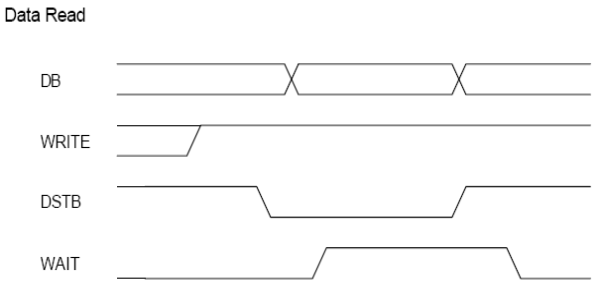

```
NET "EppWait" LOC= "N9";

NET "EppDB<0>" LOC= "R14";
```

UCF file has pins for Epp







The following signals make up the interface:

Name	Source	Description
DB0 – DB7	bidir	Data bus. The host is the source during write cycles and the peripheral is the source during read cycles.
WRITE	host	Transfer direction control. High = read. Low = write
ASTB	host	Address strobe. Causes data to be read or written to the address register
DSTB	host	Data strobe. Causes data to be read or written to a data register
WAIT	peripheral	Synchronization signal used to indicate when the peripheral is read to accept data or has data available.

Instead of viewing this as a low active wait, it may be easier to view it as a high-active GOT signal. Notice that the Epp protocol implements the full (4-way) handshake.

Extract of **IOExpansion.v**

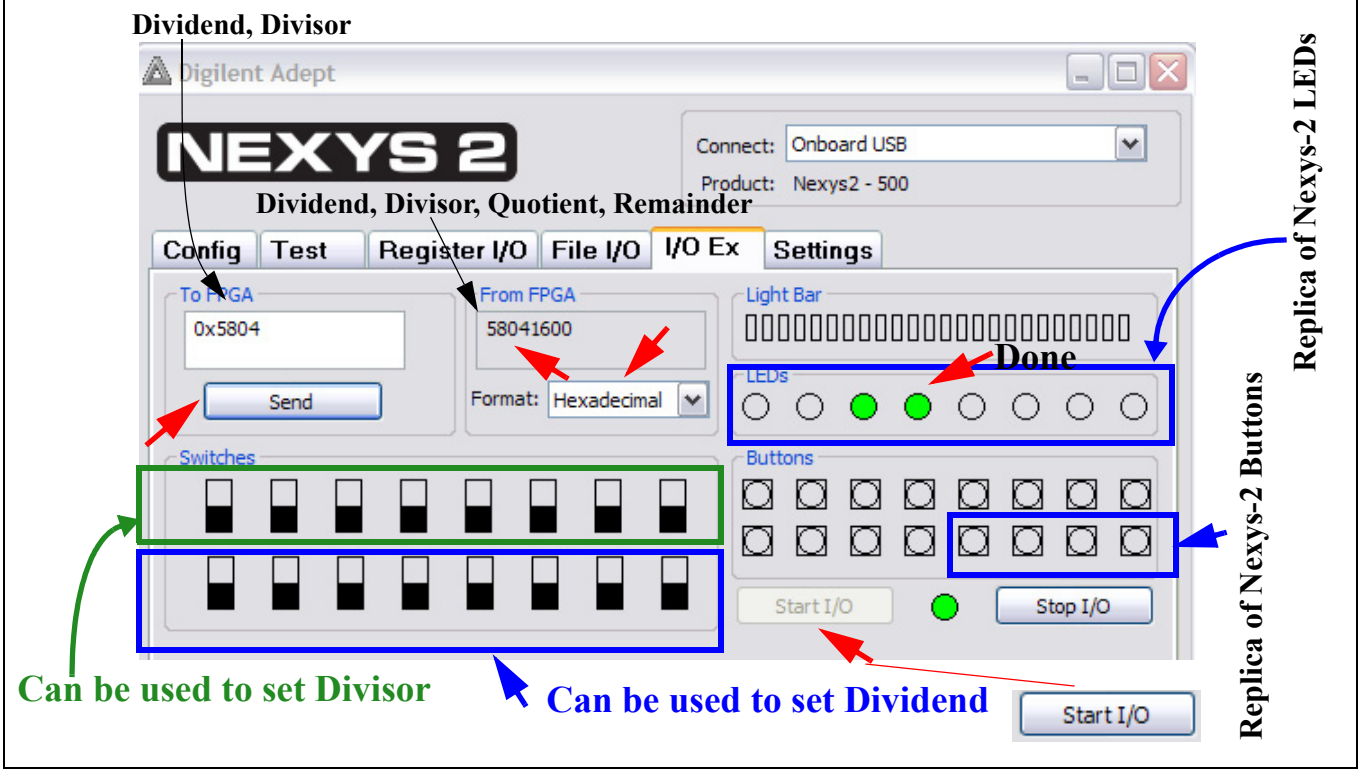
```
// EPP Address register
always @(posedge EppAstb)
begin
    if (!EppWr)
        regEppAdr <= EppDB;
end
```

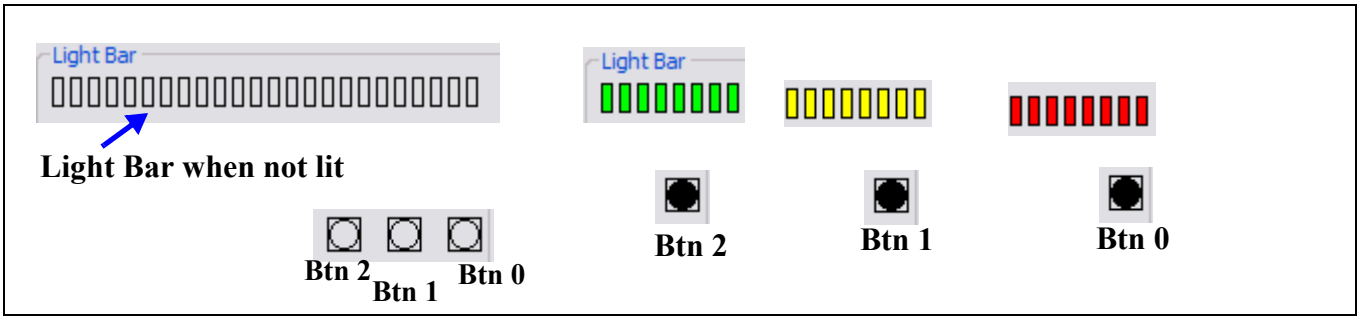
Epp Address Register is written at the end of the Epp Address Strobe because Epp Write control line is low indicating intent to write.

Your PC running Adept 2.0 (or higher) is the Epp master, which drives the three control lines:
EppAstb: Epp Address Strobe (active low, ending edge is posedge),
EppDstb: Epp Data Strobe (active low, ending edge is posedge),
EppWr: Epp Write Control (active low, low means intent to write, high means intent to read).
The **EppDB** is the Epp 8-bit data bus. During an active address or data strobe, Epp master drives data if write is true (EppWr = 0) else slave drives data if read is true (EppWr = 1).
Active-low **WAIT** (= active-high GOT) acts like a hand-shake signal between the two parties.
Address Read Cycle is not implemented in Adept Virtual I/O protocol.

ee201_divider_with_VIO_multi_step

In this example 58H is the dividend and 04H is the divisor.
The quotient is 16H and the remainder is 00H.





7. Task to be performed

Download the .zip file provided to you into your C:\xilinx_projects\ directory and extract files to form C:\xilinx_projects\ee201_divider_verilog directory with 4 sub-folders:

1. ee201_divider_simple 2. ee201_divider_with_debounce 3. ee201_divider_with_single-step
4. ee201_divider_with_VIO_multi_step

All the four folders have verilog source files, .ucf source file, a .bit file of the completed design.
After reading the code, you can download the .bit file to the Nexys-2 500K 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 project and implementing the same.

When you are done, you will submit a report to your TA your answers to questions posted under first three designs. No questions are posted for the last design.

8. Celebrate your success!!! Don't forget this step!