Laboratory 6

(Due date: **004/011**: Nov. 30th, **005**: Dec. 1st, **007**: Dec. 2nd)

OBJECTIVES

- ✓ Describe Finite State Machines (FSMs) in VHDL.
- ✓ Implement a Digital System: Control Unit and Datapath Unit.

VHDL CODING

- ✓ Refer to the Tutorial: VHDL for FPGAs for parametric code for: register, shift register, counter, adder/subtractor.

ITERATIVE SQUARE ROOT IMPLEMENTATION (100/100)

- Given an unsigned integer D, we want to design a circuit that computes the square root Q and remainder R ($R = D - Q \times Q$). There exist many algorithms for hardware implementation, and the following one has proved to outperform most others:

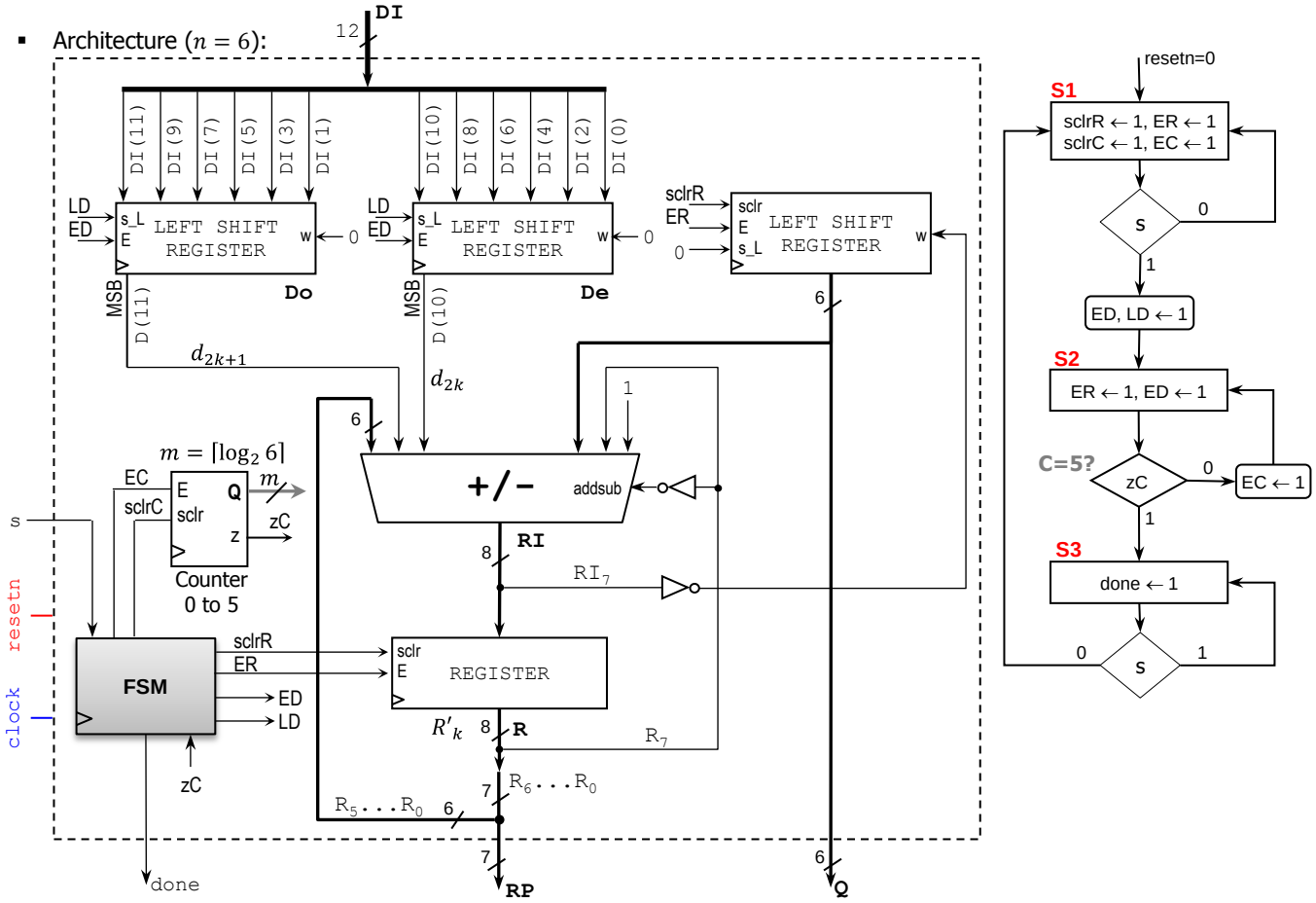
Radical ($2n$ bits): $D = d_{2n-1}d_{2n-2}d_{2n-3}d_{2n-4} \dots d_1 d_0$
We define: $Q_k = q_{n-1}q_{n-2} \dots q_k, \quad k = 0,1, \dots, n-1$
$R'_k = r'_n r'_{n-1} r'_{n-2} \dots r'_k, \quad k = 0,1, \dots, n-1$

Square Root (n bits): $Q = q_{n-1}q_{n-2} \dots q_0$
Q_k: unsigned integer with $n-k$ bits.
R'_k: signed (2C) integer with $n-k+1$ bits.

for $k = n\text{-}1 \rightarrow 0$
  if $k = n\text{-}1$ *then*
    $R'_k = d_{2k+1}d_{2k} - 01 \qquad (R'_{n-1} = d_{2n-1}d_{2n-2} - 01)$
  else
    $R'_k = \begin{cases} R'_{k+1}d_{2k+1}d_{2k} - Q_{k+1}01, & if\, q_{k+1} = 1 \\ R'_{k+1}d_{2k+1}d_{2k} + Q_{k+1}11, & if\, q_{k+1} = 0 \end{cases}$
  end
  $q_k = \begin{cases} 1, if\ R'_k \geq 0 \\ 0, if\ R'_k < 0 \end{cases}$
end

- Q and R'_{n-1} are initialized with 0's.
- This is a non-restoring algorithm, meaning that at the last iteration we might not have the correct remainder R. To get the correct value of R, an extra operation would be required.
  Remainder: $R = R_0 = \begin{cases} R'_0, & if\ R'_0 \geq 0 \\ R'_0 + Q_0 1, & if\ R'_0 < 0 \end{cases}$
- In practice, the remainder R is rarely used, and we do not implement that extra operation here.

- Architecture ($n = 6$):



Instructor: Daniel Llamocca
TAs: Sumati Gupta, Fizza Shamim, Moath Algharibeh

- Based on the algorithm, an iterative architecture for $n = 6$ is presented. The register **R** stores the 'estimated remainder' R'_k at each iteration. A square root operation is started when $s = 1$ (where the 12-bit input **DI** is captured). The signal done is asserted to indicate that the operation has been completed and the result appears in register **Q** (and R'_0 appears in **RP**)
  - ✓ At each iteration, q_k (a bit of the square root Q) is computed and shifted into the register **Q**.
  - ✓ The input data **DI** is captured into shift register **D**. The bits $d_{2k+1}d_{2k}$ correspond to the 2 MSBs of the register **D**. At every iteration, the register **D** shifts two bits to the left.
    - Register **D**: implemented by two parallel access shift registers: **Do** (for odd bits of **D**) and **De** (for even bits of **D**).
  - ✓ Note that on **RP**, we get R'_0 at the end of the computations. This is a 7-bit signed number. If it is positive, it is the correct remainder. If it is negative, an extra operation is required to get the correct remainder (not implemented in this circuit).
  - ✓ On register **Q**, we get the result, which is the square root Q, a 6-bit unsigned integer.

- The circuit includes three 6-bit parallel access left shift registers, an 8-bit register, a modulo-6 counter, an 8-bit adder/subtractor (addsub=0: add, addsub=1: subtract), and an FSM. Each sequential component has *resetn* and *clock* inputs.

- The circuit is an example of a Digital System: It includes a Control Circuit (FSM) and a Datapath Circuit. The Datapath Circuit is made from combinational and sequential components. The circuit is also called a Special-Purpose Processor. In this case, the special purpose is the integer square root.

PROCEDURE

- **Vivado: Complete the following steps:**

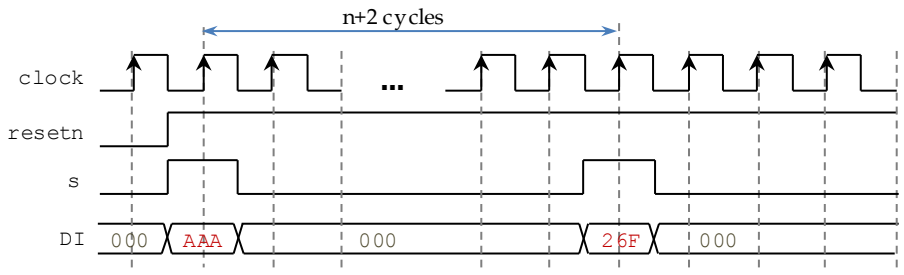
  - ✓ Create a new Vivado Project. Select the corresponding Artix-7 FPGA device (e.g.: the XC7A50T-1CSG324 FPGA device for the Nexys A7-50T).
  - ✓ Write the VHDL code for the given circuit. Synthesize your circuit to clear syntax errors.
    - Use the Structural Description: Have a separate file for the counter, parallel access shift register with sclr, register, adder/subtractor, FSM, and top file.
    - Suggestion: Use parametric code (set up the proper parameters with *generic map*) for these components:
      - Parallel access shift registers with sclr: my_pashiftreg_sclr
        * Note that some of these shift registers do not use the sclr input. In this case, that input should be tied to '0'.
      - Counter: my_genpulse_sclr (include in the top file: `use ieee.math_real.log2; use ieee.math_real.ceil;`)
      - Register with enable: my_rege
      - Adder/subtractor: my_addsub

  - ✓ Write the VHDL testbench (generate a 100 MHz input clock for your simulations) to test the following cases:

DI	**Q**	**RP**	Comments
101010101010 (2730)	110100 (52)	0110001 (0x31)	Remainder ✓
001001101111 (623)	011000 (24)	1111110 (0x7E)	Correct remainder: 1111110+0110001 = AF = **2F**
111110000010 (3970)	111111 (63)	0000001 (0x01)	Remainder ✓
000100100001 (289)	010001 (17)	0000000 (0x00)	Remainder ✓
010110111110 (1470)	100110 (38)	1001101 (0x4D)	Correct remainder: 1001101+1001101 = 9A = **1A**
100010100001 (2209)	101111 (47)	0000000 (0x00)	Remainder ✓

    - Note that **RP** is a 7-bit signed number. If it is positive, **RP** has the correct remainder. If it is negative, the correct remainder is an unsigned 7-bit number, that can be obtained by taking the 7 LSBs out of the operation $R'_0 + Q_0 1$.

    - The testbench should be written according to the timing diagram shown in the figure, where the first two values for **DI** are fed. Note that there are $n + 2 = 8$ cycles between data values.



Instructor: Daniel Llamocca
TAs: Sumati Gupta, Fizza Shamim, Moath Alharibeh

- ✓ Perform Behavioral Simulation and Timing Simulation of your design. **Demonstrate this to your TA**.
  - Behavioral Simulation: To help debug your circuit, add the internal signals (state, **`Do`**, **`De`**, **`RI`**) to the waveform view. Go to: SCOPE window: testbench → UUT. Then go to Objects Window → Signal(s) → Add to Wave Window. Then, re-run the simulation.
  - Note that you can represent data as unsigned integers (use Radix → Unsigned Decimal).
  - If your circuit works as expected, the result appears on **`Q`** and **`RP`** when done=1. Verify that the values of **`Q`** and **`RP`** (when done=1) match those listed in the previous table.

- ✓ I/O Assignment: Create the XDC file associated with your board.
  - Suggestion (for Basys 3, use `SW15` instead of `CPU_RESETN` for *resetn* input)

Board pin names	CLK100MHZ	CPU_RESETN	SW15	SW11-SW0	LED15	LED12-LED6	LED5-LED0
Signal names in code	*clock*	*resetn*	s	DI_{11}-DI_0	done	RP_6-RP_0	Q_5-Q_0

  - The board pin names (except `CPU_RESETN`) are used by all the listed boards (Nexys A7-50T/A7-100T, Nexys 4/DDR, Basys 3). I/Os: all switches and LEDs are active-high.

  - Note: synchronous circuits always require a clock and reset signal.
    - **Reset signal**: As a convention in this class, we use active-low reset (*resetn*). As a result, ensure that *resetn* is tied to the proper board resource:
      - Nexys A7-50T/A7-100T, Nexys 4/DDR: For *resetn*, use `CPU_RESETN` pin. This is an active-low push button.
      - Basys 3: There is no active low push button. Thus, for *resetn*, use `SW15`. Even though `SW15` is active high, we can still think of it as active-low *resetn*, where the circuit is reset when the switch position is OFF ('0').
    - **Clock signal**: Like other signals in the XDC file, you need to uncomment the lines associated with the clock signal and replace the signal label with name used in your code. In addition, there is parameter `-period` that is set by default to `10.00`. This is the period (in ns) that your circuit should support.
      - Nexys A7-50T: In these lines, replace the label `CLK100MHZ` with the signal name you use in your code (`clock`):
```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];
```
      - Basis 3: In these lines, replace the label `clk` with the signal name used in your code (`clock`):
```
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

- ✓ Generate and download the bitstream on the FPGA. Test the circuit (use the same input values as in the testbench). **Demonstrate this to your TA**.

- Submit (as a .zip file) all the generated files: VHDL code files, VHDL testbench, and XDC file to Moodle (an assignment will be created). DO NOT submit the whole Vivado Project.

TA signature: _____ **Date:** _____

Instructor: Daniel Llamocca
TAs: Sumati Gupta, Fizza Shamim, Moath Algharibeh