ECE-399: Selected Topics in ECE: Design with FPGAs
Report 4: Registers and Counters

Prof. Shlayan
Nithilam Subbaian

May 15, 2020

1 Objective of Report 4:

For this report, the goal is to familiarize myself with registers and counters in general. Shown at the end are select exercises to learn these concepts.

2 Materials

1. Zedboard [2]
2. 4GB SD card (purchased independently of Zedboard Kit)
3. 12V Power supply
4. Micro USB cable
5. USB Adapter: Male Micro-B to Female Standard-A
6. Xilinx Vivado Design Suite Version 2018.3 [6]

3 Lecture 7 and 8 Notes

These notes are based of Lecture 7 [3] and 8 [4].

3.1 Tasks and Functions

Subprograms are used to improve the readability and to exploit code reusability. They can be defined using: Tasks, Functions.

Tasks are more general than functions and can be used to model both combinational and sequential logic.Tasks may contain timing controls, such as timing delays, like posedge, negedge, delay, wait. Other properties of Tasks:

1. Can call other tasks and functions
2. Can have zero, one, or more arguments.
3. Values are passed to and from a task through arguments.
4. Can use global variables, when no local variables are used. When local variables are used then output is assigned only at the end of task execution.
5. Can have any number of inputs and outputs, the arguments can be input, output
6. The variables declared within the task are local to that task. The order of declaration within the task defines how the variables passed to the task are used when called
7. The variables declared within the task are local to that task. The order of declaration within the task defines how the variables passed to the task are used when called

8. It must be specifically called with a statement, unlike a function which can be used within an expression
9. Defined in the module in which they are used. It is possible to define a task in a separate file and use the compile directive 'include to include the task in the file which instantiates the task.

System Tasks:

1. $display –Print immediate values to standard output with an end-of-line character.
2. $write –Similar to the display task except it does not print an end-of-line character.
3. $monitor –Monitors the argument continuously. Whenever there is a change of value in an argument list, the entire argument list is displayed.
4. $strobe-Print the values at the end of the current timestep

Functions are used to describe combinatorial logic. Functions cannot call tasks; however, it can call other functions.
Other properties of Functions:

1. Cannot drive more than one output but can have any number of inputs
2. The variables declared within the function are local and their order of declaration dictates how the variables passed to the function are used.
3. Cannot use delay, timing, or event control constructs, i.e. posedge, negedge, delay
4. Defined in the module in which they are used. It is possible to define functions in separate files and use compile directive 'include to include the function in the file which instantiates the task.

T Flip-Flop (Toggle): Useful for constructing: binary counters, clock dividers.
Registers: Stores bits of information in such a way that systems can write to or readout all the bits simultaneously.
Counters: Asynchronous Counter (count the number of events using an event signal), Synchronous Counter (use a common clock signal)

3.2 Additional constructs for behavioral modeling

Primary mechanisms to model behavior of a design: initial - mainly used for testbenches to generate inputs at a desired time, always: mainly used to describe functionality of a circuit.
Since always statements executes continuously, they are typically controlled using either delay control or event control mechanisms.

3.3 Loop Statements

1. forever: used when the procedural statement(s) need to be executed continuously
2. repeat: used when the procedural statement(s) need to be executed for a specified number of times
3. while: procedural statement(s) are executed until certain conditions become false
4. for: used when the procedural statement(s) need to be executed for a specified number of times. Unlike the repeat statement, an index variable is used which can be initialized to any desired value, and a condition can be given to terminate the loop statement.

Parameterized Model: Reuse the same model a number of times by passing a number of parameter values. Note that te parameter is defined before it's used, in case of multiple parameter definitions, the value listed during instantiation maps to the order in which the parameters are defined, all parameters must be defined, the parameter definition can be used in the same file where it's used or in a separate file.

3.4 Timing Constraints

Delay through combinational circuits depends on: number of logic levels, number of gate inputs a net drives (fan-out), the capacitive loading on the output nets.
This affects the clock speeds at which sequential designs can be operated. Timing constrains allows communicating expected performance for appropriate placement of design in LUTs, flip-flops and registers when synthesized and implemented global timing, path specific (have higher priority and routed first).

Combinatorial design: path-to-path constraint is used
Sequential circuits: period, input delay, and output delay constraints are used

3.5 Timing Paths

Defined by the connectivity between the instances of the design. In digital logic, they are formed by a pair of sequential elements controlled by the same clock, or by two different clocks:

1. **Input Port to Internal Sequential Cell Path:** Launched outside device by port clock, reaches device port after input delay, propagates through internal logic reaching a sequential cell clocked by the destination clock.
2. **Internal Path from Sequential Cell to Sequential Cell:** Launched by sequential cell clocked by source clock, propagates through internal logic reaching a sequential cell clocked by destination clock.
3. **Internal Sequential Cell to Output Port Path:** Launched inside the device by a sequential cellclocked by the source clock, propagates through internal logic reaching the output port, captured by a port clock after an additional delay called output delay.
4. **Input Port to Output Port Path:** Propagates directly from an input port to an output port without being latched inside the device.

3.6 Timing Path Sections

1. **Source Clock Path:** the path followed by the source clock from its source point to the clock pin of the launching sequential cell
2. **Data Path:** the data path is the path between the launching and capturing sequential cells.
3. **Destination Clock Path:** the path followed by the destination clock from its source point, to the clock pin of the capturing sequential cell.

3.7 Primary Clocks

Defined by the create_clock command.It must be attached to a netlist object. This netlist object represents the point in the design from which all the clock edges originate and propagate downstream on the clock tree. In other words, the source point of a primary clock defines the time zero used by the Vivado IDE when computing the clock latency and uncertainty used in the slack equation.

3.8 Using the on-board CLK (ZedBoard)

The Zynq-7000 AP SoC's PS subsystem uses a dedicated 33.3333 MHz clock sourceAn on-board 100 MHz oscillator, supplies the PL subsystem clock input on bank 13, pin Y9.

4 Zedboard and Vivado

Helpful tutorials: [5] and [1].

4.1 Model a 4-bit register with synchronous reset and load. Develop a testbench and simulate the design. Assign Clk, D input, reset, load, and output Q. Verify the design in hardware.

```verilog
module regis (clk, d, r, l, q);
input clk, r, l;
input [3:0] d;
output reg [3:0] q;

always @(posedge clk)
    if (r)
        q <= 0;
    else if (l)
        q <= d;
endmodule
```

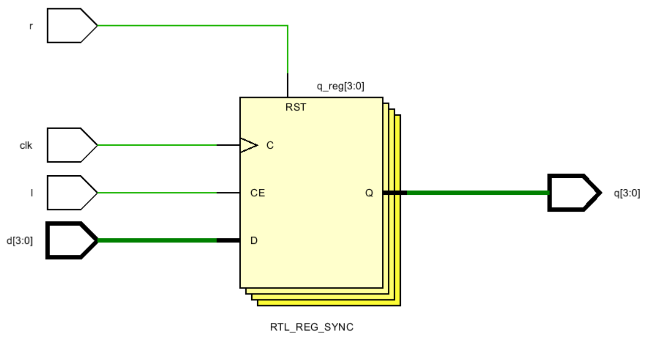
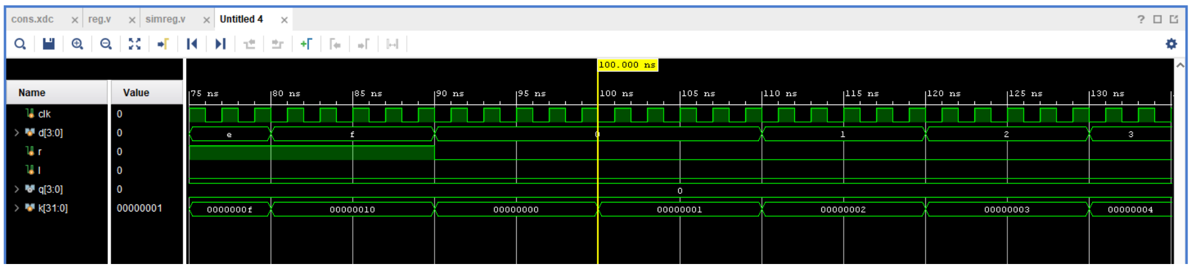


Figure 1: 4-bit register with synchronous reset and load.



Name	Value							
clk	0							
d[3:0]	0	e	f	0		1	2	3
r	0							
l	0							
q[3:0]	0	0						
k[31:0]	00000001	0000000f	00000010	00000000	00000001	00000002	00000003	00000004

Figure 2: 4-bit register with synchronous reset and load.

4.2 Model a 4-bit parallel in left shift register. Develop a testbench and simulate the design. Assign Clk, ParallelIn, load, ShiftEn, ShiftIn, RegContent, and ShiftOut. Verify the design in hardware.

```
module regis( Clk, ParallelIn, Load,
ShiftEn, ShiftIn,  RegContent, ShiftOut);

input Clk;
input [3:0] ParallelIn;
input Load;
input ShiftEn;
input ShiftIn;
output [3:0] RegContent;
output ShiftOut;
reg [3:0] shift_reg;

always @(posedge Clk)
    if (Load)
            shift_reg <= ParallelIn;
    else if (ShiftEn)
            shift_reg <= {RegContent[2:0], ShiftIn};

    assign ShiftOut = shift_reg[3];
    assign RegContent = shift_reg;

endmodule
```

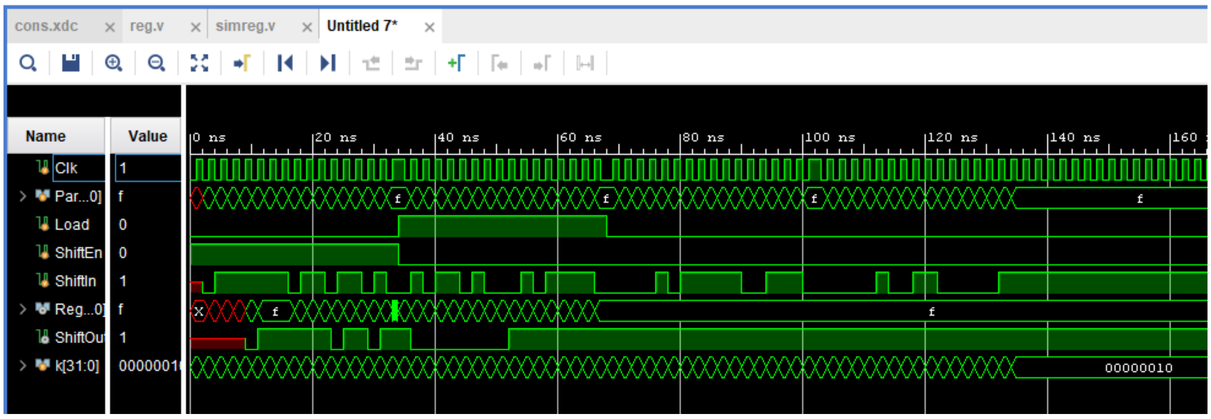


Figure 3: 4-bit parallel in left shift register

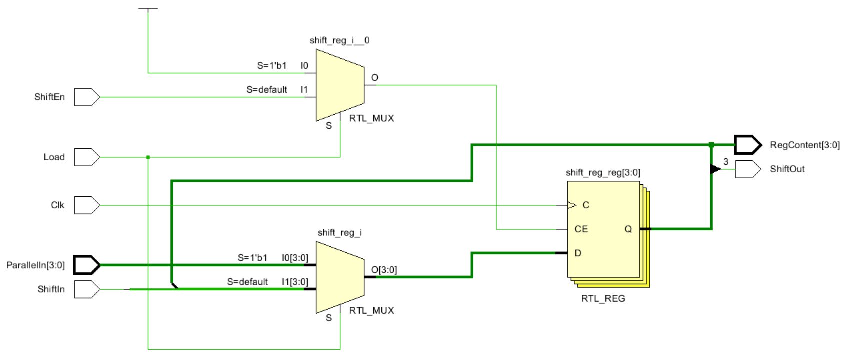
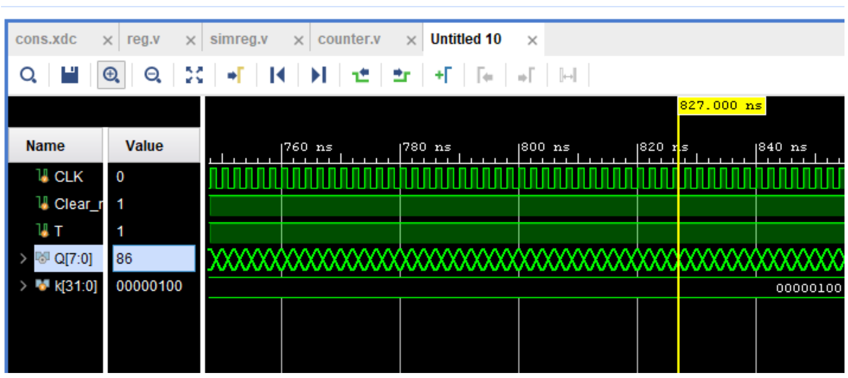


Figure 4: 4-bit parallel in left shift register

4.3 Design a 8-bit counter using T flip-flops. Us a T flip-flop in behavioral modeling and rest either in dataflow or gate-level modeling. Develop a testbench and validate the design. Assign Clock input, Clear_n, Enable, and Q. Implement the design and verify the functionality in hardware.

```
module counter( Clk,   Clear_n,  T,  Q);
input Clk;
input  Clear_n;
input T;
output [7:0] Q;

count block1(.Clk(Clk), . Clear_n( Clear_n), .T(T), .Q(Q[0]));
count block2(.Clk(Clk), . Clear_n( Clear_n), .T(T && Q[0]), .Q(Q[1]));
count block3(.Clk(Clk), . Clear_n( Clear_n), .T(T && Q[0] && Q[1]), .Q(Q[2]));
count block4(.Clk(Clk), . Clear_n( Clear_n), .T(T && Q[0] && Q[1] && Q[2]), .Q(Q[3]));
count block5(.Clk(Clk), . Clear_n( Clear_n), .T(T && Q[0] && Q[1] && Q[2] && Q[3]), .Q(Q
    [4]));
count block6(.Clk(Clk), . Clear_n( Clear_n), .T(T && Q[0] && Q[1] && Q[2] && Q[3] && Q
    [4]), .Q(Q[5]));
count block7(.Clk(Clk), . Clear_n( Clear_n), .T(T && Q[0] && Q[1] && Q[2] && Q[3] && Q
    [4] && Q[5]), .Q(Q[6]));
count block8(.Clk(Clk), . Clear_n( Clear_n), .T(T && Q[0] && Q[1] && Q[2] && Q[3] && Q
    [4] && Q[5] && Q[6]), .Q(Q[7]));

endmodule



\\ T flip flop
module count(input Clk, input  Clear_n, input T, output reg Q);

always @(negedge Clk)
    if(! Clear_n)
        Q <= 1'b0;
    else
        Q <= Q ^ T;

endmodule
```



```
Name      Value
CLK       0
Clear_n   1
T         1
Q[7:0]    86
k[31:0]   00000100
```

Figure 5: 8-bit counter using T flip-flops.

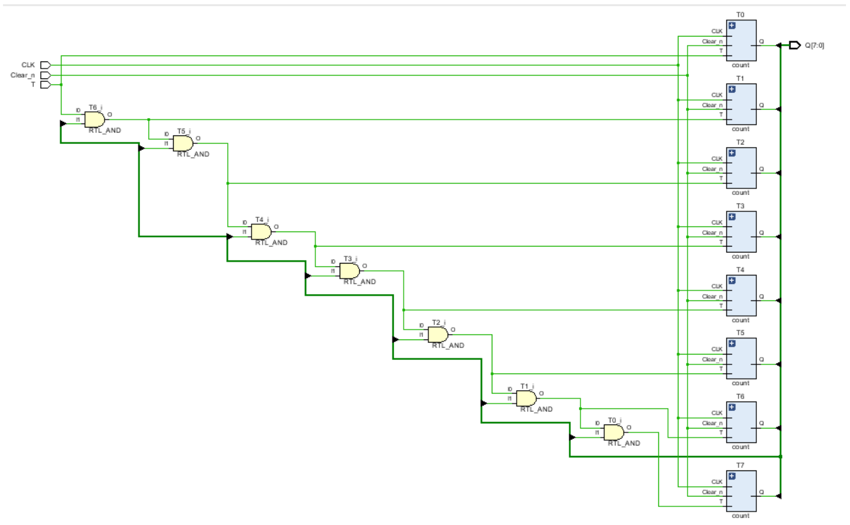


Figure 6: 8-bit counter using T flip-flops.

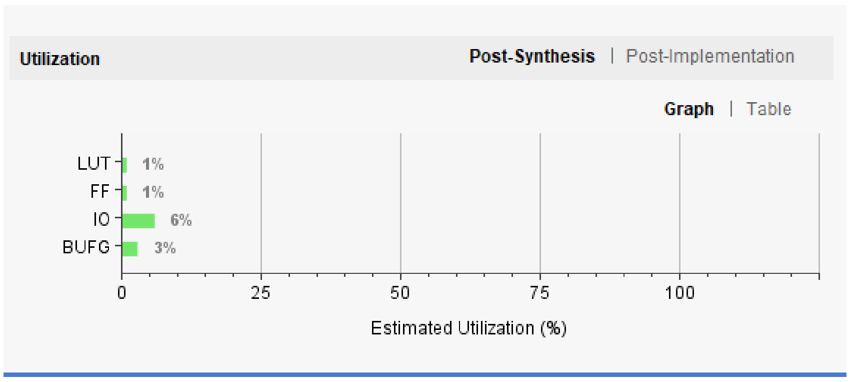
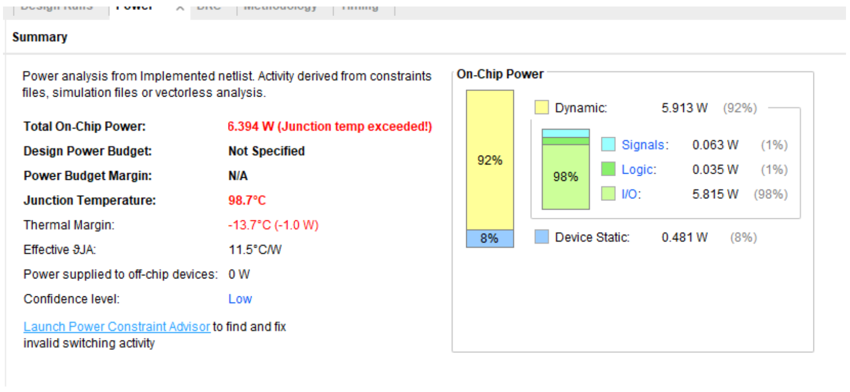


Figure 7: 8-bit counter using T flip-flops.



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**6.394 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**98.7°C**
Thermal Margin:	-13.7°C (-1.0 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	5.913 W	(92%)
Signals:	0.063 W	(1%)
Logic:	0.035 W	(1%)
I/O:	5.815 W	(98%)
Device Static:	0.481 W	(8%)

Figure 8: 8-bit counter using T flip-flops.

References

[1] DAnsermino. *Stack Exchange.* URL: https://electronics.stackexchange.com/questions/218994/8-bit-counter-from-t-flip-flops.

[2] Avnet Electronics Marketing. *Zynq^{TM}Evaluation and Development Hardware User's Guide.* URL: https://reference.digilentinc.com/_media/zedboard:zedboard_ug.pdf.

[3] Naveen Shlayan. *Sequential Logic Design Lecture 7.* URL: https://moodle.cooper.edu/moodle/pluginfile.php/82750/mod_resource/content/1/Lecture7.pdf.

[4] Naveen Shlayan. *Sequential Logic Design Lecture 8.* URL: https://moodle.cooper.edu/moodle/pluginfile.php/82902/mod_resource/content/1/Lecture8.pdf.

[5] Xlinx. *Modeling Registers and Counters.* URL: https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2015x/VHDL/docs-pdf/lab6.pdf.

[6] xlinx. *Vivado Design Suite - HLx Editions.* URL: https://www.xilinx.com/products/design-tools/vivado.html.