EEL 5721 – Reconfigurable Computing

Lab 0 – VHDL Tutorial

By

Nathan Jessurun

September 6, 2020

Introduction:

In this lab, you will be implementing and testing some basic digital circuits in VHDL to familiarize yourself with VHDL simulations and synthesis. Before starting, download all provided code off the website, which includes skeleton VHDL files and testbenches for simulations. If you are confused about the how each entity works, use google or look at the testbench code to see the correct output. Make sure to read my provided VHDL tutorial. Instructions for how to simulate and synthesis will be given in class, although I encourage you to figure it out from the ISE documentation.

You will submit this lab on e-learning, but it will not be graded. Instead, in the case of borderline final grades, I will check to see how much effort was made on this lab.

Part 1 – 2-to-4 Decoder

For part 1, you will be modifying the dec2to4.vhd file. The entity for the decoder has been provided, along with 4 empty architectures. Implement each architecture using the construct suggested by the architecture name (e.g., with select, when else, if, case). Simulate each architecture using the provided testbench dec2to4_tb.vhd. Ensure that there are no failed assertions. Synthesize your design in Xilinx ISE and ensure that there are no warnings.

Part 2 – 4-to-2 Priority Encoder

Repeat the steps of part 1 for the 4-to-2 priority encoder (enc4to2.vhd). For this part, you only need two architectures based on the if and the case statement. Assume that higher inputs have priority over lower inputs. The valid output should be asserted when any of the inputs are 1 and should be 0 when all the inputs are 0. This valid bit is needed to understand the output of "00" for an input of "0001" and for an input of "0000".

Part 3 – Adder + Register

Repeat the previous steps and create an adder followed by a register (add_pipe.vhd). As specified in the provided file, you should use a behavioral implementation. Note that the output of the adder is one bit wider than the inputs. In other words, the output should include the carry. Although it shouldn't matter, you can assume the inputs are unsigned.

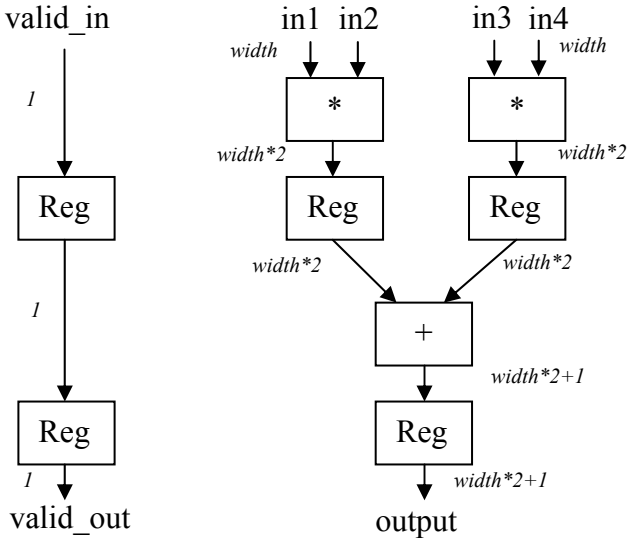
Part 4 – Multiplier + Register

Repeat part 3, but replace the adder with a multiplier (mult_pip.vhd). The output of the multiplier should be twice the width of the inputs.

Part 5 – Datapath

Implement the following datapath structurally. Use the entities from part 3 and part 4, along with the provided register entity (reg.vhd).

I



1 Working Simulations

1.1 2-to-4 Decoder

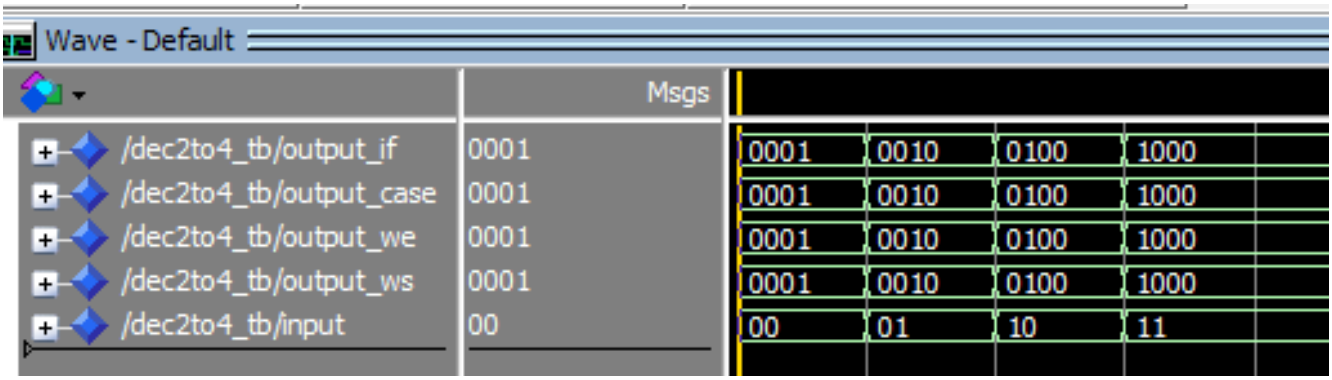


Figure 1: All architectures properly decode the input signal

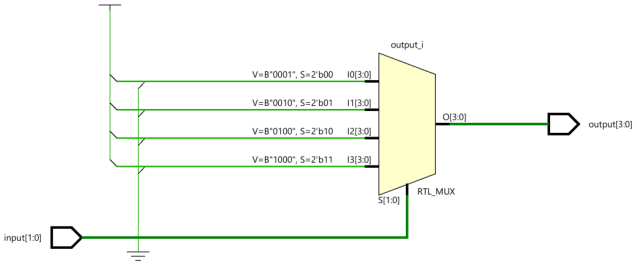


Figure 2: Synthesized schematic correctly indicates a mux implementation

1.2 4-to-2 Priority Encoder

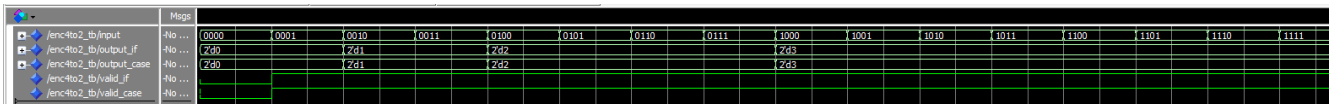


Figure 3: All architectures properly encode the input signal

1.3 Add Pipeline

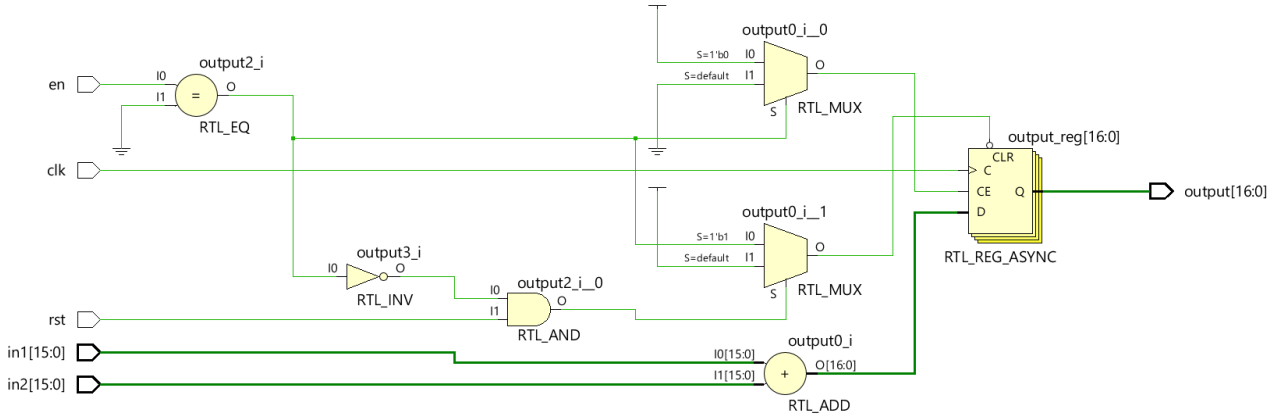


Figure 4: Add pipeline has no inferred latches

See the tabulated simulation results below indicating proper behavior when the 'done' bit indicate the final computation has appropriately propagated:

ns	delta	clk	en	rst	in1	in2	output	done
0	+2	0	1	1	0	0	0	0
5	+0	1	1	1	0	0	0	0
10	+0	0	1	1	0	0	0	0
15	+0	1	1	1	0	0	0	0
20	+0	0	1	1	0	0	0	0
25	+0	1	1	1	0	0	0	0
30	+0	0	1	1	0	0	0	0
35	+0	1	1	1	0	0	0	0
40	+0	0	1	1	0	0	0	0
45	+1	1	0	1	0	0	0	0
50	+0	0	0	1	0	0	0	0
55	+1	1	0	1	255	255	0	0
60	+0	0	0	1	255	255	0	0
65	+1	1	0	1	255	255	510	0
70	+0	0	0	1	255	255	510	0
75	+0	1	0	1	255	255	510	0
80	+0	0	0	1	255	255	510	0
85	+1	1	0	0	0	0	510	0
90	+0	0	0	0	0	0	510	0
95	+0	1	0	0	0	0	510	0
100	+0	0	0	0	0	0	510	0
105	+0	1	0	0	0	0	510	0
110	+0	0	0	0	0	0	510	0
115	+1	1	0	1	255	249	510	0
120	+0	0	0	1	255	249	510	0
125	+1	1	0	1	255	249	504	0
130	+0	0	0	1	255	249	504	0
135	+0	1	0	1	255	249	504	0
140	+0	0	0	1	255	249	504	0

1.4 Mult Pipeline

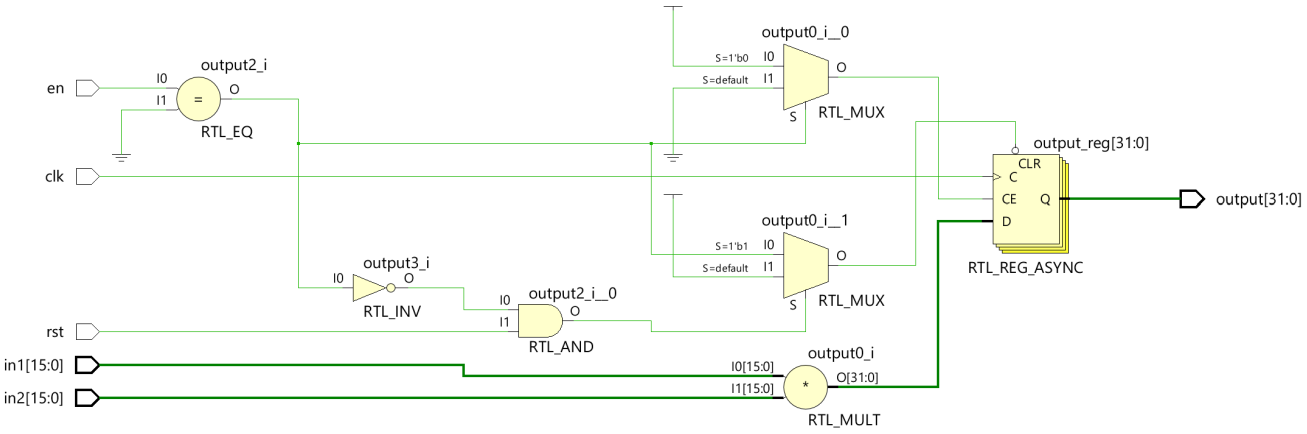


Figure 5: Mult pipeline has no inferred latches

See the tabulated simulation results below indicating proper behavior:

ns	delta	clk	en	rst	in1	in2	output	done
0	+2	0	1	1	0	0	0	0
5	+0	1	1	1	0	0	0	0
10	+0	0	1	1	0	0	0	0
15	+0	1	1	1	0	0	0	0
20	+0	0	1	1	0	0	0	0
25	+0	1	1	1	0	0	0	0
30	+0	0	1	1	0	0	0	0
35	+0	1	1	1	0	0	0	0
40	+0	0	1	1	0	0	0	0
45	+1	1	0	1	0	0	0	0
50	+0	0	0	1	0	0	0	0
55	+1	1	0	1	255	255	0	0
60	+0	0	0	1	255	255	0	0
65	+1	1	0	1	255	255	65025	0
70	+0	0	0	1	255	255	65025	0
75	+0	1	0	1	255	255	65025	0
80	+0	0	0	1	255	255	65025	0
85	+1	1	0	0	0	0	65025	0
90	+0	0	0	0	0	0	65025	0
95	+0	1	0	0	0	0	65025	0
100	+0	0	0	0	0	0	65025	0
105	+0	1	0	0	0	0	65025	0
110	+0	0	0	0	0	0	65025	0
115	+1	1	0	1	255	249	65025	0
120	+0	0	0	1	255	249	65025	0
125	+1	1	0	1	255	249	63495	0

130	+0	0	0	1	255	249	63495	0
135	+0	1	0	1	255	249	63495	0

1.5 Datapath Pipeline

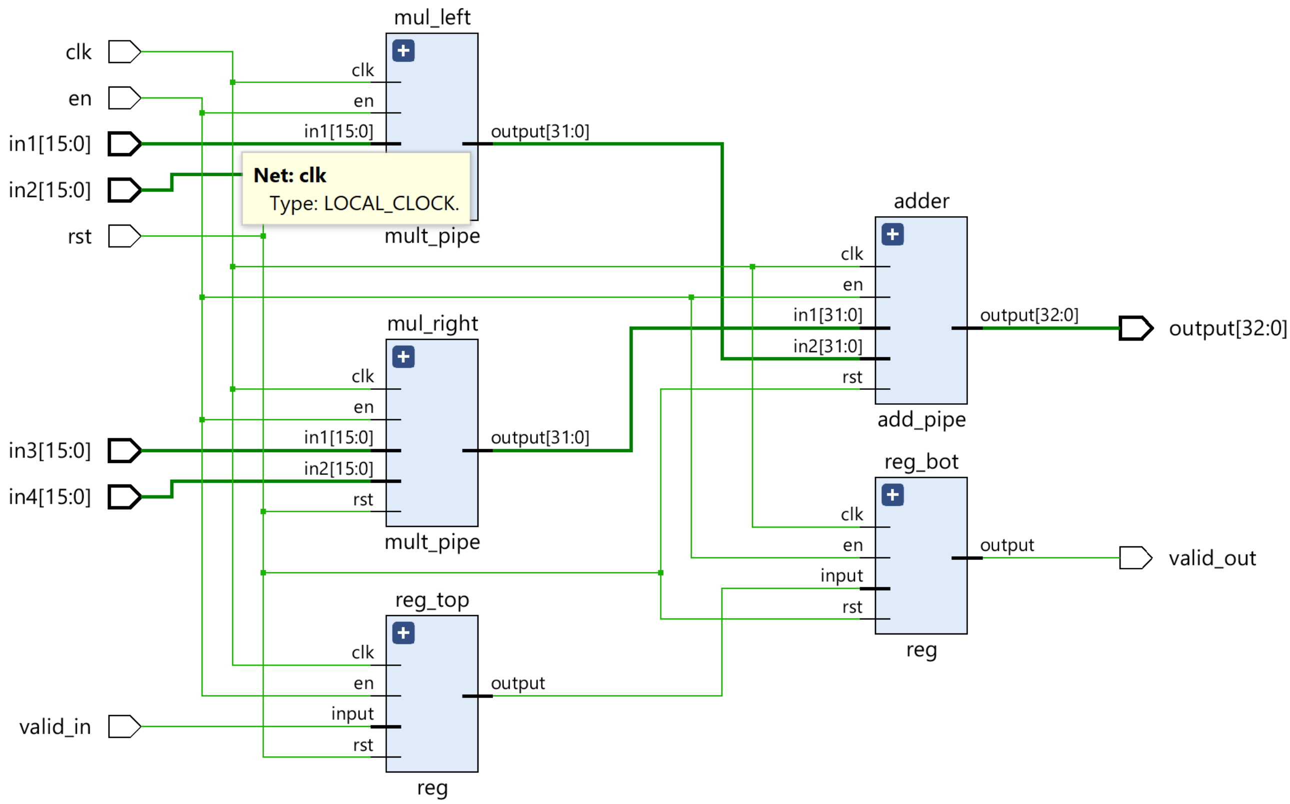


Figure 6: Datapath is correctly synthesized as an architectural diagram

See the tabulated simulation results below indicating proper behavior:

ns	delta	clk	en	rst	valid_in	valid_out	in1	in2	in3	in4	output	done
0	+2	0	1	1	0	0	0	0	0	0	0	0
5	+0	1	1	1	0	0	0	0	0	0	0	0
10	+0	0	1	1	0	0	0	0	0	0	0	0
15	+0	1	1	1	0	0	0	0	0	0	0	0
20	+0	0	1	1	0	0	0	0	0	0	0	0
25	+0	1	1	1	0	0	0	0	0	0	0	0
30	+0	0	1	1	0	0	0	0	0	0	0	0
35	+0	1	1	1	0	0	0	0	0	0	0	0
40	+0	0	1	1	0	0	0	0	0	0	0	0
45	+1	1	0	1	0	0	0	0	0	0	0	0
50	+0	0	0	1	0	0	0	0	0	0	0	0
55	+1	1	0	1	1	0	255	255	255	255	0	0
60	+0	0	0	1	1	0	255	255	255	255	0	0
65	+1	1	0	1	0	0	255	255	255	255	0	0

70	+0	0	0	1	0	0	255	255	255	255	0	0
75	+1	1	0	1	0	1	255	255	255	255	130050	0
80	+0	0	0	1	0	1	255	255	255	255	130050	0
85	+1	1	0	1	0	0	255	255	255	255	130050	0
90	+0	0	0	1	0	0	255	255	255	255	130050	0
95	+0	1	0	1	0	0	255	255	255	255	130050	0
100	+0	0	0	1	0	0	255	255	255	255	130050	0
105	+1	1	0	1	1	0	255	249	165	85	130050	0
110	+0	0	0	1	1	0	255	249	165	85	130050	0
115	+1	1	0	1	0	0	255	249	165	85	130050	0
120	+0	0	0	1	0	0	255	249	165	85	130050	0
125	+1	1	0	1	0	1	255	249	165	85	77520	0
130	+0	0	0	1	0	1	255	249	165	85	77520	0
135	+1	1	0	1	0	0	255	249	165	85	77520	0

2 Code Listings

Per instructions, all code files are zipped along with the project report.