Lab 3 - Decoders and ROMs
Hardware Design - ECE 311

January 12, 2021

Partners: Jinhan Zhang
Kevin Jiang

Abstract

In this lab, we built more complex models using what we have learned in lab 2. We built a 3 to 8 decoder, which basically required 8 2 to 1 multiplexers. We then modeled the IC chip IC74138 using dataflow modeling and the 3 to 8 decoder we built. Finally, we learned how to use ROM in Verilog and emulate a 2 bit by 2 bit multiplier by reading preset values from the ROM.

Introduction

There are three parts to this lab. The first is part is modeling a 3 to 8 binary decoder using dataflow modeling. The second part is to model the IC chip IC74138 using dataflow modeling and using the decoder constructed in the first part. The last part is emulating a 2-bit by 2-bit multiplier using ROM¿ For all three parts we need to synthesize the design, implement the design, generate the bitstream, and download it into the ZedBoard to verify the functionality. For the decoder and the IC74138, we also need a run a simulation using a provided testbench.

We were unaware during the experiment that we were meant to model IC74138 by calling our decoder module from the first part. What we did instead was directly writing the model for IC74138 using similar dataflow modeling as our model for the decoder. We did not explicitly call the decoder module, but our model functions correctly nonetheless.

Procedure

Part 1

Figure 1 shows the design schematics of the 3 to 8 decoder using dataflow modeling. The decoder takes in 3 address bits. In total there are 8 outputs, because the 3 address bits can represent 8 different output addresses. The binary number that the address bits represent is the output that will have a value of "1" assigned to it. All the other outputs get "0". So by inputting different address bits, we can select different outputs to be "1".

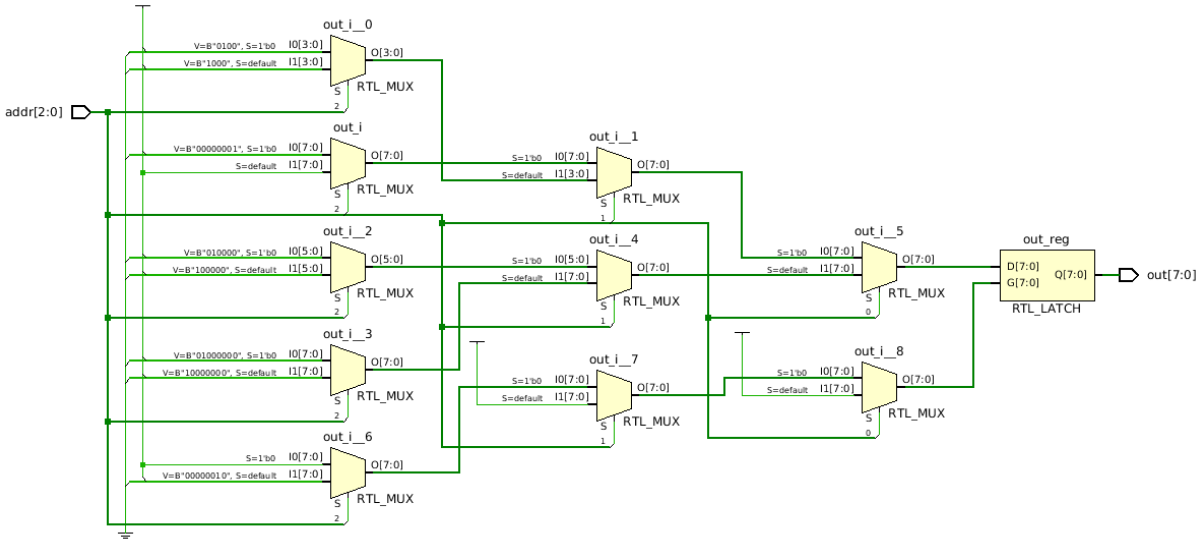

Figure 1: Design Schematics of 3 to 8 Decoder using dataflow modeling

Simulation Results

Figure 2 shows the behavioral simulation of the 3 to 8 decoder. As you can see, the simulation results match the expected results in the truth table of a typical decoder.

Synthesis Results

Figure 3 shows the synthesized schematic of the design.

Implementation Results

Project Summary is shown in Figure 4.
The implemented schematics is shown in Figure 5.
LUTs: 1% of LUTs is utilized.
IO: 4% of IO is utilized.

Summary and Discussion

Experiment	Simulation	Theoretical
The experiment was a success. We were able to select from eight different outputs by changing the address bits.	The simulation matched the expected output. Success.	From this experiment, we learned how to model a decoder by using dataflow modeling

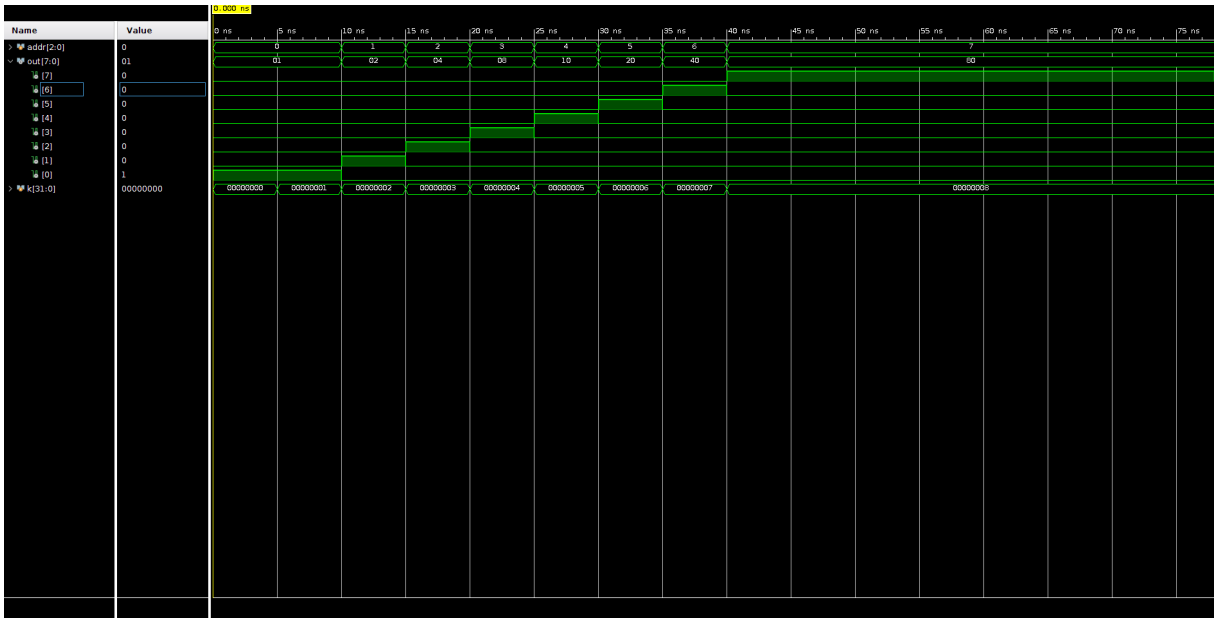

Figure 2: Behavioral Simulation of 3 to 8 Decoder using dataflow modeling

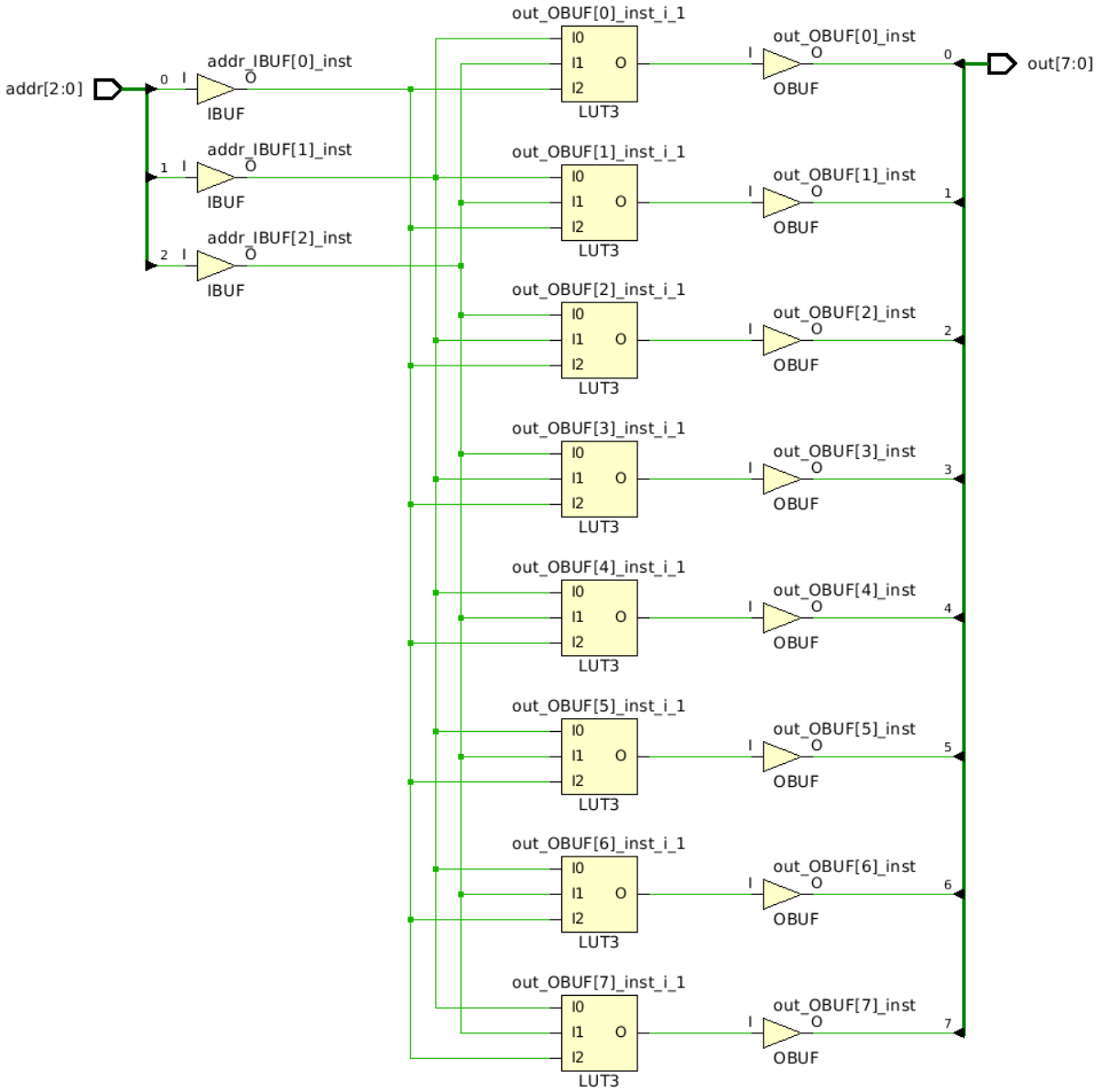

Figure 3: Synthesized Schematics of 3 to 8 Decoder using dataflow modeling

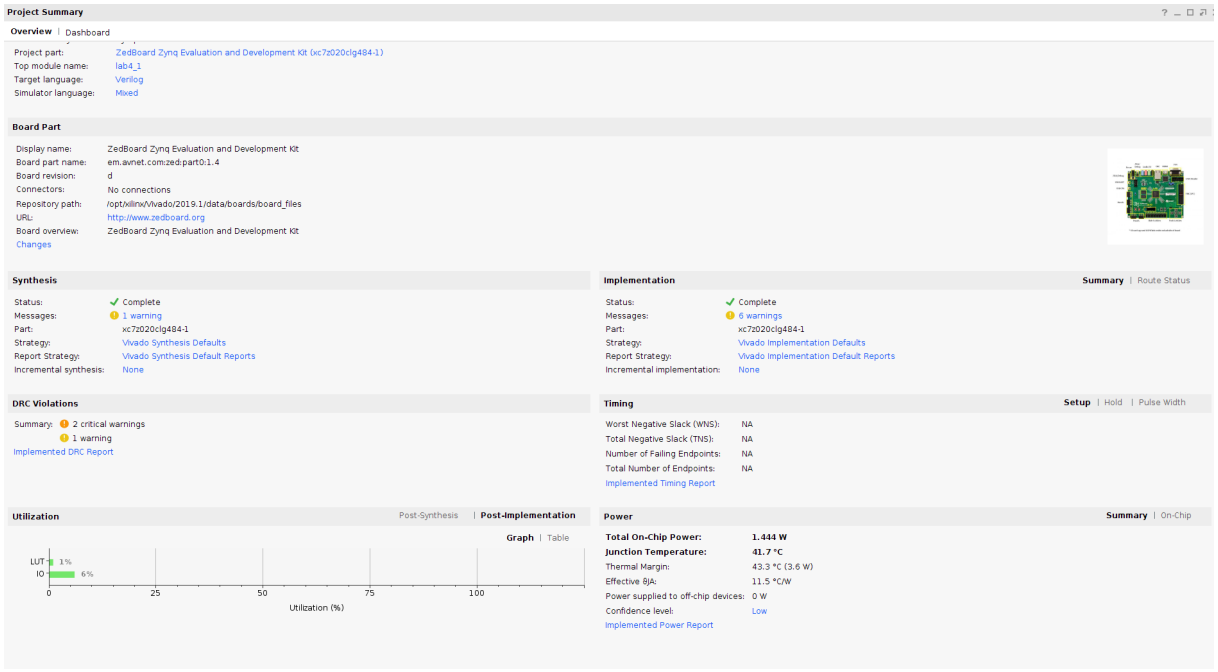

Figure 4: Project Summary of 3 to 8 Decoder using dataflow modeling

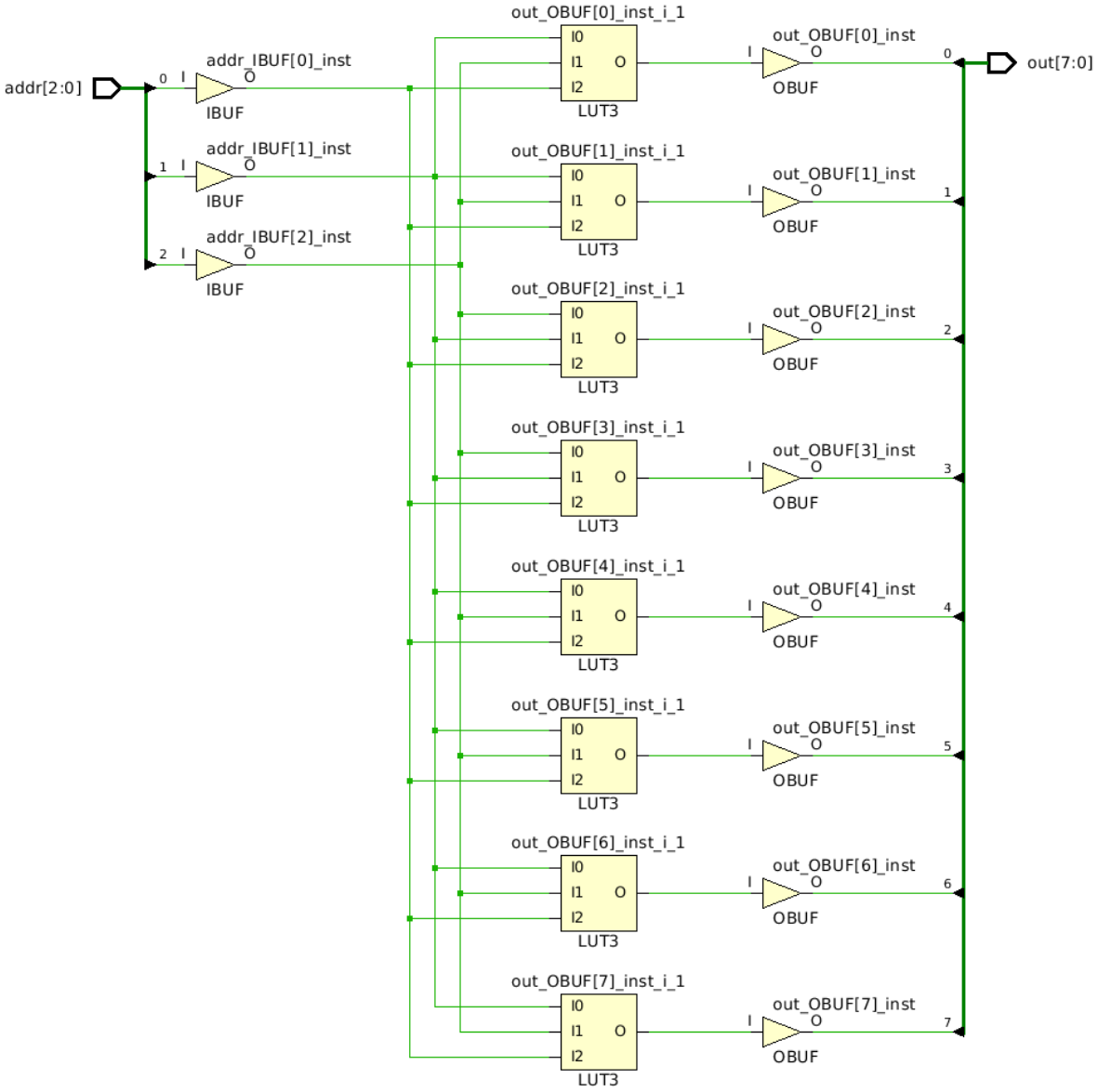

Figure 5: Implemented Schematics 3 to 8 Decoder using dataflow modeling

Part 2

Figure 6 shows the design schematics of the IC74138 chip. When g1 is "1" and g2 and g3 are "0", the chip behaves like a decoder with all the outputs going through a NOT gate. In any other case, the chip outputs all "1". As you can see from the design schematic, our model is basically the decoder (which is made up of 8 2-1 multiplexers) with some additional logic in front of the multiplexers.

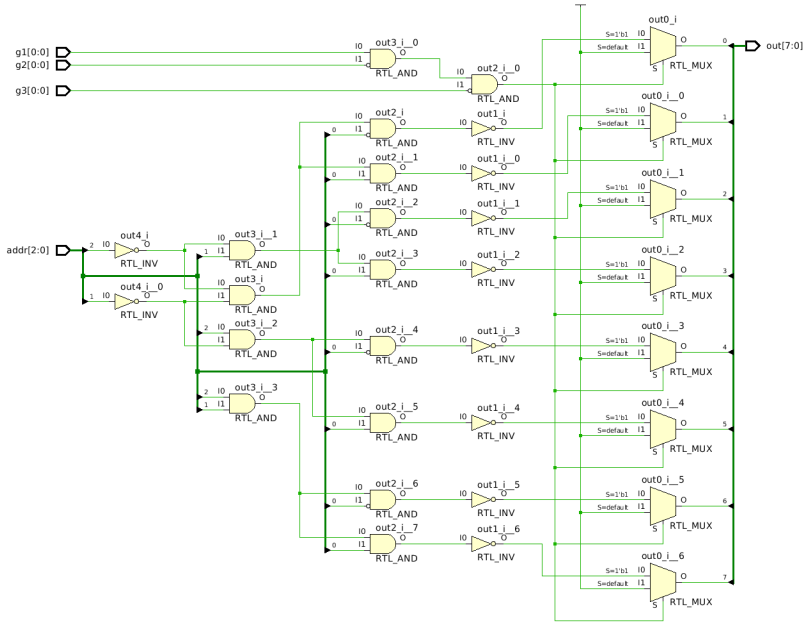

Figure 6: Design Schematics of IC74138 chip using dataflow modeling

Simulation Results

Figure 7 shows the behavioral simulation of the IC74138. As you can see, the simulation results match the expected results of the truth table shown in Figure 8.

Synthesis Results

Figure 9 shows the synthesized schematic of the design.

Implementation Results

Project Summary is shown in Figure 10.
The implemented schematics is shown in Figure 11.
LUTs: 1% of LUTs is utilized.
IO: 7% of IO is utilized.

Summary and Discussion

Experiment	Simulation	Theoretical
The experiment was a success. We were able to match the truth table of the actual IC74138	Simulation success.	From this experiment, we learned how to model an IC by using dataflow modeling and building off of our decoder.

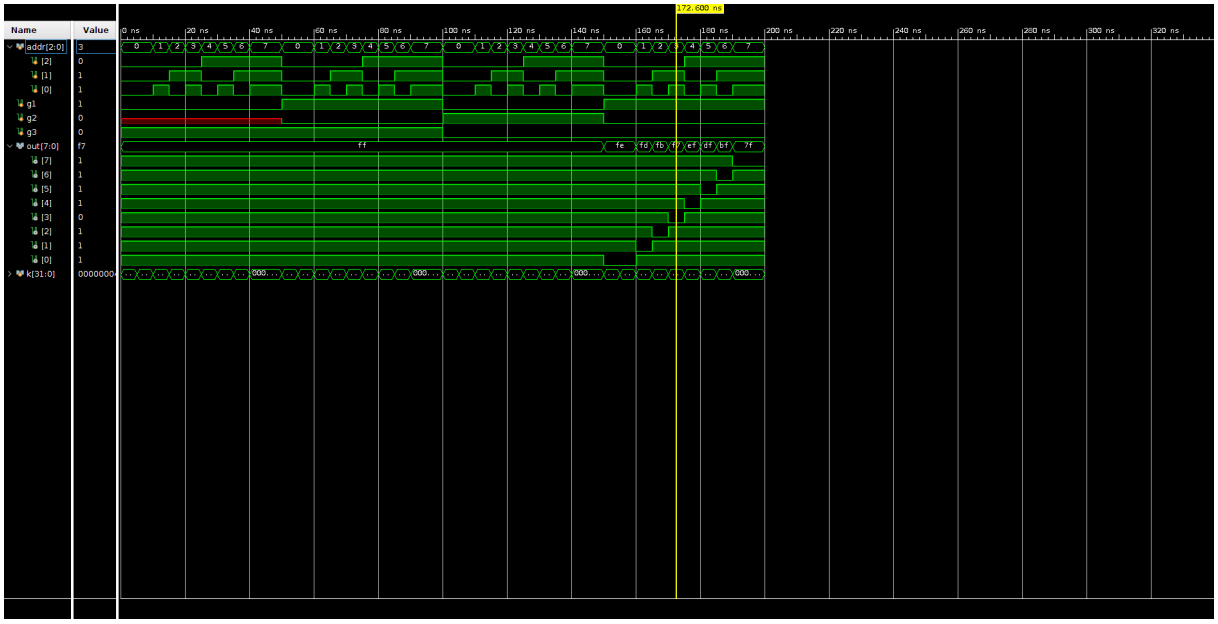

Figure 7: Behavioral Simulation of IC74138using dataflow modeling

g_1	g_{2a_n}	g_{2b_n}	x_0	x_1	x_2	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

Figure 8: Truth table of IC74138

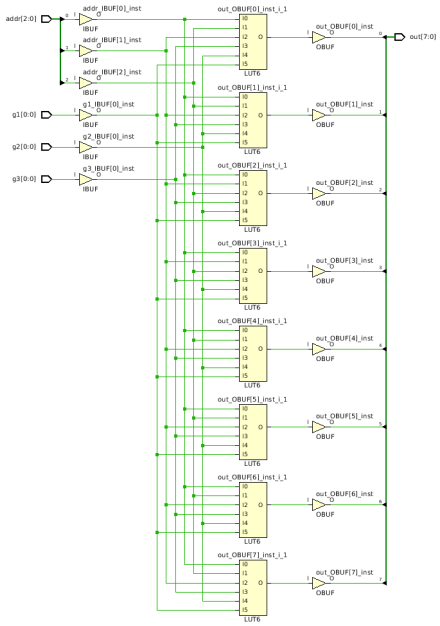

Figure 9: Synthesized Schematics of IC74138 chip using dataflow modeling

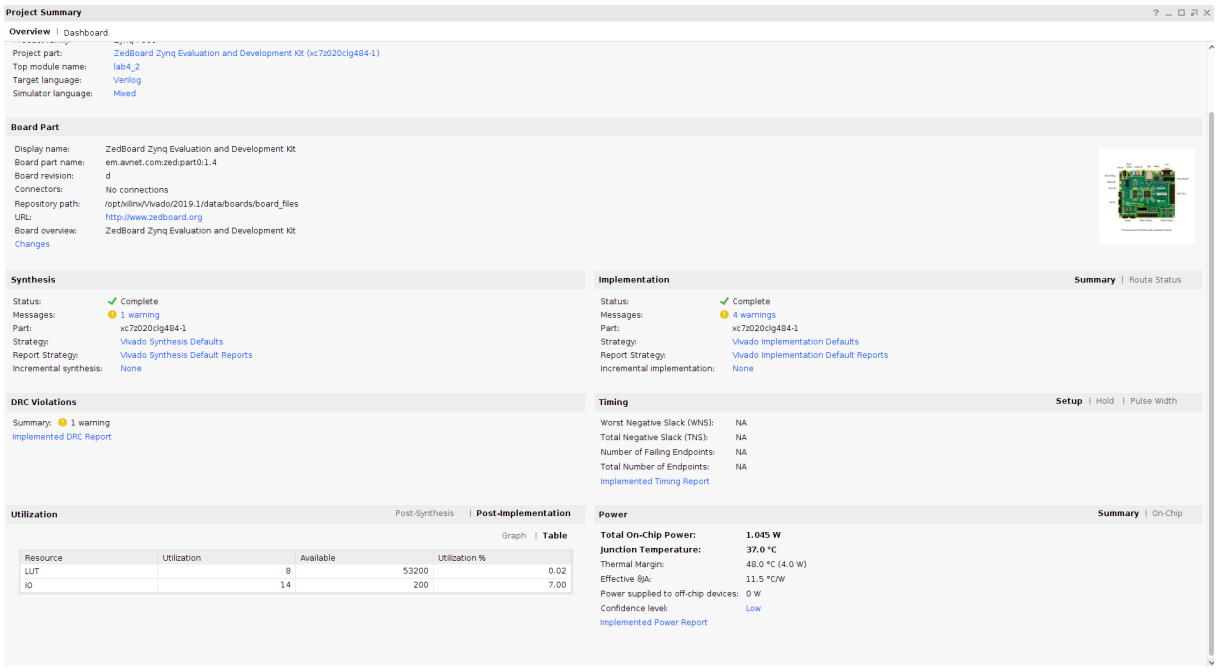

Figure 10: Project Summary of IC74138 chip using dataflow modeling

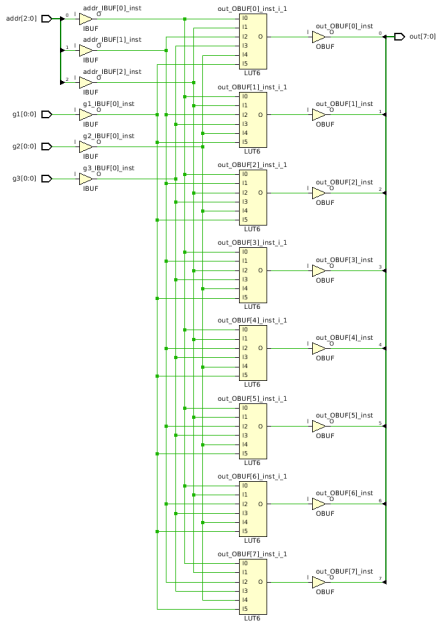

Figure 11: Implemented Schematics of IC74138 chip using dataflow modeling

Part 3

Figure 12 shows the design schematics of our 2-bit by 2-bit multiplier using ROM. The ROM has 16 different address for memory, and to access this memory we split the 4 address bits into 2 bits from one input and 2 bits from the other input. For example, if A is 10 and B is 01, your address bit is 1001, the and number stored inside ROM(1001), or ROM(9), is 10. So the output reads 10 from the ROM.

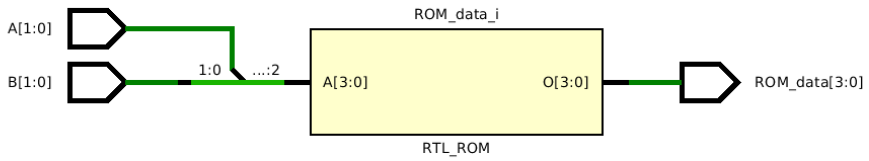

Figure 12: Design Schematics of 2 by 2 Multiplier using ROM

Synthesis Results

Figure 13 shows the synthesize schematic of the design.

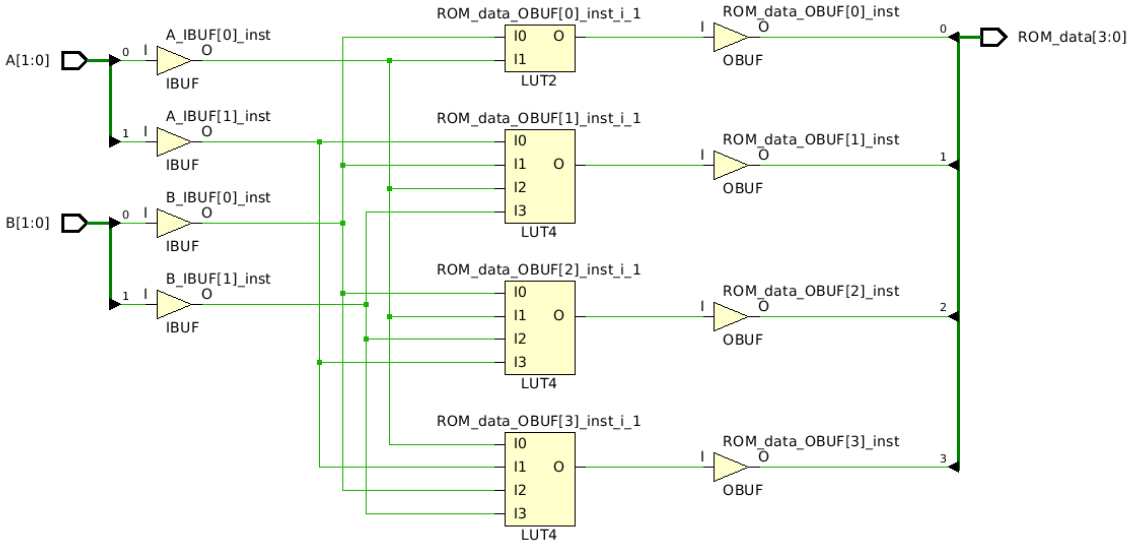

Figure 13: Design Schematics of 2 by 2 Multiplier using ROM

Implementation Results

Project Summary is shown in Figure 14.
The implemented schematics is shown in Figure 15.
LUTs: 1% of LUTs is utilized.
IO: 4% of IO is utilized.

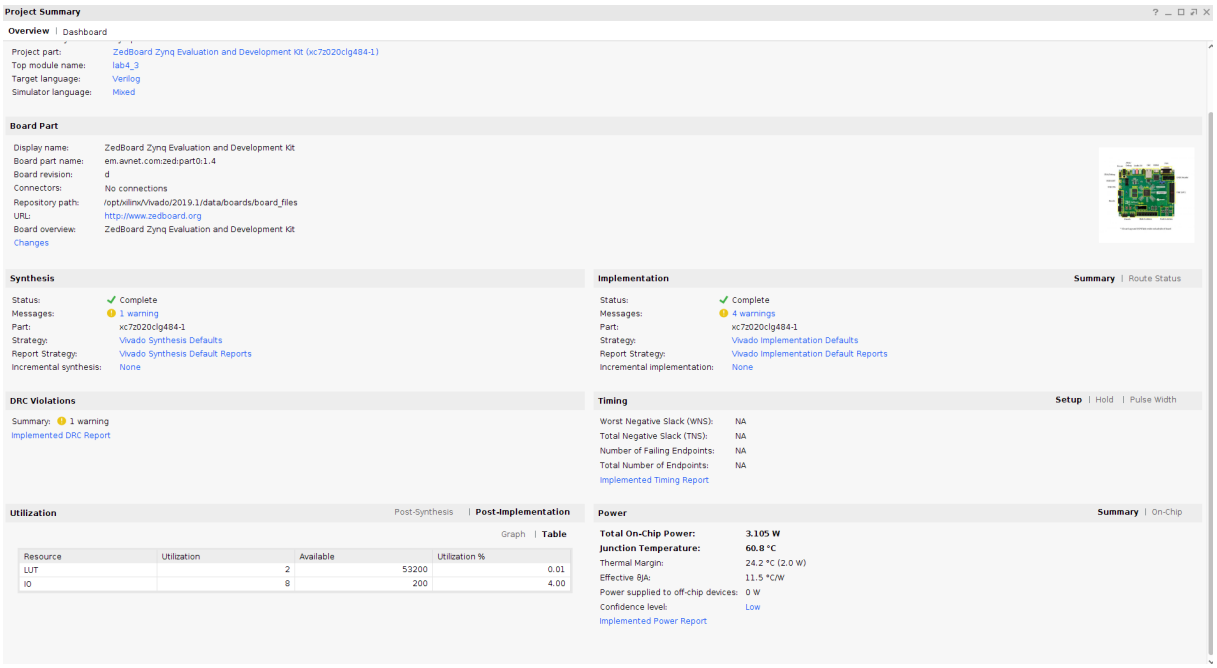

Figure 14: Project Summary of 2 by 2 Multiplier using ROM

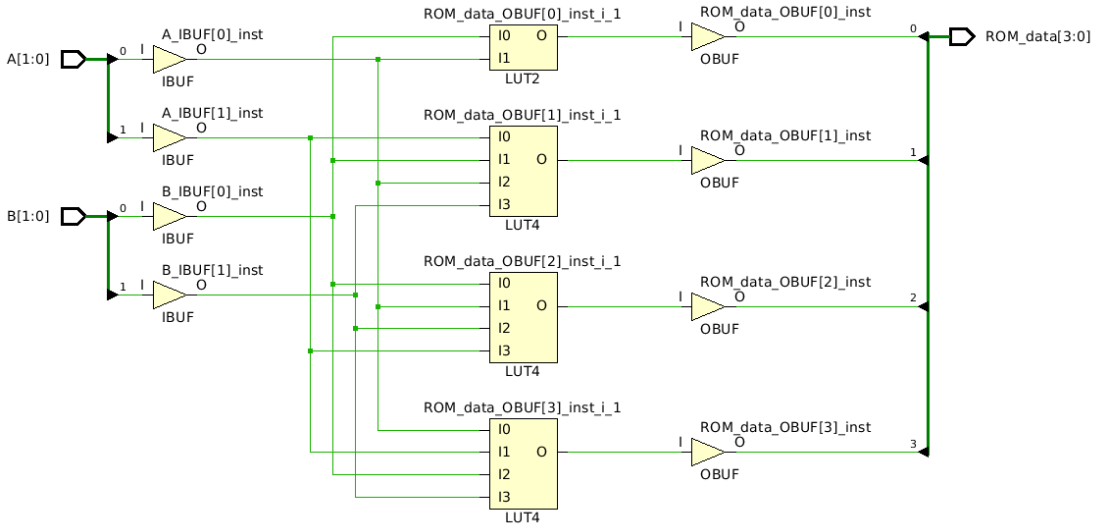

Figure 15: Implemented Schematics of 2 by 2 Multiplier using ROM

Summary and Discussion

Experiment	Simulation	Theoretical
The experiment was a success. We were able to emulate a 2 by 2 multiplier with ROM	No simulation required.	From this experiment, we learned how to model read from ROM to emulate certain circuits.

Overall Conclusions

In this lab, we applied what we learned in Lab 2 to create more complex modules. We combined multiplexers to create a decoder, and we derived a model for IC74138 based on our decoder model. We also learned a new skill, using ROMs. By using ROMs, we were able to generte the same outputs as a multipler from the same set of inputs without actually implementing the multiplier.

Appendix A Part 1 Code

```
1   `timescale 1ns / 1ps
2
3   module lab4_1(input [2:0] addr, output [7:0]out);
4       assign out[0] = (~addr[2] & ~addr[1] & ~addr[0]);
5       assign out[1] = (~addr[2] & ~addr[1] & addr[0]);
6       assign out[2] = (~addr[2] & addr[1] & ~addr[0]);
7       assign out[3] = (~addr[2] & addr[1] & addr[0]);
8       assign out[4] = (addr[2] & ~addr[1] & ~addr[0]);
9       assign out[5] = (addr[2] & ~addr[1] & addr[0]);
10      assign out[6] = (addr[2] & addr[1] & ~addr[0]);
11      assign out[7] = (addr[2] & addr[1] & addr[0]);
12  endmodule
```

Figure 16: Design Source code for Part 1

Appendix B Part 2 Code

```
1  `timescale 1ns / 1ps
2
3
4  module lab4_2(
5      addr,
6      out,
7      g1,
8      g2,
9      g3
10     );
11     input wire [0:0] g1;
12     input wire [0:0] g2;
13     input wire [0:0] g3;
14     input wire [2:0] addr; // 2 is least significant bit
15     output wire [7:0] out; // 8 outputs
16     assign out[0] = (g1 & ~g2 & ~g3) ? ~(~addr[2]&~addr[1]&~addr[0]) : 1;
17     assign out[1] = (g1 & ~g2 & ~g3) ? ~(~addr[2]&~addr[1]&addr[0]) : 1;
18     assign out[2] = (g1 & ~g2 & ~g3) ? ~(~addr[2]&addr[1]&~addr[0]) : 1;
19     assign out[3] = (g1 & ~g2 & ~g3) ? ~(~addr[2]&addr[1]&addr[0]) : 1;
20     assign out[4] = (g1 & ~g2 & ~g3) ? ~(addr[2]&~addr[1]&~addr[0]) : 1;
21     assign out[5] = (g1 & ~g2 & ~g3) ? ~(addr[2]&~addr[1]&addr[0]) : 1;
22     assign out[6] = (g1 & ~g2 & ~g3) ? ~(addr[2]&addr[1]&~addr[0]) : 1;
23     assign out[7] = (g1 & ~g2 & ~g3) ? ~(addr[2]&addr[1]&addr[0]) : 1;
24 endmodule
25
```

Figure 17: Design Source code for Part 2

Appendix C Part 3 Code

```
1  `timescale 1ns / 1ps
2
3
4  module lab4_3 (ROM_data, A, B);
5  input [1:0] A;
6  input [1:0] B;
7  output [3:0] ROM_data;
8  reg [3:0] ROM [15:0]; // defining 4x2 ROM
9  assign ROM_data = ROM[{A, B}]; // reading ROM content at the address ROM_addr
10 initial $readmemb ("Multiplier_outputs.txt", ROM, 0, 15); // load ROM content from ROM_data.txt file
11 endmodule
12
```

Figure 18: Design Source code for Part 3

Appendix D Testbench Code for Part 1

```
1  `timescale 1ns / 1ps
2
3  module lab4_1test(
4  );
5  reg [2:0] addr;
6  wire [7:0] out;
7  integer k;
8  lab4_1 DUT (.addr(addr), .out(out));
9  initial
10 begin
11 addr = 0;
12 for (k=0; k < 8; k=k+1)
13 #5 addr=k;
14 #10;
15 end
16 endmodule
```

Figure 19: Simulation Source code for Part 1

Appendix E Testbench Code for Part 2

```
1  `timescale 1ns / 1ps
2  module lab4_2testFile( );
3  reg [2:0] addr;
4  reg g1, g2, g3;
5  wire [7:0] out;
6  integer k;
7  lab4_2 DUT (.g1(g1), .g2(g2), .g3(g3), .addr(addr), .out(out));
8  initial
9  begin
10 addr = 0; g1 = 0; g3 = 1; g3 = 1;
11 for (k=0; k < 8; k=k+1)
12 #5 addr=k;
13 #10;
14 addr = 0; g1 = 1; g2 = 0; g3 = 1;
15 for (k=0; k < 8; k=k+1)
16 #5 addr=k;
17 #10;
18 addr = 0; g1 = 0; g2 = 1; g3 = 0;
19 for (k=0; k < 8; k=k+1)
20 #5 addr=k;
21 #10;
22 addr = 0; g1 = 1; g2 = 0; g3 = 0;
23 for (k=0; k < 8; k=k+1)
24 #5 addr=k;
25 #10;
26 end
27 endmodule
```

Figure 20: Simulation Source code for Part 2

Appendix F Constraint File for Part 2

```
# Assign inputs/outputs to actual pins on the FPGA
set_property PACKAGE_PIN M15 [get_ports {g1}]
set_property PACKAGE_PIN H17 [get_ports {g2}]
set_property PACKAGE_PIN H18 [get_ports {g3}]
set_property PACKAGE_PIN H19 [get_ports {addr[0]}]
set_property PACKAGE_PIN F21 [get_ports {addr[1]}]
set_property PACKAGE_PIN H22 [get_ports {addr[2]}]
set_property PACKAGE_PIN U14 [get_ports {out[0]}]
set_property PACKAGE_PIN U19 [get_ports {out[1]}]
set_property PACKAGE_PIN W22 [get_ports {out[2]}]
set_property PACKAGE_PIN V22 [get_ports {out[3]}]
set_property PACKAGE_PIN U21 [get_ports {out[4]}]
set_property PACKAGE_PIN U22 [get_ports {out[5]}]
set_property PACKAGE_PIN T21 [get_ports {out[6]}]
set_property PACKAGE_PIN T22 [get_ports {out[7]}]
# Define voltage levels (3.3 for LEDs and 1.8 for Switches)
set_property IOSTANDARD LVCMOS18 [get_ports {g1}]
set_property IOSTANDARD LVCMOS18 [get_ports {g2}]
set_property IOSTANDARD LVCMOS18 [get_ports {g3}]
set_property IOSTANDARD LVCMOS18 [get_ports {addr[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {addr[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {addr[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[7]}]
```

Figure 21: Constraint File for Part 2

Appendix G Constraint File for Part 3

```
# Assign inputs/outputs to actual pins on the FPGA
set_property PACKAGE_PIN M15 [get_ports {g1}]
set_property PACKAGE_PIN H17 [get_ports {g2}]
set_property PACKAGE_PIN H18 [get_ports {g3}]
set_property PACKAGE_PIN H19 [get_ports {addr[0]}]
set_property PACKAGE_PIN F21 [get_ports {addr[1]}]
set_property PACKAGE_PIN H22 [get_ports {addr[2]}]
set_property PACKAGE_PIN U14 [get_ports {out[0]}]
set_property PACKAGE_PIN U19 [get_ports {out[1]}]
set_property PACKAGE_PIN W22 [get_ports {out[2]}]
set_property PACKAGE_PIN V22 [get_ports {out[3]}]
set_property PACKAGE_PIN U21 [get_ports {out[4]}]
set_property PACKAGE_PIN U22 [get_ports {out[5]}]
set_property PACKAGE_PIN T21 [get_ports {out[6]}]
set_property PACKAGE_PIN T22 [get_ports {out[7]}]
# Define voltage levels (3.3 for LEDs and 1.8 for Switches)
set_property IOSTANDARD LVCMOS18 [get_ports {g1}]
set_property IOSTANDARD LVCMOS18 [get_ports {g2}]
set_property IOSTANDARD LVCMOS18 [get_ports {g3}]
set_property IOSTANDARD LVCMOS18 [get_ports {addr[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {addr[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {addr[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out[7]}]
```

Figure 22: Constraint File for Part 3