ENGR 272 Lab 0: Digital Logic 4-bit Unsigned Multiplier Example Report

Sep 10, 2019
Craig Munsee

0.1 Objectives:

The objective of this lab is to design and build a 4-bit Unsigned Multiplier using a logic gate-level design. This is to be simulated and demonstrated on the DE-10-Lite FPGA. Additionally, this report is intended to be an example of what is expected for the lab reports in this class.

0.2 Equipment/Parts/Materials:

1. Quartus Prime Lite Edition V. 18.1
2. DE10-Lite kit with MAX10 10M50DAF484C7G FPGA
3. USB to USB-B cable
4. The ECE 271 textbook, Digital Design and Computer Architecture by Drs. David and Sarah Harris CH. 5 pg. 253

0.3 Procedure:

0.3.1 Design:

The process of designing the multiplier was started by doing some research on how a Multiplier is designed. In chapter 5 of the textbook an example of how a multiplier is designed was discussed. The method used involved the ANDing of the bits and then adding the shifted left values. the Full adder Logic is given in the Sum of Products Form is given below (Section 2.2.2 of Textbook)

$SUM = A \oplus B \oplus Cin$

$Cout = AB + ACin + BCin$

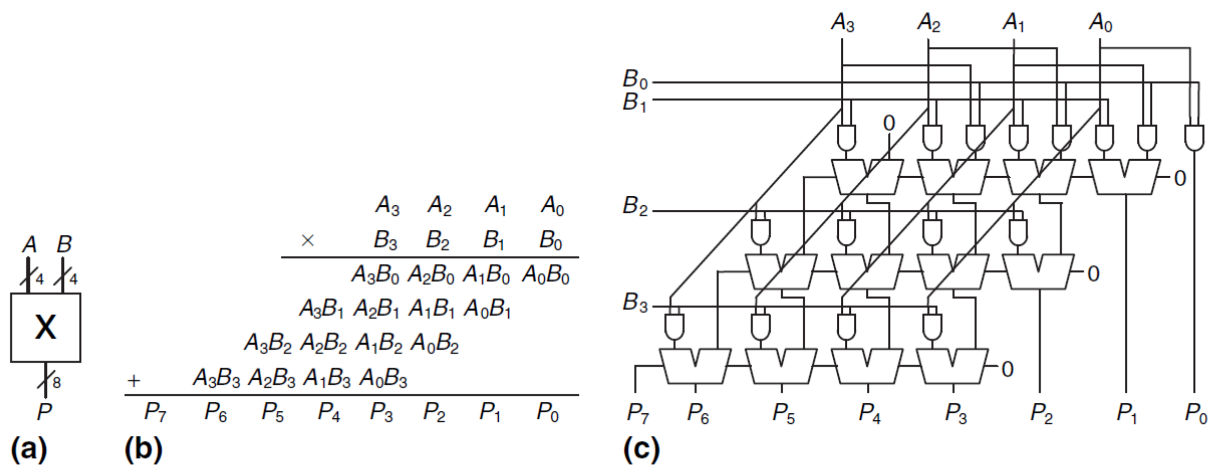
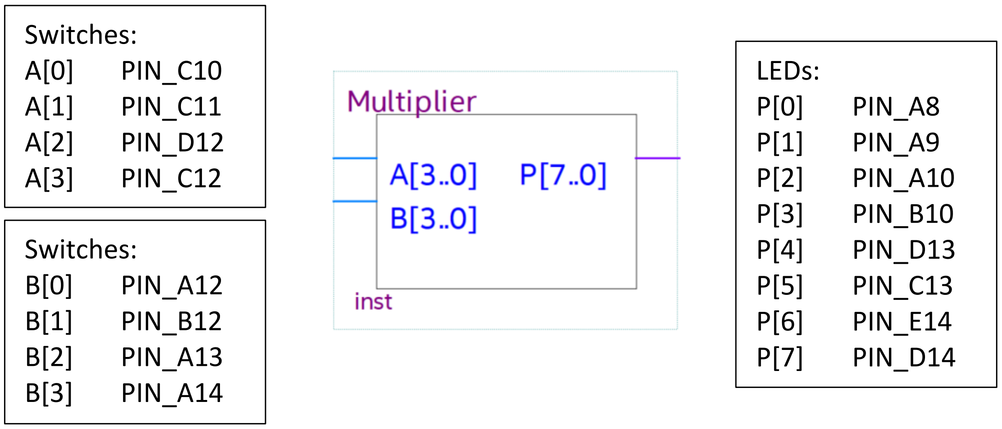

A B
4 4
X
8
P
A_3 A_2 A_1 A_0
× B_3 B_2 B_1 B_0
A_3B_0 A_2B_0 A_1B_0 A_0B_0
A_3B_1 A_2B_1 A_1B_1 A_0B_1
A_3B_2 A_2B_2 A_1B_2 A_0B_2
+ A_3B_3 A_2B_3 A_1B_3 A_0B_3
P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0
B_0
B_1
B_2
B_3
0
(a)
(b)
(c)

Figure 5.20 from Digital Design and Computer Architecture by Drs. David and Sarah Harris is shown below that demonstrates the design of a Multiplier.

0.3.2 Design Entry:

The gate-level design was implemented in Quartus Prime Lite. A high-level block diagram of the 4-bit Multiplier is shown in Figure 0.1. Figure 0.2 shows the entire layout of the 4-bit Multiplier design with AND gates and full adder circuits. Figure 0.3 shows the gate-level design of the full adder circuit.



Switches:

A[0]	PIN_C10
A[1]	PIN_C11
A[2]	PIN_D12
A[3]	PIN_C12

Switches:

B[0]	PIN_A12
B[1]	PIN_B12
B[2]	PIN_A13
B[3]	PIN_A14

LEDs:

P[0]	PIN_A8
P[1]	PIN_A9
P[2]	PIN_A10
P[3]	PIN_B10
P[4]	PIN_D13
P[5]	PIN_C13
P[6]	PIN_E14
P[7]	PIN_D14

Figure 0.1: A high-level block diagram of the 4-bit Multiplier

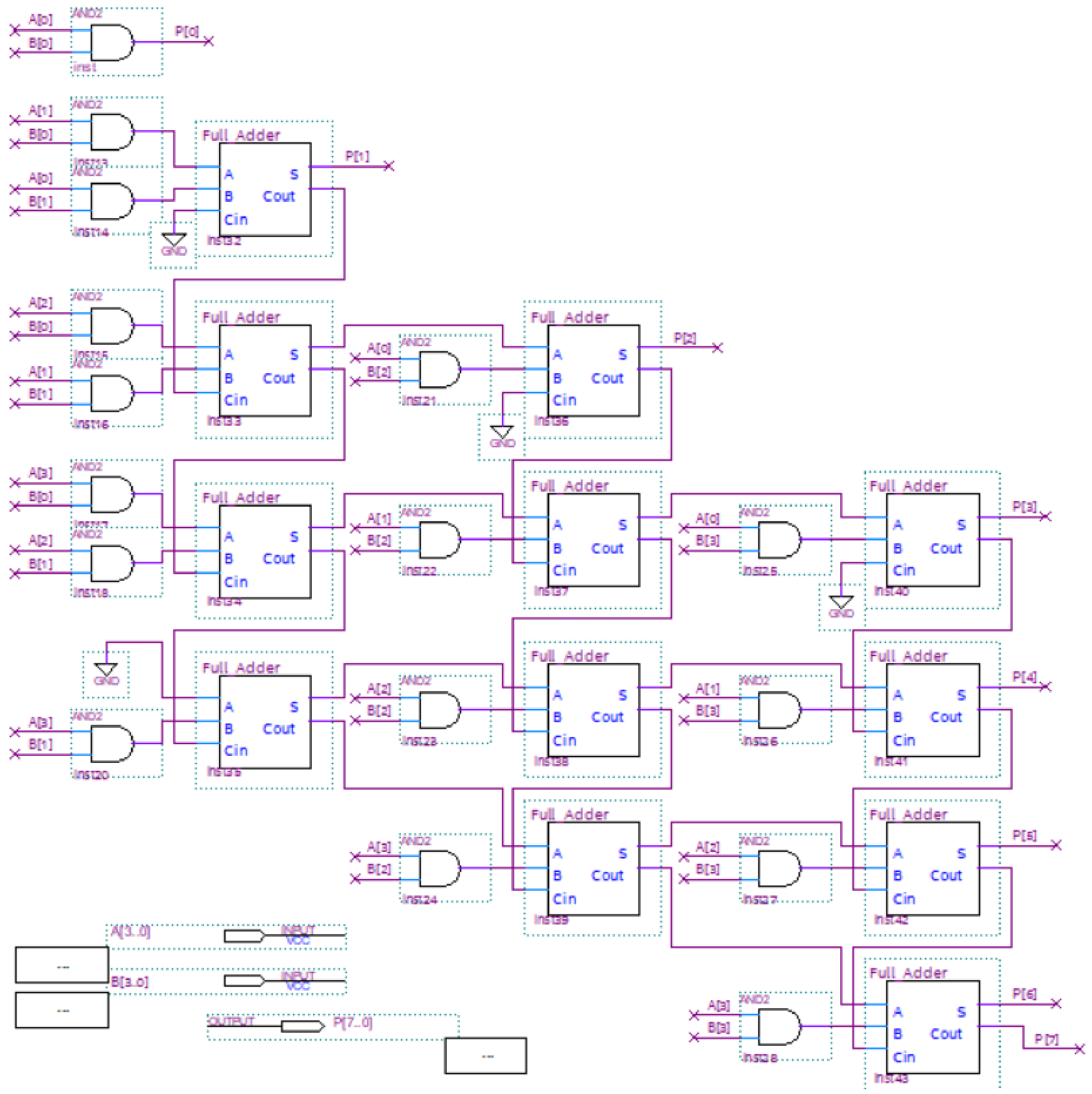


Figure 0.2: The entire layout of the 4-bit Multiplier design with AND gates and full adder circuits.

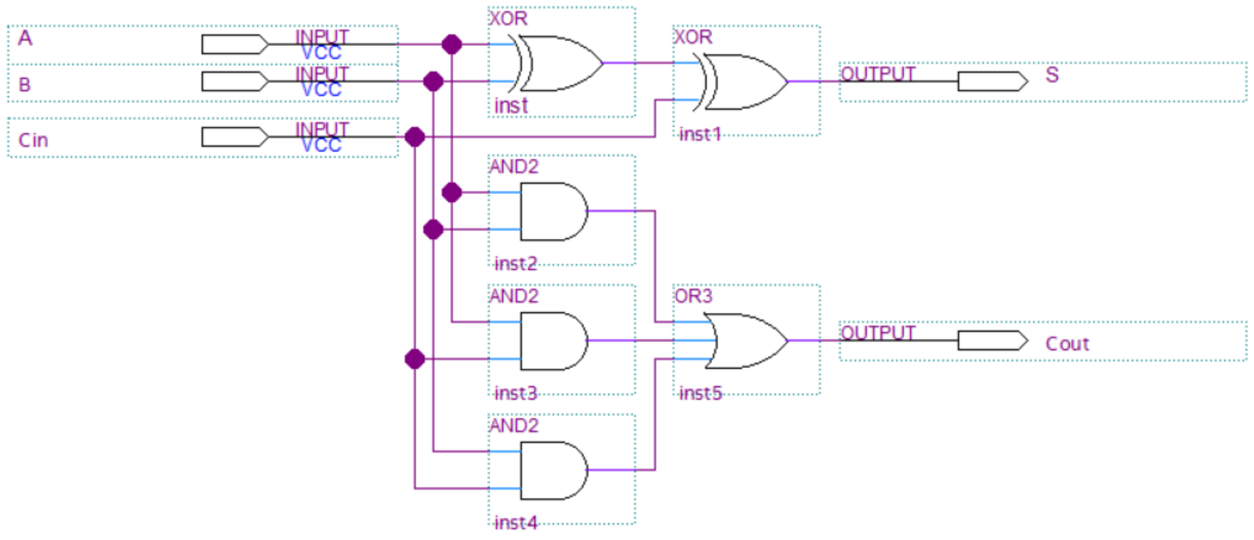
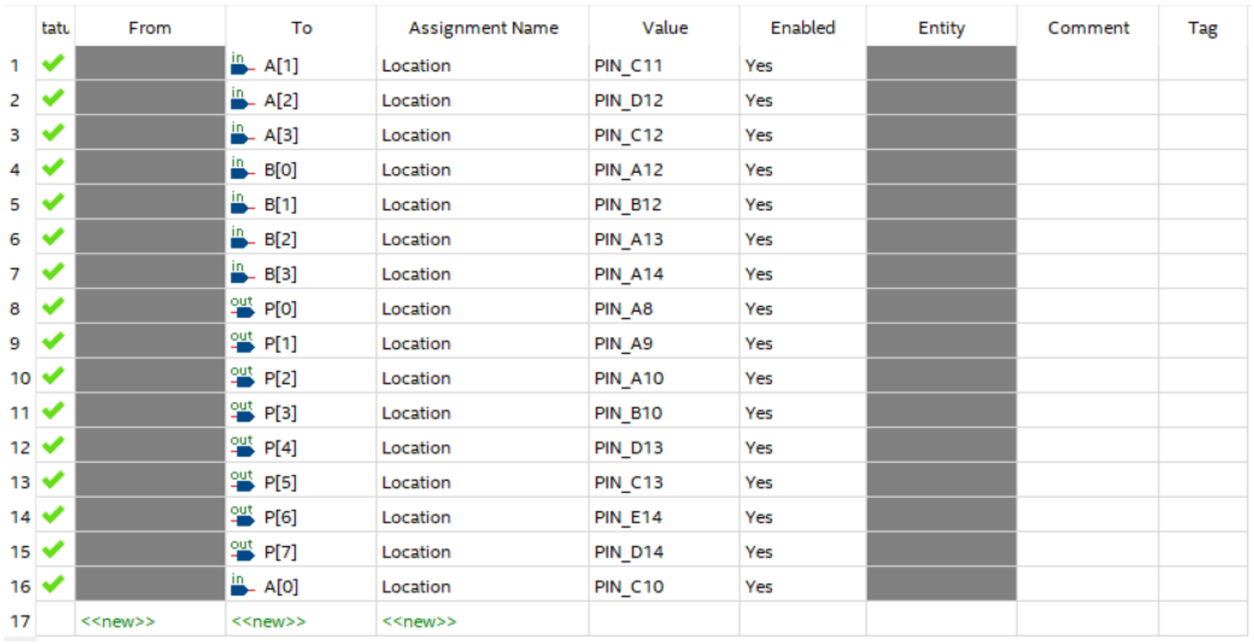


Figure 0.3: The gate-level design of the full adder circuit.



	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		A[1]	Location	PIN_C11	Yes			
2	✓		A[2]	Location	PIN_D12	Yes			
3	✓		A[3]	Location	PIN_C12	Yes			
4	✓		B[0]	Location	PIN_A12	Yes			
5	✓		B[1]	Location	PIN_B12	Yes			
6	✓		B[2]	Location	PIN_A13	Yes			
7	✓		B[3]	Location	PIN_A14	Yes			
8	✓		P[0]	Location	PIN_A8	Yes			
9	✓		P[1]	Location	PIN_A9	Yes			
10	✓		P[2]	Location	PIN_A10	Yes			
11	✓		P[3]	Location	PIN_B10	Yes			
12	✓		P[4]	Location	PIN_D13	Yes			
13	✓		P[5]	Location	PIN_C13	Yes			
14	✓		P[6]	Location	PIN_E14	Yes			
15	✓		P[7]	Location	PIN_D14	Yes			
16	✓		A[0]	Location	PIN_C10	Yes			
17		<<new>>	<<new>>	<<new>>					

Figure 0.4: The pin assignments used in Quartus.

0.3.3 Design Simulation:

Table 0.1 shows a partial 4-bit Multiplier truth table that was used during the verification process. Modelsim was used for the simulation of the Multiplier circuit. Verilog files for each of the schematics were generated using the process below.

To generate the Verilog Hardware Description Language file for use in Modelsim, go to File -> create/update -> create HDL Design File for current file -> Verilog

A Verilog file was created for both the Adder circuit and the Multiplier circuit. The code for each of the Verilog files is included in the appendix of this report.

Figure 0.5 shows the ModelSim simulation showing the results for the first 14 values of the truth table. Figure 0.6 shows the ModelSim simulation showing the results for the last 14 values of the truth table. The ModelSim simulation matched the values in Table 0.1.

Operand1 (Binary)	Operand1 (Decimal)	X	Operand2 (Binary)	Operand2 (Decimal)	=	Value (8-bit Binary)	Value (Unsigned Decimal)	Value (Hexadecimal)
0b 0000	0	X	0b 0000	0	=	0b 0000 0000	0	0
0b 0001	1	X	0b 0001	1	=	0b 0000 0001	1	1
0b 0001	1	X	0b 0010	2	=	0b 0000 0010	2	2
0b 0001	1	X	0b 0100	4	=	0b 0000 0100	4	4
0b 0001	1	X	0b 1000	8	=	0b 0000 1000	8	8
0b 0010	2	X	0b 0001	1	=	0b 0000 0010	2	2
0b 0010	2	X	0b 0010	2	=	0b 0000 0100	4	4
0b 0010	2	X	0b 0100	4	=	0b 0000 1000	8	8
0b 0010	2	X	0b 1000	8	=	0b 0001 0000	16	10
0b 0100	4	X	0b 0001	1	=	0b 0000 0100	4	4
0b 0100	4	X	0b 0010	2	=	0b 0000 1000	8	8
0b 0100	4	X	0b 0100	4	=	0b 0001 0000	16	10
0b 0100	4	X	0b 1000	8	=	0b 0010 0000	32	20
0b 1000	8	X	0b 0001	1	=	0b 0000 1000	8	8
0b 1000	8	X	0b 0010	2	=	0b 0001 0000	16	10
0b 1000	8	X	0b 0100	4	=	0b 0010 0000	32	20
0b 1000	8	X	0b 1000	8	=	0b 0100 0000	64	40
0b 1111	15	X	0b 0001	1	=	0b 0000 1111	15	F
0b 1111	15	X	0b 0010	2	=	0b 0001 1110	30	1E
0b 1111	15	X	0b 0100	4	=	0b 0011 1100	60	3C
0b 1111	15	X	0b 1000	8	=	0b 0111 1000	120	78
0b 1111	15	X	0b 1001	9	=	0b 1000 0111	135	87
0b 1111	15	X	0b 1010	10	=	0b 1001 0110	150	96
0b 1111	15	X	0b 1011	11	=	0b 1010 0101	165	A5
0b 1111	15	X	0b 1100	12	=	0b 1011 0100	180	B4
0b 1111	15	X	0b 1101	13	=	0b 1100 0011	195	C3
0b 1111	15	X	0b 1110	14	=	0b 1101 0010	210	D2
0b 1111	15	X	0b 1111	15	=	0b 1110 0001	225	E1

Table 0.1: Partial 4-bit Multiplier Truth Table

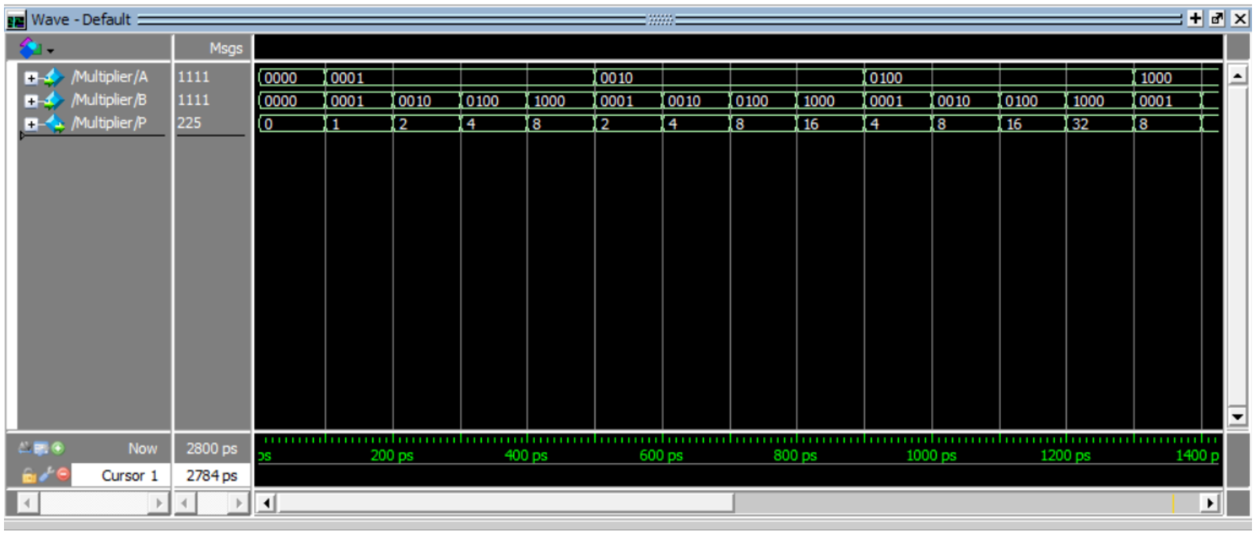


Figure 0.5: ModelSim simulation showing the results for the first 14 values of the truth table.

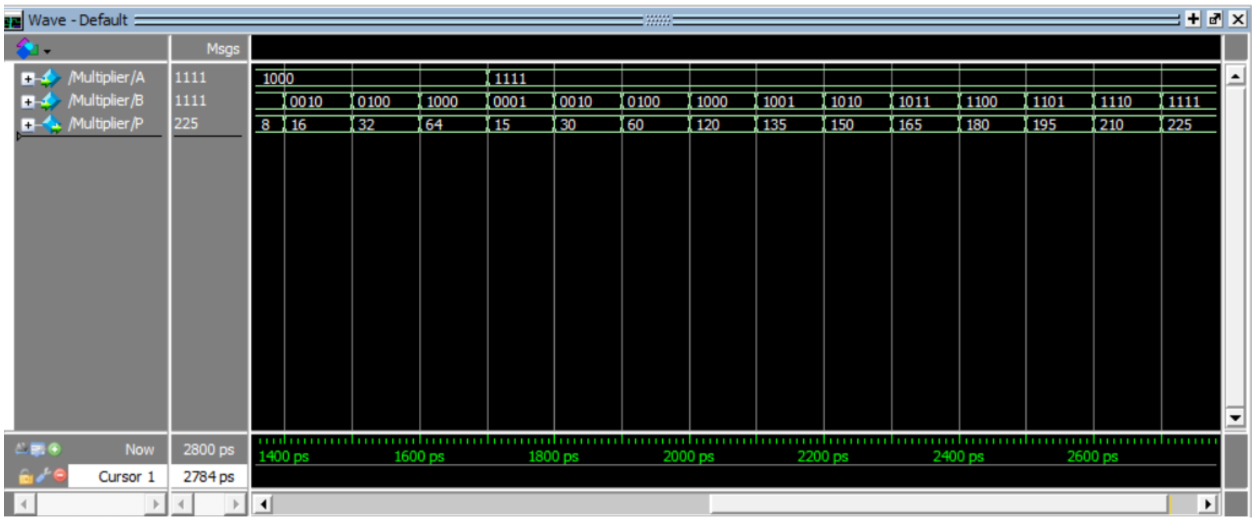


Figure 0.6: ModelSim simulation showing the results for the last 14 values of the truth table.

0.3.4 Program and Test Hardware

After simulating the design and verifying that the design worked as expected the program was uploaded to the FPGA. Figure 0.7 shows a picture of the programmed board with the binary value of 0b 1110 0001 (decimal 225).

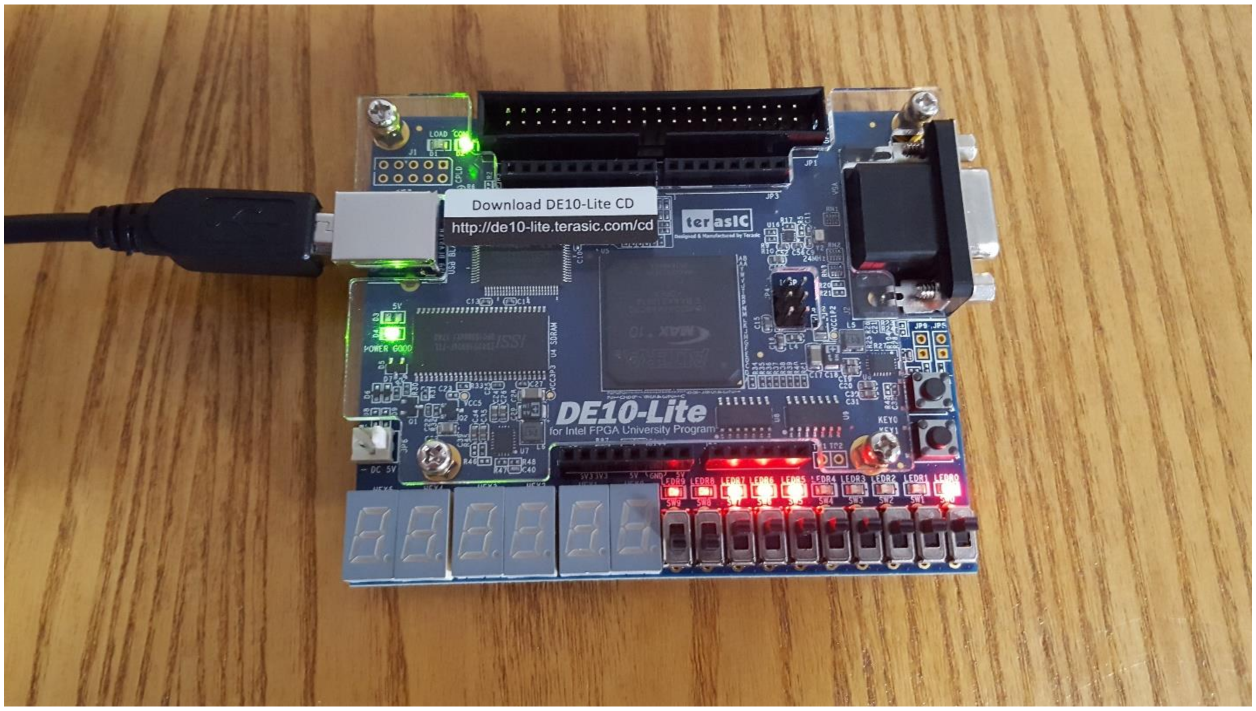

Download DE10-Lite CD
http://de10-lite.terasic.com/cd
DE10-Lite
for Intel FPGA University Program

Figure 0.7: Picture of the programmed board with the binary value of 0b 1110 0001 (decimal 225).

0.4 Observations

For the most part, the process went well. The one hang-up was how Quartus names the wires on a BUS. It is a bit different than how you would name the wires in an HDL. One thing that may have improved this design would have been to include the unused LEDs and connect them to the ground rather than leaving them floating so that they didn't partially turn on as shown in Figure 0.7.

0.5 Conclusion

In this lab, a 4 x 4 Unsigned Multiplier was designed and implemented at a logic gate level using the Quartus Prime Lite software. The design that was used is from the textbook, Digital Design and Computer Architecture by Drs. David and Sarah Harris CH. 5 pg. 253. This design uses a method of ANDing the bits and adding the partial products. The layout of the Multiplier is shown in Figure 0.2. The design was simulated in ModelSim and the simulation matched the expected results of the partial 4-bit Multiplier Truth Table shown in Table 0.1. The design was loaded onto a DE10-Lite MAX10 10M50DAF484C7G FPGA and the working multiplier was demonstrated.

0.6 Study Questions

1. How many gates were used for this design?

- For the design I used there were 6 gates for the Full_Adder Circuits with 12 Full_Adders in the circuit plus 16 AND Gates. This brings the total used to 88 gates.

2. Estimate the number of transistors that would be needed for this design. What scale of Integration would this design be?
   - Based on the different logic gates discussed in class there are 6 transistors used for the AND2 Gates, 6 for the OR2 gates, and 8 for the XOR gates. This design used a total of 736 transistors, so this design is considered a Large-Scale Integration (LSI).

Logic Gate Type:	Number of Transistors:	Number Used:	Total:
AND2	6	64	384
OR2	6	16	96
XOR	8	32	256
Total:			736

Table 0.2: Table used to estimate the number of transistors used for the design of the Multiplier.

0.7 References

1. Harris, David, and Sarah Harris. Digital Design and Computer Architecture: ARM Edition, Elsevier Science & Technology, 2015. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/linnbenton-ebooks/detail.action?docID=5754460.
2. Terasic Inc. (2017) DE10-Lite User Manual https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=1021&FID=a13a2782811152b477e60203d34b1baa
3. Introduction to ModelSim: Installation and Simulating a Decoder https://eecs.oregonstate.edu/sites/eecs.oregonstate.edu/files/tekbots/docs/ece272/introduction-to-modelsim.pdf

0.8 Appendix

Full_Adder.v

```
// Copyright (C) 2018  Intel Corporation. All rights reserved.
// Your use of Intel Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Intel Program License
// Subscription Agreement, the Intel Quartus Prime License Agreement,
// the Intel FPGA IP License Agreement, or other applicable license
// agreement, including, without limitation, that your use is for
// the sole purpose of programming logic devices manufactured by
// Intel and sold by Intel or its authorized distributors.  Please
// refer to the applicable agreement for further details.

// PROGRAM		"Quartus Prime"
// VERSION		"Version 18.1.0 Build 625 09/12/2018 SJ Lite Edition"
// CREATED		"Wed Sep 11 10:07:27 2019"

module Full_Adder(
	A,
	B,
	Cin,
	Cout,
	S
);

input wire	A;
input wire	B;
input wire	Cin;
output wire	Cout;
output wire	S;

wire	SYNTHESIZED_WIRE_0;
wire	SYNTHESIZED_WIRE_1;
wire	SYNTHESIZED_WIRE_2;
wire	SYNTHESIZED_WIRE_3;


assign	SYNTHESIZED_WIRE_0 = A ^ B;
assign	S = SYNTHESIZED_WIRE_0 ^ Cin;
assign	SYNTHESIZED_WIRE_3 = A & B;
assign	SYNTHESIZED_WIRE_1 = A & Cin;
assign	SYNTHESIZED_WIRE_2 = B & Cin;
assign	Cout = SYNTHESIZED_WIRE_1 | SYNTHESIZED_WIRE_2 | SYNTHESIZED_WIRE_3;

endmodule
```

Multiplier.v

```
// Copyright (C) 2018  Intel Corporation. All rights reserved.
// Your use of Intel Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Intel Program License
// Subscription Agreement, the Intel Quartus Prime License Agreement,
// the Intel FPGA IP License Agreement, or other applicable license
// agreement, including, without limitation, that your use is for
// the sole purpose of programming logic devices manufactured by
// Intel and sold by Intel or its authorized distributors.  Please
// refer to the applicable agreement for further details.

// PROGRAM		"Quartus Prime"
// VERSION		"Version 18.1.0 Build 625 09/12/2018 SJ Lite Edition"
// CREATED		"Wed Sep 11 10:07:43 2019"

module Multiplier(
	A,
	B,
	P
);

input wire	[3:0] A;
input wire	[3:0] B;
output wire	[7:0] P;

wire	[7:0] P_ALTERA_SYNTHESIZED;
wire	SYNTHESIZED_WIRE_0;
wire	SYNTHESIZED_WIRE_1;
wire	SYNTHESIZED_WIRE_2;
wire	SYNTHESIZED_WIRE_3;
wire	SYNTHESIZED_WIRE_4;
wire	SYNTHESIZED_WIRE_5;
wire	SYNTHESIZED_WIRE_6;
wire	SYNTHESIZED_WIRE_7;
wire	SYNTHESIZED_WIRE_8;
wire	SYNTHESIZED_WIRE_9;
wire	SYNTHESIZED_WIRE_10;
wire	SYNTHESIZED_WIRE_11;
wire	SYNTHESIZED_WIRE_12;
wire	SYNTHESIZED_WIRE_13;
wire	SYNTHESIZED_WIRE_14;
wire	SYNTHESIZED_WIRE_15;
wire	SYNTHESIZED_WIRE_16;
wire	SYNTHESIZED_WIRE_17;
wire	SYNTHESIZED_WIRE_18;
wire	SYNTHESIZED_WIRE_19;
wire	SYNTHESIZED_WIRE_20;
wire	SYNTHESIZED_WIRE_21;
wire	SYNTHESIZED_WIRE_22;
wire	SYNTHESIZED_WIRE_23;
wire	SYNTHESIZED_WIRE_24;
wire	SYNTHESIZED_WIRE_25;
```

```
wire      SYNTHESIZED_WIRE_26;
wire      SYNTHESIZED_WIRE_27;
wire      SYNTHESIZED_WIRE_28;
wire      SYNTHESIZED_WIRE_29;
wire      SYNTHESIZED_WIRE_30;
wire      SYNTHESIZED_WIRE_31;
wire      SYNTHESIZED_WIRE_32;
wire      SYNTHESIZED_WIRE_33;
wire      SYNTHESIZED_WIRE_34;
wire      SYNTHESIZED_WIRE_35;

assign    SYNTHESIZED_WIRE_2 = 0;
assign    SYNTHESIZED_WIRE_9 = 0;
assign    SYNTHESIZED_WIRE_14 = 0;
assign    SYNTHESIZED_WIRE_26 = 0;

assign    P_ALTERA_SYNTHESIZED[0] = A[0] & B[0];
assign    SYNTHESIZED_WIRE_0 = A[1] & B[0];
assign    SYNTHESIZED_WIRE_1 = A[0] & B[1];
assign    SYNTHESIZED_WIRE_3 = A[2] & B[0];
assign    SYNTHESIZED_WIRE_4 = A[1] & B[1];
assign    SYNTHESIZED_WIRE_6 = A[3] & B[0];
assign    SYNTHESIZED_WIRE_7 = A[2] & B[1];
assign    SYNTHESIZED_WIRE_10 = A[3] & B[1];
assign    SYNTHESIZED_WIRE_13 = A[0] & B[2];
assign    SYNTHESIZED_WIRE_16 = A[1] & B[2];
assign    SYNTHESIZED_WIRE_19 = A[2] & B[2];
assign    SYNTHESIZED_WIRE_22 = A[3] & B[2];
assign    SYNTHESIZED_WIRE_25 = A[0] & B[3];
assign    SYNTHESIZED_WIRE_28 = A[1] & B[3];
assign    SYNTHESIZED_WIRE_31 = A[2] & B[3];
assign    SYNTHESIZED_WIRE_34 = A[3] & B[3];

Full_Adder        b2v_inst32(
          .A(SYNTHESIZED_WIRE_0),
          .B(SYNTHESIZED_WIRE_1),
          .Cin(SYNTHESIZED_WIRE_2),
          .S(P_ALTERA_SYNTHESIZED[1]),
          .Cout(SYNTHESIZED_WIRE_5));


Full_Adder        b2v_inst33(
          .A(SYNTHESIZED_WIRE_3),
          .B(SYNTHESIZED_WIRE_4),
          .Cin(SYNTHESIZED_WIRE_5),
          .S(SYNTHESIZED_WIRE_12),
          .Cout(SYNTHESIZED_WIRE_8));
```

```
Full_Adder	b2v_inst34(
		.A(SYNTHESIZED_WIRE_6),
		.B(SYNTHESIZED_WIRE_7),
		.Cin(SYNTHESIZED_WIRE_8),
		.S(SYNTHESIZED_WIRE_15),
		.Cout(SYNTHESIZED_WIRE_11));


Full_Adder	b2v_inst35(
		.A(SYNTHESIZED_WIRE_9),
		.B(SYNTHESIZED_WIRE_10),
		.Cin(SYNTHESIZED_WIRE_11),
		.S(SYNTHESIZED_WIRE_18),
		.Cout(SYNTHESIZED_WIRE_21));


Full_Adder	b2v_inst36(
		.A(SYNTHESIZED_WIRE_12),
		.B(SYNTHESIZED_WIRE_13),
		.Cin(SYNTHESIZED_WIRE_14),
		.S(P_ALTERA_SYNTHESIZED[2]),
		.Cout(SYNTHESIZED_WIRE_17));


Full_Adder	b2v_inst37(
		.A(SYNTHESIZED_WIRE_15),
		.B(SYNTHESIZED_WIRE_16),
		.Cin(SYNTHESIZED_WIRE_17),
		.S(SYNTHESIZED_WIRE_24),
		.Cout(SYNTHESIZED_WIRE_20));


Full_Adder	b2v_inst38(
		.A(SYNTHESIZED_WIRE_18),
		.B(SYNTHESIZED_WIRE_19),
		.Cin(SYNTHESIZED_WIRE_20),
		.S(SYNTHESIZED_WIRE_27),
		.Cout(SYNTHESIZED_WIRE_23));


Full_Adder	b2v_inst39(
		.A(SYNTHESIZED_WIRE_21),
		.B(SYNTHESIZED_WIRE_22),
		.Cin(SYNTHESIZED_WIRE_23),
		.S(SYNTHESIZED_WIRE_30),
		.Cout(SYNTHESIZED_WIRE_33));
```

```
Full_Adder        b2v_inst40(
            .A(SYNTHESIZED_WIRE_24),
            .B(SYNTHESIZED_WIRE_25),
            .Cin(SYNTHESIZED_WIRE_26),
            .S(P_ALTERA_SYNTHESIZED[3]),
            .Cout(SYNTHESIZED_WIRE_29));


Full_Adder        b2v_inst41(
            .A(SYNTHESIZED_WIRE_27),
            .B(SYNTHESIZED_WIRE_28),
            .Cin(SYNTHESIZED_WIRE_29),
            .S(P_ALTERA_SYNTHESIZED[4]),
            .Cout(SYNTHESIZED_WIRE_32));


Full_Adder        b2v_inst42(
            .A(SYNTHESIZED_WIRE_30),
            .B(SYNTHESIZED_WIRE_31),
            .Cin(SYNTHESIZED_WIRE_32),
            .S(P_ALTERA_SYNTHESIZED[5]),
            .Cout(SYNTHESIZED_WIRE_35));


Full_Adder        b2v_inst43(
            .A(SYNTHESIZED_WIRE_33),
            .B(SYNTHESIZED_WIRE_34),
            .Cin(SYNTHESIZED_WIRE_35),
            .S(P_ALTERA_SYNTHESIZED[6]),
            .Cout(P_ALTERA_SYNTHESIZED[7]));

assign   P = P_ALTERA_SYNTHESIZED;

endmodule
```