CMPE-260 Laboratory Exercise 3

ALU

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students; however, other than code provided by the instructor for this exercise, all code was developed by me.

_____

Erez Binyamin
Performed 10/23/17
Submitted 10/24/17
Lab Section 1
Instructor: Richard Cliver
TA: Mark the TA
TA: David Lin

Lecture Section 1
Professor: Richard Cliver

ABSTRACT:

The intent of this exercise was to design and thoroughly test a hardware design for an arithmetic logic unit (ALU). This was accomplished by using Xilinx ISE to develop hardware description code for a 1-bit full-adder, n-bit full-adder, n-bit multiplier, and an n-bit logic unit capable of executing AND, OR, XOR, and the NOT operation on the first input. These components functionalities were all written using behavioral hardware description, then a top-level wrapper was written to connect the components together. A testbench was written to exhaustively test the multiplier unit, using file/IO to do so. Finally, a testbench was written for the top-level design, behavioral and post route waveforms were simulated using Modelism, and the design was programmed onto a Nexys board. This report will detail the functionality of these components as well as the various tests that verify the projects functionality. Overall this exercise proved to be successful.

DESIGN METHODOLOGY:

A 1-bit full adder is a simple logical unit that only ever takes in 1-bit inputs. The inputs and outputs of a 1-bit full adder are described in table 1.

Table 1 1-bit Full Adder I/0

Name	In/Out	Num Bits
A	IN	1
B	IN	1
Cin	IN	1
Sum	OUT	1
Cout	OUT	1

The 1-bit full adder unit is the fundamental building block of an n-bit full adder. An n-bit full adder is constructed by generating ‘n’ full adders and connecting the ‘n-1’th Cout bit to the n’th adder’s Cin. This was achieved by creating a signal called “carry array”. The signal carry array contains n+1 bits where the ‘n+1’th bit is the final Cout of the n-bit full adder (The MSB of the sum). The inputs and outputs of an n-bit full adder are described in table 2.

Table 2 n-bit Full Adder I/0

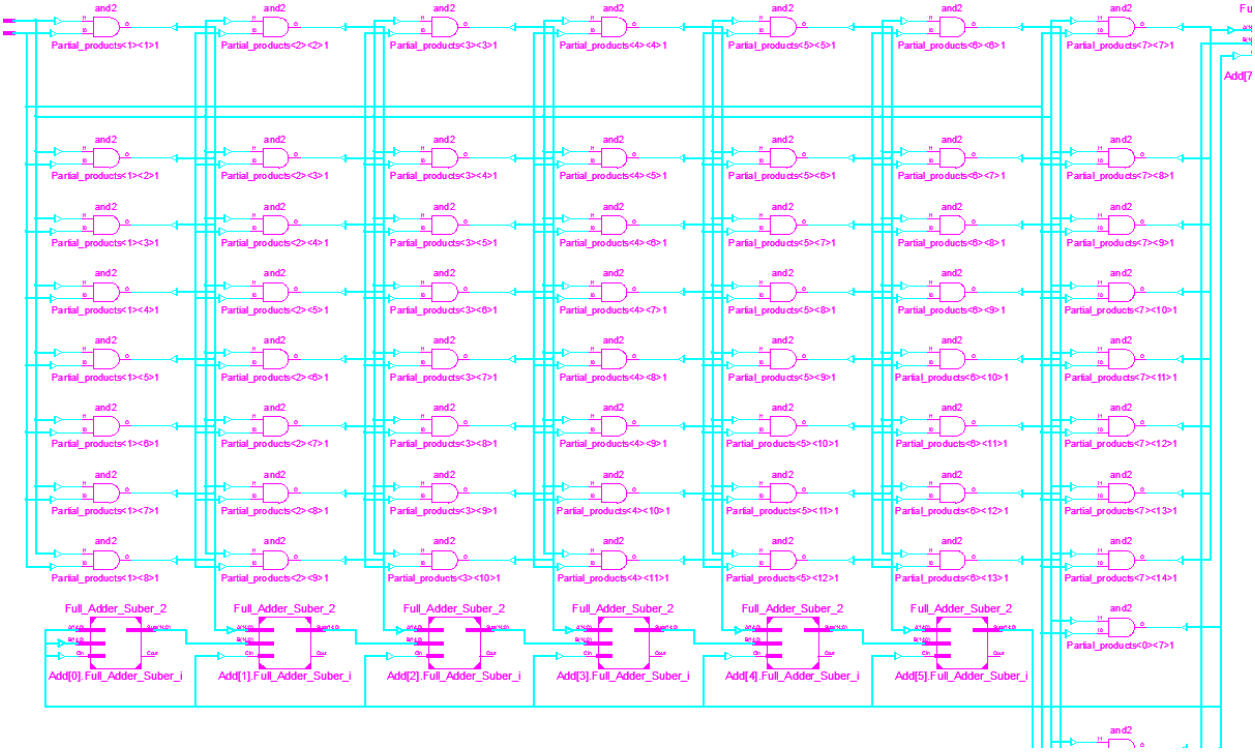
Name	In/Out	Num Bits
A	IN	N
B	IN	N
Cin	IN	1
Sum	OUT	N
Cout	OUT	1

The n-bit full adder also works as an “n-bit subtractor”. The Cin bit serves as a control for the adder/subtractor functionality, 0 and 1 being addition mode or subtraction mode respectively. The B input is always converted to an n-bit “New_B” signal before being passed to the generated

1-bit full adders. New_B is simply each bit of B XOR'ed with the Cin bit. This will effectively serve to invert the B signal if Cin is a 1.

Another functionality of the n-bit ALU is an n-bit multiplier unit. Multiplication involves a combination of "AND" logic and Addition logic. The balance of the two will effect the final hardware complexity and slice count. An ideal multiplication groups similar hardware together. A frequently used and very poor design would be the *carry save* multiplier. The carry save multiplier performs AND logic and Addition logic in an alternating pattern. First AND'ing corresponding A and B bits, and then calculating the sum of their product before cascading to the next row. This ends up using generating n-1 rows of n full adders in between AND logic. Another approach would be to first calculate all of the partial products and then add them together at the end. This approach serves to group similar hardware together, thereby allowing most FPGA design suites (i.e. ISE) to optimize the routing more effectively (decreasing propagation delay) and making a schematic much more readable. The grouped hardware approach consistently synthesized to produce a lower slice count than the traditional carry save design approach. A carry save design would look like the image in Fig. 1 as opposed to the grouped design seen in Fig. 2.

Fig 1 Carry Save RTL
Fig 2 Grouped Hardware RTL



Delving into the specifics of the grouped multiplier design, the inputs are simply n-bit A and B input vectors and the output is a 2n-bit vector. Each bit of A and B are AND'ed (1-bit multiplier) together in a 2d matrix like pattern to produce an array of n, 2n-1 bit vectors. These partial products are then added together using an accumulator array of 2n-1 bit vectors and an array of carry result bits. The final product's MSB is the last carry from the carry array and the rest of its bits given by the last value in the accumulator. This process is illustrated graphically in Fig. 3-4.

FIG 3. Generation of partial products (P_0 and P_1)

	x A_1	A_0	A_{input}
	B_1	B_0	B_{input}
0	A_1B_0	A_0B_0	P_0
A_1B_1	A_0B_1	0	P_1

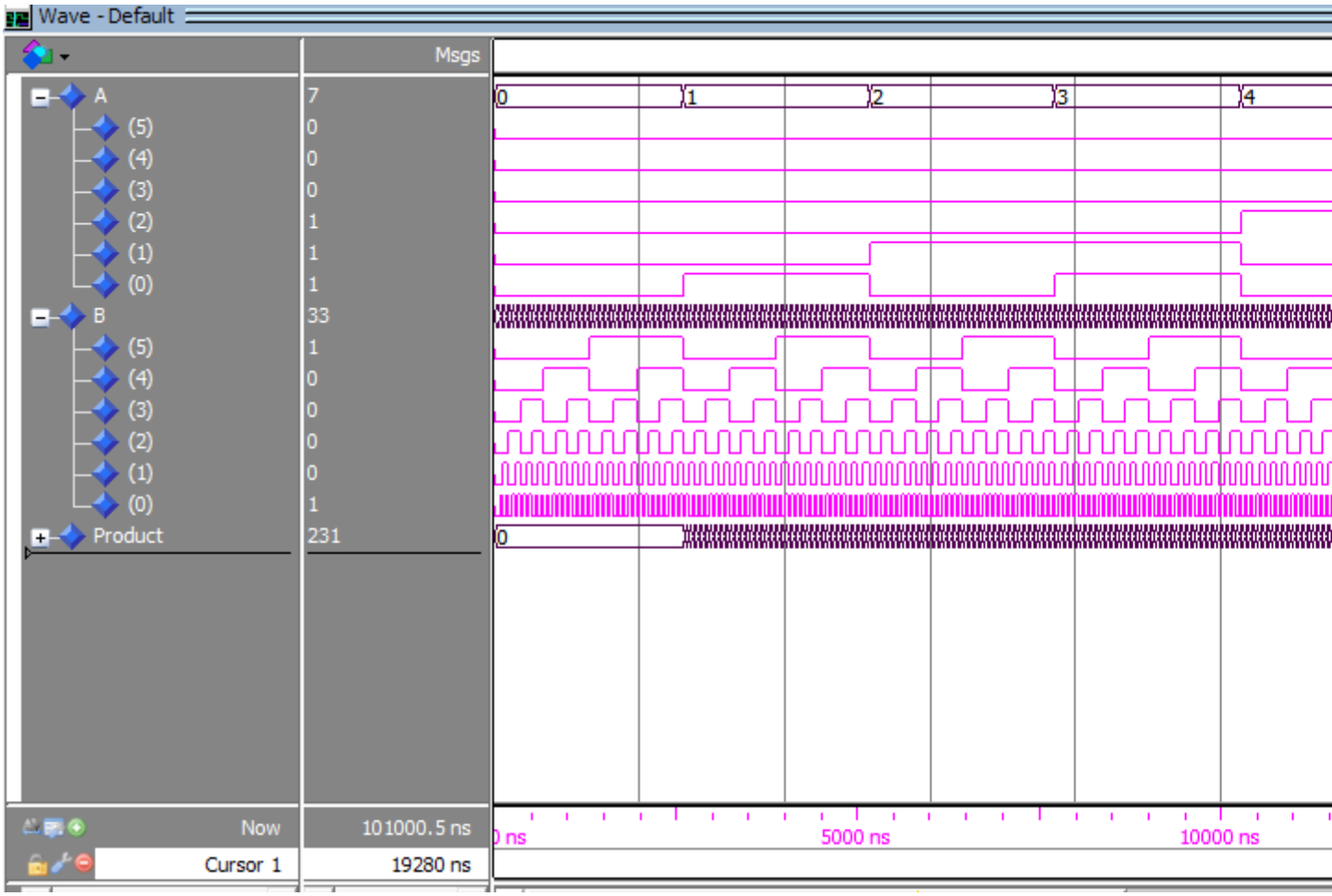
FIG 4. Addition of partial products using Accumulator ($A_{0\text{-}2}$)

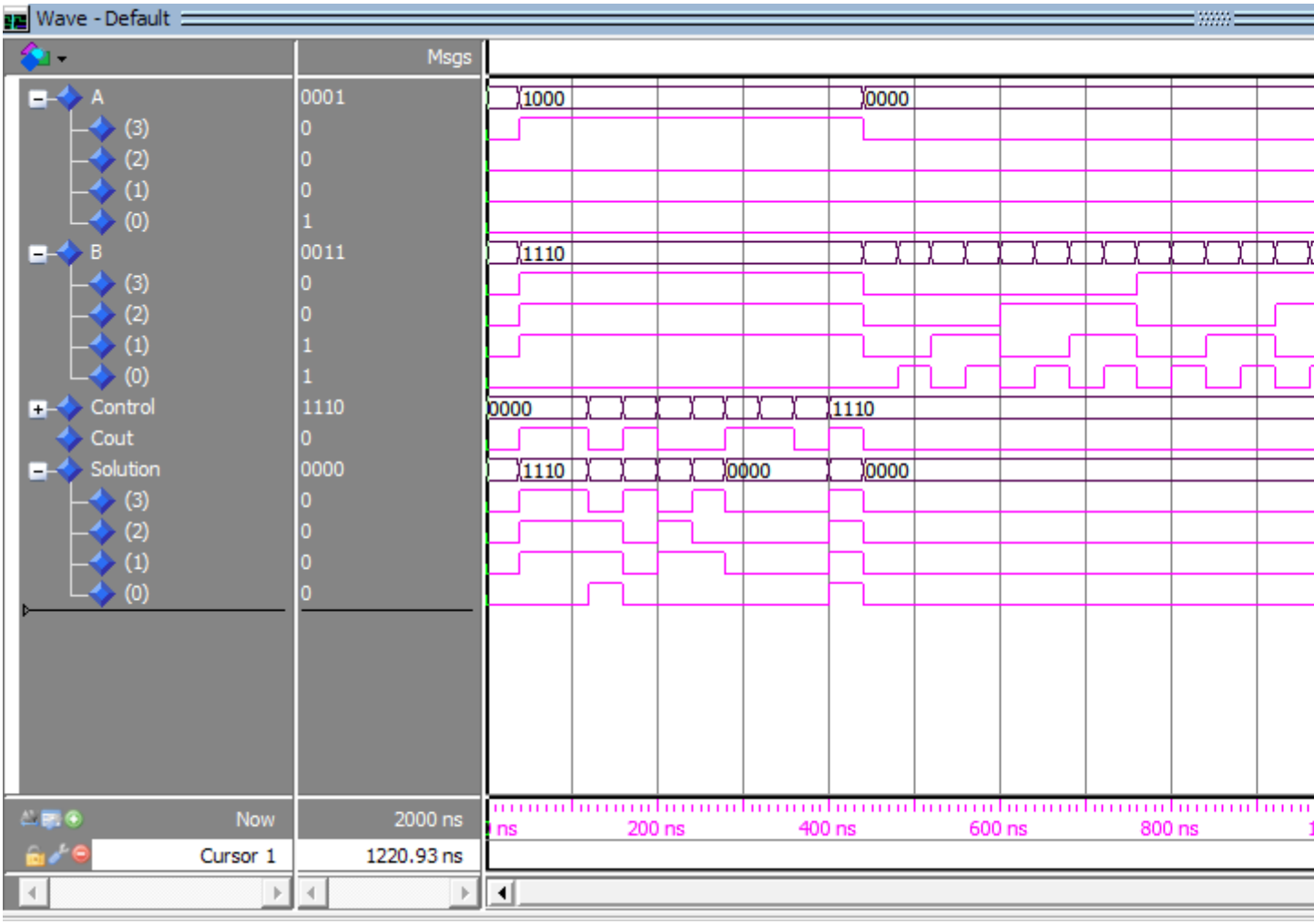
+ 0	0	0	A_0
0	A_1B_0	A_0B_0	P_0
0	A_1B_0	A_0B_0	A_1
+ A_1B_1	A_0B_1	0	P_1
A_1B_1	$A_0B_1+A_1B_0$	A_0B_0	A_2

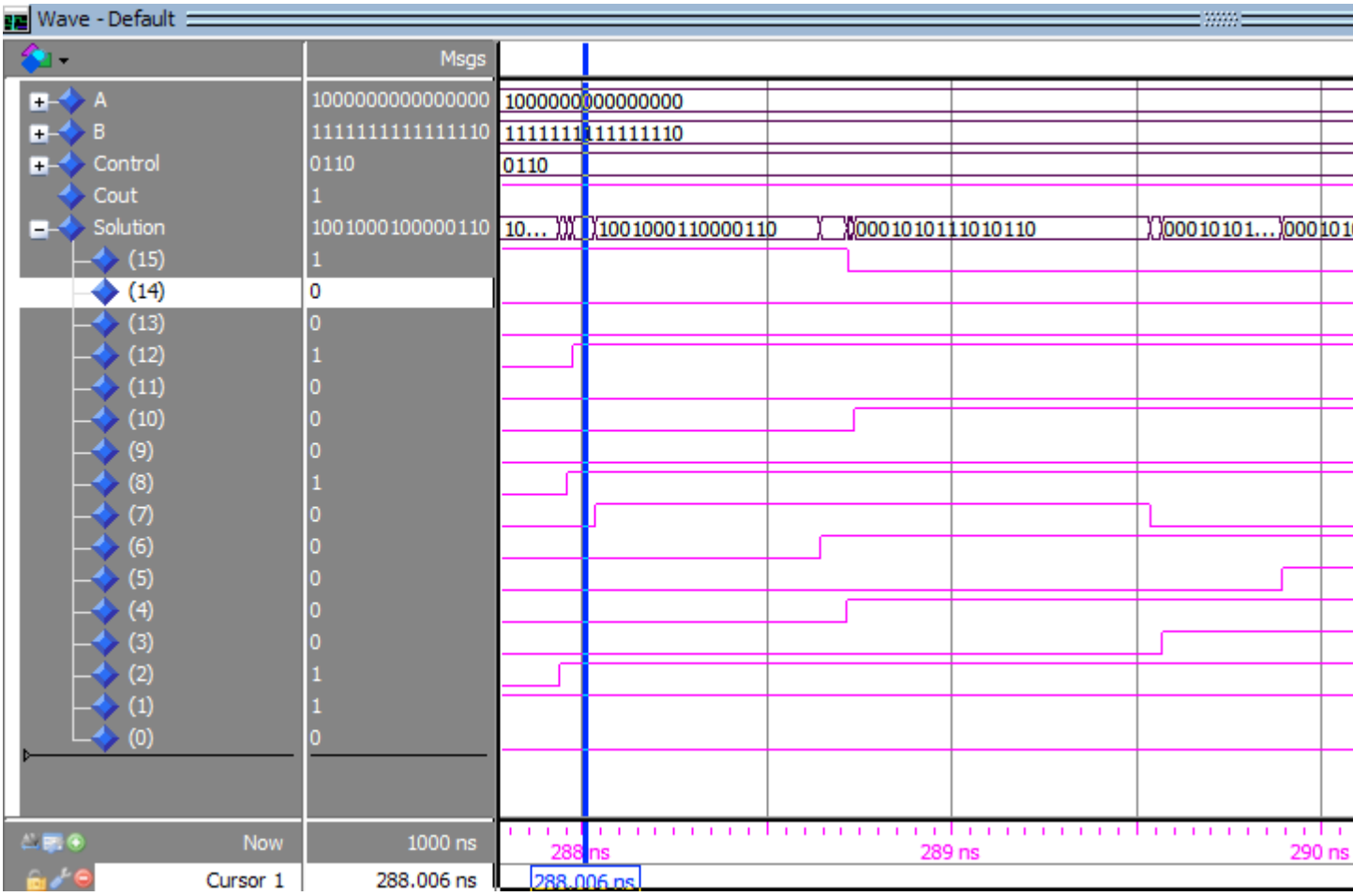
The output assignment was as simple as assigning the bottom 2n-2 bits to the last vector of the accumulator array and the MSB (2n-1'th bit) to the last carry from the carry array. The n-bit logic unit was a simple design using the built in VHDL logical operators and simply selecting which operation to perform based off a 2-bit control signal.

The full n-bit ALU takes in n-bit A and B inputs, a 4-bit control signal input, and produces an n-bit output signal. Fort the multiplication operation the bottom half of bits A and B are used as inputs to the multiplier. This means that in a 4-bit build of the ALU the operation 15 * 2 would actually calculate 3*2.

RESULTS AND ANALYSIS:


Wave - Default
Msgs
A
7
(5)
0
(4)
0
(3)
0
(2)
1
(1)
1
(0)
1
B
33
(5)
1
(4)
0
(3)
0
(2)
0
(1)
0
(0)
1
Product
231
0
1
2
3
4
Now
101000.5 ns
Cursor 1
19280 ns
0 ns
5000 ns
10000 ns


Wave - Default
Msgs
A
0001
(3)
0
(2)
0
(1)
0
(0)
1
B
0011
(3)
0
(2)
0
(1)
1
(0)
1
Control
1110
Cout
0
Solution
0000
(3)
0
(2)
0
(1)
0
(0)
0
1000
0000
1110
0000
1110
1110
0000
0000
Now
2000 ns
Cursor 1
1220.93 ns
ns
200 ns
400 ns
600 ns
800 ns



CONCLUSIONS:

This successful exercise demonstrates the underlying hardware that governs basic memory read/write operations. A major hurdle faced in the development of this hardware was the concept of changing signals on the rising edge and attempting to accept them at the same time. It was discovered that when writing testbenches it is crucial that inputs be toggled on the falling edge of a clock to ensure that the “correct” inputs be assigned at the “correct” times. However, despite the many hurdles, this exercise proved to be successful.

QUESTIONS:

Area used:

- Number of occupied slices: 66
- FF’s used: 128
- LUT’s used: 68

Timing Results:

- Best Case Achievable: 1.730 ns