Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

Objective:

The purpose of this lab is to examine, design, test and implement arithmetic circuits that add, subtract, multiply, and divide 32 bit integers. Use 32 bit registers to store operands you input with switches on the board. All blocks you design in this lab have to store 32 bit operands in registers, as we explained in the class. The result of each operation has to be displayed using 7- segment display and LEds. Use unsigned arithmetic only.

For the sake of simplicity of explanation, we use all operands less than 32 bits. You have to design 32 bit digital ciurcuits.

1. Design and test (write a test-bench in vhdl and test in ModelSIM) 8 bit accumulator, as shown in Fig. 2. Extend the circuit to perform addition or subtraction by adding 1 bit opcode and additional required circuitry.
2. Design and test(write a test-bench in vhdl and test in ModelSIM) an array multiplier circuit, as shown in Fig. 4
3. Design and test (write a test-bench in vhdl and test in ModelSIM) an array multiplier circuit, implemented using *n*-bit adders, as shown in Fig. 5.
4. Design and test (write a test-bench in vhdl and test in ModelSIM) a registered multiplier circuit, integrating two registers to store the operands and another register to store the result, as shown in Fig. 6. Notice the difference in register size.
5. Design and test (write a test-bench in vhdl and test in ModelSIM) a parallel adder tree circuit, as shown in Fig. 7.
6. Redesign and test (write a test-bench in vhdl and test in ModelSIM) an array multiplier circuit, as shown in Fig. 5, using a parallel tree adder instead of *n*-bit adder,. as shown in Fig. 5.
7. Design and test (write a test-bench in vhdl and test in ModelSIM) a circuit to compute quotient and remainder when dividing two integers. To design the divider, follow detailed description in PART . VI, and as shown in Figures 8, 9, 10, 11.
8. Prepare a detailed report on all 7 designs following the required format! Prepare a demo video. Please be ready to show a working project if asked.

Each circuit will be described in VHDL and implemented on an Altera DE2-series board.

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

Part I

Consider again the four-bit ripple-carry adder circuit used in previous lab; its diagram is reproduced in Figure 1.

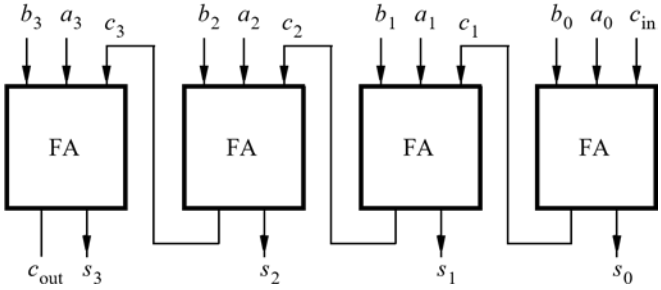

Figure 1: A four-bit ripple carry adder.

This circuit can be implemented using a '+' sign in VHDL. For example, the following code fragment adds *n*-bit numbers A and B to produce output *sum*:

```
Library ieee; use ieee.std
logic 1164.all; use ieee.std
logic arith.all; use ieee.std
logic signed.all; ...
 signal sum : std logic vector(n-1 downto 0);
...
sum <= A + B;
```

Use this construct to implement a circuit shown in Figure 2.

Design and compile your circuit with Quartus II software, download it onto a DE2-series board, and test its operation as follows:

1. Create a new Quartus II project. Select the appropriate target chip that matches the FPGA chip on the AlteraDE2-series board. Implement the designed circuit on the DE2-series board.

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

2. Write VHDL code that describes the circuit in Figure 2.

3. Connect input A to switches SW_{7-0}, and use KEY_0 as an active-low asynchronous reset and KEY_1 as a manual clock input. The sum output should be displayed on red $LEDR_{7-0}$ lights and the carry-out should be displayed on the red $LEDR_8$ light.

4. Assign the pins on the FPGA to connect to the switches and 7-segment displays by importing the appropriate pin assignment file.

5. Compile your design and use timing simulation to verify the correct operation of the circuit. Once the simulation works properly, download the circuit onto the DE2-series board and test it by using different values of A. Be sure to check that the *Overflow* output works correctly.

6. Open the Quartus II Compilation Report and examine the result**s reported by the Timing Analyzer.** What is the **maximum operation frequency, f_{max},** of your circuit? What is the **longest path in the circuit** in terms of delay?

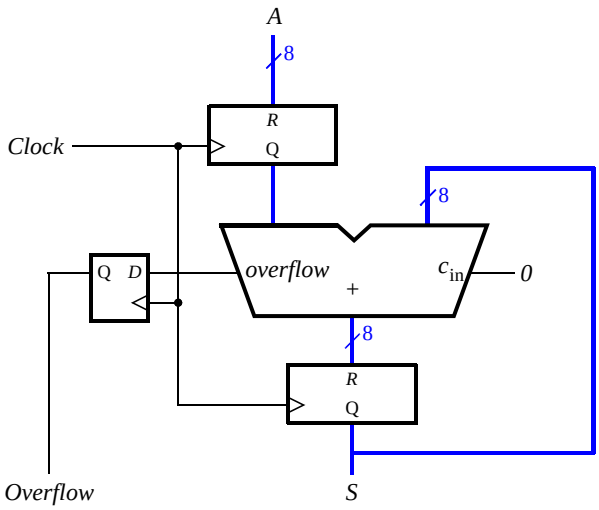

Figure 2: An eight-bit accumulator circuit.

Part II

Extend the circuit from Part I to be able to both add or subtract numbers. **To do so, add an *add sub* input to your circuit. When *add _sub* is 1, your circuit should subtract A from S, and add A and S** as in Part I otherwise.

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

PART III

MULTIPLICATION

Figure 3 gives an example of paper-and-pencil multiplication $P = A \times B$, where $A = 11$ and $B = 12$.

				1	0	1	1
				1	1	0	0
				0	0	0	0
			0	0	0	0	
		1	0	1	1		
	1	0	1	1			
1	0	0	0	0	1	0	0

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

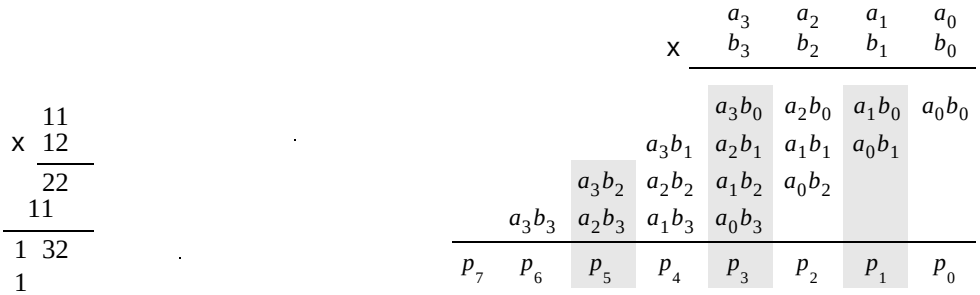

Figure 3.Multiplication of binary numbers.

We compute $P = A \times B$ as an addition of summands. The first summand is equal to A times the ones digit of B. The second summand is A times the tens digit of B, shifted one position to the left. We add the two summands to form the product $P = 132$.

Part b of the figure shows the same example using four-bit binary numbers. To compute $P = A \times B$, we first form summands by multiplying A by each digit of B. Since each digit of B is either 1 or 0, the summands are either shifted versions of A or 0000. Figure 3c shows how each summand can be formed by using the Boolean AND operation of A with the appropriate bit of B.

A four-bit circuit that implements $P = A \times B$ is illustrated in Figure 4. Because of its regular structure, this type of multiplier circuit is called an ***array multiplier (systolic array).*** The shaded areas correspond to the shaded columns in Figure 3c. In each row of the multiplier AND gates are used to produce the summands, and full adder modules are used to generate the required sums.

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

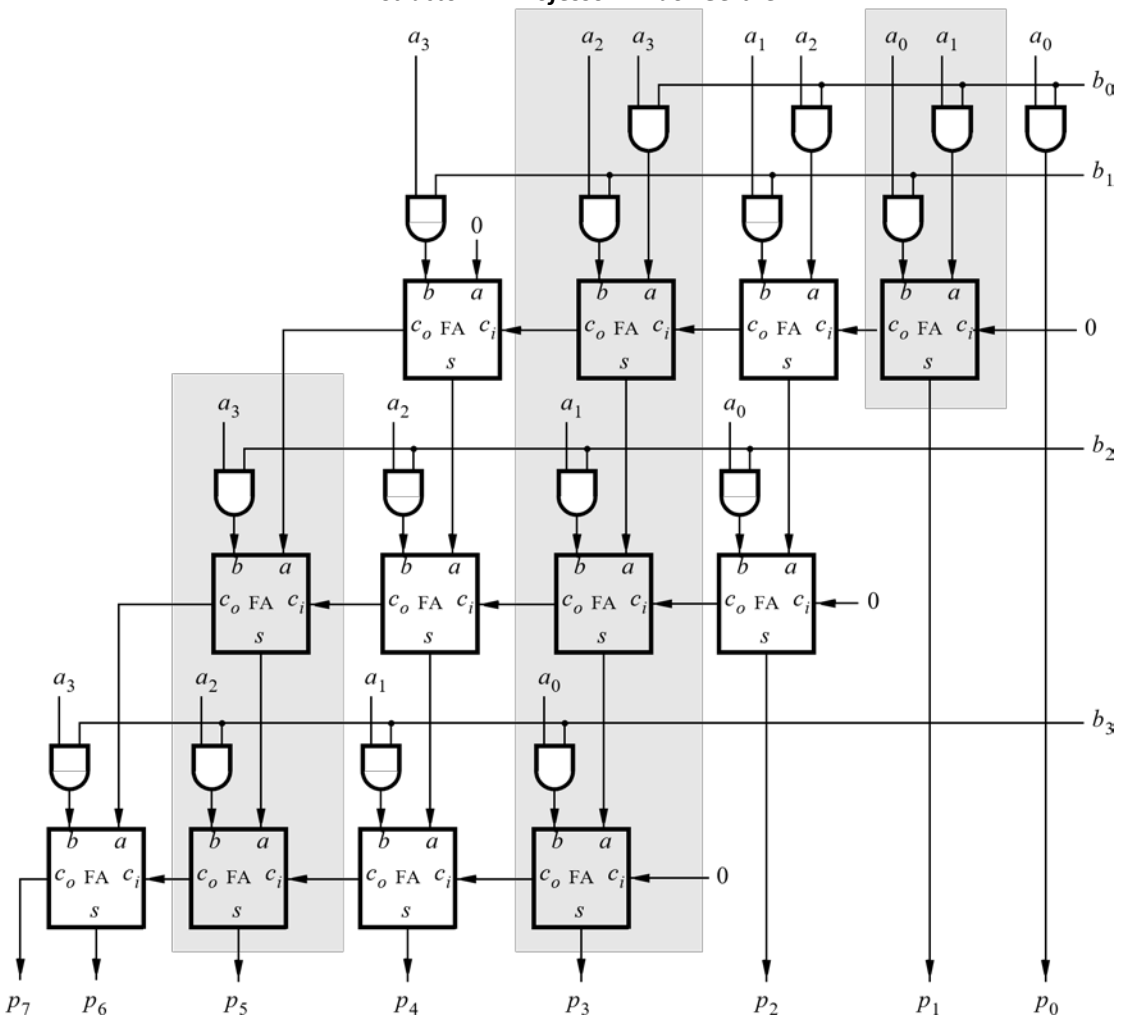

Figure 4: An array multiplier circuit.

Perform the following steps to implement the array multiplier circuit:

1. Create a new Quartus II project to implement the desired circuit on the Altera DE2-series board.
2. Generate the required VHDL file, include it in your project, and compile the circuit.
3. Use functional simulation to verify your design.
4. Augment your design to use switches SW_{11-8} to represent the number A and switches SW_{3-0} to represent

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

B. The hexadecimal values of A and B are to be displayed on the 7-segment displays *HEX6* and *HEX4*, respectively. The result $P = A \times B$ is to be displayed on *HEX1* and *HEX0*.

5. Assign the pins on the FPGA to connect to the switches and 7-segment displays by importing the appropriate pin assignment file.
6. Recompile the circuit and download it into the FPGA chip.
7. Test the functionality of your circuit by toggling the switches and observing the 7-segment displays.

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

Part IV

In Part III, an array multiplier was implemented using full adder modules. At a higher level, a row of FA functions as an n-bit adder and the array multiplier circuit can be represented as shown in Figure 5.

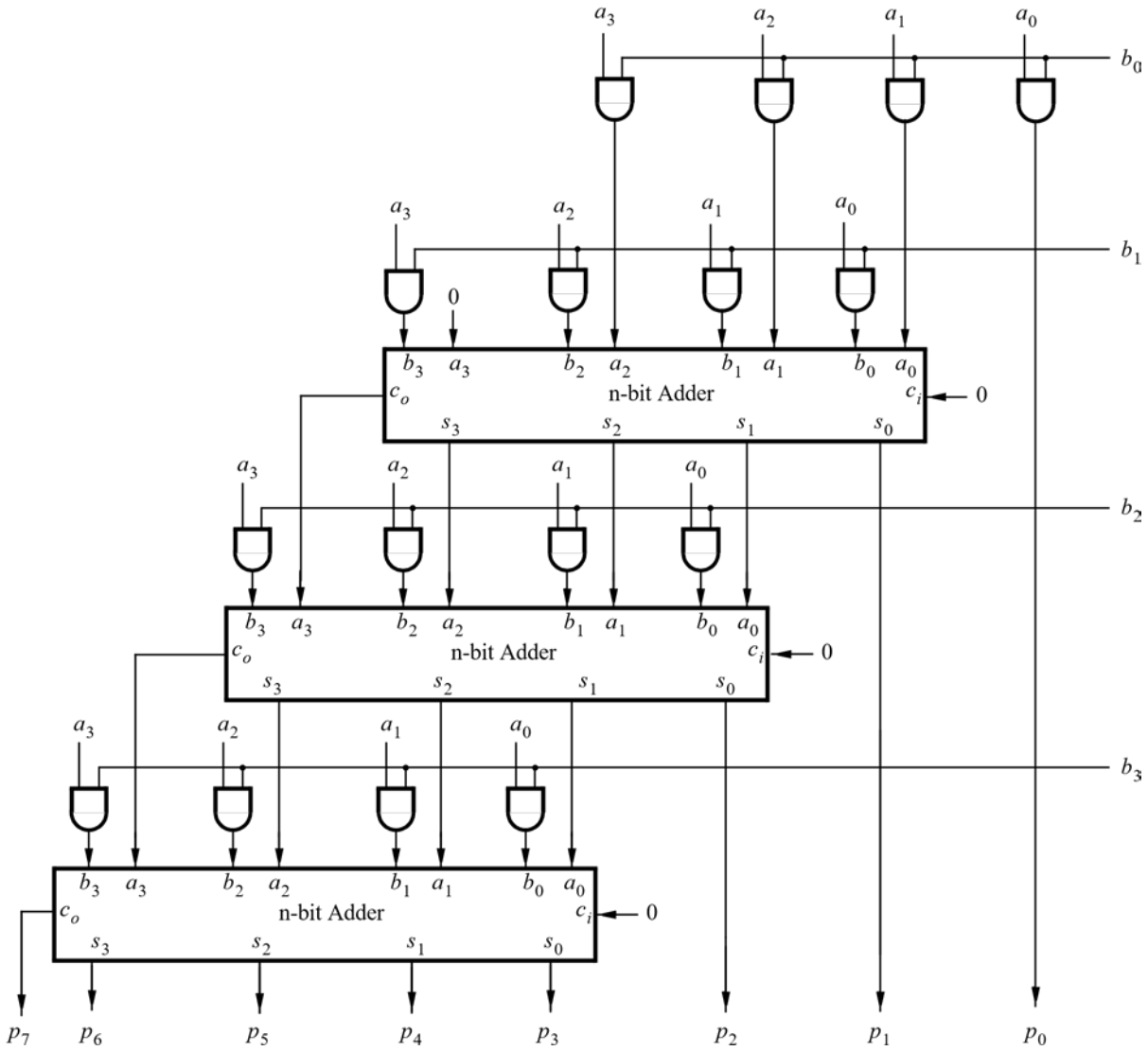

Figure 5: An array multiplier implemented using n-bit adders.

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

Each n-bit adder adds a shifted version of A for a given row and the partial sum of the row above. Abstracting the multiplier circuit as a sequence of additions allows us to build larger multipliers. The multiplier should consist of n-bit adders arranged in a structure shown in Figure 5. Use this approach to implement an 8x8 multiplier circuit with registered inputs and outputs, as shown in Figure 6.

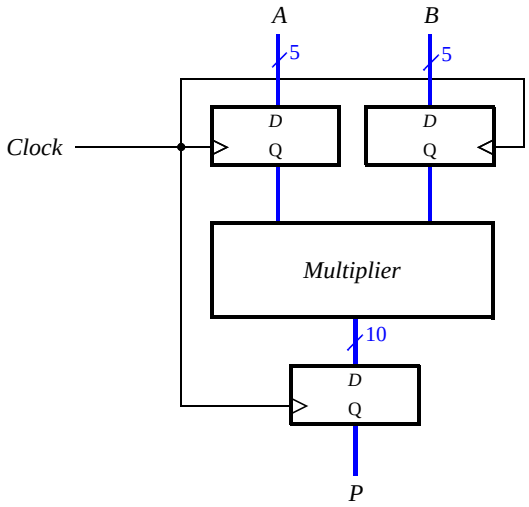

Figure 6: A registered multiplier circuit.

Perform the following steps:

1. Create a new Quartus II project.
2. Write the required VHDL file, include it in your project, and compile the circuit.
3. Use functional simulation to verify your design.
4. Augment your design to use switches SW_{15-8} to represent the number A and switches SW_{7-0} to represent B. The hexadecimal values of A and B are to be displayed on the 7-segment displays *HEX7-6* and *HEX5-4*, respectively. The result $P = A \times B$ is to be displayed on *HEX3-0*.
5. Assign the pins on the FPGA to connect to the switches and 7-segment displays.
6. Recompile the circuit and download it into the FPGA chip.
7. Test the functionality of your design by toggling the switches and observing the 7-segment displays.
8. How large is the circuit in terms of the number of logic elements?
9. What is the f_{max} for this circuit?

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

Part V

Part IV showed how to implement multiplication $A \times B$ as a sequence of additions, by accumulating the shifted versions of A one row at a time. Another way to implement this circuit is to perform addition using an adder tree.

An adder tree is a method of adding several numbers together in a parallel fashion. This idea is illustrated in Figure 7. In the figure, numbers A, B, C, D, E, F, G, and H are added together in parallel. The addition $A+B$ happens simultaneously with $C+D$, $E+F$ and $G+H$. The result of these operations are then added in parallel again, until the final sum P is computed.

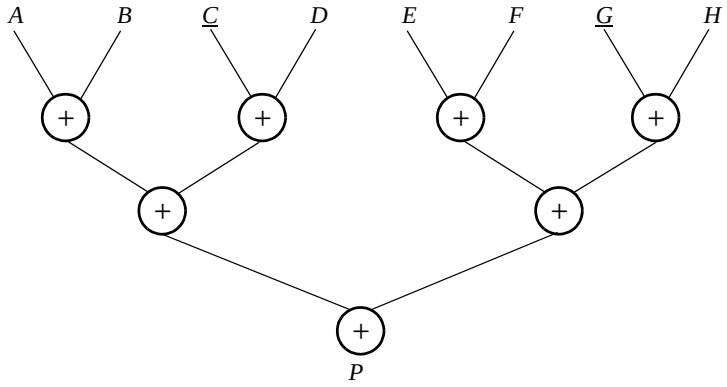

Figure 7: An example of adding 8 integers using an tree adder.

In this part you are to implement an 8x8 array multiplier that computes $P = A \times B$. Use an adder tree structure to implement operations shown in Figure 5. **Inputs A and B, as well as the output P should be registered as in Part IV. What is the f_{max} for this circuit?**

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

Part VI

The binary division requires to iterate the subtraction and the bit shifting, as the following example shows: the divisor is compared with the dividend, starting from the most significant digits, subtracting the divisor from the dividend only if it is possible. That way, the integer division result (quotient) is made of 1s when the subtraction is possible and 0s otherwise, whereas the remainder is what remains after all the subtractions.

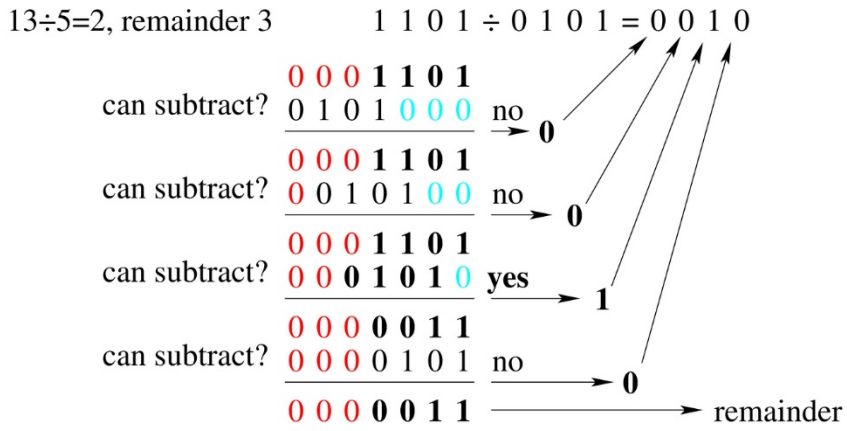

Figure 8: Division by subtraction and shifting methods for four bit unsigned numbers.

Try the above for different four bit strings on paper. To implement this circuit, we first need to create a full subtractor. Copy the following VHDL code, compile and create a block symbol for it. Call it **FS** (full subtractor).

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity FS is

port(

    S : in STD_LOGIC;   --Subtrahend

    M : in STD_LOGIC;   --Minuend

    Bi : in STD_LOGIC;  --Borrow In

    D : out STD_LOGIC;  --Difference
```

```
    Bo : out STD_LOGIC );  --Borrow Out

end FS;

architecture arch of FS is

begin

      D <= M xor S xor Bi;

       Bo <= ((not M) and Bi ) or ((not M) and S) or (S and Bi);

end arch;
```

Now create a block diagram file and import the **FS** symbol. You will have inputs: Minuend, Subrahend in, Borrow In and OK in. The outputs will be: Subrahend out, Borrow out, OK out and Difference. Make the following connections as shown in the next picture. Use a 2 to 1 Multiplexer (Go to Symbols and search "21mux", this will allow you to use a multiplexer from the library). The multiplexer has inputs A and B and a control input, make sure you do the right connections to A and B. The control input is OKi.

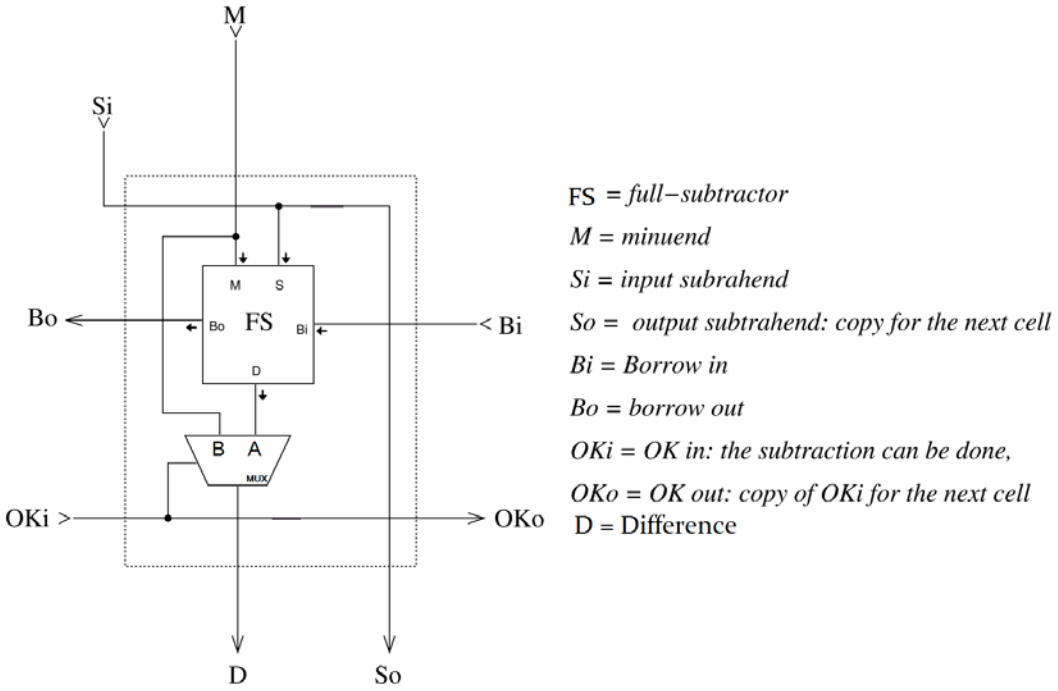

Figure 9: The DIV circuit that uses a full subtractor and a 2 to 1 multiplexer.

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

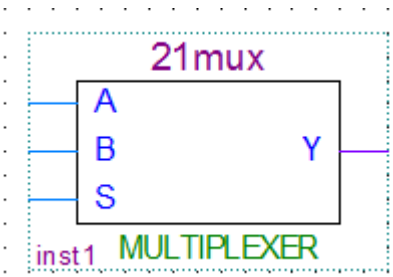

Figure 10: The 2 to 1 Multiplexer from Quartus. A and B are the inputs, S is the control input and Y is the output.

When you finish connecting the input pins and output pins correctly save your block diagram file and name it **DIV.** Compile and create a block symbol for it.

We are now ready to implement the final circuit. Use 16 instances of your **DIV** symbol and connect them as shown in the following picture. You also need 4 NOT gates**. You inputs are : Dividend[3..0] and Divisor[3..0]. Your outputs are Quotient[3..0] and Remainder[3..0]. They are all four bit strings.** Notice that you also need to Ground some inputs. Go to Symbols and look for "gnd" for the Ground symbol

When you finish, simulate your circuit and load it to the Altera Board.

Laboratory Project:

Add/Sub, Accumulator, and Multipliers, Integer Divider

Simulate, and Implement, Demo on DE 2 Board + video

Final Report, Video DEMO, *Due April 17, 2019 by 12:00 2:00 PM*

Instructor: Professor Izidor Gertner

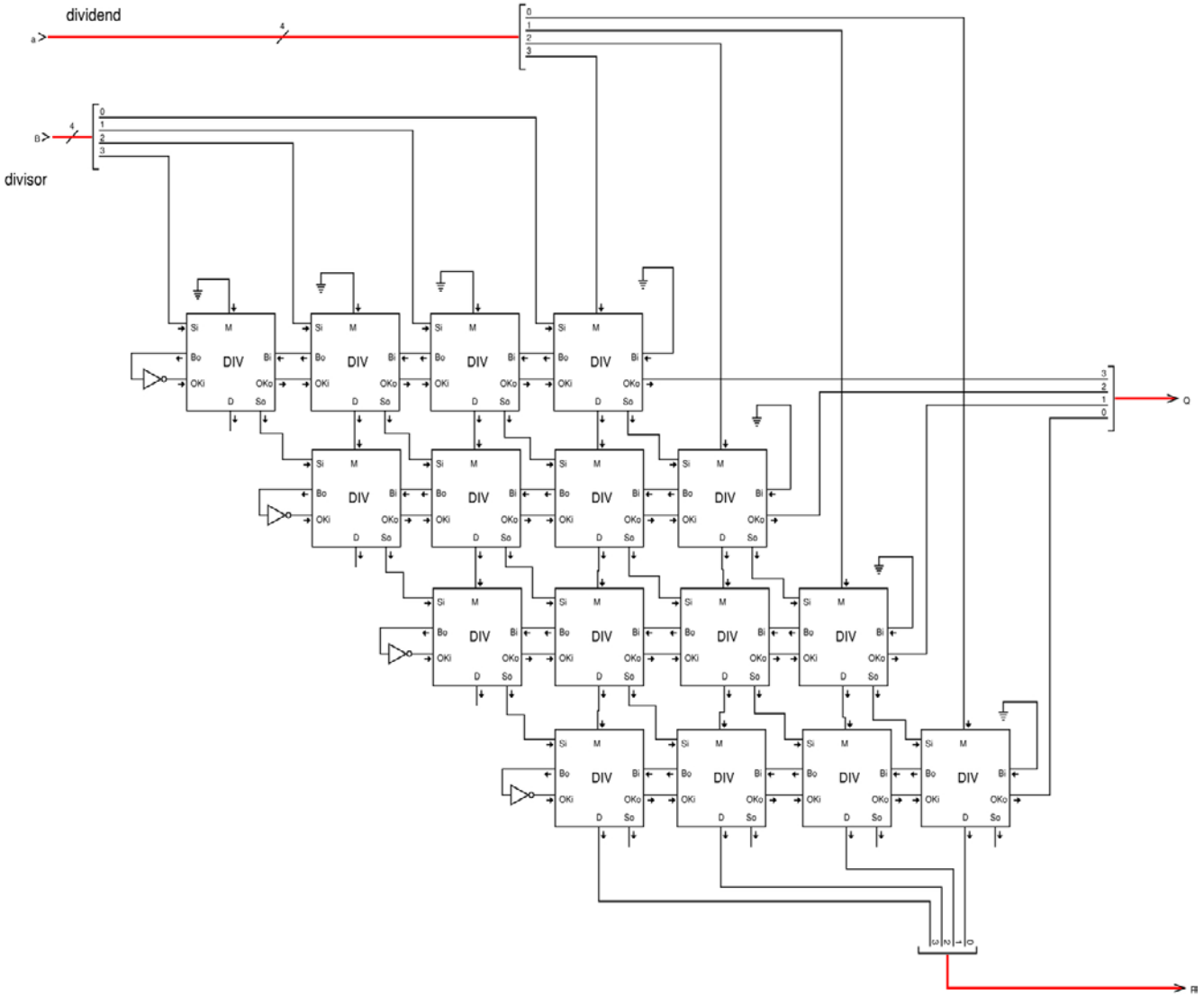

Figure 11: The Division Circuit built with 16 DIV symbols. The circuit takes as input a four bit Dividend and a four bit Divisor and outputs a four bit Quotient and Remainder. This is accomplished by bit shifting and subtraction when it is necessary.