Laboratory Exercise 6

Adders, Subtractors, and Multipliers

The purpose of this exercise is to examine arithmetic circuits that add, subtract, and multiply numbers. Each circuit will be described in VHDL and implemented on an Intel® FPGA DE10-Lite, DE0-CV, DE1-SoC, or DE2-115 board.

Part I

Consider again the four-bit ripple-carry adder circuit used in lab exercise 2; its diagram is reproduced in Figure 1.

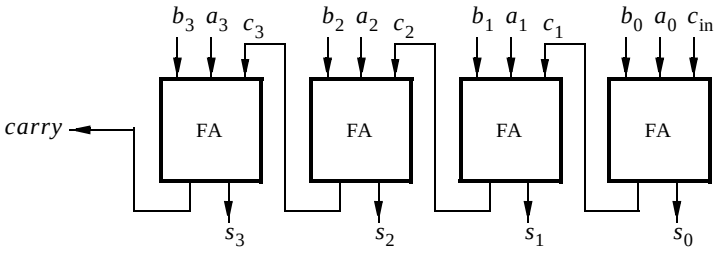

Figure 1: A four-bit ripple carry adder.

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 outputs sum and $carry$:

```
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. This circuit, which is often called an *accumulator*, is used to add the value of an input A to itself repeatedly. The circuit includes a carry out from the adder, as well as an *overflow* output signal. If the input A is considered as a 2's-complement number, then *overflow* should be set to 1 in the case where the output *sum* produced does not represent a correct 2's-complement result.

Perform the following steps:

1. Create a new Quartus® project. Write VHDL code that describes the circuit in Figure 2.
2. Connect input A to switches SW_{7-0}, use KEY_0 as an active-low asynchronous reset, and use KEY_1 as a manual clock input. The sum from the adder should be displayed on the red lights $LEDR_{7-0}$, the registered carry signal should be displayed on $LEDR_8$, and the registered *overflow* signal should be displayed on $LEDR_9$. Show the registered values of *A* and *S* as hexadecimal numbers on the 7-segment displays HEX3 − 2 and HEX1 − 0.

3. Make the necessary pin assignments needed to implement the circuit on your DE-series board, and compile the circuit.

4. Use timing simulation to verify the correct operation of the circuit. Once the simulation works properly, download the circuit onto your DE-series board and test it by using different values of A. Be sure to check that the *overflow* output works correctly.

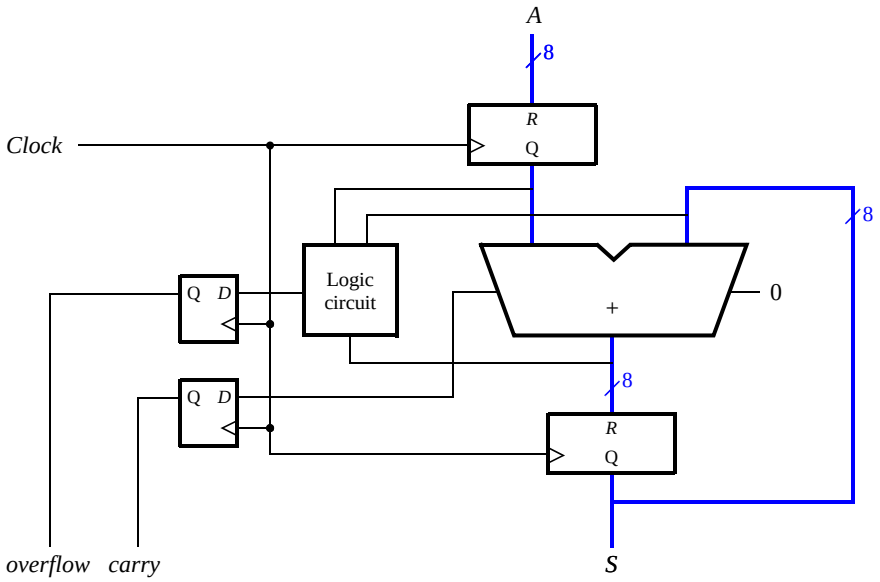

Figure 2: An eight-bit accumulator circuit.

Part II

Extend the circuit from Part I to be able to both add and subtract numbers. To do so, introduce an *add_sub* input to your circuit. When *add_sub* is 1, your circuit should subtract A from S, and when *add_sub* is 0 your circuit should add A to S as in Part I.

Part III

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

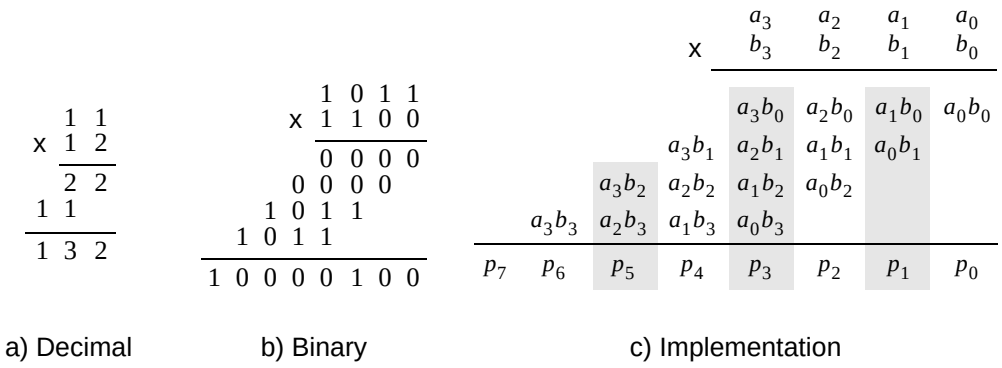

Figure 3: Multiplication of binary numbers.

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*. 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.

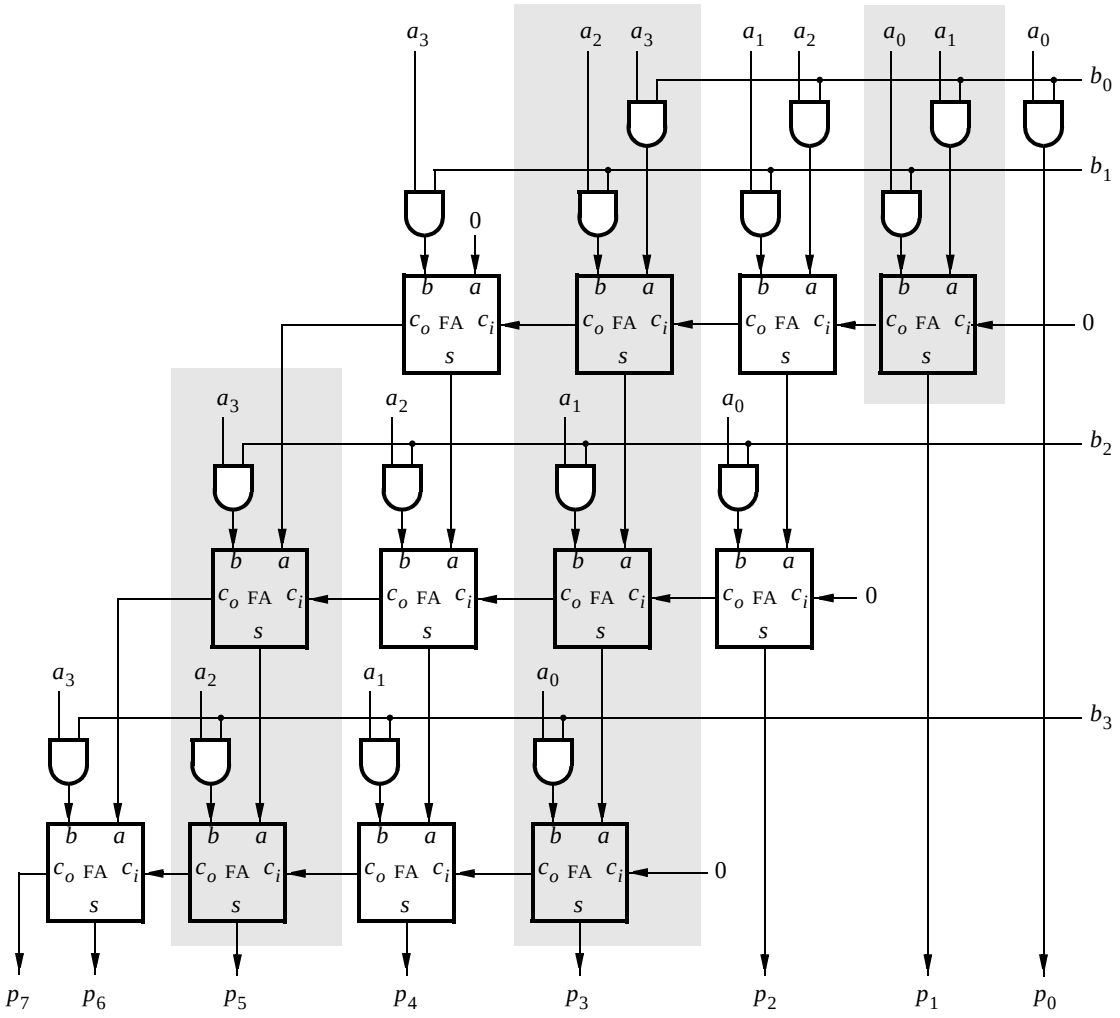

Figure 4: An array multiplier circuit.

Perform the following steps to implement the array multiplier circuit:

1. Create a new Quartus project.
2. Generate the required VHDL file. Use switches SW_{7-4} to represent the number A and switches SW_{3-0} to represent B. The hexadecimal values of A and B are to be displayed on the 7-segment displays *HEX*2 and

*HEX*0, respectively. The result $P = A \times B$ is to be displayed on *HEX*$5-4$.

3. Make the necessary pin assignments needed to implement the circuit on your DE-series board, and compile the circuit.
4. Use simulation to verify your design.
5. Download your circuit onto your DE-series board and test its functionality.

Part IV

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

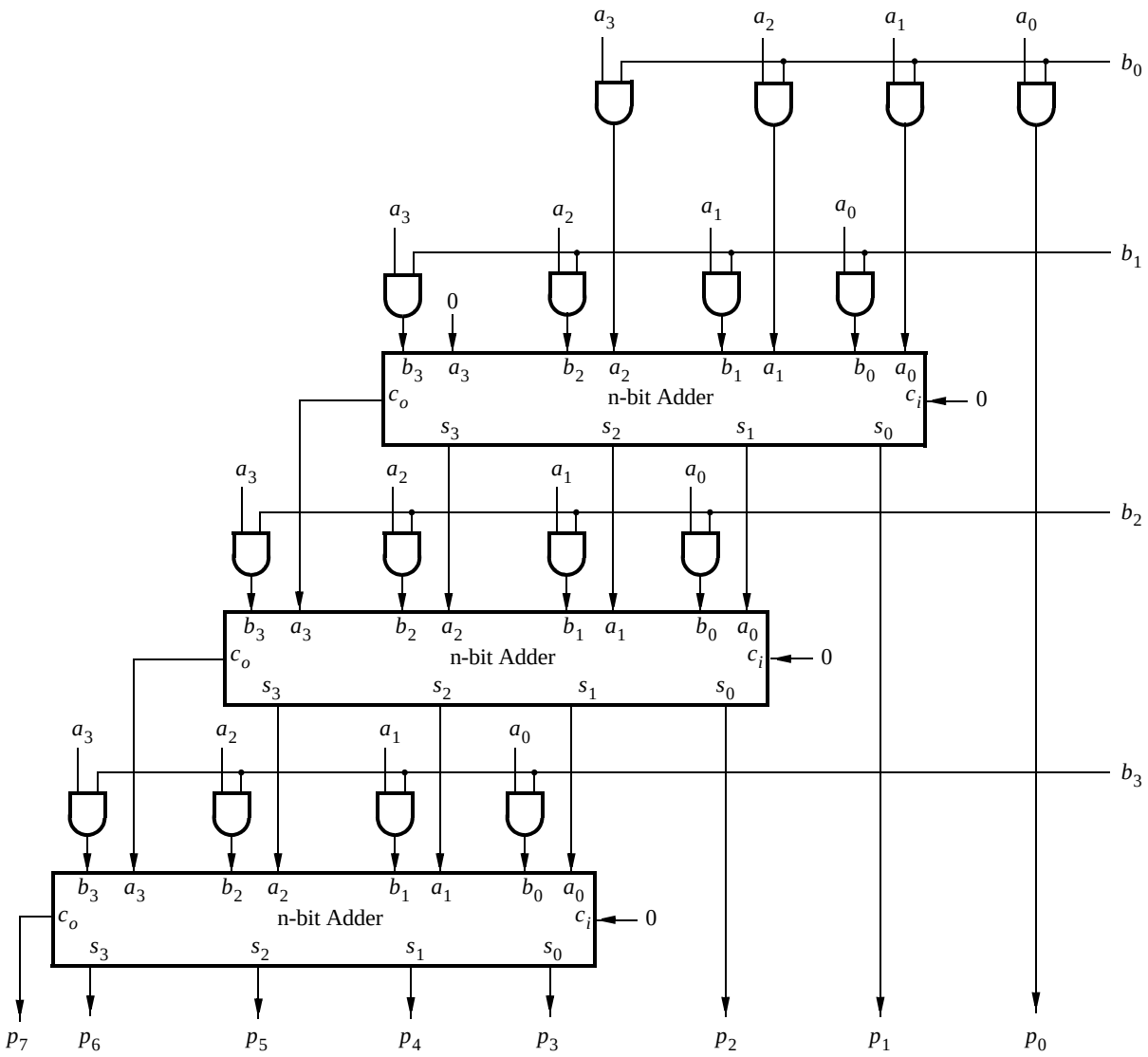

Figure 5: An array multiplier implemented using n-bit adders.

Each n-bit adder adds a shifted version of A for a given row and the *partial product* 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 8 x 8 multiplier circuit with registered inputs and outputs, as shown in Figure 6.

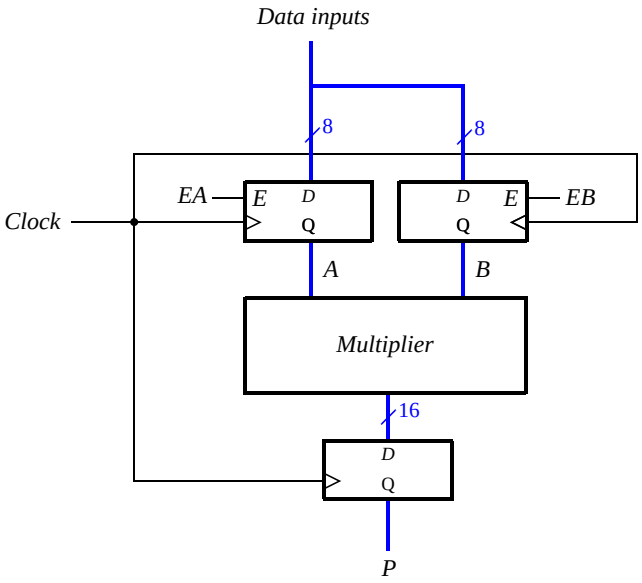

Figure 6: A registered multiplier circuit.

Perform the following steps:

1. Create a new Quartus project and write the required VHDL file.
2. Use switches SW_{7-0} to provide the data inputs to the circuit. Use SW_9 as the enable signal *EA* for register *A*, and use SW_8 as the enable for register *B*. When $SW_9 = 1$ display the contents of register A on the red lights LEDR, and display the contents of register B on these lights when $SW_8 = 1$. Use KEY_0 as a synchronous reset input, and use KEY_1 as a manual clock signal. Show the product $P = A \times B$ as a hexadecimal number on the 7-segment displays *HEX3-0*.
3. Make the necessary pin assignments needed to implement the circuit on your DE-series board, and compile the circuit.
4. Test the functionality of your design by inputting various data values and observing the generated products.

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.

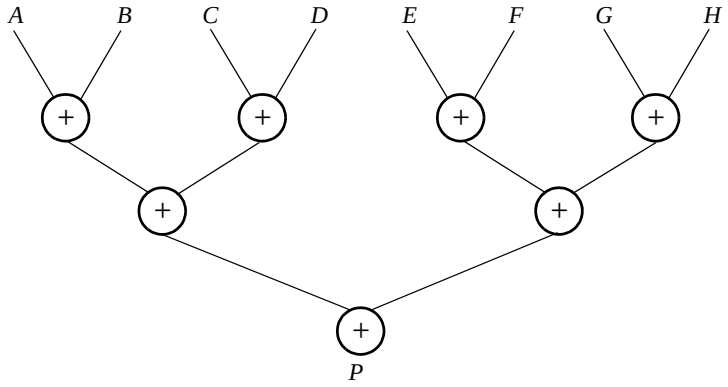

Figure 7: An example of adding 8 numbers using an adder tree.

In this part you are to implement an 8 x 8 multiplier circuit by using the adder-tree approach. Inputs A and B, as well as the output P should be registered as in Part IV.

Copyright © Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Avalon, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

*Other names and brands may be claimed as the property of others.