Lab 5: Adders, Subtractors, and Multipliers

CSC 258, University of Toronto

About this Lab

Learning Objectives

1. The purpose of this exercise is to examine arithmetic circuits that add, subtract, and multiply numbers.
2. This lab also introduces accumulators, a component of CPU architecture.

Marking Scheme

All items in this document labeled **PRELAB TODO:** are to be completed before arriving to the lab, and are worth one of the four marks for the labs. Items marked **INLAB TODO:** are to be completed in lab and will all be marked. One mark is given for completing parts I, II and III, and another mark is given for completing part IV. The last mark is given for completing one of the two Advanced tasks outlined in Section V.

1 Pre-lab Assignment

You are required to write the Verilog code for Parts I to V. To receive your pre-lab mark from the teaching assistants, you need to bring your Verilog code for Parts I to IV, your answer to the questions asked in Part II, and a print-out of your simulation for parts I and II.

2 Prelab

2.1 Part I

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

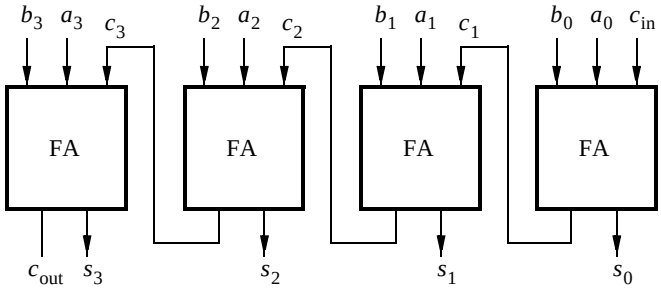

Figure 1: A four-bit ripple carry adder.

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

```
wire [n-1:0] sum;
wire carry;
...
assign {carry, sum} = A + B;
```

PRELAB TODO: Using this construct, implement the four-bit ripple carry adder from Figure 1, and simulate it to verify its correctness. **Note:** hand in a print-out of your simulation along with the code itself. Also note that you are just implementing an adder in this section for use in subsequent parts of the lab – no need for flip flops here!

2.2 Part II

Assume you're about to add ten 8-bit numbers together. One way would be to have ten 8-bit input lines and have an array of adders to add all the numbers together. Although the circuit is correct in theory, it is impractical due to the high number of input lines (80 input lines in this case) and the number of adders included. Thus, a better solution would be to provide numbers to the system one at a time. In this case, we would be providing one number at every clock cycle. Numbers are added to a register in every clock cycle. We continue the operation until all ten numbers are added, that is *accumulated* into the register.

An accumulator is a construct that we use to add multiple numbers together over time. A rough equivalent of this construct in software programming would be the "+=" operator.

PRELAB TODO: Write Verilog code that describes the circuit in Figure 2. Simulate your code by choosing appropriate data for inputs *Clock* and *A*. Demonstrate the output of *S* and *Overflow* in your simulation waveforms.

1. Create a new Quartus II project. Select as the target chip the Cyclone II EP2C35F672C6, which is the FPGA chip on the Altera DE2 board. Select Cyclone II EP2C35F672C6 device to implement the designed circuit on the DE2 board.
2. 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 Overflow output should be displayed on the red led $LEDR_8$.
   PRELAB TODO: What does active-low asynchronous reset mean? How is it different from a synchronous reset?
3. Assign the pins on the FPGA to connect to the switches and 7-segment displays by importing the *DE2_pin_assignments.csv* file.
4. Compile your design and use timing simulation to verify the correct operation of the circuit. Test the circuit by using different values of A. Be sure to check that the *Overflow* output works correctly.

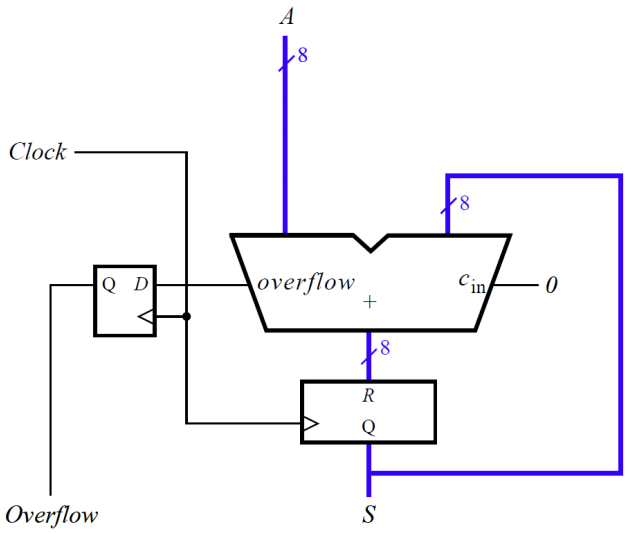

Figure 2: An eight-bit accumulator circuit.

2.3 Part III

Extend the circuit from Part II to be able to both add and 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, otherwise it should add A to S.

2.4 Part IV

Figure 3*a* gives an example of paper-and-pencil multiplication $P = A \times B$, where $A = 11$ and $B = 12$. 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 3*c* shows how each summand can be formed by using the Boolean AND operation of A with the appropriate bit of B.

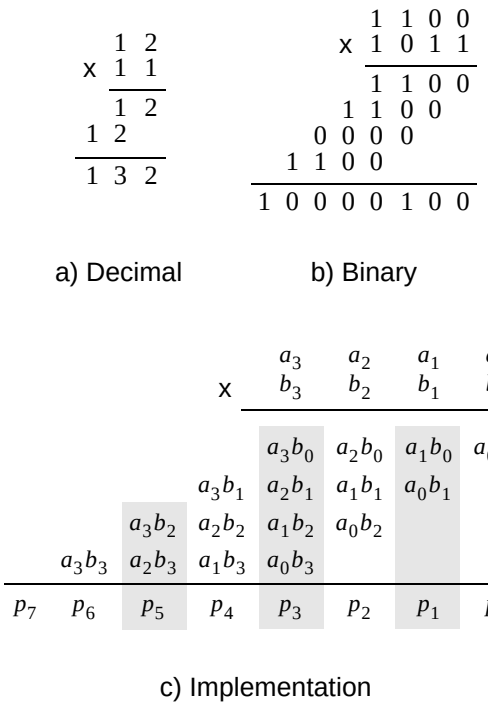

Figure 3: Multiplication of binary numbers.

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 3*c*. 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.

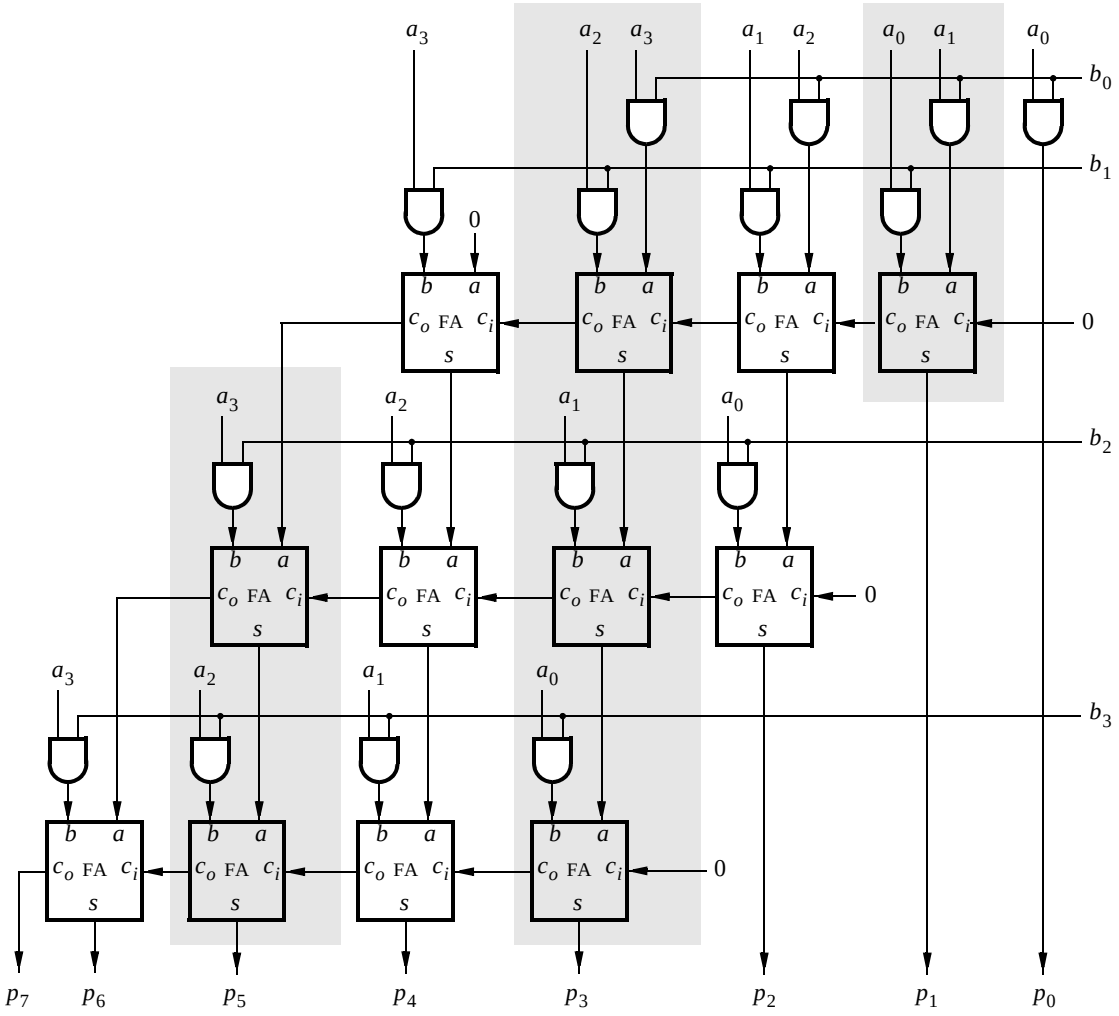

Figure 4: An array multiplier circuit.

PRELAB TODO: Implement the circuit shown in Figure 4 in Verilog.

1. Create a new Quartus II project to implement the desired circuit on the Altera DE2 board.
2. Include your Verilog file in your project, and compile the project.
3. Use functional simulation to verify your design.

INLAB TODO: Do the following steps in the lab

4. 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 *HEX1* and *HEX0*, respectively. The result $P = A \times B$ is to be displayed on *HEX3, HEX2*.
5. Assign the pins on the FPGA to connect to the switches and 7-segment displays by importing the *DE2_pin_assignments.csv* file.
6. Compile 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.

Part V - Advanced

In Part IV, 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.

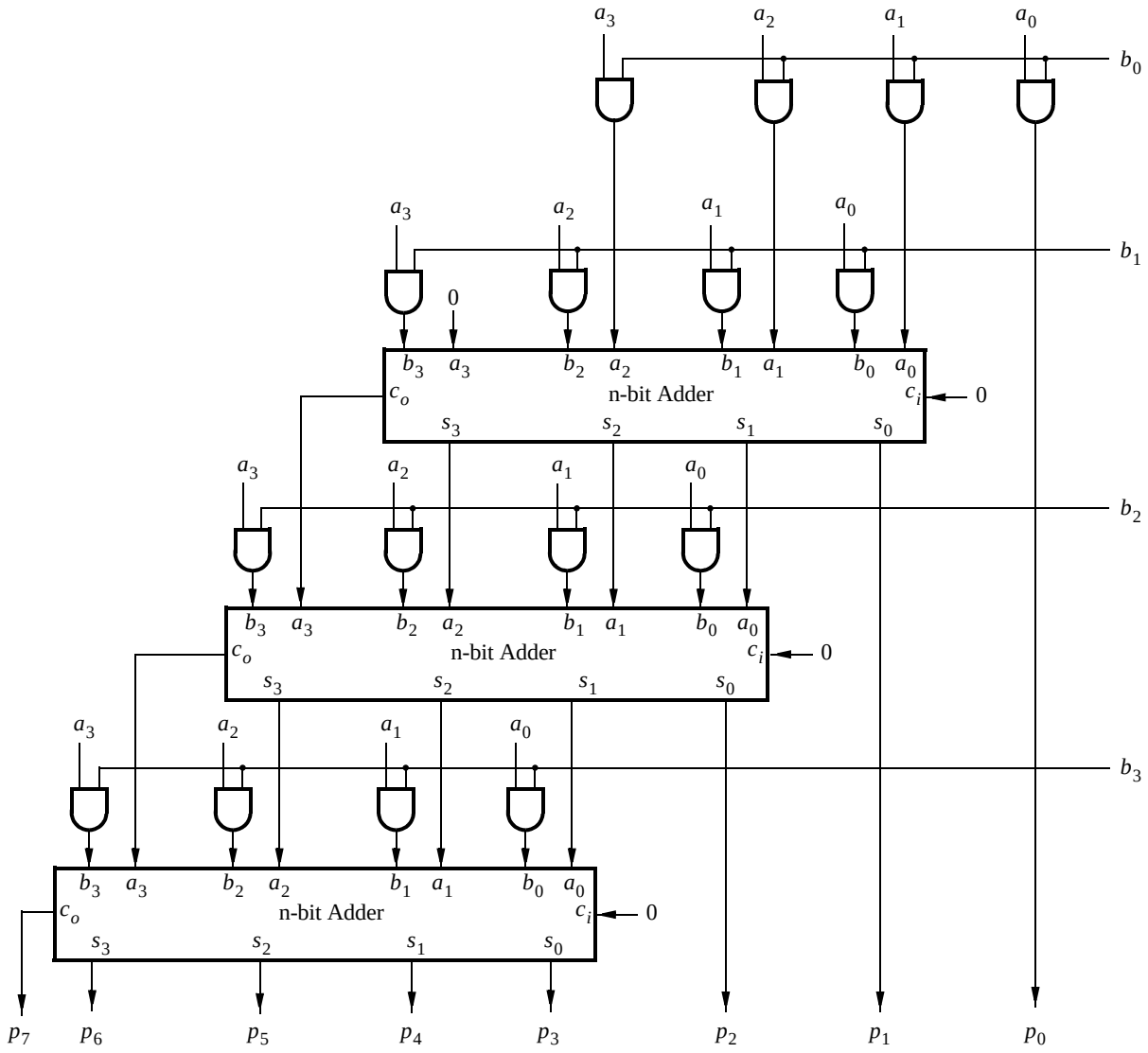

Figure 5: An array multiplier implemented using n-bit adders.

Each n-bit adder adds a shifted version of A for a given row and the partial sum of the row above. Abstracting the multiplier circuit as a sequence of additions allows us to build larger multipliers. In the case of a 4x4 multiplier, the circuit should consist of 4-bit adders arranged in a structure as shown in Figure 5.

For this advanced section, you have a choice. You can either implement the circuit in Figure 5 using TTL logic, or you can use Verilog to implement this approach to a 4x4 multiplier circuit with registered inputs and outputs, as shown in Figure 6.

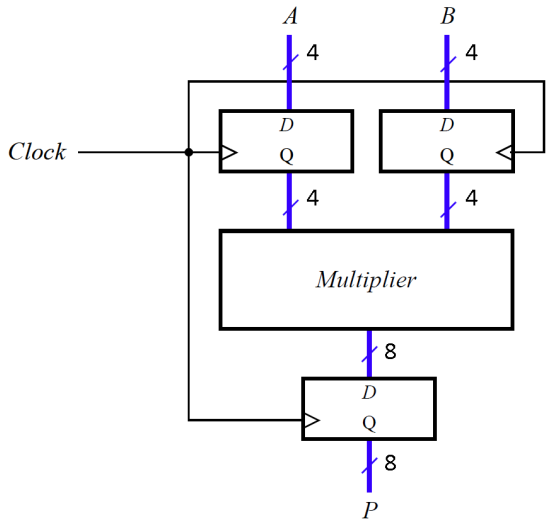

Figure 6: A registered multiplier circuit.

INLAB TODO: Either implement the circuit using TTL logic chips, or perform the following steps:

1. Create a new Quartus II project.
2. Write the required Verilog 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_{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 *HEX1* and *HEX0*, respectively. The result $P = A \times B$ is to be displayed on *HEX3-2*.
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?

3 In lab work

For parts II-IV, compile and download your code to the FPGA. Test the functionality of your design by toggling the switches and observing the displays.

1. **INLAB TODO:** Demonstrate, on the board, your code for Part II to your TA.
2. **INLAB TODO:** Demonstrate, on the board, your code for Part III to your TA.
3. **INLAB TODO:** Demonstrate, on the board, your code for Part IV to your TA.
4. **INLAB TODO:** Advanced: Demonstrate, on the board, your code for Part V to your TA.

Clean up your station once you are done with the lab.