"FPGA-Programming" Exercise Sheet VI:

Adders, Subtractors, and Multipliers

The purpose of this exercise is to examine arithmetic circuits that add, subtract, and multiply numbers.

As target hardware always choose FPGA chip Cyclone V SoC 5CSEMA5F31C6

1) *Accumulator Circuit*

   Following figure shows a 4-bit wide ripple carry adder which has been discussed in exercise sheet II).

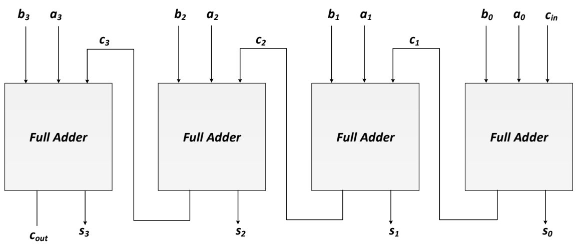

Figure 1.1: 4-bit wide ripple carry adder

Instead of describing the behaviour of the adder by using Boolean expressions, the circuit can be described in VHDL using the + symbol.
By including the functions of **std_logic_1164**, **std_logic_arith** and **std_logic_signed** it is possible to write the addition operation of two n-bit wide numbers as:

```
slv_S_int <= '0' & slv_A_int + '0' & slv_B_int;
```

Goal of this exercise is to use this syntax in order to describe the circuit of following figure:

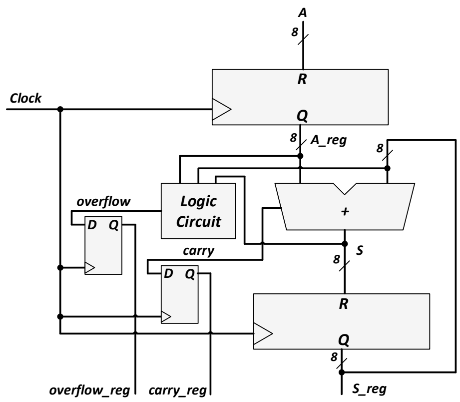

Figure 1.2: 8-bit wide accumulator circuit

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

Input **A** should be provided by switches $SW[7-0]$. Push-button $KEY[0]$ should be used as asynchronous low-active reset, push-button $KEY[1]$ as manual clock input. The sum of the adder **S** should be displayed on $LEDR[7-0]$, the output of the *carry flip-flop* **carry_reg(0)** should be displayed on $LEDR[8]$, the output of the *overflow flip-flop* **overflow_reg(0)** should be displayed on $LEDR[9]$. The output signals of the registers A (**A_reg**) and S (**S_reg**) should be displayed as hexadecimal numbers on the 7-segment displays $HEX3$, $HEX2$ and $HEX1$, $HEX0$ respectively.

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_accumulator**.

ii) First, describe the entity of the 7-segment decoder **e_hex7seg**. Use a concurrent conditional signal assignment.

iii) Describe the entity of a parametrisable n-bit wide register **e_regn**. The register features a asynchronous low-active reset input. In the case of an active reset, the output vector **Q** should be reset. Otherwise at rising edge of the clock the output should take the input **R**.

iv) Describe the top-level entity (**e_my_accumulator**) of the circuit. Consider that one additional bit is necessary for the creation of the $n+1$-bit sum **S**. For both n-bit values **A_reg** and **S_reg** use the concatenation operator $\&$ and add leading zero bit.

v) For the creation of the **overflow** bit determine and describe the logic function / conditions. Complete the description of the accumulator circuit.

vi) Import the necessary pin assignments, compile the circuit and download it into the FPGA. Verify the correct behaviour of the circuit with different values of **A**. Also check for the correct behaviour of the **overflow** output signal.

vii) Instead of using **std_logic_arith** and **std_logic_signed** it is advised to use the functions of **numeric_std** instead. Here individual signals can be defined as **signed** or **unsigned**. Therefore modify the top-level entity (**e_my_accumulator**) to be based on the preferred variant.

viii) Develop a VHDL test bench for the circuit.

2) *Accumulator Circuit with subtraction capability*

Goal of this part is to extend the circuit of the accumulator of part 1) in a way, to be able to both add and subtract numbers. To do so, introduce an **Add_Sub** input to the circuit. When **Add_Sub** is high, the circuit should subtract A from S, and when **Add_Sub** is low the circuit should add A to S. Use switch $SW[9]$ for the **Add_Sub** signal.

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_accaddsub**.

ii) For the description of the 7-segment decoder **e_hex7seg** use the one of part 1).

iii) For the description of the parametrisable n-bit wide register **e_regn** use the one of part 1).

iv) For the realisation of the top-level entity (**e_my_accaddsub**) extend the circuit of part 1) accordingly. Use the functions of **numeric_std**. Consider that a subtraction can be expressed as an addition using two's complement representation (inverting the operand and adding a 1). Describe **S** in a way to be able to handle both the addition as well as the subtraction operation.

v) For the creation of the **overflow** bit modify the logic function / conditions accordingly. Complete the description of the accumulator circuit with subtraction capability.

vi) Import the necessary pin assignments, compile the circuit and download it into the FPGA. Verify the correct behaviour of the circuit with different values of **A** for both addition and subtraction operation. Also check for the correct behaviour of the **overflow** output signal.

3) *Multiplier Circuit based on Full-Adders*

Following figure illustrates the paper-and-pencil multiplication $P = A \cdot B$, where $A = 11$ and $B = 12$.

```
      1 1
x     1 2
---------
      2 2
    1 1
---------
    1 3 2
```

Figure 3.1: Decimal paper-and-pencil multiplication

Here the product $P = A \cdot B$ is calculated as addition of summands. The first summand is equal to A times the ones digit of B, therefore $11 \cdot 2 = 22$. The second summand is A times the tens digit of B, shifted one position to the left, therefore $(11 \cdot 1) \times 10 = 110$. The addition of both summands yields $P = A \cdot B = 22 + 110 = 132$.

Following figure shows the same example using four-bit binary numbers. To compute $P = A \cdot B$, first form the 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.

```
          1 0 1 1
x         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
```

Figure 3.2: Binary paper-and-pencil multiplication

As can be seen in next figure, each of the summands can be described by a simple Boolean expression (the AND operation of one bit of A with a bit of B).

				a_3	a_2	a_1	a_0
			x	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

Figure 3.3: Boolean expressions for the binary paper-and-pencil multiplication

Due to the regular structure, this kind of multiplier circuit is called ***Array Mutliplier***. A 4-bit wide multiplier circuit is given in following figure.

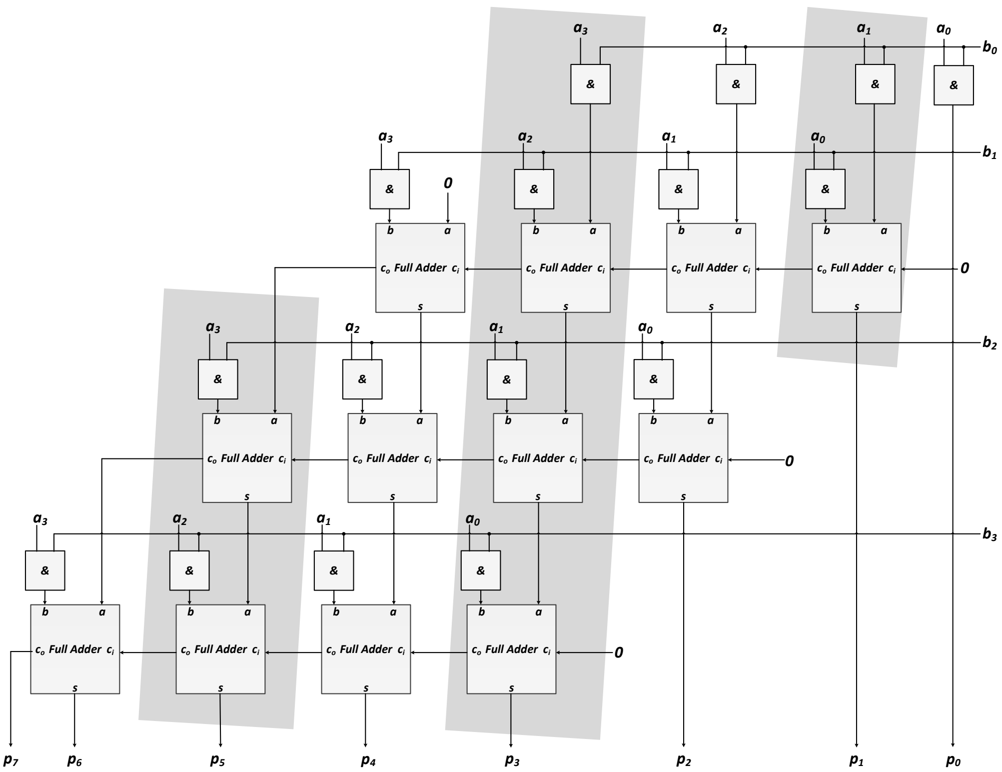

Figure 3.4: 4-bit wide Array Multiplier using Full-Adder cells

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_arraymult**.

ii) Describe the VHDL entity of a 7-segment decoder **e_hex7seg**.
Use a *combinatorial process* and a **case - when** statement.

iii) Describe the VHDL entity of a full-adder **e_fulladder**. Only use Boolean expressions here (cf. exercise II) part 3)).

iv) Describe the top-level entity (**e_my_arraymult**) in order to implement above circuit. Use switches $SW[7-4]$ for the representation of the number A and switches $SW[3-0]$ for the representation of number B. The numbers A and B should be displayed in hexadecimal notation on the 7-segment displays $HEX2$ and $HEX0$. The result of the multiplication $P = A \cdot B$ should be displayed on the 7-segment displays $HEX5$ and $HEX4$.

v) Import the necessary pin assignments, compile the circuit and download it into the FPGA. Verify the correct functionality of the circuit with different values of A and B.

4) *Multiplier Circuit based on n-bit Adder*

In the previous part the multiplier circuit has been realised using discrete full-adder cells. 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 following figure.

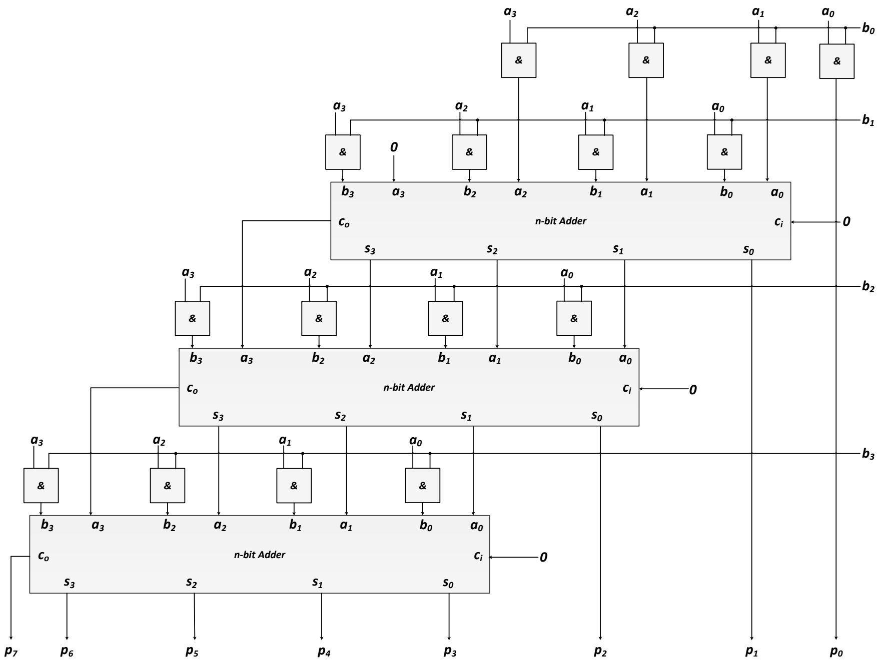

Figure 4.1: 4-bit wide Array Multiplier using n-bit adder

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. Use this approach to implement a 8×8 multiplier circuit with buffered inputs and outputs, as shown in following figure.

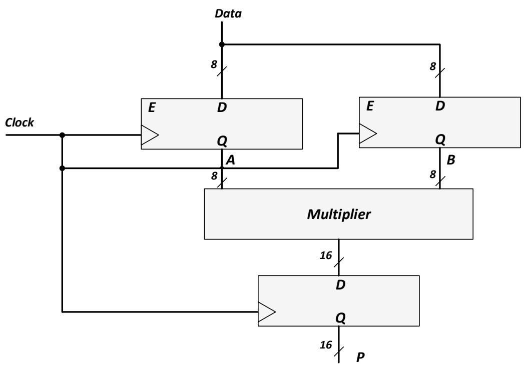

Figure 4.2: A buffered 8-bit multiplier circuit

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_mult**.

ii) Use the same VHDL entity of a 7-segment decoder **e_hex7seg** as in part 3).

iii) Describe the top-level entity (**e_my_mult**).
Use a *sequential process* with asynchronous low-active reset. During reset, the outputs of the register **A**, **B** and **P** should be set to zero. Push-button $KEY[0]$ should be used for the reset signal. The registers of A and B both exhibit an enable input. Use switch $SW[9]$ for the enable input of register A and switch $SW[8]$ for the enable input of register B. Based on the setting of the enable signals the switches $SW[7-0]$ should be used to store the value of A and B, respectively. In addition, based on the setting of the enable signal the stored value of register A or B should be displayed on $LEDR[7-0]$. Push-button $KEY[1]$ should be used as manual clock input.

iv) Identify an appropriate way for the description of the sum outputs of the n-but adder by using a concurrent *conditional signal assignment* and the concatenation operator $\&$ together with constant zero padding bits.

v) Import the necessary pin assignments, compile the circuit and download it into the FPGA. Verify the correct functionality of the circuit by trying different values of A and B with a manual clock.

vi) Use Quartus **TimeQuest Analyzer** to find out the maximum possible frequency of the circuit.

5) *Multiplication by using an Adder-Tree*

Part 4) 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. The basic principle is illustrated in following figure.

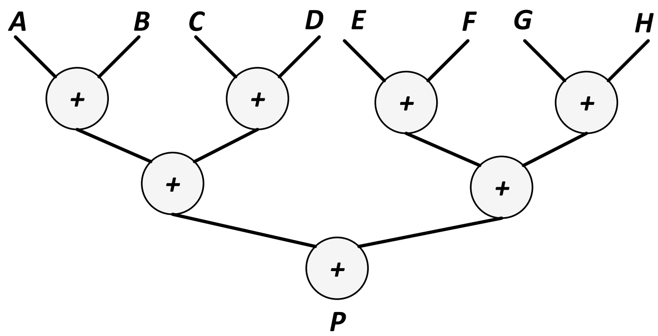

Figure 5.1: An example of adding 8 numbers using an adder-tree

In this part the 8×8 multiplier circuit of part 4) should be implemented by using the adder-tree approach. The multiplier inputs as well as the result should be buffered, as in part 4).

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_multaddertree**.

ii) Use the same VHDL entity of a 7-segment decoder **e_hex7seg** as in part 4).

iii) Describe the top-level entity (**e_my_multaddertree**). Use the top-level entity of part 4) as a basis and change it appropriately to implement an adder-tee.

iv) Import the necessary pin assignments, compile the circuit and download it into the FPGA. Verify the correct functionality of the circuit by trying different values of A and B with a manual clock.

v) Use Quartus **TimeQuest Analyzer** to find out the maximum possible frequency of the circuit. Compare the result with the one of part 4).