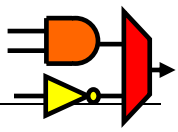


POLITECNICO DI TORINO

– Elettronica dei Sistemi Digitali –
Lab#3

Data-Path Elements

This purpose of this lab is the examination of the arithmetic circuits that add, subtract, and multiply binary numbers. Addition, subtraction and multiplication are fundamental operations usually implemented in a Data-Path (DP). The DPs are generally sequential circuits that numerically process some kind of digital input stream. For complex circuits, it is often necessary to combine multiple adders or multipliers. **Please note that some parts are optional and will be evaluated separately.**

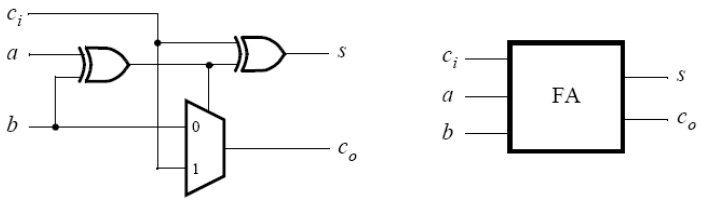
Contents:

Abbreviations and acronyms:
CBA – Carry Bypass Adder
CSA – Carry Select Adder
DP – Data Path
IC – Integrated Circuit
LED – Light Emitting Diode
MUX – Multiplexer
RCA – Ripple Carry Adder
VHDL – Very high speed integrated circuits Hardware Description Language
[VHDL cookbook: http://www.onlinefreeebooks.net/engineering-ebooks/electrical-engineering/the-vhdl-cookbook-pdf.html]

1 – 8-bit Sequential RCA

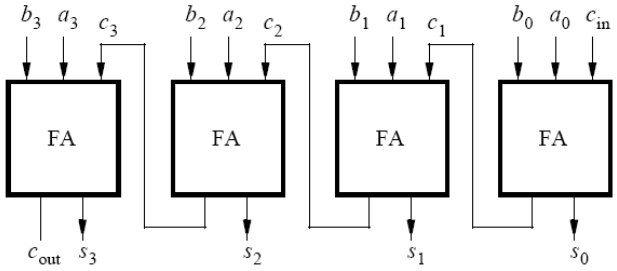
Figure 1a shows a possible implementation of a *full adder*. It reads the three inputs a, b, and c_i, and drives the outputs s and c_o. Parts b and c of the figure show a circuit symbol and the truth table of the full adder. This fundamental logic block, used massively in every digital design, generates a 2-bit array of binary sums $<c_o\ s> = a + b + c_i$, where c_o and s are called Carry-Out and Sum, respectively. Figure 1d shows how four instances of this full adder can be used for designing a circuit that adds up two 4-bit numbers. This type of circuit is usually called *ripple-carry* adder. The term "ripple" refers to the way the carry-out is propagated from a full adder to the next one. As a matter of fact, in this topology, we have to wait until the carry is propagated until the last Full Adder to have all the five sum bits ready. Create an 8-bit version of the adder and include it in the circuit shown in Figure 2b. Your circuit should be designed to support signed

numbers in 2's-complement form, and the *Overflow* output should be set to 1 whenever the sum produced by the adder does not provide the correct sign value. Do the steps below.



a) Full adder circuit b) Full adder symbol

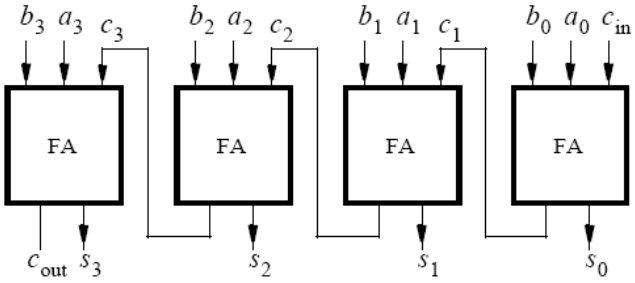
b	a	c_i	c_o	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



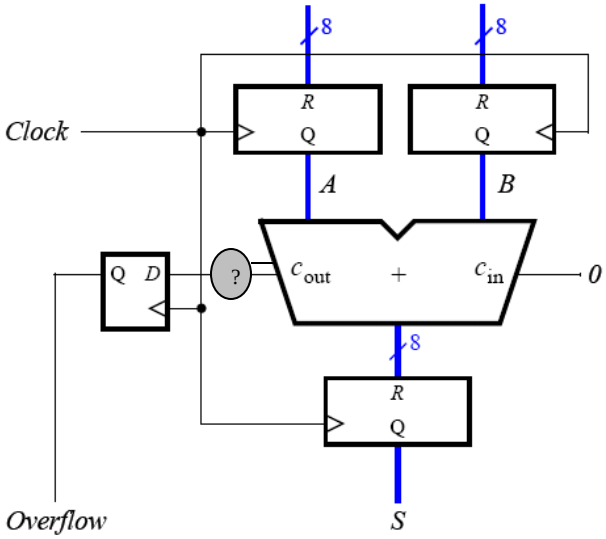
c) Full adder truth table d) Four-bit ripple-carry adder circuit

Figure 1 - A ripple-carry adder circuit.

1. **Create a new Quartus II project** and write VHDL code that describes the circuit in Figure 2b. Use the circuit structure in Figure 2a to describe your adder. Use the templates shown in Figure 3 to describe the D-FF and the register.
2. **Include the required input and output ports** in your project to implement the adder circuit on the DE2 board. Connect the inputs *A* and *B* to switches *SW*15-8 and *SW*7-0, respectively. Use *KEY*0 as an active-low asynchronous reset input, and use *KEY*1 as a manual clock input. Display the sum outputs of the adder on the red *LEDR*7-0 lights and display the overflow output on the green *LEDG*8 light. The hexadecimal values of *A* and *B* should be shown on the displays *HEX7-6* and *HEX5-4*, and the hexadecimal value of *S* should appear on *HEX1-0*.
3. **Compile your code** and use timing simulation to verify the correct operation of the circuit. Once the simulation works properly, download the circuit onto the DE2 board and test it by using different values of *A* and *B*. Be sure to check the proper functionality of the *Overflow* output.
4. Open the Quartus II Compilation Report and examine the **results reported by the Timing Analyzer**. What is the maximum operating frequency f_{max} of your circuit? What is the longest path in the circuit in terms of delay?



a) Four-bit ripple-carry adder circuit



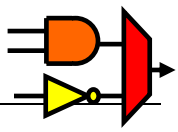
b) Eight-bit registered adder circuit

Figure 2 - An 8-bit signed adder with registered inputs and outputs.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY regn IS
      GENERIC ( N : integer:=8);
      PORT (R                 : IN  SIGNED(N-1 DOWNTO 0);
            Clock, Resetn     : IN  STD_LOGIC;
            Q                 : OUT SIGNED(N-1 DOWNTO 0));
END regn;

ARCHITECTURE Behavior OF regn IS
BEGIN
      PROCESS (Clock, Resetn)
      BEGIN
            IF (Resetn = '0') THEN
                  Q <= (OTHERS => '0');
```



```
            ELSIF (Clock'EVENT AND Clock = '1') THEN
                  Q <= R;
            END IF;
      END PROCESS;
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
      PORT (D, Clock, Resetn  : IN  STD_LOGIC;
            Q                 : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
      PROCESS (Clock, Resetn)
      BEGIN
            IF (Resetn = '0') THEN -- asynchronous clear
                  Q <= '0';
            ELSIF (Clock'EVENT AND Clock = '1') THEN
                  Q <= D;
            END IF;
      END PROCESS;
END Behavior;
```

Figure 3 - VHDL code of the register and the D-FF.

2 - 8-bit Sequential Adder/Subtractor (optional)

GOAL: Modify the circuit in figure 1 in such a way it becomes an adder/subtractor. Leave all the inputs untouched and add the ADD/SUB control.

Modify your circuit from section 1 in such a way it can perform both addition and subtraction of eight-bit numbers. Use switch SW_{16} to specify whether addition or subtraction should be done. Connect the switches, LED, and displays as described in section 1.

1. **Simulate your adder/subtractor circuit** and show that it works properly.
2. **Download it** onto the DE2 board and test it by using different switch settings.
3. Open the Quartus II Compilation Report and examine the **results reported by the Timing Analyzer**. What is the maximum operating frequency f_{max} of your circuit? What is the longest path in the circuit in terms of delay?

3 – 16-bit RCA, Carry-Bypass Adder and Carry-Select Adder

GOAL: Implement a 16-bit version of RCA, Carry-Bypass and Carry-Select adders and compare them in terms of maximum operating frequency. REPORT your results. NOTE: Carry-Bypass adder is OPTIONAL

Modify your circuit based on a ripple-carry adder of section 1 in such a way that it can perform additions of 16-bit numbers.

1. **Simulate** your Ripple-Carry adder circuit and show that it functions properly.
2. Open the Quartus II Compilation Report and examine the **results reported by the Timing Analyzer**. What is the f_{max} of your circuit? What is the longest path in the circuit in terms of delay?

Substitute the RCA with a Carry-Bypass adder (figure 3).

3. **Simulate** your Carry-Bypass adder circuit and show that it functions properly.
4. Open the Quartus II Compilation Report and examine the **results reported by the Timing Analyzer**. What is the f_{max} of your circuit? What is the longest path in the circuit in terms of delay?

Substitute the Carry-Bypass adder with a Carry-Select adder (figure 4).

5. **Simulate** your Carry-Select Adder circuit to show that it functions properly.
6. Open the Quartus II Compilation Report and examine the results **reported by the Timing Analyzer**. What is the f_{max} of your circuit? What is the longest path in the circuit in terms of delay?
7. **Compare the timing results** and the synthesis reports of the three adders and comment them.

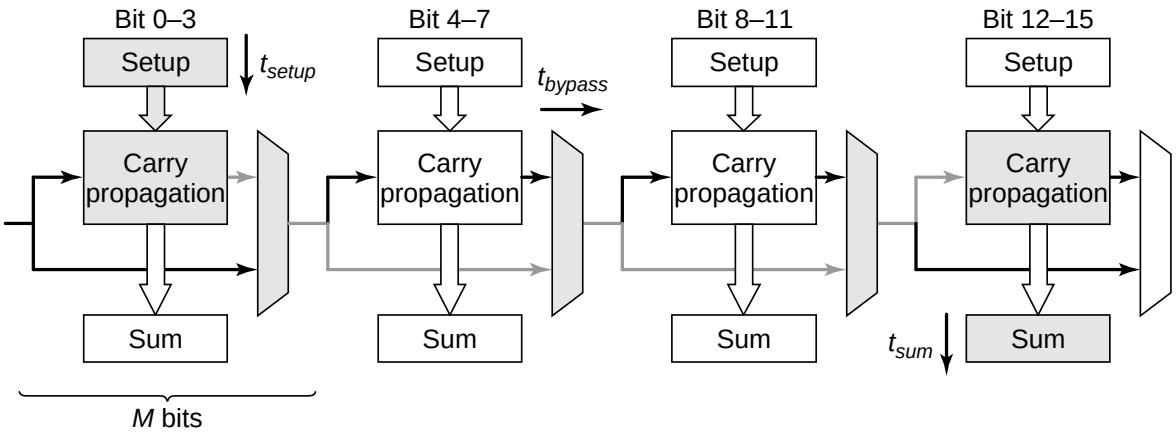


Figure 3 - Carry-Bypass Adder

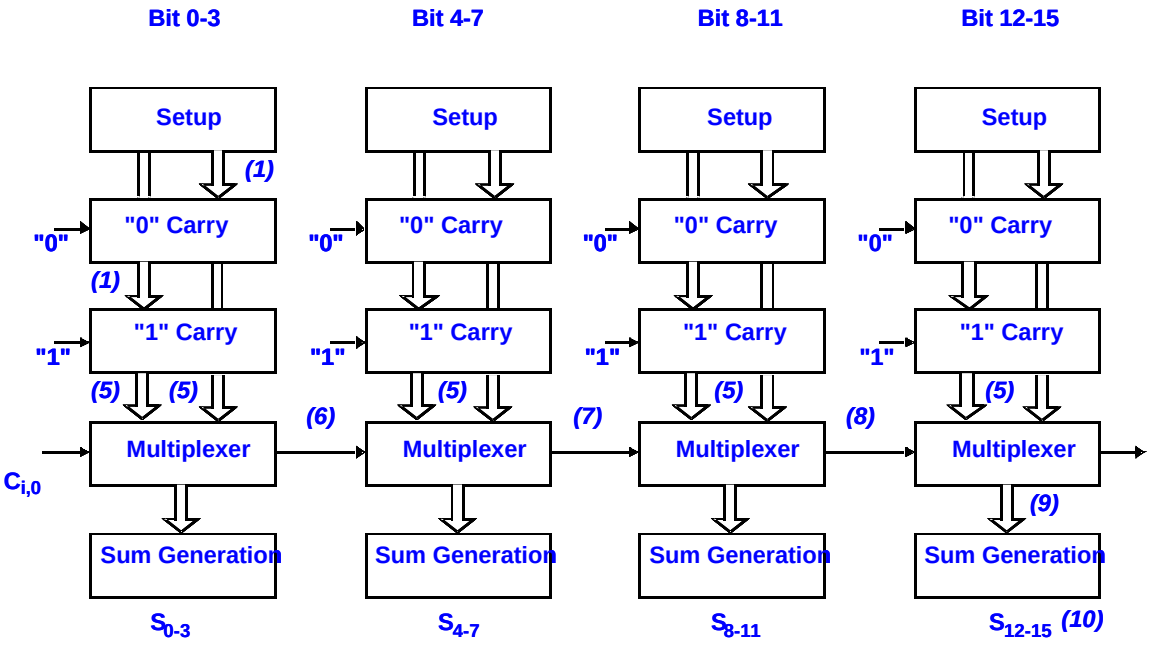


Figure 4 - Carry-Select Adder.

4 - Multiplier

GOAL: Describe a simple multiplier with the VHDL language.

Figure 5a shows the traditional paper-and-pencil multiplication $P = A \times B$, where $A = 12$ and $B = 11$. We need to add two summands that are shifted versions of A to have the product $P = 132$. The part b of the figure shows the same example by using four-bit binary numbers. Since each digit in B is either 1 or 0, the summands are either shifted versions of A or 0000. Figure 5c shows how each summand can be formed by an AND gate on A and the appropriate bit in B.

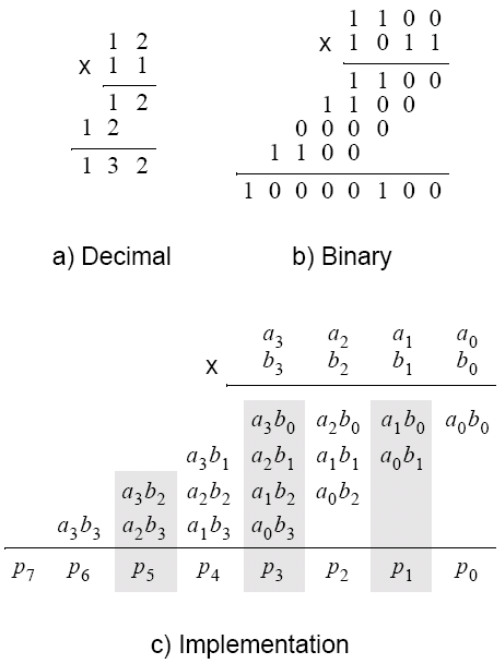


Figure 5 - Multiplication of binary numbers.

A 4-bit circuit that implements $P = A \times B$ is illustrated in Figure 6. Since its structure is regular, this type of multiplier circuit is usually called *array multiplier*. The shaded areas in the circuit correspond to the implementations of the shaded columns in Figure 5c. In each row of the multiplier the AND gates are used to generate the summands and the full adder modules are used to generate the required sums.

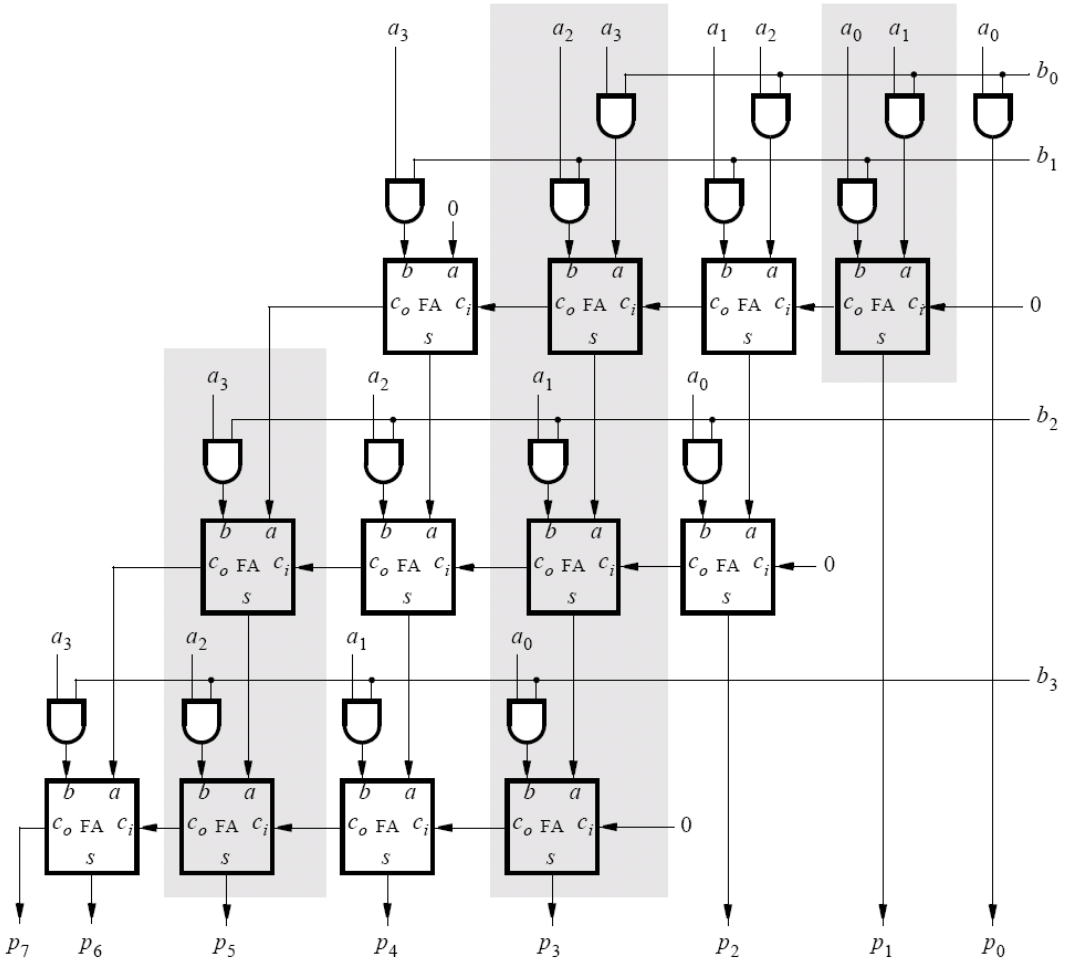


Figure 6 - An array multiplier circuit.

Do the following steps and implement the array multiplier circuit:

1. **Create a new Quartus II project** which will be used to implement the multiplier on the Altera DE2 board.
2. **Generate the VHDL file**, include it in your project, and compile the circuit.
3. Use **functional simulations** to verify that your code is correct.
4. **Augment your design** in such a way it uses switches SW_{11-8} to represent the number A and switches SW_{3-0} to represent B. The ***hexadecimal*** values of A and B should be displayed on the 7-segment displays *HEX6* and *HEX4*, respectively. The result $P = A \times B$ has to be displayed on *HEX1* and *HEX0*.
5. **Assign the pins on the FPGA** to route the connections of the switches and of the 7-segment displays as indicated in the User Manual of the DE2 board.
6. **Recompile the circuit** and download it in the FPGA IC.
7. **Test the functionality of your design** by toggling the switches and observing the 7-segment displays.