Lab 4: Binary adder

Objectives

The purpose of this laboratory exercise is to design an adder. It is a type of digital circuit that performs the operation of additions of two numbers.

Materials

You will use slide switches on the CPLD expansion board (schematic) as inputs and 7-segment display on the CoolRunner-II CPLD starter board (XC2C256-TQ144, manual, schematic) as output device.

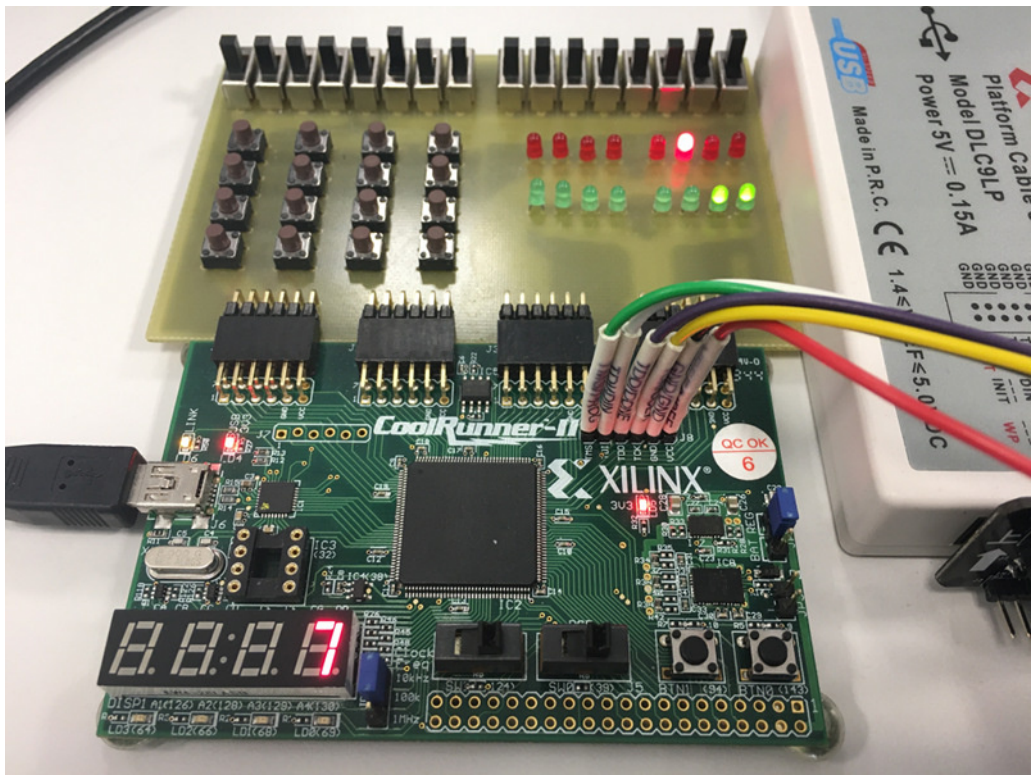

Figure 1: CoolRunner-II CPLD starter board

1 Preparation tasks (done before the lab at home)

1. A half adder has two inputs A and B and two outputs Carry and Sum. Complete the half adder truth table. Draw a logic diagram of both output functions.

B	A	Carry	Sum
0	0		
0	1		
1	0		
1	1		

2. A full adder has three inputs and two outputs. The two inputs are A, B, and Carry input. The outputs are Carry output and Sum. Complete the full adder truth table and draw a logic diagram of both output functions.

Cin	B	A	Cout	Sum
0	0	0		

Cin	B	A	Cout	Sum
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

3. Find the relationship between half adder and full adder logic diagrams.
4. See schematic of the CPLD expansion board and find out the connection of LEDs, push buttons, and slide switches.

2 Synchronize Git and create a new folder

1. Open a Linux terminal, use `cd` commands to change path to your Digital-electronics-1 working directory, and synchronize the contents with GitHub.

```
$ pwd
/home/lab661
$ cd Documents/your-name/Digital-electronics-1/
$ pwd
/home/lab661/Documents/your-name/Digital-electronics-1
$ git pull
```

2. Create a new folder `Labs/04-adder`

```
$ cd Labs/
$ mkdir 04-adder
$ cd 04-adder/
$ touch README.md
$ ls
README.md
```

3 VHDL code for half adder

1. Follow instructions from wiki, create a new project in ISE titled `binary_adder` for XC2C256-TQ144 CPLD device. Make sure the project location is `/home/lab661/Documents/your-name/Digital-electronics-1/La` , ie in **your** local folder.
2. Create a new source file **Project > New Source... > VHDL Module**, name it `half_adder` and copy/paste the following code template.

```
--------------------------------------------------------------------------------
--
-- Half adder.
-- Xilinx XC2C256-TQ144 CPLD, ISE Design Suite 14.7
--
-- Copyright (c) 2019-2020 Tomas Fryza
-- Dept. of Radio Electronics, Brno University of Technology, Czechia
-- This work is licensed under the terms of the MIT license.
--
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
```

```
--------------------------------------------------------------------------------
-- Entity declaration for half adder
--------------------------------------------------------------------------------
entity half_adder is
port (
    b_i     : in  std_logic;
    a_i     : in  std_logic;
    carry_o : out std_logic;
    sum_o   : out std_logic
);
end entity half_adder;

--------------------------------------------------------------------------------
-- Architecture declaration for half adder
--------------------------------------------------------------------------------
architecture Behavioral of half_adder is
begin
    -- Logic functions for carry and sum outputs
    -- WRITE YOUR CODE HERE
end architecture Behavioral;
```

3. Use low-level gates `and` , `or` , `not` , etc. and write logic functions for Carry and Sum. Save all files in menu **File > Save All**.

4 VHDL code for full adder

1. Create a new source file **Project > New Source... > VHDL Module**, name it `full_adder` and copy/paste the following code.

```
--------------------------------------------------------------------------------
--
-- Full adder.
-- Xilinx XC2C256-TQ144 CPLD, ISE Design Suite 14.7
--
-- Copyright (c) 2019-2020 Tomas Fryza
-- Dept. of Radio Electronics, Brno University of Technology, Czechia
-- This work is licensed under the terms of the MIT license.
--
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

--------------------------------------------------------------------------------
-- Entity declaration for full adder
--------------------------------------------------------------------------------
entity full_adder is
port (
    carry_i : in  std_logic;
    b_i     : in  std_logic;
    a_i     : in  std_logic;
    carry_o : out std_logic;
    sum_o   : out std_logic
);
end entity full_adder;

--------------------------------------------------------------------------------
-- Architecture declaration for full adder
--------------------------------------------------------------------------------
```

```
architecture Behavioral of full_adder is
    -- Internal signals between two half adders
    signal s_carry0, s_carry1, s_sum0 : std_logic;
begin

    ----------------------------------------------------------------------
    -- Sub-blocks of two half_adder entities
    HALF_ADDER_0 : entity work.half_adder
    port map (
        -- <component_signal> => <actual_signal>,
        -- <component_signal> => <actual_signal>,
        -- <other signals>...
        -- WRITE YOUR CODE HERE
    );

    HALF_ADDER_1 : entity work.half_adder
    port map (
        -- WRITE YOUR CODE HERE
    );

    -- Output carry
    -- WRITE YOUR CODE HERE

end architecture Behavioral;
```

2. A full adder can be implemented by two half adders and one OR gate. Follow the logic diagram of Satvik Ramaprasad and design a full adder.

   If top level module in Xilinx ISE has not changed automatically, do it manually: right click to **full_adder - Behavioral (full_adder.vhd)** line and select **Set as Top Module**.

3. Simulate design `full_adder` and test all input combinations according to the tutorial.

4. In menu **Tools > Schematic Viewer > RTL...** select **Start with a schematic of top-level block** and check the hierarchical structure of the module.

5 Top level implementation of 4-bit adder

1. Create a new source file **Project > New Source... > VHDL Module**, name it `top` and copy/paste the following code template.

   If top level module in Xilinx ISE has not changed automatically, do it manually: right click to **top - Behavioral (top.vhd)** line and select **Set as Top Module**.

```
----------------------------------------------------------------------------
--
-- Implementation of 4-bit adder.
-- Xilinx XC2C256-TQ144 CPLD, ISE Design Suite 14.7
--
-- Copyright (c) 2019-2020 Tomas Fryza
-- Dept. of Radio Electronics, Brno University of Technology, Czechia
-- This work is licensed under the terms of the MIT license.
--
----------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------------------------------
-- Entity declaration for top level
----------------------------------------------------------------------------
```

```
entity top is
port (
    SW0_CPLD   : in  std_logic;       -- Input A
    SW1_CPLD   : in  std_logic;
    SW2_CPLD   : in  std_logic;
    SW3_CPLD   : in  std_logic;
    SW8_CPLD   : in  std_logic;       -- Input B
    SW9_CPLD   : in  std_logic;
    SW10_CPLD  : in  std_logic;
    SW11_CPLD  : in  std_logic;
    disp_seg_o : out std_logic_vector(7-1 downto 0);
    disp_dig_o : out std_logic_vector(4-1 downto 0)
);
end entity top;

--------------------------------------------------------------------------
-- Architecture declaration for top level
--------------------------------------------------------------------------
architecture Behavioral of top is
    signal s_dataA, s_dataB : std_logic_vector(4-1 downto 0);
    signal s_carry0, s_carry1, s_carry2 : std_logic;
    signal s_result : std_logic_vector(4-1 downto 0);
    signal s_carryOut : std_logic;
begin

    -- Combine two 4-bit inputs to internal signals s_dataA and s_dataB
    -- WRITE YOUR CODE HERE


    ----------------------------------------------------------------------
    -- Sub-blocks of four full_adders
    FULL_ADDER_0 : entity work.full_adder
    port map (
        -- <component_signal> => <actual_signal>,
        -- <component_signal> => <actual_signal>,
        -- <other signals>...
        -- WRITE YOUR CODE HERE
    );

    FULL_ADDER_1 : entity work.full_adder
    port map (
        -- WRITE YOUR CODE HERE
    );

    FULL_ADDER_2 : entity work.full_adder
    port map (
        -- WRITE YOUR CODE HERE
    );

    FULL_ADDER_3 : entity work.full_adder
    port map (
        -- WRITE YOUR CODE HERE
    );


    ----------------------------------------------------------------------
    -- Sub-block of hex_to_7seg entity
    HEX2SSEG : entity work.hex_to_7seg
    port map (
```

```
        -- WRITE YOUR CODE HERE
    );

    -- Select display position
    disp_dig_o <= "1110";


    -- Show carry output bit on Coolrunner-II LED
    -- WRITE YOUR CODE HERE

    -- Show two 4-bit inputs on CPLD expansion LEDs
    -- WRITE YOUR CODE HERE

end architecture Behavioral;
```

2. Copy `hex_to_7seg.vhd` and `coolrunner.ucf` files from previous lab to current working folder and add them to the project: **Project > Add Source...**. In constraints file comment/uncomment all inputs/outputs you need in this top level design.

   Create a new constraints file with file name `cpld_board` and copy/paste the following code. The file contains pin assignments of input/output devices on CPLD expansion board. Again, comment/uncomment all inputs/outputs you need in this top level design.

```
#------------------------------------------------------------------------
#
# Constraints file with pin assignments.
# CPLD expansion board, ISE Design Suite 14.7
#
# Copyright (c) 2019-2020 Tomas Fryza
# Dept. of Radio Electronics, Brno University of Technology, Czechia
# This work is licensed under the terms of the MIT license.
#
#------------------------------------------------------------------------

#------------------------------------------------------------------------
# Buttons & switches
# 16 shared push buttons and slide switches
#------------------------------------------------------------------------
#
# 15  ...  8       7  ...  0
# +-+     +-+     +-+     +-+
# | | ... | |     | | ... | | H
# |.|     |.|     |.|     |.| L
# +-+     +-+     +-+     +-+
#
#  15  11  7   3
#  o   o   o   o   H: pressed
#  o   o   o   o   L: released
#  o   o   o   o
#  o   o   o   o
#  12  8   4   0
#
#NET SW15_CPLD           LOC = P9;
#NET SW14_CPLD           LOC = P10;
#NET SW13_CPLD           LOC = P6;
#NET SW12_CPLD           LOC = P7;
#NET SW11_CPLD           LOC = P4;
#NET SW10_CPLD           LOC = P5;
#NET SW9_CPLD            LOC = P2;
#NET SW8_CPLD            LOC = P3;
```

```
#NET SW7_CPLD            LOC = P140;
#NET SW6_CPLD            LOC = P142;
#NET SW5_CPLD            LOC = P138;
#NET SW4_CPLD            LOC = P139;
#NET SW3_CPLD            LOC = P135;
#NET SW2_CPLD            LOC = P136;
#NET SW1_CPLD            LOC = P133;
#NET SW0_CPLD            LOC = P134;

#------------------------------------------------------------------------
# 16 discrete LEDs
#------------------------------------------------------------------------
#
# 15 ... 12 11 ... 8
#  * * * *   * * * *   H: turn LED on
#  * * * *   * * * *   L: turn LED off
#  7 ... 4   3 ... 0
#
#NET LD15_CPLD            LOC = P118;
#NET LD14_CPLD            LOC = P119;
#NET LD13_CPLD            LOC = P116;
#NET LD12_CPLD            LOC = P117;
#NET LD11_CPLD            LOC = P114;
#NET LD10_CPLD            LOC = P115;
#NET LD9_CPLD             LOC = P112;
#NET LD8_CPLD             LOC = P113;
#NET LD7_CPLD             LOC = P103;
#NET LD6_CPLD             LOC = P104;
#NET LD5_CPLD             LOC = P101;
#NET LD4_CPLD             LOC = P102;
#NET LD3_CPLD             LOC = P98;
#NET LD2_CPLD             LOC = P100;
#NET LD1_CPLD             LOC = P96;
#NET LD0_CPLD             LOC = P97;
```

3. Use sub-blocks of hexadecimal to seven segment decoder, four sub-blocks of 1-bit full adders, interconnect all blocks, use slide switches/LEDs on CPLD expansion boards, seven-segment display on CoolRunner board, and implement 4-bit adder.
4. In menu **Tools > Schematic Viewer > RTL...** select **Start with a schematic of top-level block** and check the hierarchical structure of the module.
5. In menu **Project > Design Summary/Reports** check **CPLD Fitter Report (Text)** for implemented functions in section `********** Mapped Logic **********` .

6 Clean project and synchronize git

1. In Xilinx ISE, clean up all generated files in menu **Project > Cleanup Project Files...** and close the project using **File > Close Project**.

   Warning: In any file manager, make sure the project folder does not contain any **large** (gigabyte) files. These can be caused by incorrect simulation in ISim. Delete such files.

2. Use `cd ..` command in Linux terminal and change working directory to `Digital-electronics-1` . Then use git commands to add, commit, and push all local changes to your remote repository. Check the repository at GitHub web page for changes.

```
$ pwd
/home/lab661/Documents/your-name/Digital-electronics-1/Labs/04-adder

$ cd ..
```

```
$ cd ..
$ pwd
/home/lab661/Documents/your-name/Digital-electronics-1

$ git status
$ git add <your-modified-files>
$ git status
$ git commit -m "[LAB] Adding 04-adder lab"
$ git status
$ git push
$ git status
```

Experiments on your own

1. Add one control line `Subtract` and create a combined 4-bit adder-subtractor. The control line `Subtract` holds a binary value of either 0 or 1 which determines that the operation being carried out is addition or subtraction. Note, two's complement form is used to create an opposite number.
2. Complete your `README.md` file with notes and screenshots from simulation and implementation.