City University of Hong Kong

Department of Electrical Engineering

EE 2000 – Lab Manual 3

Simple ALU design

Course Leader: CHIN Lip Ket

Objectives:

- To practice designing combinational logic circuit;
- To reuse the materials and knowledge from previous labs;
- To learn how to perform type conversion in VHDL Code;
- To apply bit slicing and bit concatenation technique in VHDL.
- To learn how to use a logic analyzer to trace the digital signals.

There are 3 Checkpoints on Pages 8 and 11.
For each checkpoint, please take notes/photos/screenshots for your report.
Show all checkpoints to the lab tutor or demonstrator for marking.

Experiment 1: ALU Implementation

i. An arithmetic logic unit (ALU) is to perform arithmetic and bitwise operation on integer binary numbers. In this experiment, you will implement a simple 3- bit ALU. Given two 3-bit operands **A**, **B** and the 2-bit operator **op,** the result is shown in the following table.

op	Result	carry
00	A **ADD** B	It depends
01	A **AND** B	0
10	A **XOR** B	0
11	A **<< 1**	0

Here, we give a template for you to write your own code **alu.vhd**. In this design, we have a duplicated copy of the output signal result and carry, defined as dup_result and dup_carry in the following code. For these two copies of output, one is used to drive the LED and another is output to PMOD connector JA, so that we can use logic analyzer to probe such signal. In your code, result_t and carry_t are used to store the required output.

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

Entity ALU is
   Port (
      A: in STD_LOGIC_VECTOR (2 DOWNTO 0);
      B: in STD_LOGIC_VECTOR (2 DOWNTO 0);
      op: in STD_LOGIC_VECTOR (1 DOWNTO 0);
      result: out STD_LOGIC_VECTOR (2 DOWNTO 0);
      carry: out STD_LOGIC;

      --dup_result and dup_carry are used for observation using Logic Analyzer
      dup_result: out STD_LOGIC_VECTOR (2 DOWNTO 0);
      dup_carry: out STD_LOGIC
   );
end ALU;

Architecture Behavioral of ALU is
SIGNAL result_t: STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL carry_t: STD_LOGIC;

--You can add your declaration here.
Begin
   dup_result <= result_t;
   dup_carry <= carry_t;
   result <= result_t;
   carry <= carry_t;

   --Write your own code to implement this ALU.
end Behavioral;
```

ii. Here are some hints for you to write your own VHDL code:

(1) In VHDL, if the type is STD_LOGIC_VECTOR, you can directly use the following operators.

```
A + B          -- Addition
A AND B        -- Logical and
A OR B         -- Logical or
A XOR B        -- Logical xor
```

(2) VHDL has logical shift operator **sll** (shift left) and **srl** (shift right), but they are for BIT_VECTOR. If A and result are both STD_LOGIC_VECTOR, then type conversion is needed.

```
result <= to_stdlogicvector(to_bitvector(A) sll 1);
```

(3) You can get the sum and carry of two 3-bit inputs by expanding inputs to 4- bit, that is

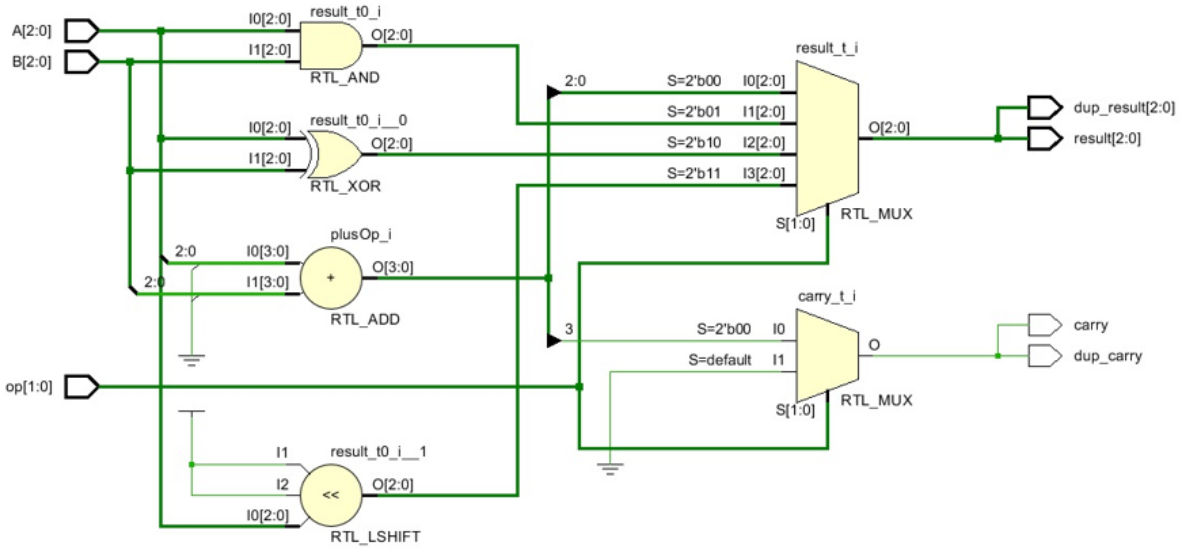
```
tmp <= ('0'&A) + ('0'&B);
-- A and B are both 3-bit inputs
-- tmp is a 4-bit result and tmp(2 downto 0) is the sum and tmp(3) is the carry value
```

(4) Now you can learn how to use the **WHEN** statement. Here is a simple example for you. It is a simple decoder from SIGNAL sel to SIGNAL result_out. Note the use of ',', and ';' carefully.

```
WITH sel SELECT
    result_out <= "0001" WHEN "00",
                  "0010" WHEN "01",
                  "0100" WHEN "10",
                  "1000" WHEN "11",
                  "ZZZ" WHEN others;
```

iii. **Add sources** alu.vhd to your own project.

iv. **Perform RTL analysis** on the source file. Get the schematic and see how Vivado infer logic gates according to your code. The schematic might be different from the one shown below.



v. Write an **I/O constraint** file and add it to your project. It is similar as Lab 2. Note that the bank voltage for IO Bank 33 is fixed to 3.3V on ZedBoard. In this experiment, we use SW0-SW2 as input "A", SW3-SW5 as input "B", SW6- SW7 as input "op", LD0-LD2 as output "result", LD3 as output carry. JA1-JA3 are connected with output port dup_result[0]-dup_result[2] and JA4 with dup_carry.

```
# write your I/O constraint here

set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 33]];

# Set the bank voltage for IO Bank 34 to 1.8V by default.
set_property IOSTANDARD LVCMOS18 [get_ports -of_objects [get_iobanks 34]];

# Set the bank voltage for IO Bank 35 to 1.8V by default.
set_property IOSTANDARD LVCMOS18 [get_ports -of_objects [get_iobanks 35]];

# Note that the bank voltage for IO Bank 13 is fixed to 3.3V on ZedBoard.
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 13]];
```

vi. **Simulate** the Design using the XSim Simulator.

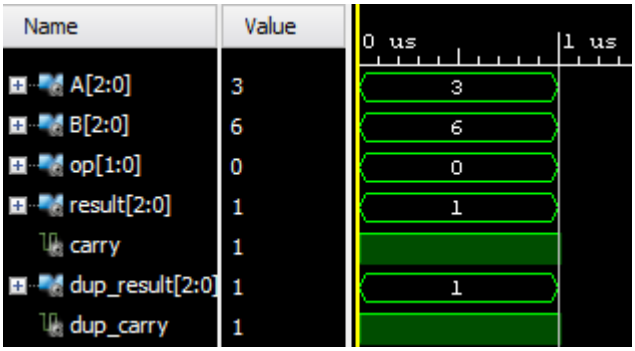
In order to test your design, we should write a testbench to generate the input signal. In this design, input signals include operand A and B, operator op. In the simulation process, we should assign different value to A, B and op, then verify carry and result are correct or not. For testbench, we still use the same VHDL grammar to generate expected stimulations. Here we provide a simple test bench as follows to you.

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;

Entity ALU_tb is
end ALU_tb;

Architecture Behavioral of ALU_tb is
  COMPONENT ALU is
   Port (
     A: in STD_LOGIC_VECTOR (2 DOWNTO 0);
     B: in STD_LOGIC_VECTOR (2 DOWNTO 0);
     op: in STD_LOGIC_VECTOR (1 DOWNTO 0);
     result: out STD_LOGIC_VECTOR (2 DOWNTO 0);
     carry: out STD_LOGIC;
     dup_result:out STD_LOGIC_VECTOR (2 DOWNTO 0);
     dup_carry: out STD_LOGIC
   );
  End COMPONENT;
  SIGNALA: STD_LOGIC_VECTOR (2 DOWNTO 0);
  SIGNAL B: STD_LOGIC_VECTOR (2 DOWNTO 0);
  SIGNAL op: STD_LOGIC_VECTOR (1 DOWNTO 0);
  SIGNAL result: STD_LOGIC_VECTOR (2 DOWNTO 0);
  SIGNAL carry: STD_LOGIC;
  SIGNAL dup_result: STD_LOGIC_VECTOR (2 DOWNTO 0);
  SIGNAL dup_carry: STD_LOGIC;
begin
  uut: ALU PORT MAP (A, B, op, result, carry, dup_result, dup_carry);
```

```
    A <= "011";
    B <= "110";
    op <= "00";
end Behavioral;
```



You can use a **PROCESS** block to generate more stimulations. Try to write a test bench, alu_tb, to verify all the functionality of your design. Variables declaration and design instantiation are provided as follows.

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;

Entity ALU_tb is
end ALU_tb;

Architecture Behavioral of ALU_tb is
   COMPONENT ALU is
      Port (
         A: in STD_LOGIC_VECTOR (2 DOWNTO 0);
         B: in STD_LOGIC_VECTOR (2 DOWNTO 0);
         op: in STD_LOGIC_VECTOR (1 DOWNTO 0);
         result: out STD_LOGIC_VECTOR (2 DOWNTO 0);
         carry: out STD_LOGIC;
         dup_result:out STD_LOGIC_VECTOR (2 DOWNTO 0);
         dup_carry: out STD_LOGIC
      );
```

```
    End COMPONENT;
    SIGNALA: STD_LOGIC_VECTOR (2 DOWNTO 0);
    SIGNAL B: STD_LOGIC_VECTOR (2 DOWNTO0);
    SIGNAL op: STD_LOGIC_VECTOR (1 DOWNTO 0);
    SIGNAL result: STD_LOGIC_VECTOR (2 DOWNTO 0);
    SIGNAL carry: STD_LOGIC;
    SIGNAL dup_result: STD_LOGIC_VECTOR (2 DOWNTO 0);
    SIGNAL dup_carry: STD_LOGIC;
begin
    uut: ALU PORT MAP (A, B, op, result, carry, dup_result, dup_carry);
        siggen: PROCESS
          begin
          -- Add your code here to set values of A, B and op.
        end PROCESS siggen;
end Behavioral;
```

Checkpoint 1: show your testbench and the waveform window captures (that shows ADD, AND, XOR, shift operation) to your demonstrator.

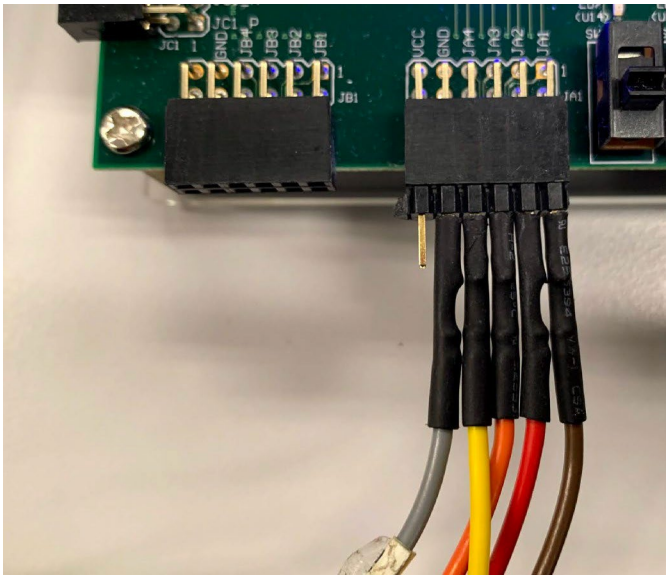
Run synthesis, **Run implementation, Generate bitstream** as the experiment 1. Then, we can **program** the bitstream file to the ZedBoard. After that, you can verify the design by setting different input settings.

Checkpoint2: Show your board and results to your demonstrator and verify your design properly. Please remember to take the screen capture, and photo for the lab report writing.

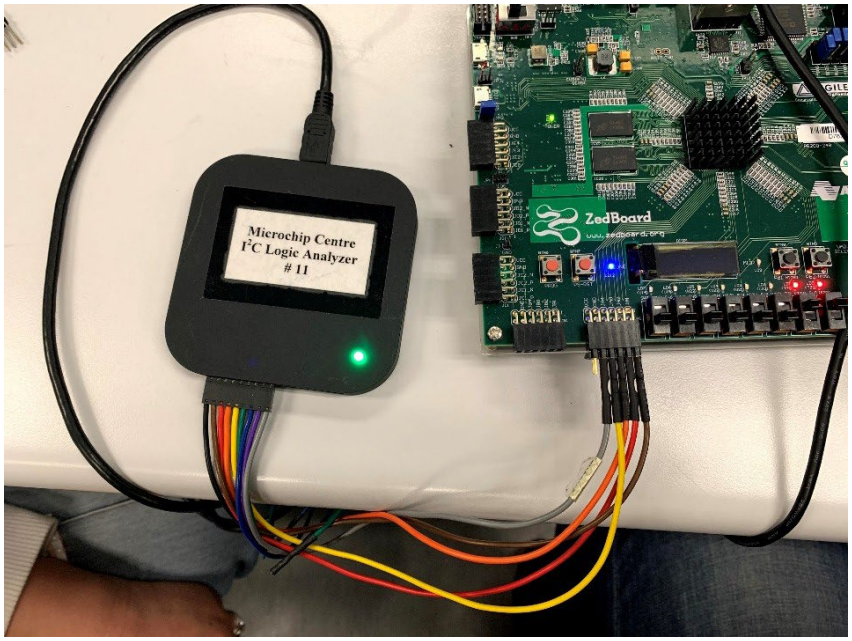
Experiment 2: Use Logic Analyzer to Track Signals

i. To use the physical logic analyzer, first, we should connect the rainbow cable to the black hub as shown in the figure below. Make sure that the gray cable should be aligned with the ground icon on the back of the logic analyzer.

ii. Then, you can connect the cable with JA pmod connector, i.e. connect gray cable with GND port, brown cable with JA1, red cable with JA2, orange cable with JA3 and yellow cable with JA4. The gray one should connect with GND pin. Please recall Experiment 1.v, in Page 5, you can remember the signals connected to JA1, JA2, JA3, JA4. What are these signals?

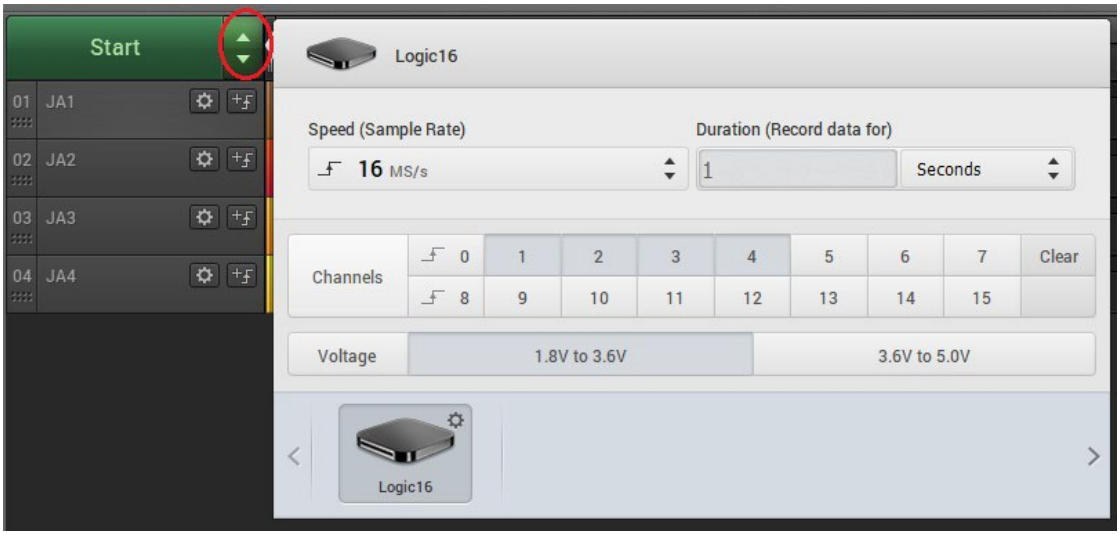


iii. Then, connect the logic analyzer with computer using the micro-usb cable.



iv. Open “logic 1.1.15” software on the desktop of your computer in the lab.

v. Click the arrow key beside “Start”. Adjust the settings as shown in the following figure. (Logic 16. Speed 16 MS/s. Duration 1 Second. Channels 1,2,3,4. Voltage 1.8V to 3.6V)



vi. After setting all the parameters correctly, click Start.

vii. The waveform is transferred once you click the start button.

Checkpoint 3: show your physical logic analyzer waveform window and your Zedboard to the demonstrator in your lab session. You should demonstrate the captured waveform for the ALU Design.

~ END ~