1 Discussion

The purpose of this lab is to design, simulate, and implement an Arithmetic and Logic Unit (ALU) using a Hardware Description Language (HDL). HDLs are used to describe the architecture and behavior of binary digital systems. HDLs were developed to deal with increasingly complex designs. An analogy is often made between hardware and software languages:

| machine code | transistors |
| assembly language | netlists |
| high-level languages | HDLs |

The HDLs typically support a mixed-level description, where structural or netlist constructs can be mixed with behavioral or algorithmic descriptions. With this mixed-level capability, one can describe system architectures at a high level of abstraction; then incrementally refine a design into a particular component-level or gate-level implementation. Alternatively, one can use a compiler (synthesizer) to synthesize a gate-level implementation automatically. A design methodology that uses HDLs has several fundamental advantages over a traditional gate-level design methodology. Some of the advantages are listed below.

- The functionality of a HDL design can be verified earlier in the design process and immediately simulated.
- Design simulation at a high level, before implementation at the gate level, allows the designer to test architectural and design decisions.
- The design logic can be synthesized and optimized automatically -- converting a HDL description to a gate-level implementation.
- Synthesis eliminates the gate-level design bottleneck (errors introduced when a HDL specification is hand-translated to gates) thereby reduces circuit design time.
- With a synthesizer, the design can be automatically transformed to a smaller and faster circuit. The designer can apply information gained from the synthesized and optimized circuits back to the HDL description, perhaps to fine-tune architectural decisions.
- A HDL description supplies technology-independent documentation of a design and its functionality.
- A HDL description is easier to read and understand than a netlist or schematic description. Because the initial HDL design description is technology-independent, you can later reuse it to generate the design in a different technology, without having to translate from the original technology.
There are two IEEE standard HDL languages, Verilog and VHDL, which means that such description can be processed by any design tool with HDL support by any vendor. In case of schematic design entry, one must learn a particular design tool, and there is almost no way to transfer a design from one platform to another.

So, mastering the HDL approach is a must for those who are planning to work in the hardware design field during his/her career.

In this lab, we will describe the Lab 6 ALU using an HDL called VHDL (which stands for VHSIC Hardware Description Language, where the acronym VHSIC refers to Very High-Speed Integrated Circuit). VHDL is one of a few HDLs in widespread use today. VHDL is recognized as a standard HDL by the Institute of Electrical and Electronics Engineers (IEEE Standard 1076, ratified in 1987) and by the United States Department of Defense (MIL-STD-454L).

VHDL divides digital systems (components, circuits, or systems) into an external or visible part (entity name and connections) and an internal or hidden part (entity algorithm and implementation). After you define the external interface to a digital system, other digital systems can use this digital system as a building block. This concept of internal and external views is central to the VHDL view of system design. A digital system is defined by (1) its connections to the outside world and (2) its internal behavior. You can explore alternative implementations (architectures) of a digital system without changing the rest of the design. After you define an entity for one design, you can reuse it in other designs as needed. You can develop libraries of entities to use with many designs or a family of designs. A VHDL hardware model is shown in Figure 1.

A VHDL entity has one or more input, output, or inout ports that are connected (wired) to neighboring systems. An entity is composed of interconnected entities, processes, and components, all of which operate concurrently. Each entity is defined by a particular architecture, which is composed of VHDL constructs such as arithmetic, signal assignment, or component instantiation statements. In VHDL, sequential (clocked) circuits are modeled using independent processes, using flip-flops and latches, and combinatorial (unclocked) circuits are
modeled using only logic gates. Processes can define and call (instantiate) subprograms (subdesigns). Processes communicate with each other by signals (wires). A signal has a source (driver), one or more destinations (receivers), and a user-defined type, such as “color” or “number between 0 and 15”. VHDL provides a broad set of constructs. With VHDL, you can describe discrete electronic systems of varying complexity (systems, boards, chips, or modules) with varying levels of abstraction. VHDL language constructs are divided into three categories by their level of abstraction: behavioral, dataflow, and structural. These categories are described as follows:

- **Behavioral**: The functional or algorithmic aspects of a design, expressed in a sequential VHDL process.
- **Dataflow**: The view of data as flowing through a design, from input to output. An operation is defined in terms of a collection of data transformations, expressed as concurrent statements.
- **Structural**: The view closest to hardware; a model where the components of a design are interconnected. This view is expressed by component instantiations.

To learn the entire language is beyond the scope of this course, but we will try to introduce some features of it. To do this we will use a so-called a mixed schematic/HDL approach, which is easy for us to start with. We will use a familiar schematic capture tool by Xilinx to draw a data-path of our design (block diagram) and use VHDL to describe the building blocks of this design (for example, MUX or ADDER).

In this lab, you are to implement the ALU with the following functions:

<table>
<thead>
<tr>
<th>Operation Code OPER[2:0]</th>
<th>Operation Mnemonic</th>
<th>Operation Description</th>
<th>Operation Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>ADD</td>
<td>R=A+B</td>
<td>Arithmetic</td>
</tr>
<tr>
<td>001</td>
<td>SUB</td>
<td>R=A-B</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>MIN</td>
<td>R=Min(A, B)</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>MAX</td>
<td>R=Max(A, B)</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>AND</td>
<td>R=A and B</td>
<td>Logic</td>
</tr>
<tr>
<td>101</td>
<td>OR</td>
<td>R=A or B</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>XOR</td>
<td>R=A xor B</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>PASS</td>
<td>R=A</td>
<td>Data Transfer</td>
</tr>
</tbody>
</table>

To simplify this lab, there will be no SIG_OV, UNSIG_OV and ZERO from the ALU.
2 Pre-Lab

Look over the VHDL descriptions of the 3-to-8 decoder in Appendix A and the ALU in Appendix C. These examples use case statements. As you can see, it would be a simple matter to redefine the truth table to specify any combinational circuit. From these examples, generate the VHDL code for a hex-to-7-segment-decoder (see Lab 4) using the template in Appendix B. This template includes an entity definition (symbol) consisting of a 4-bit input bus called HEX (HEX3–HEX0) and a 7-bit output bus called SEG (SEG6–SEG0), as well as a part of the decoder architecture description. The decoder's architecture (function) should be described using a case statement. Write a VHDL code for the hex-to-7-segment-decoder.

The pre-Lab must be done before you come to lab.

3 In-lab

In the lab, we will use Xilinx ISE software to write a VHDL code, draw, simulate, and implement the ALU into the PLDT-3 board.

Task A: Create a new project (use name lab7). Download the ALU source file ALU.VHD from the course website, and add it into your project. Double click the file name, the source code of the file will be displayed on the right window.

Observe this source code written in VHDL. Green color highlighted text are comments which can be used to explain what the author (you) is describing. Use comments extensively in your programs. It helps you and others to analyze the code. The pink and blue colors highlighted words are called key words of the VHDL language. You can't use the same words in your program for signals and variables; it will cause an error during the program syntax-checking phase. Finally, the black text is the rest of the program code.

Select Synthesis-XST (Xilinx Synthesis Tool) in the Processes for Current Source window to generate the hardware (gates and registers) for the described circuit. If you will see a green checkmark appear, the process was run successfully. After synthesis, in the same window, click Create Schematic Symbol of Design Utilities. A symbol, alu, will be generated.

Task B: Go back to the Source in Project again. Create a new source file. In the New Source window, select VHDL Module and name the file HEXSEG.VHD. In the Define VHDL Source window, write

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Direction</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX</td>
<td>in</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>SEG</td>
<td>out</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Click Next, Next. Type in the source code for the hex-to-7-segment decoder you developed in the pre-lab between begin and end. Note that you need to add process (HEX) … end process and begin case HEX is … end case to indicate that the algorithm used here is a sequential VHDL process and a case statement construct.
Use Synthesis-XST to compile this circuit. If there is an error report, please read it, then correct the source code if you can, or ask the instructor if you can’t figure out what has happened. If the process was run successfully, generate a symbol, hexseg. Close the HDL Editor and go to the Source in Project.

Task C: Create a new schematic, Lab7, in the Source in Project window. Add the symbols, alu, (Task A) and hexseg (Task B), on this schematic diagram.

Task D: Draw wires to extend the inputs and outputs, A(A3~A0), B(B3~B0), OPER (OPER2~OPER0) and R(R3~R0) as well as X(X3~X0) and HEXSEG(SEG6~SEG0) without terminals. Since the VHDL compiler considers that you use 4-bit buses for the inputs and outputs, you will see the heavy lines that represent 4-bit buses are drawn. Meanwhile, add ibufs and obufs to the schematic and connect them to input/output markers (Don’t connect them to busses at the moment). Name the markers as A3, …, A0, B3,…,B0, OPER2, …, OPER0 and SEG6,…, SEG0.

Task E: In order to connect individual inputs (or outputs) to a bus, you need to add bus taps between them. The Add Bus Tap icon is on the left side of the icon Add I/O Marker. Left-click on the Add Bus Tap, choose the bus tap direction in the Options window (For example, right for an input and left for an output) and touch the bus tap to the bus A, and connect it to the input buffer of A0 by drawing a wire between them. Left-double-click on the bus tap, the Object Properties window will open. In the Category, there are two items, which are the attributes of the connected net in two ends (If NOTHING SHOW IN THE WINDOW, ZOOM IN YOUR SCHEMATIC AND CHECK IF THE BUS TAP CONNECTS THE BUS AND THE INPUT). Click the first item, change it into \texttt{A(3:0)} in the Value column and click Apply. Click the second item, change it into \texttt{A(0)} and click Apply. This indicates that the bus tap connect the bus \texttt{A(3:0)} and assign the input as \texttt{A(0)}. Do the same to the rest bits of A bus, B bus, OPER bus and HEXSEG bus.

Figure 2 Naming two ends of the bus tap
**Task F:** Similar to Lab 6, add a flip-flop, FD4, between the outputs of ALU and the inputs of HEXSEG. Use bus taps to connect the outputs of the ALU and the inputs of the FD4 as well as the outputs of the FD4 and the inputs of the HEXSEG. NOTICE THAT YOU NEED TO DRAW WIRES BETWEEN BUS TAPS AND THE PINS OF FD4.

**Task G:** Locate the inputs and outputs to the following pins of the PLCDT-3 board. Download the generated programming data into the PLDT-3 trainer board. Check your circuit to determine if it is working properly.

**Table 1**

<table>
<thead>
<tr>
<th>Input</th>
<th>Pin Location</th>
<th>Output</th>
<th>Pin Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK to flip-flop</td>
<td>P25</td>
<td>a</td>
<td>P15</td>
</tr>
<tr>
<td>A0</td>
<td>P5</td>
<td>b</td>
<td>P18</td>
</tr>
<tr>
<td>A1</td>
<td>P6</td>
<td>c</td>
<td>P23</td>
</tr>
<tr>
<td>A2</td>
<td>P7</td>
<td>d</td>
<td>P21</td>
</tr>
<tr>
<td>A3</td>
<td>P11</td>
<td>e</td>
<td>P19</td>
</tr>
<tr>
<td>B0</td>
<td>P1</td>
<td>f</td>
<td>P14</td>
</tr>
<tr>
<td>B1</td>
<td>P2</td>
<td>g</td>
<td>P17</td>
</tr>
<tr>
<td>B2</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>P4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPER0</td>
<td>P51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPER1</td>
<td>P52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPER2</td>
<td>P53</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**4 Lab Report**

Demonstrate the correct operation of your designs to your lab instructor. Print out the source code of your decoder (hex-to-7-segment) and the schematic diagram of the entire circuit.
Appendices

A. VHDL Source Code of 3-to-8 Decoder

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity DEC is
    port (A: in std_logic_vector (2 downto 0);
          Y: out std_logic_vector (7 downto 0));
end DEC;

architecture DEC_arch of DEC is
begin
    process(A)
    begin
        case A is
            when "000" => Y <= "00000001";
            when "001" => Y <= "00000010";
            when "010" => Y <= "00000100";
            when "011" => Y <= "00001000";
            when "100" => Y <= "00010000";
            when "101" => Y <= "00100000";
            when "110" => Y <= "01000000";
            when "111" => Y <= "10000000";
            when others => Y <= "00000000";         -- only for simulation
        end case;
    end process;
end DEC_arch;
```

B. Template for Hex-to-7-Segment Decoder

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity HEXSEG is
    port (HEX: in std_logic_vector (3 downto 0);    -- inputs
          SEG: out std_logic_vector (6 downto 0));   -- to segments, SEG(6) for seg a
end HEXSEG;

architecture HEXSEG_arch of HEXSEG is
begin
    process(HEX)
    begin
        case HEX is
            when …
            … put your code here
```
C. VHDL Source Code of ALU

```vhdl
-- 4-bit Arithmetic & Logic Unit
-- Operation code (Oper[2:0])
-- 000 - ADD ... R = A + B  (addition)
-- 001 - SUB ... R = A - B  (subtraction)
-- 010 - MIN ... R = min(A, B) (minimum)
-- 011 - MAX ... R = max(A, B) (maximum)
-- 100 - AND ... R = A and B (logic AND)
-- 101 - OR  ... R = A or  B (logic OR)
-- 110 - XOR ... R = A xor B (logic XOR)
-- 111 - PASS ... R = A (pass A)

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-- ALU inputs and outputs declaration
entity ALU is
port (Oper: in  std_logic_vector(2 downto 0);   -- operation
     A, B: in  std_logic_vector(3 downto 0);   -- input operands
     R:    out std_logic_vector(3 downto 0));  -- result
end ALU;

-- ALU behavior description
architecture ALU_arch of ALU is
begin
  ALU_process:  process(A, B, Oper) is -- sensitivity list
    begin                              -- architecture description begins here
    case Oper is                     -- select the operation
    when "000"  => R <= A + B;     -- ADD
    when "001"  => R <= A - B;     -- SUB
    when "010"  => if (A < B) then -- MIN
      R <= A;       -- A < B
      else          -- A >= B
        R <= B;
      end if;
    when "011"  => if (A < B) then -- MAX
      R <= B;       -- A < B
      else          -- A >= B
        R <= A;
      end if;
    when "100"  => R <= A and B;  -- AND
    when "101"  => R <= A or  B;  -- OR
    when "110"  => R <= A xor B;  -- XOR
    when "111"  => R <= A;        -- PASS A
    when others => R <= "0000";   -- no operation (for simulation)
    end case;
  end process ALU_process;
end ALU_arch;
```