Laboratory 2

2. Extending your design: Seven Segment display

2.1 Objectives

Design, implement and test

- **The Seven Segment Display**
- **A simple Arithmetic Logic Unit (ALU)**

Deeper Knowledge of:

- Xilinx® ISE WebPack CAD tools
- Xilinx® Synthesis Technology (XST) **XST User Guide**
  - **Chapter 2: XST HDL Coding Techniques**
  - **Chapter 6: XST VHDL Language Support**
- Digilent Development Boards **(DDB)**
  - Digilent Basys Board – Reference Manual
  - Digilent Basys 2 Board – Reference Manual
  - Digilent Basys 3 Board – Reference Manual

2.2 4-Digit Seven Segment Display

The development board comes equipped with a 4-digit Seven Segment Display (SSD). This interface uses seven LEDs for each digit; each digit is enabled by an anode signal. All the connections (7 common cathode and 4 distinct anode signals) to the SSD interface are active low. The cathode signals control the LEDs of the digit to be displayed, which is selected by the active anode signal.

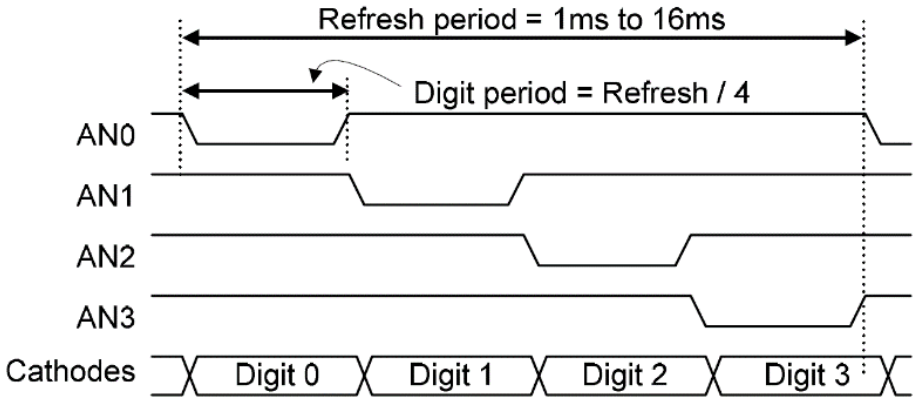

Figure 2-1: SSD Timing Diagram [3], [4], [5]

In order to display 4 different digits on the SSD, you have to implement a circuit that sends the digits to the cathode signals of the SSD according to the timing diagram presented in Figure 2-1. The refresh period is chosen in order to accommodate the human eye (at least 60 Hz refresh rate) with the cycling display of the digits (actually, only one digit is displayed at one time). Read the section about seven segment display methodology in the development boards Reference Manual [3], [4], [5].

The figure below describes a possible implementation of the seven-segment display circuit. The inputs are 4 4-bit signals and the clock signal; the outputs are represented by the anode (an) and cathode (cat) signals (active low).

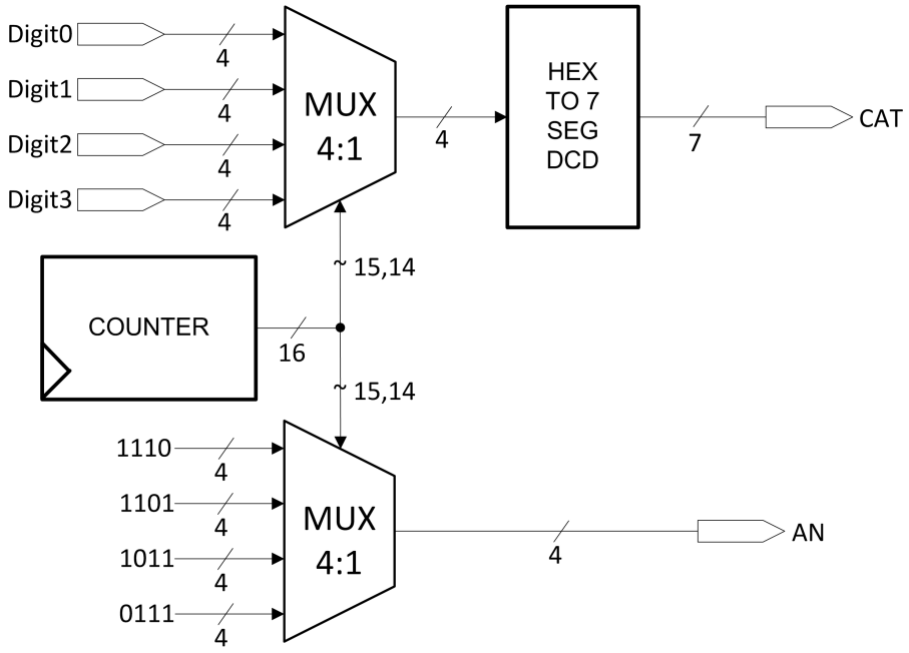

Figure 2-2: Schematic of the SSD circuit

2.3 General laboratory design for the development boards

Once you will complete assignment 2.5.1, all your future designs for this laboratory will resemble the next figure. This design will provide the necessary interfaces with the development boards. All of your circuit descriptions (behavioral or component instantiation) will be placed inside the “cloud”.

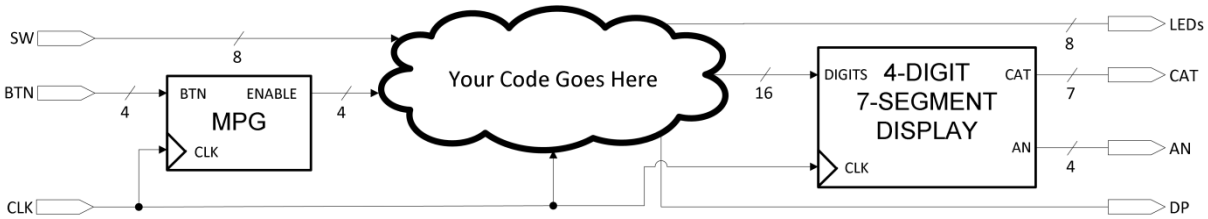

Figure 2-3: Laboratory Design

2.4 Simple ALU operations

Each of the following simple ALU operations can be implemented in VHDL in one line of code. You can also use processes for your implementations.

2.4.1 Adders

An adder is a digital circuit that performs addition of numbers. In modern computers, adders reside in the arithmetic logic unit (ALU) where other operations are performed. The equations for 1-bit full adder are given below:

$$Sum = A\ xor\ B\ xor\ Cin$$
$$Cout = (A\ and\ B) or\ (A\ and\ Cin)\ or\ (B\ and\ Cin)$$

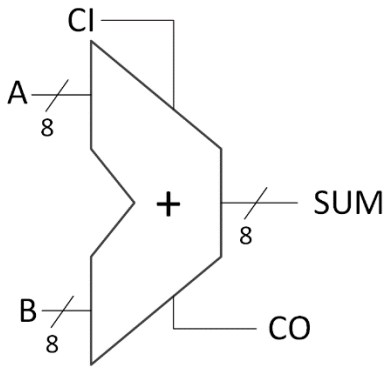

Figure 2-4: 8-Bit Adder with Carry in and Carry out

IO Pins	Description
A, B	Add Operands
CI	Carry In
CO	Carry Out
SUM	Add Result

Table 2.1: 8-Bit Adder with Carry In and Carry Out Pin Descriptions

Implementation in VHDL – Use simple assignment:

```
SUM <= A + B;
```

If the input signals are extended by at least one bit, the Xilinx Synthesizer takes care of the carry input and output signals.

2.4.2 Subtractors

A subtractor is a digital circuit that performs subtraction. In 2's complement subtraction is the same as adding the negative of the number and setting the Carry in to 1.

$$A - B = A + \overline{B} + 1$$

Implementation in VHDL – Use simple assignment:

```
DIFF <= A – B;
```

2.4.3 Shifters

A shifter is a digital circuit that can shift a word of data by a specified number of bits. There are two kinds of shifters:

- logical shifters – value shifted in is always 0
- arithmetic shifters – on right shifts sign extend

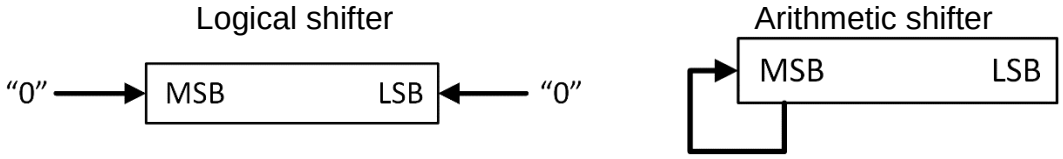

Figure 2-5: Logical and Arithmetic Shift Operations

Xilinx defines a shifter as a combinatorial circuit with two inputs and one output.

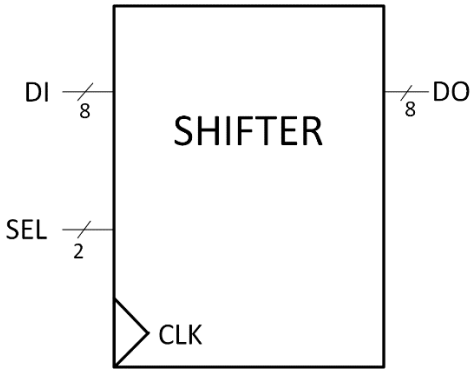

Figure 2-6: Shifter

IO Pins	Description
DI	Data Input
SEL	Shift Distance Selector
DO	Data Output

Table 2.2: Shifter Pin Description

Implementation in VHDL – For simple shifters when the shift amount is fixed use signal concatenation. Example << 2:

```
DO(7 downto 0) <= DI(5 downto 0) & "00";
```

A possible implementation for a combinational shifter with a variable shift amount (SEL) is presented in appendix 3.

2.4.4 Zero Detector

The n-bit zero detector is an n-bit input, 1-bit output NOR circuit, used in Arithmetic/Logic units especially for condition detection for Branch on Equal instructions.

Implementation in VHDL: use a simple line of code (mux 2:1 type):

Zero <= ‘1’ when DI=0 else ‘0’;

2.4.5 Sign/Zero Extender

The Sign and Zero extender for MIPS CPU are used in arithmetical/logical operations in which the 16-bit immediate value is implied. The extension is necessary because the ALU works on 32-bit operands, and the other involved operand has 32 bits.

Example: ADDI (ADD Immediate) instruction, memory access, branch address computation, logical operations with immediate, etc.

Implementation in VHDL: Use concatenation with zeros for the Zero Extension circuit and with the Sign Bit for the Sign Extension circuit.

2.4.6 Comparators

A comparator is a digital circuit that compares two numbers in binary form and generates a one or a zero at its output depending on whether they are the same or not. A comparator can be simulated by subtracting the two values (A & B) in question and checking if the result is zero.

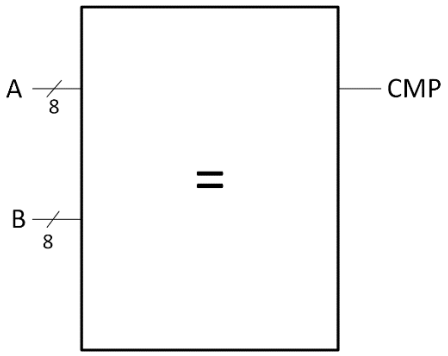

Figure 2-7: Unsigned 8-Bit Equal Comparator

IO Pins	Description
A, B	Comparison Operands
CMP	Comparison Result

Table 2.3: Unsigned 8-Bit Greater or Equal Comparator Pin Description

Implementation in VHDL – ALU performs subtraction between the two operands and the Zero Detection circuit is used over the ALU result. Another approach is to use the implementation provided in the Zero Detector:

CMP <= ‘1’ when A = B else ’0’;

2.5 Laboratory Assignments

2.5.1 4-Digit Seven Segment Display

You have to work in the previous design from laboratory 1 (*test_env* project). Design and implement a new component (a separate entity) for the 4-digit seven-segment display interface (see Figure 2-2 for details). **Use only processes to implement the counter and the two multiplexers, and use the Language Templates to implement the “HEX TO 7 SEG DCD” (the component that transforms a 4-bit hexadecimal digit to its corresponding seven-segment LEDs representation).**

Declare the component in the *test_env* project and instantiate it in the *test_env* architecture. Connect a 16-bit counter to the 4 input data signals of the seven-segment display. The counter is incremented when a button is pressed (use the MPG component to validate the increment of the counter). Your final design must resemble Figure 2-3.

2.5.2 Simple Arithmetic Logic Unit Circuit

Using the previous design (containing the MPG and the SSD), implement a simple ALU circuit with the following functions: ADD, SUB, SHIFT LEFT 2, and SHIFT RIGHT 2.

The results (16-bit) of the ALU operations are displayed on the SSD interface (Digit3...Digit0). Use a 2-bit counter (controlled by the MPG) to select the desired ALU operation.

The input operands for the ALU are the switches of the development board. The ALU design is presented in the figure below (describe it in the architecture of the *test_env* entity, no other components are required, use only internal signals, processes, and concurrent assignments).

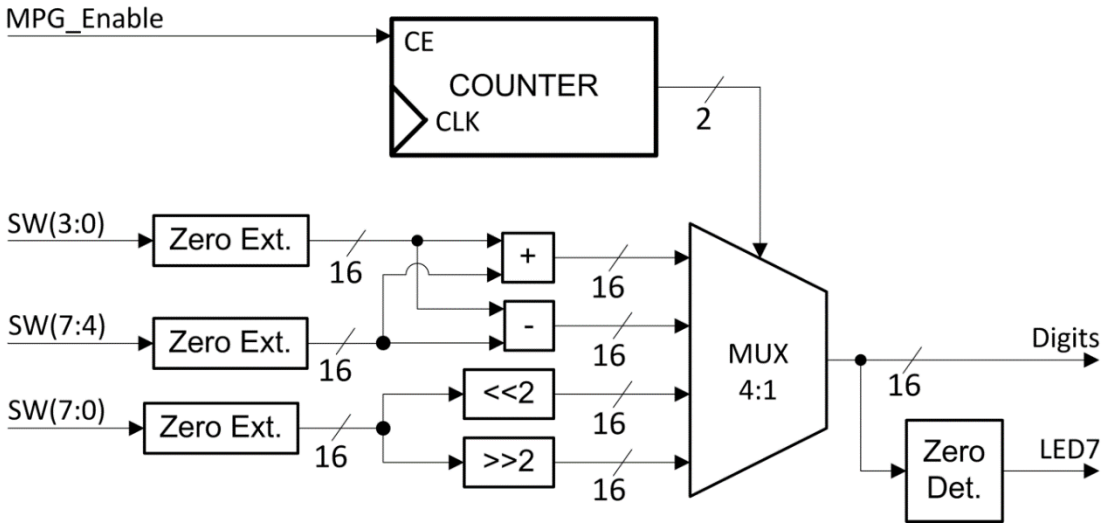

Figure 2-8: Simple ALU design.

2.5.3 Homework: Combinational Shifter with Variable Shift Amount

Read appendix 3 at the end of this laboratory guide. Draw (paper and pencil) the diagram of this shifter (described in the example from the appendix). Implement the circuit on the development board.

2.6 References

[1] XST User Guide, Chapter 2: XST HDL Coding Techniques
[2] XST User Guide, Chapter 6: XST VHDL Language Support
[3] Digilent Basys Board – Reference Manual
[4] Digilent Basys 2 Board – Reference Manual
[5] Digilent Basys 3 Board – Reference Manual