ENGG2410: Digital Design

Lab 7: Data Path Design "Arithmetic Logic Unit"

via *VHDL*

Shawki Areibi
School of Engineering, University of Guelph
Fall 2023

Start Date: *Week#10 2023*
Duration: 2 Weeks
DEMO: *Week#11*
Report Due Date: *Week#12, In the Lab*

1 Objectives:

- Understand the steps involved for designing the "Data Path" of a digital system.
- Implement the Arithmetic Logic Unit (ALU) of a simple CPU.
- Emphasize the importance of modular and hierarchical design.

2 Introduction

A digital system is a sequential circuit made up of interconnected flip-flops and gates. To specify a large digital system with state tables is very difficult, if not impossible, because the number of states is prohibitively large. To overcome this difficulty, digital systems are designed using a modular, hierarchical approach. The system is partitioned into modular subsystems, each of which performs some functional task. The modules are constructed hierarchically from functional blocks such as registers, counters, decoders, multiplexers, buses, arithmetic elements, and primitive gates.

In most digital system designs, we partition the system into two types of modules: *a data-path*, which performs data-processing operations, and a *control unit*, which determines the sequence of those operations. Data-paths are best defined by their registers and operations that are performed on binary data stored in the registers. Instead of having each individual register perform its micro-operations directly, computer systems often employ a number of storage registers in conjunction with a shared operation unit called an *arithmetic/logic unit* i.e ALU.

3 The Arithmetic/Logic Unit

The Arithmetic Logic Unit (ALU) is a combinational circuit that performs a set of basic arithmetic and logic micro-operations. The ALU has a number of selection lines used to determine the operation to be

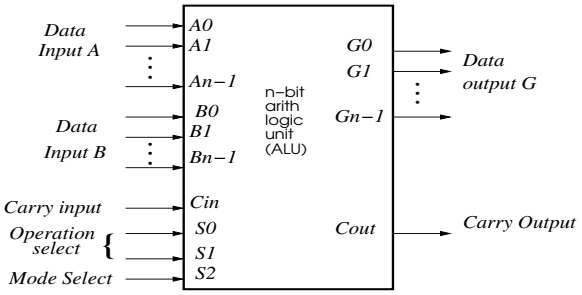

Figure 1: Symbol for an n-bit ALU.

performed. The selection lines are decoded within the ALU, so that k selection lines can specify up to 2^k distinct operations. Figure 1 shows the n-bit ALU. The n data inputs from **A** are combined with the n data inputs from **B** to generate the result of an operation at the **G** outputs. The mode-select input S_2 distinguishes between arithmetic and logic operations. The two function-select inputs S_1 and S_0 specify the particular arithmetic or logical operations to be performed. With three selection lines, it is possible to specify four arithmetic operations with S_2 at 0 and four logic operations with S_2 at 1. The input and output carries have meaning only during an arithmetic operation.

We will perform the design of this ALU (using VHDL) in three stages. First we design the arithmetic section. Then we design the logic selection, and finally, we combine the two sections to form the ALU.

3.1 Arithmetic Circuit

The basic component of an arithmetic circuit is a parallel adder, which is constructed with a number of full-adder circuits connected in cascade. By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations. The block diagram in Figure 2 demonstrates a configuration in which one set of inputs to the parallel adder is controlled by the select lines S_1 and S_0. There are n bits in the arithmetic circuit, with two inputs **A** and **B** and output **G**. The n inputs from **B** go through the **B** input logic to the **Y** inputs of the parallel adder. The input carry C_{in} goes in the carry input of the full adder in the least significant bit position. The output carry C_{out} is from the full adder in the most significant bit position. The output of the parallel adder is calculated from the arithmetic sum $G = X + Y + C_{in}$ where X is the n-bit binary number from the inputs and Y is the n-bit binary number from the **B** input logic. C_{in} is the input carry, which equals 0 or 1. **See section 7-7 in your text book (or lecture notes) for more details**.

3.1.1 Design of the B-input Logic

Table 1 shows the arithmetic operations that are obtainable by controlling the value of **Y** with the two selection inputs S_1 and S_0.

1. If the inputs from **B** are ignored and we insert all 0's at the Y inputs, the output sum becomes $G = A + 0 + C_{in}$. This gives G = A when $C_{in} = 0$ and G = A + 1 when $C_{in} = 1$. In the first

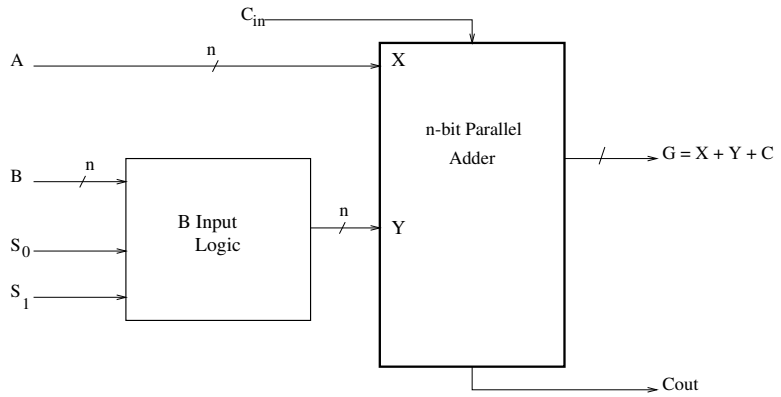

Figure 2: Block Diagram of an Arithmetic Circuit.

Select		**Input**	G=A/B/C_{in}	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0's	G = A (transfer)	G=A+1 (increment)
0	1	B	G = A + B (add)	G=A+B+1
1	0	$\bar{B}$	G = A + $\bar{B}$	G=A+$\bar{B}$+1 (subtract)
1	1	all 1's	G = A - 1 (decrement)	G=A (transfer)

Table 1: Function Table for Arithmetic Circuit

case, we have a direct transfer from input **A** to output **G**. In the second case, the value of **A** is incremented by 1.

2. For a straight arithmetic addition, it is necessary to apply the **B** inputs to the Y inputs of the parallel adder. This gives G = A+B when C_{in}=0.

3. Arithmetic subtraction is achieved by applying the complement of inputs B to the Y inputs of the parallel adder, to obtain $G = A + \bar{B} + 1$ when C_{in}=1. This gives A plus the 2's complement of B, which is equivalent to 2's complement subtraction.

4. All 1's is the 2's complement representation for -1. Thus applying all 1's to the Y inputs with C_{in}=0 produces the decrement operation G=A-1.

To design a one stage arithmetic circuit you will need to do the following:

- Implement the B-input Logic using a separate VHDL file.
- Implement a full adder (i.e. 1-bit) using a second VHDL file.
- A single stage arithmetic unit would simply consist of a full adder and the B-input logic. Declare each as a component in the third VHDL file and use structural declaration to connect the two into a single module.

3.2 One Stage of Logic Circuit

The logic micro-operations manipulate the bits of the operands by treating each bit in a register as a binary variable, giving bit-wise operations. There are four commonly used logic operations – AND, OR, XOR and NOT – from which others can be conveniently derived. A one stage logic circuit can

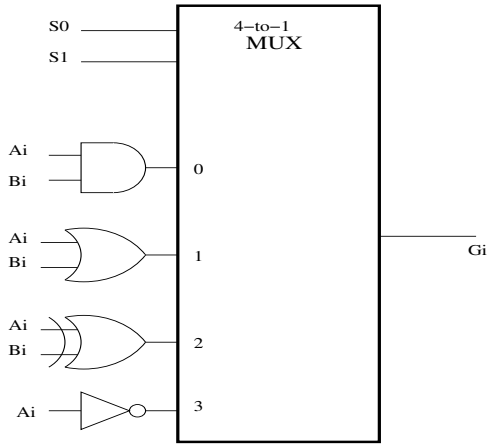

Figure 3: One Stage of Logic Circuit.

be obtained by using a multiplexer and four gates. Each of the four logic operations is generated through a gate that performs the required logic. The outputs of the gates are applied to the inputs of the multiplexer with two selection variables S_0 and S_1. These choose one of the data inputs of the multiplexer and direct its value to the output. Figure 3 shows a one stage possible implementation of the Logic Circuit. To design a one stage logic unit you will need to do the following:

- Implement the circuit shown in Figure 3 using VHDL in a separate file.

3.3 Arithmetic/Logic Unit (One Stage)

The logic circuit can be combined with the arithmetic circuit to produce an ALU. Selection variables S_1 and S_0 can be common to both circuits, provided that we use a third selection variable to differentiate between the two. The configuration for one stage of the ALU is illustrated in Figure 4. The outputs of the arithmetic and logic circuits in each stage are applied to a 2-to-1 multiplexer with selection variable S_2. When S_2=0, the arithmetic output is selected, and when S_2=1, the logic output is selected. Note that the diagram shows just one typical stage of the ALU; the circuit must be repeated n times for an n-bit ALU.

To design a one stage ALU you will need to do the following:

- You have to write the VHDL code of a one stage ALU. The code utilizes two components i.e.,
    1. 1-stage arithmetic unit designed earlier.
    2. 1-stage logic unit designed earlier.

Once you successfully design and test the one stage ALU, declare it as a component that can be used later in an n-bit ALU.

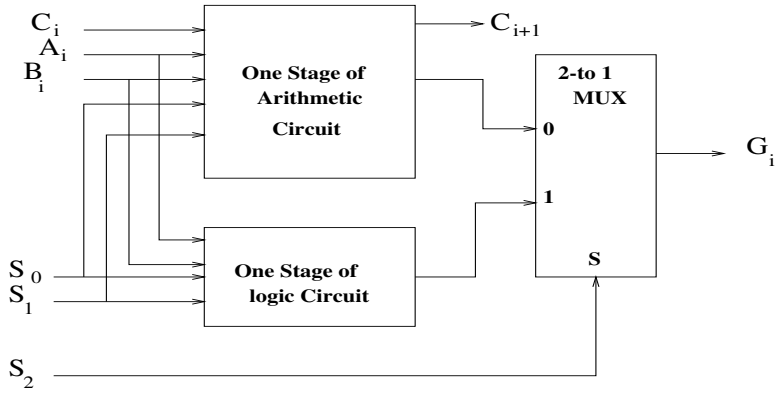

Figure 4: One Stage of ALU.

S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	G = A	Transfer A
0	0	0	1	G = A+1	Increment A
0	0	1	0	G = A+B	Addition
0	0	1	1	G = A+B+1	Add with carry input of 1
0	1	0	0	$G = A + \bar{B}$	A plus 1's complement of B
0	1	0	1	$G = A + \bar{B} + 1$	Subtraction
0	1	1	0	G = A - 1	Decrement A
0	1	1	1	G = A	Transfer A
1	0	0	X	$G = A \wedge B$	AND
1	0	1	X	$G = A \vee B$	OR
1	1	0	X	$G = A \oplus B$	XOR
1	1	1	X	$G = \bar{A}$	NOT (1's complement)

Table 2: Function Table for ALU

4 4-bit ALU

The ALU provides eight arithmetic and four logic operations. Each operation is selected through the variables S_2, S_1, S_0 and C_{in}. Table 2 lists the ALU operations. The first 8 are arithmetic operations and are selected with $S_2 = 0$. The next 4 are logic operations and are selected with $S_2 = 1$. The input carry C_{in} has no effect during the logic operations and is marked with X to indicate that its value may be either 0 or 1.

Use the one stage ALU component to create a 4-bit ALU.

5 Requirements

- **Design and implement** the single stage of the arithmetic unit (in VHDL), simulate it using the Xilinx Foundation tools.
- **Design and implement** the single stage of logic unit (in VHDL).

- **Combine** the one stage arithmetic unit and logic unit into a single stage arithmetic logic unit (in VHDL).
- **Declare** the one stage ALU as a component that can be used in building a 4-bit ALU circuit using structural VHDL.
- **Test** the *complete* 4-bit ALU circuit as explained in the preparation section and map it on the NEXYS 3 FPGA board.
- **Connect** a 7-segment decoder (you have built in previous labs) to the output of the ALU and demonstrate several operations for the ALU.

6 Deliverables

1. **Lab Preparation File**:
   - Name your file as follows: ENG2410_F23_LAB7_Section(Mon,Tue,Thu,Fri)_Group#_Preparation.pdf
   - The preparation file can be hand written or typed. But has to be neat.
   - The preparation file should consist of the Truth Table, Boolean Equations, Block Diagram, circuit diagram and partial VHDL Code.
   - Only one submission is allowed. Multiple submissions will not be marked.
2. **Lab Final Report File**:
   - Name your file as follows: ENG2410_F23_LAB7_Section(Mon,Tue,Thu,Fri)_Group#_FinalReport.pdf
   - Check the format of the final report below in the next section.
   - Keep the first page only for the title page.
   - Only one submission is allowed. Multiple submissions will not be marked.
3. **Zipped Project File**:
   - Name your file as follows: ENG2410_F23_LAB7_Section(Mon,Tue,Thu,Fri)_Group#_Project.zip
   - The project file should have a README file that explains the switches and LEDs you used in your design.
4. **DEMO**:
   - You will need to demonstrate to the Teaching Assistant that your design fulfills the requirements.
   - The DEMO will take place during the LAB hours and not outside these hours.

7 Report

Below is the general format of the report required:

1. Title Page:
   - Course #, and Date
   - Lab # and name of experiment
   - Your Group #, and Names
2. Start a new page and explain how you implemented your design by providing the following:
   (a) **Problem Statement**,
      i. Briefly describe the problem solved in the lab.
   (b) **Assumptions and Constraints**.
      i. Constraints could be for example using only NAND gates or NOR gates in your design.
   (c) **How you solved the Problem**,
      i. Truth Table.
      ii. Boolean Equations.
      iii. Circuit Diagram.
   (d) **VHDL Code**,
      i. List the VHDL code that you have written.
      ii. Make sure that the code has comments and documented.
   (e) **System Overview & Justification of Design**
      i. Give an overview of the system to be designed.
      ii. Briefly explain how the system works and reasons behind the design.
3. **System Design**
   (a) Block Diagram.
   (b) Hierarchical Design Approach.
4. **Circuit Diagram**
   (a) Brief explanation of the hardware.
   (b) The schematic you produced using Xilinx ISE tools.
5. **Error Analysis**
   (a) Printed **Simulation** waveform data, showing that the simulation coincides with the expected truth table of your function.
   (b) Include the VHDL **Test Bench** in an Appendix.
   (c) Describe any problems with the system.
   (d) If no problems in the final system, describe problems/errors encountered during the development and how they were resolved.