## LAB 3                    Arithmetic Logic Unit

### 1. Purpose:

In this lab students will design, simulate, build and test an Arithmetic Logic Unit (ALU), employing Quartus II as a development environment and the Altera DE2-115 board as experimental platform. ALU has to execute 16 different operations on two operands of 4 bits and will provide a 4-bit result along with 4 status bits (oVerflow, Zero, Negative, Carry). The input operands will be generated by slide switches (or by "input_generator" component if you are testing your design remotely), while the result and the status bits will be displayed on LEDs.

### 2. Requirements of the Lab:

The following will be submitted in your report.
  * Functional and truth tables, equations and schematics of your design
  * Log of what you did
  * Screen shots of all schematics and waveform diagrams
  * Compilation, simulation and downloading messages (if any)
  * Test results

### 3. Equipment and Supplies:

  * Quartus II (student edition or web edition)
  * Altera DE2-115 board with USB-Blaster cable and Power supply 12 VDC, 2A

### 4. References:

4.1. Chapter1 - 4 of the Textbook: *Computer System Architecture*, Morris Mano, 3rd Ed.

4.2. Course notes and

4.3. *DE2-115 User Manual* posted in the *Documentation* section under the *Laboratories* tab of CEG2136 *Virtual Campus*.

### 5. PreLab – Design of the ALU

#### 5.1. ALU structure

The Central Processing Unit (CPU) consists of a Control Unit (CU) and a Datapath (execution unit – EU) as shown in the block diagram of Figure 1.
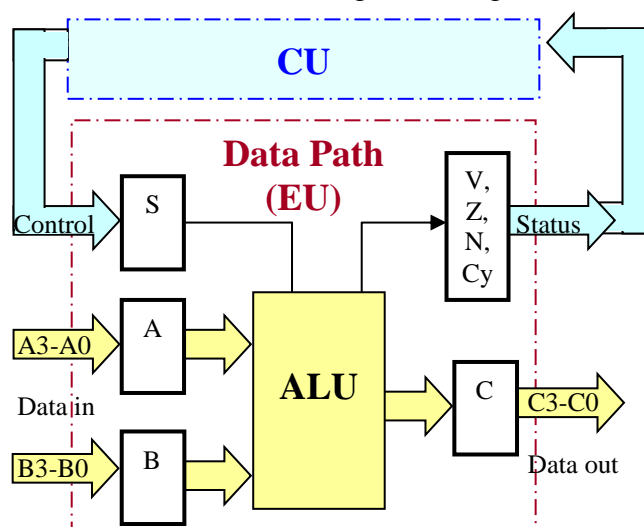


**Figure 1: ALU block diagram**

The CPU's datapath contains registers to store data (A, B, C) and control (S) / status (V,Z,N,Cy) information, along an Arithmetic and Logic Unit (ALU). You have to design CPU's datapath that can perform arithmetic, shift and logic operations. Your datapath module will handle two types of signals:

- Data (inputs: A3-A0 and B3-B0, and outputs C3-C0) – marked in yellow
- Control (operations codes) / status: S3-S0 / (V, Z, N, Cy) – marked in light blue in Fig. 1

Figure 2(a) presents the ALU datapath. For testing purposes, since CU (the light-blue block in Figure 2) is not designed and built in this lab, CU is replaced and emulated by 4 DIP switches for generating the ALU control inputs (S3 - S0) and 4 LEDs to display the ALU's state indicators (V,Z,S,O). Operands A and B are generated manually by 8 DIP switches, while the result C is displayed by other 4 LEDs.
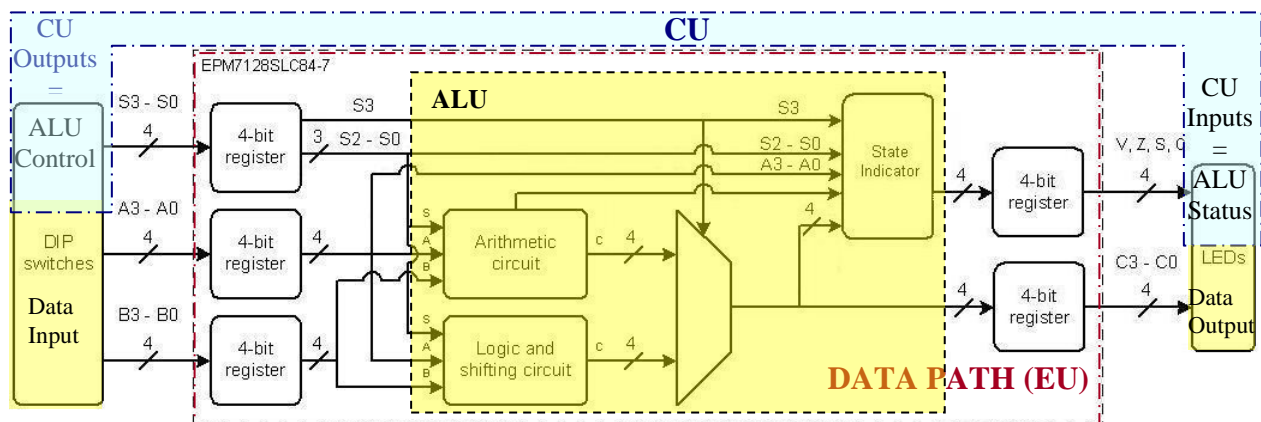


**Figure 2(a): ALU datapath**

**Note: This term we will not use DIP switches to generate the test inputs (A, B) and control signals (S) for experimental verification of your ALU. Instead, we are going to use a pre-designed component named "input_ generator" which generates sequences of test vectors A, B, S. Figure 2(b) shows the block diagram of the datapath that we will use for this purpose.**
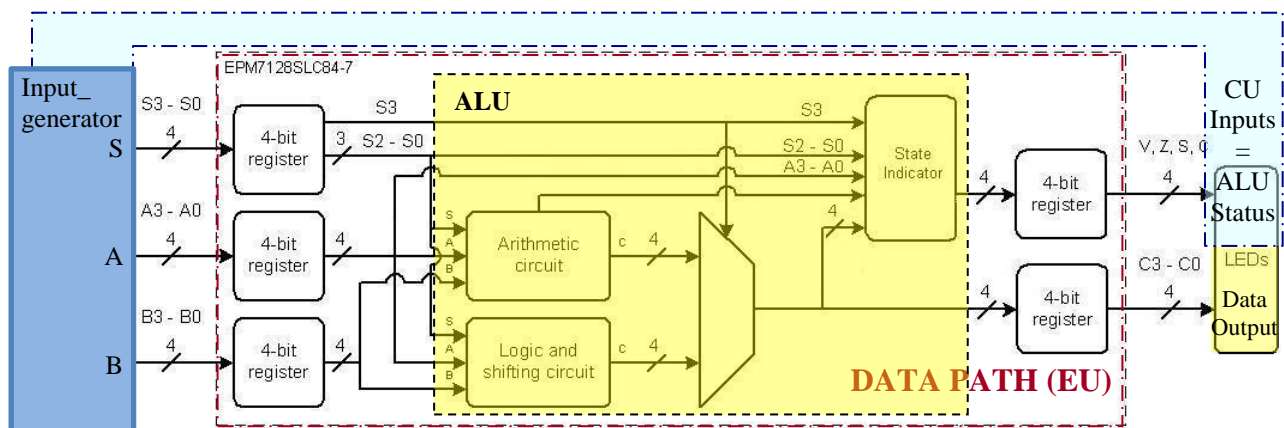


**Figure 2(b): ALU datapath**

ALU is a <u>combinational circuit</u> which can execute any of the 16 micro-operations of Table 1. The operation to be performed is set by the *control word* (*operation code*) S3-S0 as explained below:
- the control signal S3 selects either the Arithmetic Circuit (AC) or the Logic and Shifting Circuit (LSC) to perform the operation and send the result to the output C;
- the control signals S2, S1, and S0 define the operations to be performed by each circuit (AC or LSC).

Both inputs and outputs are stored in buffer registers (D flip-flops synchronized with the same clock), A, B, and C, as shown in Figure 2. To operate at the highest speed, the ALU datapath should be provided with both operands (A and B) and control bits (S3-S0) at every consecutive clock. To insure a correct operation of the datapath, the total delay introduced by the ALU's circuits should be smaller than the clock's period and, as such, the datapath can execute a micro-operation at every clock. The results (C3-C0) and V, Z, S, Cy are provided on the next pulse following the change of A, B and S.

**Table 1: arithmetic and logic micro-operations**

| Circuit | Control Word S3 S2 S1 S0 | ALU Data Output | Micro-operation Description |
|---|---|---|---|
| Arithmetic circuit (AC) | 0  0  0  0 | $C \leftarrow A + B$ | Addition |
| | 0  0  0  1 | $C \leftarrow A + B + 1$ | Add with carry |
| | 0  0  1  0 | $C \leftarrow A$ | Transfer A |
| | 0  0  1  1 | $C \leftarrow A + 1$ | Increment A |
| | 0  1  0  0 | $C \leftarrow A + \overline{B}$ | Subtraction A - B with borrow (using 1's complement of B gives a result lower by 1 than a conventional subtraction). |
| | 0  1  0  1 | $C \leftarrow A + \overline{B} + 1$ | Subtraction A - B (use 2's complement of B) |
| | 0  1  1  0 | $C \leftarrow \overline{A}$ | NOT A (complement A) |
| | 0  1  1  1 | $C \leftarrow \overline{A} + 1$ | 2's complement of A |
| Logic and shift circuit (LSC) | 1  0  0  0 | $C \leftarrow$ "0000" | Reset C |
| | 1  0  0  1 | $C \leftarrow$ "1111" | Set C |
| | 1  0  1  0 | $C \leftarrow A \wedge B$ | A AND B |
| | 1  0  1  1 | $C \leftarrow A \vee B$ | A OR B |
| | 1  1  0  0 | $C \leftarrow A \oplus B$ | A EXCLUSIVE-OR B |
| | 1  1  0  1 | $C \leftarrow A \wedge \overline{B}$ | Reset A bits selected by "mask" B |
| | 1  1  1  0 | $C \leftarrow a\text{shl } A$ | Shift A left (signed multiplication by 2) |
| | 1  1  1  1 | $C \leftarrow a\text{shr } A$ | Shift A right (signed division by 2) |

### 5.2. **Hierarchical design**

Since both the input and the output signals of ALU have 4 bits, the most effective way to implement ALU is to devise it in a hierarchical manner. Files on the lowest level are 1-*bit full adder* and 1-*bit logic and shift circuit* (LSC). The intermediate files will consist of a 4-*bit register*, a 4-*bit arithmetic circuit* (AC), a 4-*bit logic and shift circuit* (LSC), and a *state circuit*. Finally, the file at the highest level will cover the complete circuit, loadable to the Altera DE2-115 board. The relation between these files is shown in figure 3. The parts in yellow will be explained further in this document. For the preparation of this lab, you are required to design and test the circuits coloured in green, i.e., the 1-bit LSC, the 4-bit LSC, the 4-bit AC, and the state circuit.
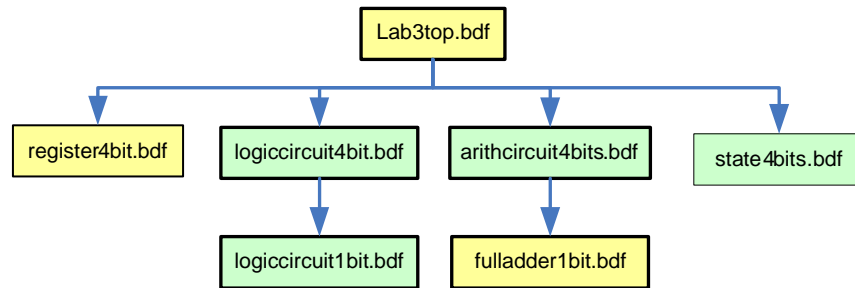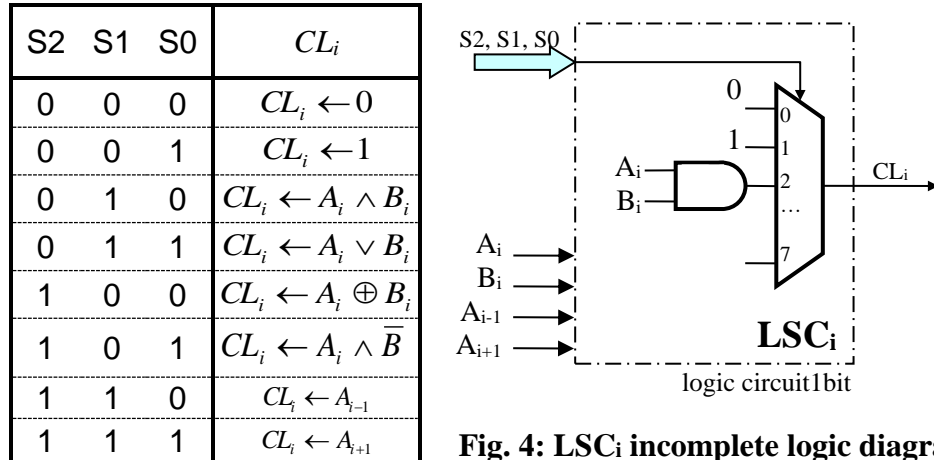
**Figure 3:  Hierarchy of the files**

### 5.3. Design of Logic and Shift Circuit (LSC)

LSC operates at the bit level, which means that, e.g. C = A AND B is interpreted as {C3, C2, C1, C0} = {A3 AND B3, A2 AND B2, A1 AND B1, A0 AND B0}.

Design a circuit which implements all logic and shift functions as shown in the bottom half of Table 1 (with S3=1). LSC is built from four **1-bit Logic and Shift Circuit (LSC)**.  Such a 1-bit module has 3 control inputs (S2, S1, and S0), two logic inputs ($A_i$ and $B_i$), two additional inputs for left and right shifts ($A_{i-1}$ and $A_{i+1}$), and an output ($CL_i$) whose truth table is detailed below.  A starting point for the logic diagram of **1-bit LSC** is suggested below;

>    5.3.1.   Add all the logic gates that are required to complete the $LSC_i$ logic diagram of Fig. 4.

| S2 | S1 | S0 | $CL_i$ |
|----|----|----|--------|
| 0 | 0 | 0 | $CL_i \leftarrow 0$ |
| 0 | 0 | 1 | $CL_i \leftarrow 1$ |
| 0 | 1 | 0 | $CL_i \leftarrow A_i \wedge B_i$ |
| 0 | 1 | 1 | $CL_i \leftarrow A_i \vee B_i$ |
| 1 | 0 | 0 | $CL_i \leftarrow A_i \oplus B_i$ |
| 1 | 0 | 1 | $CL_i \leftarrow A_i \wedge \overline{B}$ |
| 1 | 1 | 0 | $CL_i \leftarrow A_{i-1}$ |
| 1 | 1 | 1 | $CL_i \leftarrow A_{i+1}$ |



**Fig. 4: $LSC_i$ incomplete logic diagram**

>    5.3.2.   Use the 1-bit LSC module that you designed above to <u>build a **4-bit logic and shift circuit LSC**</u>. Your final design should have 3 control inputs (S2 - S0), 8 data inputs (A3 - A0 and B3 - B0), and 4 data outputs (CL3 - CL0). Your design will be captured in the *logiccircuit4bit* file.

### 5.4. Design of Arithmetic Circuit (AC)

>    5.4.1.   The block diagram of the Arithmetic Circuit (AC) is shown in Fig. 5. AC consists of a **4-bit full adder** ($\sum$) which calculates the output CA as a function of operands op1 and op0, and of the micro-operation select bits S2, S1 and S0, as described in Table 1.

Complete the following truth table with the corresponding values derived from Table 1.

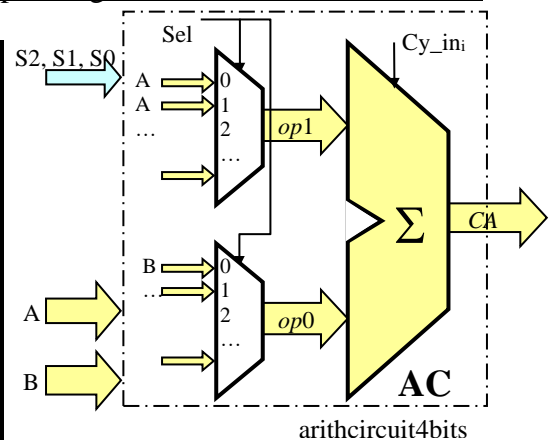| S2 | S1 | S0 | op1 | op2 | Cy_in | CA output |
|----|----|----|-----|-----|-------|-----------|
| 0  | 0  | 0  | A   | B   | 0     | $CA \leftarrow A + B$ |
| 0  | 0  | 1  | A   | B   | 1     | $CA \leftarrow A + B + 1$ |
| 0  | 1  | 0  | A   | 0   | 0     | $CA \leftarrow A$ |
| 0  | 1  | 1  |     |     |       | $CA \leftarrow A + 1$ |
| 1  | 0  | 0  |     |     |       | $CA \leftarrow A + \overline{B}$ |
| 1  | 0  | 1  |     |     |       | $CA \leftarrow A + \overline{B} + 1$ |
| 1  | 1  | 0  |     |     |       | $CA \leftarrow \overline{A}$ |
| 1  | 1  | 1  |     |     |       | $CA \leftarrow \overline{A} + 1$ |



**Fig. 5: AC block diagram**

5.4.2. Note that the data inputs A3 - A0 and B3 - B0 should not be connected directly to the inputs of multiplexers for op1 and op2, since you might need some minimal logic to insure full compliance with the truth table. A trivial solution is to implement op1 and op0 with 8-to-1 multiplexers. Start your design by analyzing what inputs are to be brought to op1 and op0 for each combination of S2 and S1, only.
Derive a cost effective solution from this analysis using a 4-to-1 multiplexer.

5.4.3. Derive the equation of Cy_in in terms of the select bits S2, S1 and S0.

5.4.4. Complete the block diagram of ACi of Fig. 6 with logic components that implement the conclusions you drew above (5.4.1 - 5.4.2):

- replace the multiplexor symbols with the ones resulted at 5.4.2;
- add more wires and other logic components between module's inputs ($A_i$, $B_i$ , S2, S1 and S0) and multiplexers' inputs to implement the AC table;
- implement Cy equation.

This $AC_i$ schematic will be later captured in the *fulladder1bit.bdf* file in Quartus II.

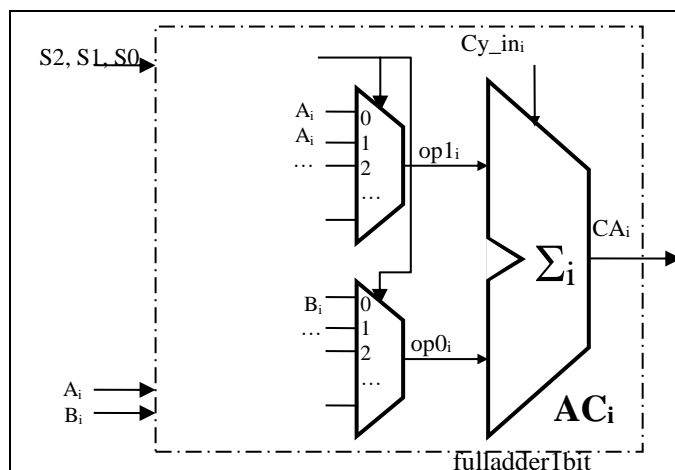Note: $\sum_i$ is based on the *full adder* shown in Figure 9.



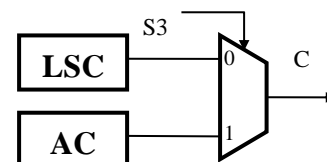**Fig. 6: $AC_i$ incomplete logic diagram**

**Fig. 7: ALU Data Output**

The ALU Data Output is provided by AC or LSC as dictated by the control signal S3. The block diagram of Fig. 7 employs 2-to-1 multiplexers to select AC or LSC to be loaded into register C.

Your final circuit of the ALU data flow should have 3 control inputs (S2 - S0), 8 data inputs (A3 - A0 and B3 - B0), 4 data outputs (C3 - C0) and the most significant carry bits, which will be used for the status circuit.

### 5.5. **ALU status register**

The ALU results are characterized by four status bits (C, S, Z, and V), which are defined as follows:

- Bit Cy (carry) is set to 1 only when the operation is an arithmetic operation and its carry output is 1. It is cleared to 0 if the carry is 0.
- Bit S (sign) is set to 1 if the most significant bit of the result C3 is one.
- Bit Z (zero) is set to 1 only if the output of the ALU contains all 0's. It is set to 0 otherwise.
- Bit V (overflow) is set to 1 only when an overflow occurs when performing operations on signed numbers in 2's complement representation.

Derive the logic equations of the status bits and implement them with conventional gates.

## 6.  **Procedure**
## **PART I – Design**

**6.1.** Use Quartus II to capture the following graphic file of Fig. 8, which implements a 4-bit register. Use D-type flip-flops (symbol "dff"). Save your file under the name "register4bits.bdf" (using standard names will help your TAs to check your circuit if needed).
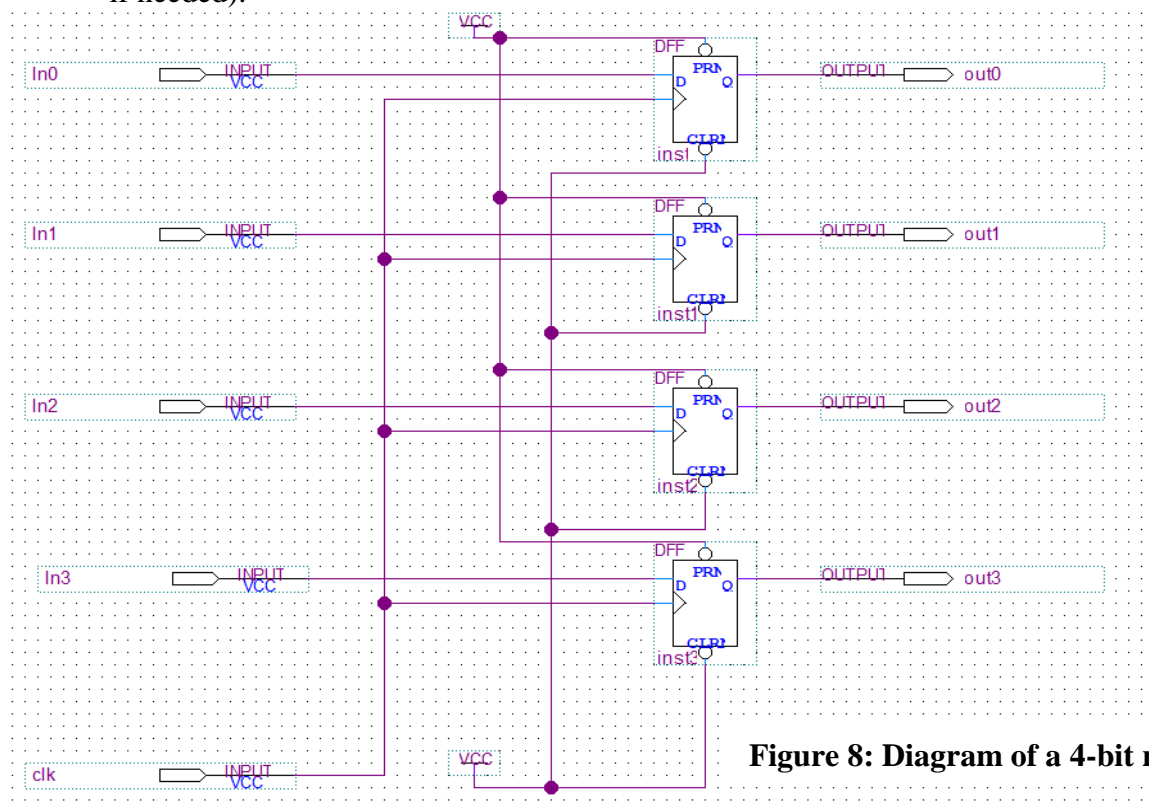


**Figure 8: Diagram of a 4-bit register**

---

6.2.  Choose *Set as Top Level Entity* (assign the open file to the project), select **EP4CE115F29C7** and make sure you compile your file without errors (you do not need to assign pines at this stage).

Go to the section *Fitter Summary* to check the percentage of the resources of the chip that are used by the file "*register4bits.fit.rpt*". Create a symbol for the register which could be used later on.

6.3. Use the graphic editor of Quartus to draw the circuit diagram of your 1-*bit Logic and Shift Circuit* (*LSC*) into a schematic file and save it as "logiccircuit1bit.bdf".

To enter the symbol of multiplexer *74151* make a right-click, select *Insert >> Symbol*, and type "74151" in the dialogue window *Name*. **If you have decided to use the multiplexer lpm_mux (to obtain bonus points), the block symbol can be created by launching the MegaWizard Plug-In Manager under the Symbol window. Select "*Create a new custom megafunction variation*", specify the output files name and proceed to the on-screen instructions.** If the multiplexer does not appear in the form that you prefer, you can make a right click, and you can reverse it horizontally or vertically. Assign the file open to the project, compile the file, and create a symbol.

6.4. Capture your 4-bit logic and shift circuit LSC to a file which you will call "*logiccircuit4bits.bdf*". Assign the open file to the project, compile it, and create a symbol. Which percentage of the resources of the chip do you use for this file?

6.5. Capture the full adder of Figure 9 in a file called "*fulladder1bit.bdf*." Add the other logic components that are needed to implement the logic circuit ACi that you developed from Fig. 6. Repeat the necessary steps to create the corresponding symbol.
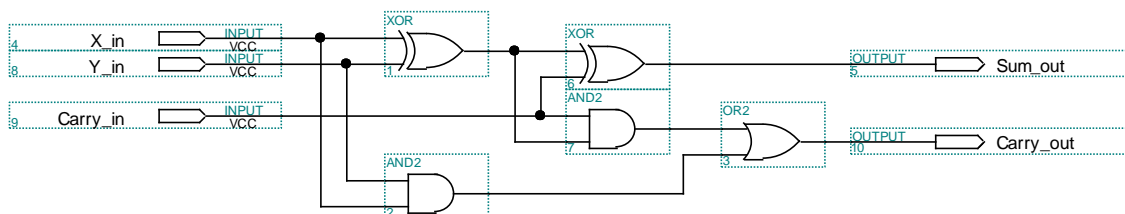


**Figure 9: Logic diagram of a full adder** (marked $\sum$ in Fig. 6)

6.6. Use the graphic editor of Quartus to draw the circuit diagram of your **4-bit full adder (AC)** into a schematic file and save it as "arithcircuit4bits.bdf". Repeat the necessary steps to create the corresponding symbol. What percentage of the resources of the chip you use for this file? Is the *arithmetic circuit* more complex than the logic and shift circuit?

6.7. Enter the ALU status circuit (called "State Indicator" in the diagram of Fig. 2) in a graphic file named "state4bits.bdf". Repeat the necessary steps to create its symbol.

6.8. Combine your registers (A, B, C), arithmetic circuit (AC), logic and shift circuit (LSC) and the status indicator in a top-level graphic file (named Lab3top.bdf in Fig. 3.) to synthesize the ALU datapath as presented in Fig.2. (The top level diagram of a similar but different ALU is presented as an example in Figure 15-annexed at the end of this document).

**NOTE: For experimental verification with physical presence in the lab, skip to page 12 with Sections II.2 and III.2**

**Otherwise, for online access and remote experimental verification just continue with the following, including Sections II.1 and III.1**

## Remote experimental verification

Your ALU will be tested by simulation and experimental means. The test vectors (operations' codes S, as well as the operands A and B) are generated by "input_generator" modules as shown in Figure 2b and they are provided. The operands' values (A and B) generated by this module may differ from one team to another and are determined by a 3 bit code (code 2, code1, and code 0) which you have to apply to your "input_generator," as explained next.

Code 0, code1 and code 2 are determined by calculating the remainder of division of your team number by 5 (i.e., team number %5) and converting it to binary format. For example, if you are in group 23, the result is 3. By converting 3 to binary we get 011. As a result, code2=0, code1=1, code0=1. You will apply these values to the inputs of the component using "vcc" and "gnd" which can be fount in **Insert > symbol > primitives > other.**

Next, connect the outputs A, B, and S of the input_generator component to their respective inputs of the 4bit registers so that they can be used as the inputs to the ALU. Figure 10 demonstrates how you can use the input generator component. Code 011 has been used as the input (teams 3,8,13, …) in this example.
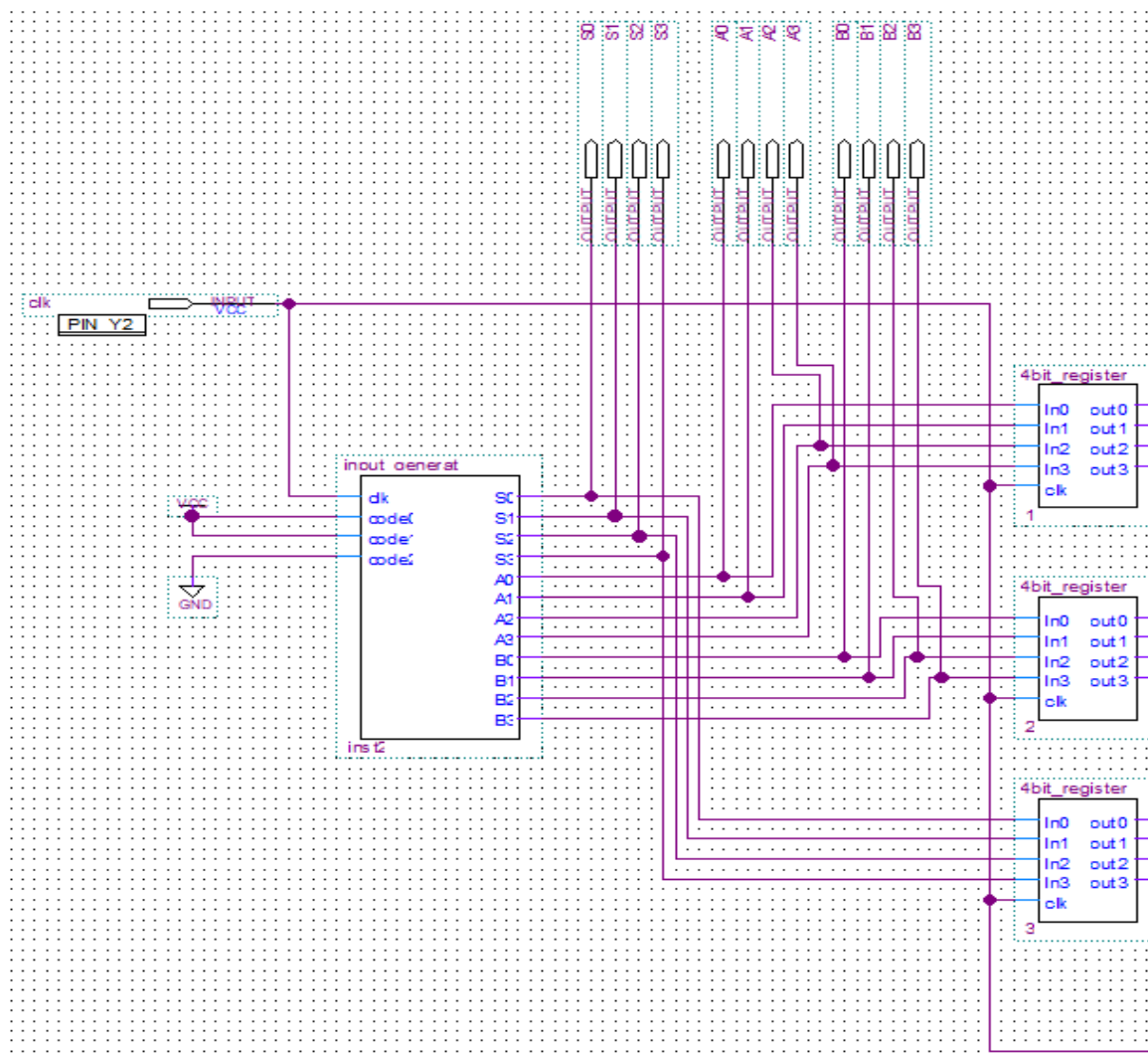


Figure 10: How to connect the input_generator component to 4-bit registers

The "input_generator_high_freq" changes its outputs every approximatively 240 ns and will be used for simulations at high frequency. For experimental tests, "input_generator_low_freq" will be used since its test vectors change every second, such that one can observe the values S, A, B and C with naked eyes.

## PART II.1 - Simulation

6.9.  Download the input_generator files (a total of 4) from BrightSpace and add them to project just like you did for lab 1 and lab 2.
These files are input_generator_high_freq.vhd, input_generator_high_freq.bsf, input_generator_low_freq.vhd, and input_generator_low_freq.bsf.
The high frequency component will be used for simulation, while the low frequency component is used for practical testing on the board.
Add the "input_generator_high_freq" component to your design. Compile your design.

6.10.  Create a file of test vectors (*.vwf*) to verify your ALU datapath as follows:

a)  Choose *Edit >> Grid Size…* and enter 50ns. Note that your circuit should function in a satisfactory way with a size of grid of 20ns and a period of clock of 40ns, provided that you devised it correctly; even the most ineffective circuits should have enough time between each clock pulse!

b)  Choose *Edit >> End Time…* and enter 10.0μs.

c)  Choose *Edit >> Insert Node or Bus …* in order to import all the inputs and all the outputs in *.vwf*.

d)  In the (.vwf) file, group A3, A2, A1, A0 together and name it "A". To group A3-A0, first select A3, then press the shift key and click on A0. Then, right click and choose *Group…,* give a name to this group, choose the format of posting into binary, and select *OK*. Repeat the same for B3-B0 and S3-S0, C3-C0 and name them "B", "Select", "Out" respectively. Make sure that A3, B3, S3, C3 are actually the most significant bits. If they are not, right click on the group name and select **Reverse group or bus bit order.**

e)  To check the correctness of your results, you should first fill table 5, knowing that the values of A and B generated by your input_generator_high_freq" remain constant at all times and are determined based on your group number. To find the specific values of A and B for your group, check table 6. You need to calculate the remainder of division of your group number by 5, just like you did earlier. For example, if you are in team 12, then 12%5=2 and A=1010 and B=1100. Put these values for A and B in all of the 16 states of the excitation table. The value of S changes from 0000 (at clock cycle 1) to 1111 (at clock cycle 16). Fill in the empty columns of the Table 5 with the appropriate values of the control signals (S3, S2, S1, S0) and the 2 operands (A and B) that are needed to perform the corresponding RTL micro-operations. Evaluate the result (C) and the ALU state indicators (V,Z,N,Cy) the execution of each micro-operation, and fill out corresponding cells accordingly. Convert these values to hexadecimal ($S_{16}$, $A_{16}$, $B_{16}$, $C_{16}$, $St_{16}$) to simplify comparing with the simulations' results.

| Clock Cycle | A3 | A2 | A1 | A0 | B3 | B2 | B1 | B0 | S3 | S2 | S1 | S0 | C | V,Z,N,Cy | RTL Micro-operation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | |

**Table 5**

| Team_number%5 | A | B |
|---|---|---|
| 0 | 0101 | 0011 |
| 1 | 0110 | 0101 |
| 2 | 1010 | 1100 |
| 3 | 1100 | 0111 |
| 4 | 1101 | 1001 |

**Table 6**

6.11. Start simulation and compare the outputs of your simulated ALU with the calculated values of C and St in your Table 5. **Note that there is a time delay at the ALU output which is caused by the input/output registers!** If some elements of your circuit do not function correctly, it is imperative to correct them at this stage. Finally make a capture of the simulation screen for your lab report.

## PART III.1 - Testing Experiment

6.12. In your top-level entity where you have your final ALU circuit, replace "input_generator_high_freq" component with "input_generator_low_freq" component. "input_generator_low_freq" will generate the inputs at a lower frequency such that you can see the results on the board.

6.13. Use *Assignments >> Pins…* to assign the pins. Use table 7 for pin assignment. After assigning the PINs compile the project.

**Table 7: Pin Assignments for the final circuit**

| Signal | Pin Location | Component |
|--------|--------------|-----------|
| C3 | PIN_F21 | LEDR3 |
| C2 | PIN_E19 | LEDR2 |
| C1 | PIN_F19 | LEDR1 |
| C0 | PIN_G19 | LEDR0 |
| V | PIN_E24 | LEDG3 |
| Z | PIN_E25 | LEDG2 |
| S | PIN_E22 | LEDG1 |
| Cy | PIN_E21 | LEDG0 |
| CLK | PIN_Y2 | 50 MHz oscillator |

6.14.  Select **Tools → Programmer** in the **Quartus II** window.

Select the **Hardware Setup** tab and the **Hardware Setup** box will pop up; in the **Hardware Settings** tab, double click on **USB-Blaster** from the **Available hardware items** table (but do not click **Add Hardware**); click **Close.**

Note: if the USB-Blaster doesn't show-up in the list of **Available hardware items**, close the window and open it again. You might have to repeat this process a few times.

**6.15.**  Make sure **Program/Configure** is checked-in and then Click **Start.**

**6.16.**  Open VLC media player and click on **media > open capture device > play.**

Verify the correctness of your results by checking them with the values in excitation table (table 5).

NOTE: Make sure the RUN/PROG switch (SW19; leftmost toggle switch) is set to RUN. Do not worry about this if you are testing your project remotely.

### Submission of the report

Your lab report should observe the structure presented in the General Instructions from BS, while the following parts must be included:

1. Diagrams of your circuits along with the truth tables and the equations that you derived;
2. A short discussion on how you designed your circuit; and
3. An image of your file of simulation (*.vwf*). You can break it in several images if it is necessary.

The TA must check that your circuit functions both in simulation **AND** on the DE2-115 board **before** you leave the laboratory.

## PHYSICALLY PRESENT IN THE LAB OPTION

### PART II.2 - Simulation

6.9. Assign the current file to the project. Assign pine 83 to the clock, and pine 1 to CLRN, but leave all the other signals without assignments. Compile the file: the engine of routing and placement will assign all the other pins for you.

6.10.  Create a file of test vectors (*.vwf*) to verify your ALU datapath as follows:

f) Choose *Edit >> Grid Size…* and enter 50ns. Note that your circuit should function in a satisfactory way with a size of grid of 20ns and a period of clock of 40ns, provided that you devised it correctly; even the most ineffective circuits should have enough time between each clock pulse!

g) Choose *Edit >> End Time…* and enter 2.0μs.

h) Choose *Edit >> Insert Node or Bus …* in order to import all the inputs and all the outputs in *.vwf*.

i) Set up the clock signals and CLRN; note that CLRN (reset signal for the D flip flops of the 4-bit register) is active low, such that the registers are asynchronously forced to 0 until CLRN is deactivated by bringing it to high).

j) The sequence of micro-operations that you have to use in order to test the correctness of your circuit is given in Table 5. Fill in the empty columns of the Table 5 with the appropriate values of the control signals (S3, S2, S1, S0) and the 2 operands (A and B) that are needed to perform the corresponding RTL micro-operations. Evaluate the result (C) and the ALU state indicators (V,Z,N,Cy) the execution of each micro-operation, and fill out corresponding cells accordingly. Convert these values to hexadecimal ($S_{16}$, $A_{16}$, $B_{16}$, $C_{16}$, $St_{16}$) to simplify comparing with the simulations' results.

**Table 5: Sequence of micro-operations to simulate**

| Clock Cycle | RTL Micro-operations | S3 | S2 | S1 | S0 | A←op1 | B←op2 | C | V,Z,N,Cy | $S_{16}$ | $A_{16}$ | $B_{16}$ | $C_{16}$ | $St_{16}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $A \leftarrow "1010", B \leftarrow "0011", C \leftarrow A \wedge \overline{B}$ | | | | | | | | | | | | | |
| 2 | $A \leftarrow "0110", C \leftarrow a\text{shl } A$ | | | | | | | | | | | | | |
| 3 | $A \leftarrow "0011", B \leftarrow "0101", C \leftarrow A + B$ | | | | | | | | | | | | | |
| 4 | $A \leftarrow "1100", C \leftarrow A + 1$ | | | | | | | | | | | | | |
| 5 | $A \leftarrow "0011", B \leftarrow "0101", C \leftarrow A \oplus B$ | | | | | | | | | | | | | |
| 6 | $A \leftarrow "1010", C \leftarrow \overline{A}$ | | | | | | | | | | | | | |
| 7 | $C \leftarrow "0000"$ | | | | | | | | | | | | | |
| 8 | $A \leftarrow "0101", B \leftarrow "0011", C \leftarrow A + \overline{B} + 1$ | | | | | | | | | | | | | |
| 9 | $A \leftarrow "1110", C \leftarrow A$ | | | | | | | | | | | | | |
| 10 | $A \leftarrow "0110", C \leftarrow \overline{A} + 1$ | | | | | | | | | | | | | |
| 11 | $A \leftarrow "0101", B \leftarrow "0011", C \leftarrow A \wedge B$ | | | | | | | | | | | | | |
| 12 | $A \leftarrow "0001", B \leftarrow "0010", C \leftarrow A + B + 1$ | | | | | | | | | | | | | |
| 13 | $A \leftarrow "1101", C \leftarrow a\text{shr } A$ | | | | | | | | | | | | | |
| 14 | $A \leftarrow "0110", B \leftarrow "0101", C \leftarrow A + \overline{B}$ | | | | | | | | | | | | | |
| 15 | $C \leftarrow "1111"$ | | | | | | | | | | | | | |
| 16 | $A \leftarrow "1100", B \leftarrow "1010", C \leftarrow A \vee B$ | | | | | | | | | | | | | |

Set up the ALU control (S) and data (A and B) input signals of the .*vwf* file in accord with the values you just put in Table 5. It is strongly recommended to group your signals to simplify the data entry and reading. To group A3 - A0, assuming that A3 is the MSB and A0 is the LSB, please make sure that A3 is on the upper part of your time diagrams while A2, A1 and A0 should follow below it. To group A3-A0, first select A3, then press on the shift key and click on A0. Then, made a right click and choose ***Group…,*** give a name to this group, choose the format of posting into binary, and select ***OK***. To enter a value for the group, use the mouse to select a signal on the beach of time wanted, and then click on the icon ![icon] on the right of the screen. A window will open and ask for a new value.

6.11. Start simulation, and compare the outputs of your simulated ALU with the calculated values of C and St in your Table 5. **Note that there is a time delay at the ALU output which is caused by the input/output registers!** If some elements of your circuit do not function correctly, it is imperative to correct them at this stage. Finally make a capture of the simulation screen for your lab report.

6.12. Use ***Assignments >> Pins…*** to assign the rest of the pins as indicated in Table 6.

**Table 6: Pin Assignments for the final circuit**

| Signal | Pin Location | Component |
|--------|--------------|-----------|
| S3 | PIN_Y23 | SW17 |
| S2 | PIN_Y24 | SW16 |
| S1 | PIN_AA22 | SW15 |
| S0 | PIN_AA23 | SW14 |
| A3 | PIN_AD27 | SW3 |
| A2 | PIN_AC27 | SW2 |
| A1 | PIN_AC28 | SW1 |
| A0 | PIN_AB28 | SW0 |
| B3 | PIN_AB26 | SW7 |
| B2 | PIN_AD26 | SW6 |
| B1 | PIN_AC26 | SW5 |
| B0 | PIN_AB27 | SW4 |
| C3 | PIN_F21 | LEDR3 |
| C2 | PIN_E19 | LEDR2 |
| C1 | PIN_F19 | LEDR1 |
| C0 | PIN_G19 | LEDR0 |
| V | PIN_E24 | LEDG3 |
| Z | PIN_E25 | LEDG2 |
| S | PIN_E22 | LEDG1 |
| Cy | PIN_E21 | LEDG0 |
| CLK | PIN_Y2 | 50 MHz oscillator |
| CLRN | PIN_AA24 | SW13 |

## PART III.2 - Testing Experiment

6.13. Connect the ALU input pins (S, A, B) to slide switches, and its output pins (C, St) to LEDs. Program EP4CE115F29C7 and verify that your ALU functions correctly for each row of Table 5. Use your simulation and the calculated values of C and St to check the outputs displayed on LEDs.

NOTE: Make sure the RUN/PROG switch (SW19; leftmost toggle switch) is set to RUN.

## Submission of the report

The following parts must be included in your report:

1. Diagrams of your circuits along with the truth tables and the equations that you derived;
2. A short discussion on how you designed your circuit; and
3. An image of your file of simulation (*.vwf*). You can break it in several images if it is necessary.

The TA must check that your circuit functions both in simulation **AND** on the DE2-115 board **before** you leave the laboratory.

## APPENDICES

In the following are given the functional descriptions of some of the multiplexers you may want to use to implement your ALU circuit.

The symbol and truth table of an **8-to-1 multiplexer (74151)** are given below. Note how in the table, A, B, and C are the lines of selection of the multiplexer, and thus do not correspond to A, B, and C of our ALU!
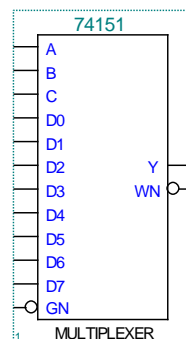
| Inputs | | | | Output |
|--------|---|---|--------|--------|
| Selection | | | Enable | |
| C | B | A | GN | Y |
| X | X | X | 1 | 0 |
| 0 | 0 | 0 | 0 | D0 |
| 0 | 0 | 1 | 0 | D1 |
| 0 | 1 | 0 | 0 | D2 |
| 0 | 1 | 1 | 0 | D3 |
| 1 | 0 | 0 | 0 | D4 |
| 1 | 0 | 1 | 0 | D5 |
| 1 | 1 | 0 | 0 | D6 |
| 1 | 1 | 1 | 0 | D7 |

**Figure 11: Symbol of multiplexer 74151**

**Table 7: Truth table of multiplexer 74151**

The symbol and the truth table of *dual* **multiplexer 4-to-1 74153** are shown below. Note that the two multiplexers share the same lines of selection A and B (it is an unfortunate coincidence, as these A and B do not have any relationship with the ALU inputs which carry the same name!)
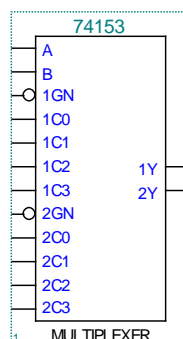
| Inputs | | | Output |
|--------|---|--------|--------|
| Selection | | Enable | |
| B | A | GN | Y |
| X | X | 1 | 0 |
| 0 | 0 | 0 | C0 |
| 0 | 1 | 0 | C1 |
| 1 | 0 | 0 | C2 |
| 1 | 1 | 0 | C3 |

**Figure 12: Symbol of the dual multiplexer 74153**

**Table 8: Truth table of the dual multiplexer 74153**

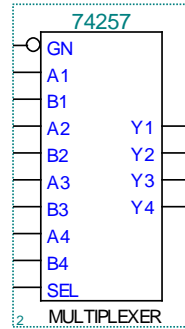The truth table and the symbol of the **quadruple multiplexer 2-to-1 74257** are given below:



| | | 74257 |
| GN | | |
| A1 | | |
| B1 | | |
| A2 | Y1 | |
| B2 | Y2 | |
| A3 | Y3 | |
| B3 | Y4 | |
| A4 | | |
| B4 | | |
| SEL | | |
| MULTIPLEXER | | |

| Inputs | | Output |
|---|---|---|
| **Selection** | **Enable** | |
| **A** | **GN** | **Y** |
| X | 1 | Z (high impedance) |
| 0 | 0 | A |
| 1 | 0 | B |

**Figure 13:  Symbol of the quadruple multiplexer 74257**

**Table 9:  Truth table of the quadruple multiplexer 74257**

**Note** that the use of multiplexers 74151, 74153 and 74257 is not compulsory. In fact, you can get *5% bonus* if you use the component *lpm-mux* instead of these multiplexers. Figure 17 gives an example of this multiplexer.

### EXAMPLES

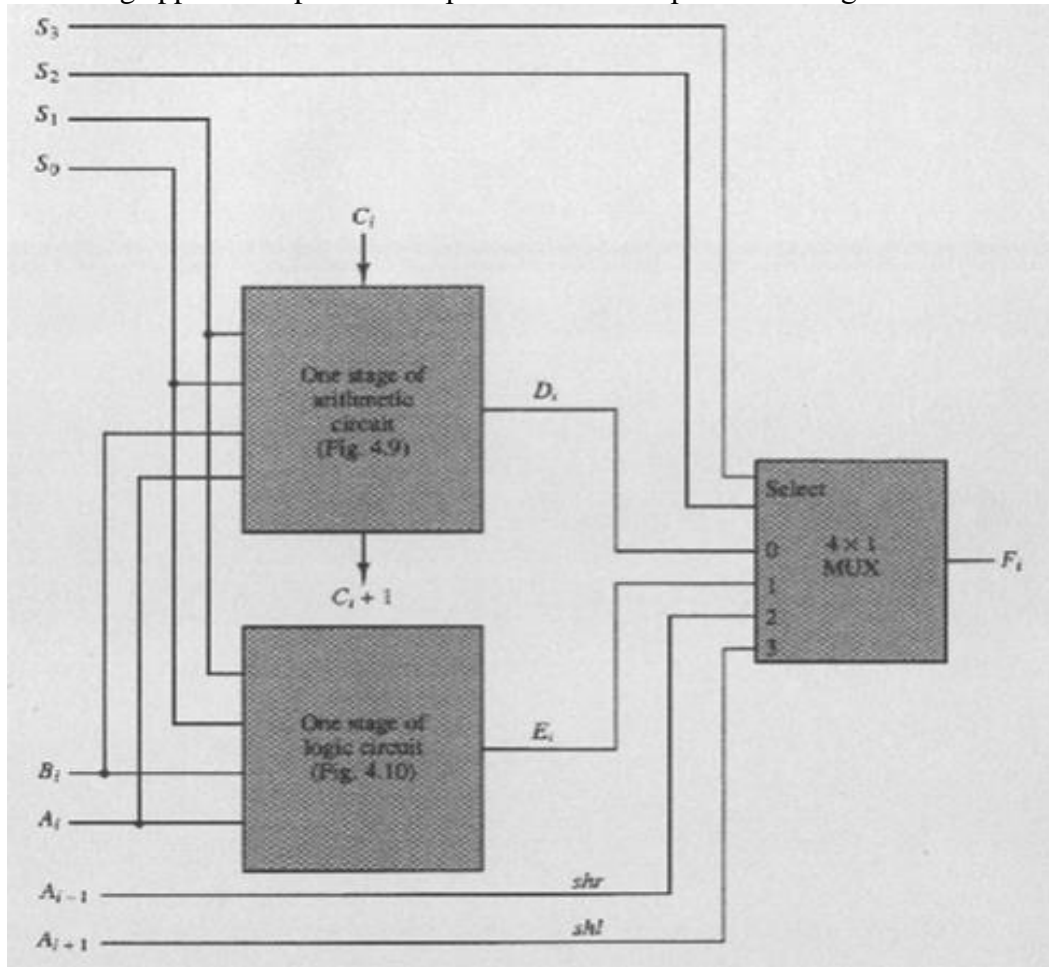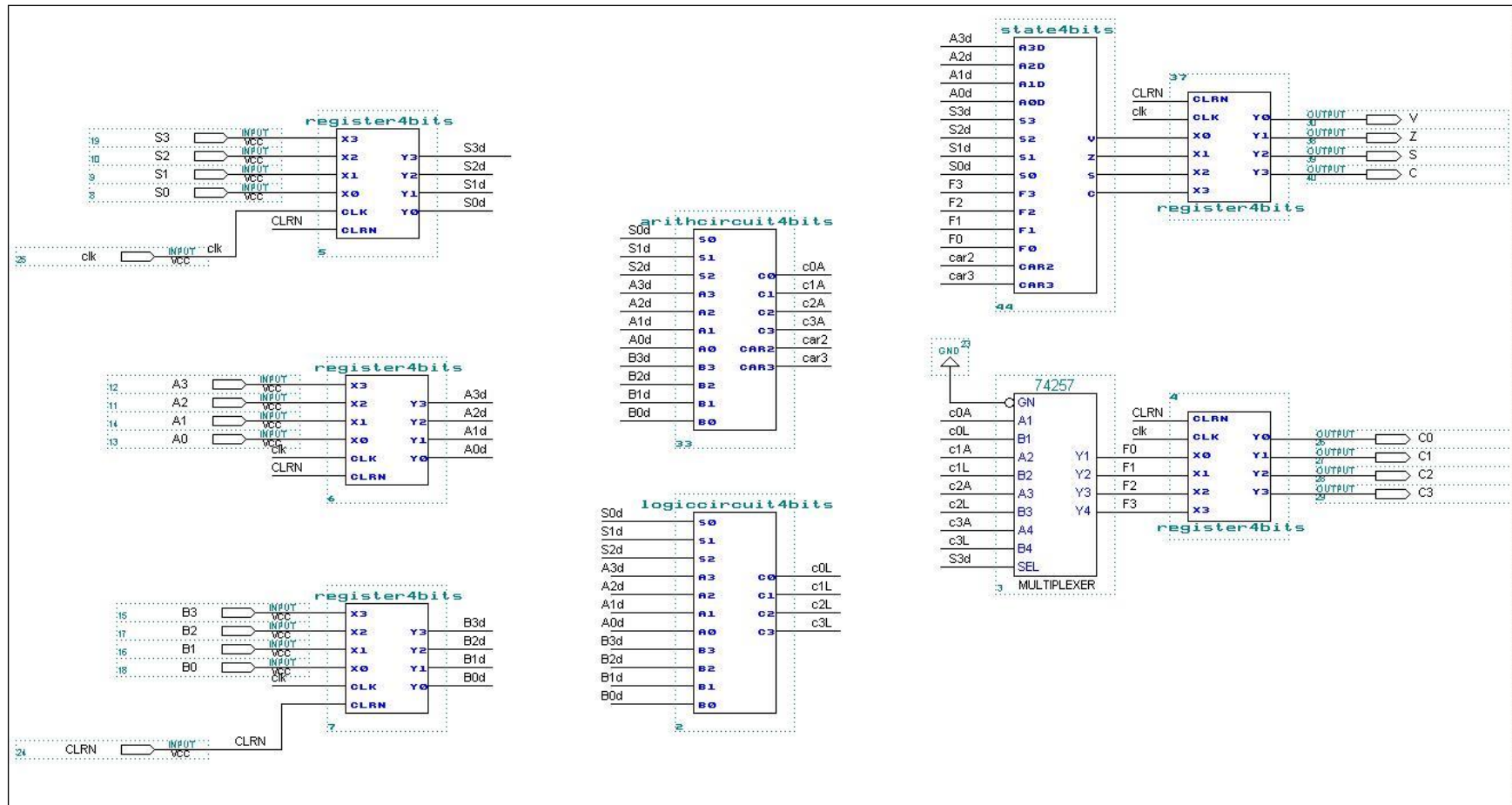The following appendices present samples from related previous designs of other ALU's.



**Figure 14:  Arithmetic logic unit**
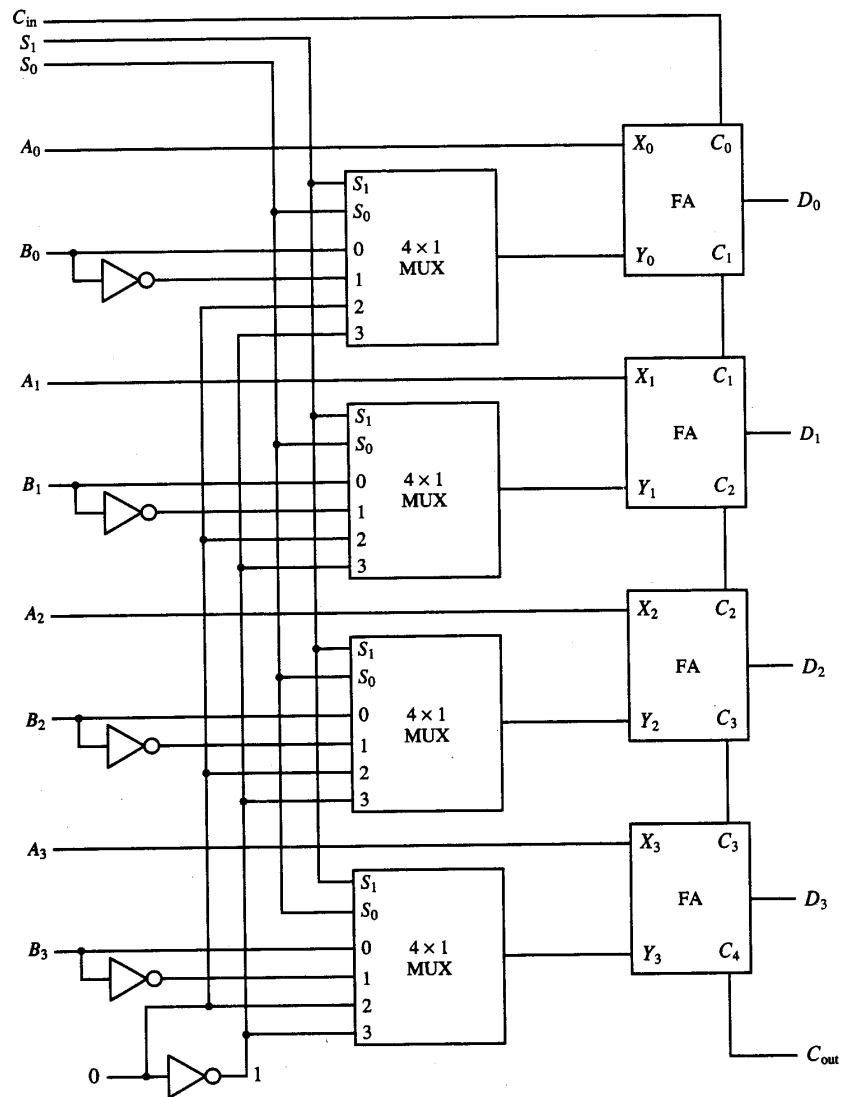
---

**Annexed Figure 15:  Diagram of complete ALU**

**Figure 16: Arithmetic circuit of 4-bit**
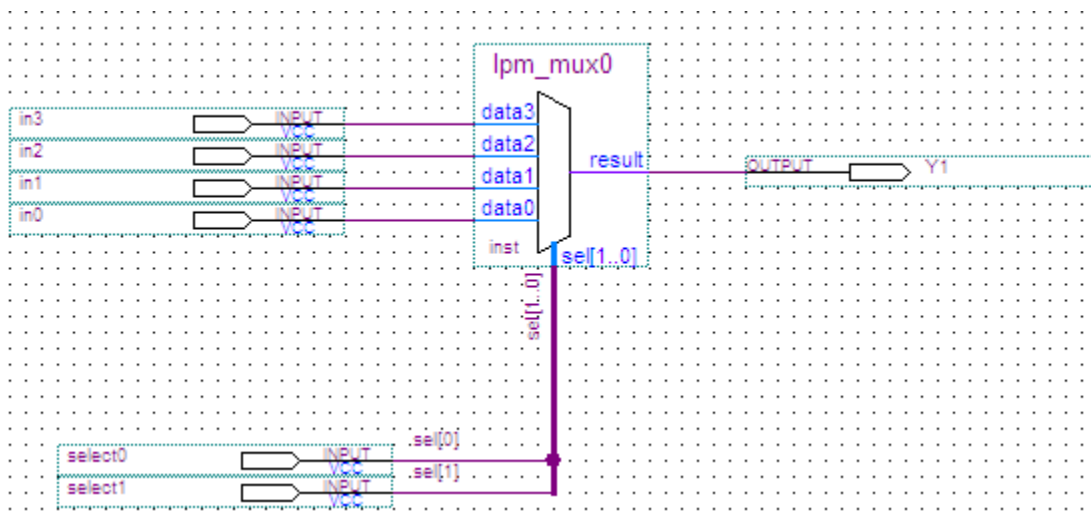


Figure 17: Example of a multiplexer lpm-mux

**Table 7: Shift and function table of an arithmetic logic unit**

| Operation select | | | | | | |
|---|---|---|---|---|---|---|
| $S_3$ | $S_2$ | $S_1$ | $S_0$ | $C_{in}$ | Operation | Function |
| 0 | 0 | 0 | 0 | 0 | $F = A$ | Transfer $A$ |
| 0 | 0 | 0 | 0 | 1 | $F = A + 1$ | Increment $A$ |
| 0 | 0 | 0 | 1 | 0 | $F = A + B$ | Addition |
| 0 | 0 | 0 | 1 | 1 | $F = A + B + 1$ | Add with carry |
| 0 | 0 | 1 | 0 | 0 | $F = A + \overline{B}$ | Subtract with borrow |
| 0 | 0 | 1 | 0 | 1 | $F = A + \overline{B} + 1$ | Subtraction |
| 0 | 0 | 1 | 1 | 0 | $F = A - 1$ | Decrement $A$ |
| 0 | 0 | 1 | 1 | 1 | $F = A$ | Transfer $A$ |
| 0 | 1 | 0 | 0 | $\times$ | $F = A \wedge B$ | AND |
| 0 | 1 | 0 | 1 | $\times$ | $F = A \vee B$ | OR |
| 0 | 1 | 1 | 0 | $\times$ | $F = A \oplus B$ | XOR |
| 0 | 1 | 1 | 1 | $\times$ | $F = \overline{A}$ | Complement $A$ |
| 1 | 0 | $\times$ | $\times$ | $\times$ | $F = \text{shr } A$ | Shift right $A$ into $F$ |
| 1 | 1 | $\times$ | $\times$ | $\times$ | $F = \text{shl } A$ | Shift left $A$ into $F$ |



Figure 18: Logic unit