The George Washington University
Electrical and Computer Engineering Department
Fall 2001

Mid-Term Project Documentation

Design of 4-bit Arithmetic and Logic Unit

Course: ECE 122

Professor: John E. Korman
GTA: Ritabrata Roy

Table of Content

List of Illustrations

List of Tables

Abstract

This project is to design a 4-bit ALU core which, if designed successfully, is supposed to be integrated into an IC used to control a new color touch pad display. Among several requirements imposed on the design, are the speed of clock signal that the ALU are to operate in being 30 MHz, the source voltage being 5 V … The design should be implemented with smallest layout area. Some other requirements involve the way we will simulate the design and the technology used to generate the layout. (using Hewlett-Packard 0.5 um n-well technology)

The project is given about three weeks ago. However, due to various reasons, it can't be started until about two weeks prior to the due date. Among the reasons that make early initialization of the project impossible was the theoretical materials given in the lecture being not sufficient up to the point.

However, althrough being started a little late, the project basically has been completed one week before the due date. Unfortunately, again due to some misunderstanding of the statement of the project by the author, especially of the meanings of the Carry and Overflow bit of a ALU, the author has to make some correction before it can be called a completed working project. And this is done just two day before the due date.

In summary, the project has been successfully completed after three weeks of designing, simulation, testing, and documentation. All of the goals given by the project statement have been beautifully achieved with one exception of the layout area the definition of which is not very obvious so as how small is considered as small. However, according to the estimated number of MOSFETs that Pr. Can E. Korman told us, the number of MOSFETs used in this design is not a bad one at all. And hence, we can say the project is successful in terms of timeliness and good functionality of the design.

Acknowledgement

I would like to thank Pr. John E. Korman who has been very patient explaining to me various practical concepts and ideas behind a real ALU's working functionality such as synchronization of input before going in the ALU and output data before going out the ALU. He also spent lots of his time clarifying the idea of how the carry and overflow bits are set in a real, practical working ALU as well as giving very interesting lectures on the electronic and digital design course. I also thank Ritabrata Roy, my GTA for this course, for his help on various practical problems that I encountered while conducting the project as well as in regular lab hours. I also thank many of friends who shared with me very enjoyable moments while we are busy with our tasks looking for solutions to the project. Without any of these people, I can hardly imagine how can complete this project in a timely manner while not compromising the quality of my work.

I- Project Introduction

Project Requirements and Basic Goals

The task is to design a 4-bit arithmetic logic unit (ALU) core which will be part of an IC to control a new color touch pad display. The primary emphasis of the project is on a working ALU design with minimum core layout area.
Besides the specific technical requirements which are related to input/output characterizations and relationship that will be stated clearly in the next section, there are other requirements regarding to the technology used to implement the layout as well as the working environment in which the ALU must function correctly. These are:

- The ALU must be able to operate at a 30 MHz clock and 5V-power source. (We are not responsible for the design of the clock).
- All designs will employ the SCMOS logic gate library that is in TannerLB.
- All layouts will employ the Hewlett Packard 0.5 μm n-well technology.

Top Level Design and Input/Output Relationship

Before going into details of the design, this section will give a more specific description of the requirement in terms of technical aspect by giving a clear statement of the input/output characterization and relationship. From that point, a sketch of top-level design will be derived based on that input/output characterization as well as other requirements which are relevant to this stage as stated by the project specifications.

From the top-level, the design will be a circuit with three sets of inputs and two sets of outputs as shown in Figure 1.1

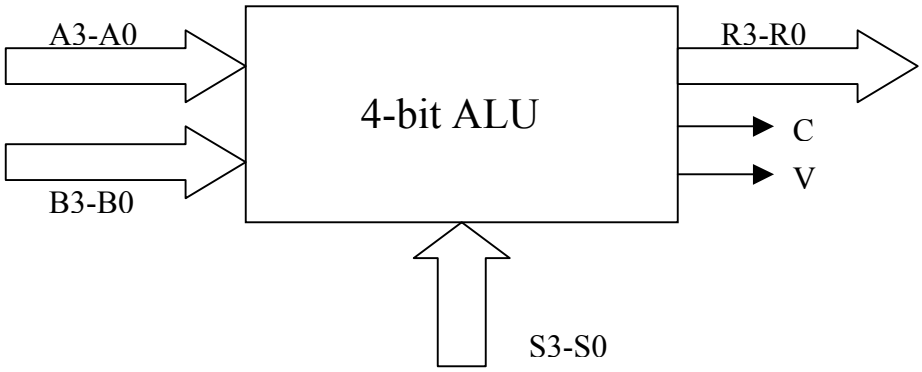


Figure 1.1. Top-Level Design

As shown in figure 1.1, the circuit accepts three 4-bit input vectors A, B, and S. More specifically, A and B are 4-bit data input vectors and S is a 4-bit command input vector. At the output end, the circuit will generate a 4-bit data result as well as two 1-bit status

signals C and V representing the Carry bit and Overflow bit, respectively for the operations.
The relationship between A, B, S, R, C, and V can be more specifically listed in the following table (Table 1.1)

Hex. Code	S3	S2	S1	S0	R	Comments	C	V
0	0	0	0	0	A	A is outputted	0	0
1	0	0	0	1	B shifted	1 bit left circular shift	0	0
2	0	0	1	0	A - B	Complement-Two Subtraction	0/1	0/1
3	0	0	1	1	B	B is outputted	0	0
4	0	1	0	0	A + 1	Increment A	0/1	0/1
5	0	1	0		A - 1	Decrement A	0/1	0/1
6	0	1	1	0	B + 1	Increment B	0/1	0/1
7	0	1	1	1	A + B	Add A and B	0/1	0/1
8	1	0	0	0	B - 1	Decrement B	0/1	0/1
9	1	0	0	1	A OR B	bit by bit OR	0	0
A	1	0	1	0	A shifted	1 bit right circular shift	0	0
B	1	0	1	1	A AND B	bit by bit AND	0	0
C	1	1	0	0	A XOR B	bit by bit XOR	0	0
D	1	1	0	1	Max(A, B)	Select the maximum	0	0
E	1	1	1	0	-B	Complement-Two Negation	0	0
F	1	1	1	1	- A	Complement-Two Negation	0	0

Table 1.1 Input/Output Relationship

According to the input/output relationship listed in table 1.1, we can specify further the way we interprete the meaning of the data input vectors A and B, and the data output vector R respectively so that the corresponding operation is meaningful. This is straightforward. For those operations other than addition, subtraction, increment, and decrement (including Max (A,B)), A and B are considered as two sets of bits. For the rests, A, B, and R are 4-bit signed numbers. In such circumstances, their most significant bits (A3, B3, and R3) are *sign* bits. Also note that we will use complement-two representations for negative numbers or in cases that involve subtraction operation.

One more thing important enough to be mentioned is that we can't assume each bit of our data inputs and command inputs which are meaningful to the supposed operation will be available to the ALU simultaneously. Hence, it is appropriate to implement some kind of circuit that can synchronize the inputs; and make sure that our circuit is operating on the

correct data. At the output end, we will do the same thing so that our output will always contain meaningful data.

Design Procedure with Block Diagrams to Get Input/Output Relationships

Up to this point, the basic requirements and specifications of the problem have been well stated; and therefore, we are ready for the designing process. It is certain that, while the basic requirements will need to be perfectly met, we will probably impose some more specifications and requirements in order to improve and/or to facilitate our version of implementation. We will do this along our way to implement the circuit at any stage of our designing and/or testing processes.
As the basic requirements and specifications of the problem suggest, the circuit will accepts three sets of inputs (A, B, and S) and giving the values of output (R,C, and V) accordingly. Also as stated, we always keep in mind that, whatever our approach is, the meaningful input data of our top-level circuit will not come to our ALU at the same time. However, for the time being, we just assume that the synchronizing mechanism has been achieved; and our data and command inputs are already synchronized. So at the top-level without taking into account the synchronizing circuitry, we can model the rest of the circuit as of a logic function given as f = f(A, B, S) whereas f is the output vector that includes R, C, and V as a whole. More specifically, we can implement R, C, and V separately as follows: R = R(A, B, S), C = C(A, B, S), and V = V(A, B, S); and f = [R, C, V]. Doing so, we can approach the task by introducing an initial version of our design of R as given in figure 1.2

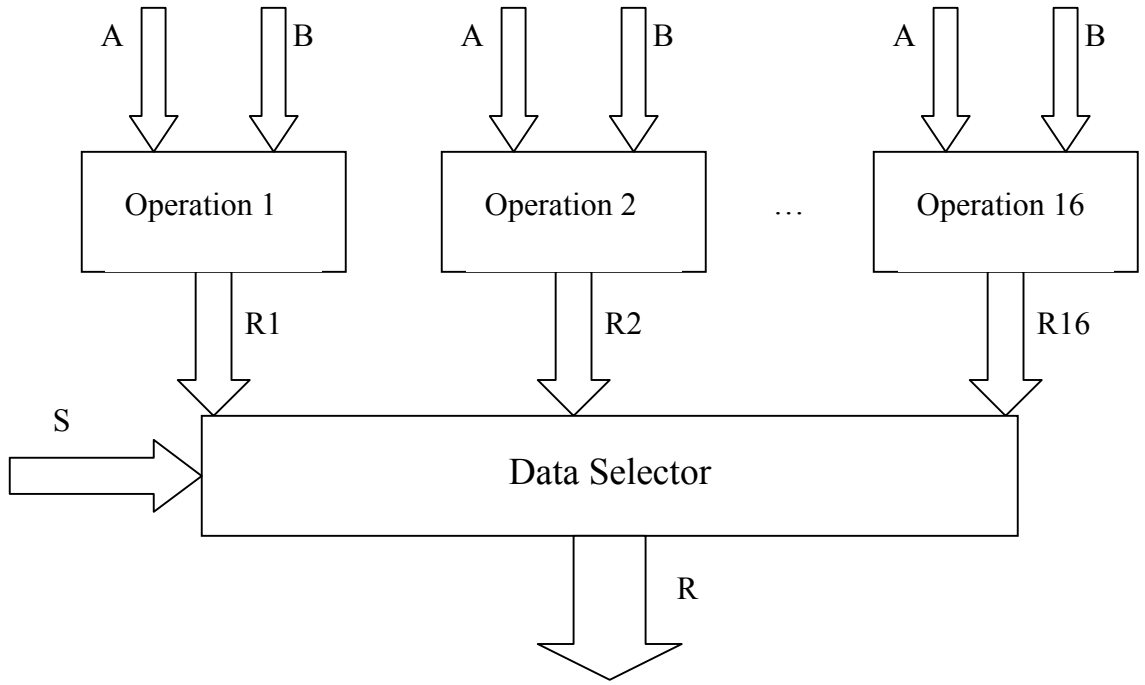


Figure 1.2. Block diagram to implement data output vector R

As shown in Figure 1.2, our circuit can comprise 16 operation components , each of which performs one of the 16 operations given by the statement of the problem (see table 1.1). Also from the figure, A, B, S, and R are all 4-bit vectors. The basic idea behind this scheme is that because we have 16 operations (each operation corresponds to a combination of 4-bit command input S – there are 16 of such combinations), we can implement each component as a separate combinational circuit receiving A and/or B as data inputs and giving 16 output vectors R1, R2, …, R16 respectively, in general sense, at the same time. Because there is only one out of these 16 data is meaningful at a time, the unique bit pattern of S will choose among these the unique correct output vector.
Similarly, we can use the same approach to obtain the circuitry for C and V as shown in figure 1.3. After implementing each part individually, we can put them together giving the complete and integrated circuit as desired.

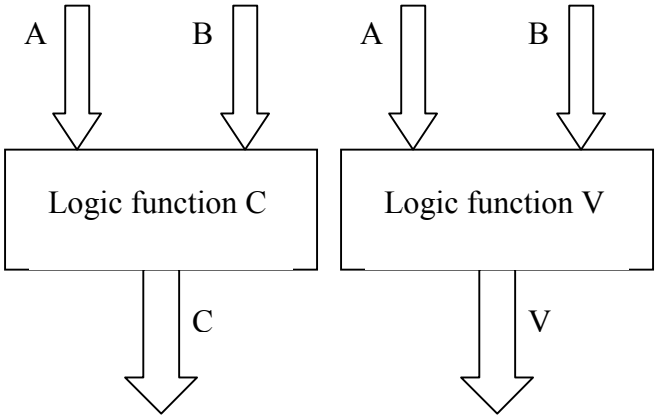


Figure 1.3. Block diagram for the circuitry of C and V functions

Comment: We can see that the material from the two sections just described above is sufficient to implement the circuit in terms of logic functions. However, as stated from the beginning of the project statements, the layout with minimum core area is also desired so that the core can be appropriately integrated into a larger chip. It turns out that the described approach is not a good one by small area criteria. Also, from figure 1.2, if this approach is adopted, we will need a 4-bit 16:1 multiplexer so that we can select the correct data output at the output end of the ALU. However, this is very impractical because it may impose a heavy current-driving capability on the selector inputs**.** For that reason, we will adopt a more practical approach by modifying the initial version described above. In this approach, we will try to group our data into a smaller number of units by sharing some of the components among members of each group. This approach will help reduce the total area of the circuit in order to meet the minimum area layout requirement. In order to do so, we will begin with the basic functions which may be reuseable by other operations and more importantly, can't be broken up into smaller pieces. The candicate operations for separate and independent implementation are: OR, AND, and XOR. The operations to output A or B, of course, can use either one of the above if appropriate inputs are applied; hence these operations are removed from the list. The interesting cases are the shift operations (having hex code 1, and A). Should this be implemented as independent module or not depends on how we choose to implement the functions. One option could be using a 4-bit register with a clock signal being implemented internally so that at the output end of the circuit, the ALU behaves just like a combinational logic circuit. We can use the same register with extra logic to perform the two *shift* operations. However, the added logic

circuit may increase the performance (adding delay time) of the operations as well as the total area of the ALU as a whole. Or, we can use two registers, each for one operation. If the latter one is adopted, then there is an even better solution: using fixed connection so that no registers nor logic circuits are needed by outputting directly. We will go back to this issue when we are done with other operations. Now, besides OR, AND, XOR, A, B, and the two *shift* operations, all the outputs of the other operations can be implemented sharing the adder circuitry except for the operation D (Max(A, B)). These are done in the following manner:

Operation 7 (A + B) is clear. The adder will receive two input vectors A and B and generates the output vector A + B using regular addition performance.
Operation E (0 minus B) is implemented by applying 0 at one input of the adder. The other input vector is B complemented and added 1.
Operation F (0 minus A) is done in the similar fashion.
Operation 4 (A + 1) is just like operation 7 except for one of the operands now being a constant 1.
Operation 6 (B+1) is similar to the operation 4.
Operations 5 and 8 (A – 1 and B – 1 respectively) are performed by adding the relevant operand (A or B) to –1 (The 4-bit complement-two of –1 is binarily represented as 1111).
Operation 2 (A – B) is performed by first negating B and add this to A itself.

The discussion above indicates that at least the outputs of the operations 2, 4, 5, 6, 7, 8, E, and F can be taken from the sum output of the adder.

So far, we have 4 units whose outputs will be fed to the output end of the ALU: the circuit of the adder, and three circuits that performs the OR, AND, and XOR respectively. Also, we have at least one more unit that needs its output to be directed to the output end (at least one shift operation). Because the inputs of the Output Selector component shown in figure 1.2 now exceeds four. It may be attempted to use a 4-bit 8:1 multiplexer to do the task.

The operation D itself, can be implemented using the XOR function which is assumably already built as an independent function. Depending on the comparison, either A or B is outputted. Again we can build this unit as one output unit or just by outputting A or B directly based on the outcome of the comparision. The first choice requires extra logic circuit to switch the outputs. The second one just outputs A or B directly; thus, requires two more vector inputs at the 4-bit 8:1 multiplexer giving the total inputs already used at the multiplexer to 6.

Now, come back to the *shift* operations, because we still have two more 4-bit inputs available at the data selector component, we will choose the last option to implement the *shift* operations, that is, the one using fixed, direct connection of input to output. As discussed, this will need two vector inputs at the data selector component making the total numbers of inputs needed for the data selection component to be 8.

From the discussion above, we will have our modified block diagram shown in figure 1.4, in which we also incorporated the logic circuit to implement C and V bits. Also, what not shown explicitly from the diagram is the synchronization mechamism to synchronize our inputs and outputs.

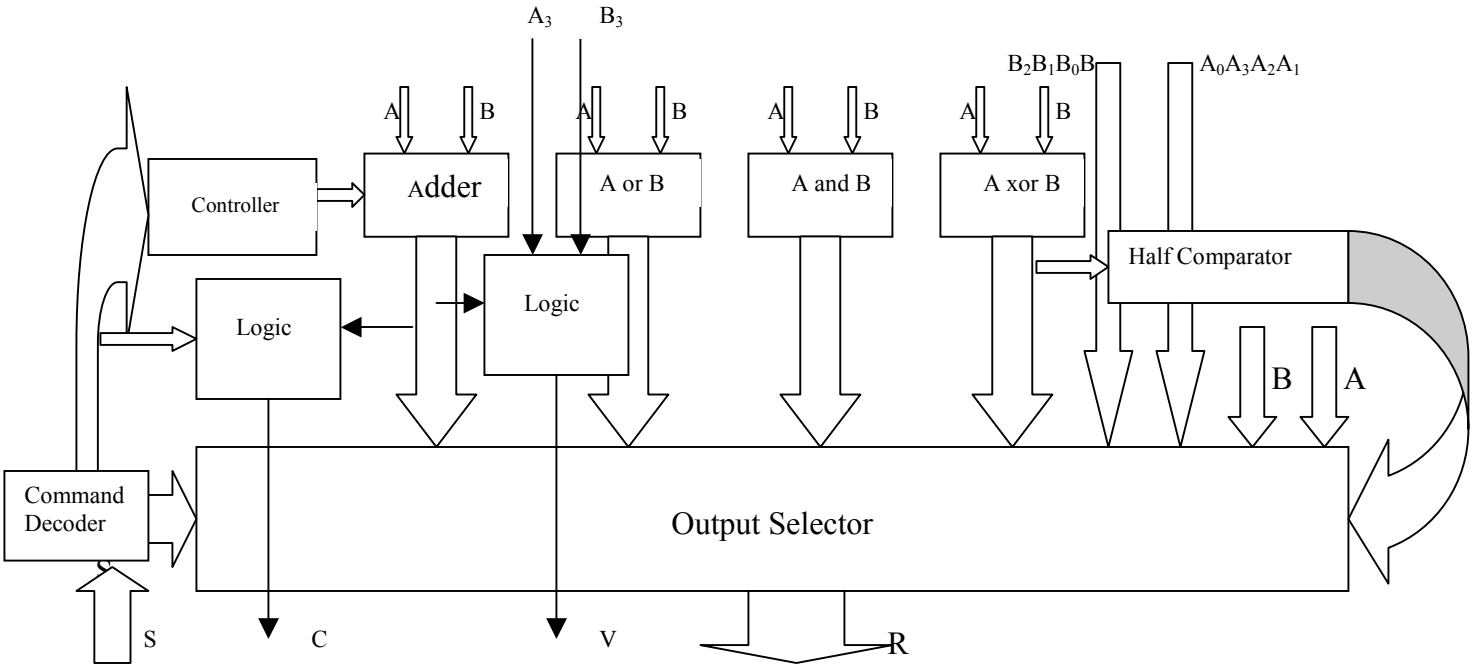


Figure 1.4. Modified Block Diagram of the Circuit

Implementation of Design Using EDA Tools

There are three main stages in implementing the circuit using EDA tools.

- S-Edit will be used first to create the circuitry of the design including all modules together with the top module. We also use S-Edit to input values of some other parameters such as source voltage, and relevant information of the pulses, if any, so that it is ready for simulation. (For instance, the clock signal will have period of less than 33.33 ns to emulate the condition of 30 MHz clock as required).
- T-Spice will be used to simulate the circuit. According to the requirement of the project, we will set the chanel length L = 0.5 μm through parameter l.
- W-Edit will be used to view the waveforms of our output together with its corresponding inputs.

II- Project Breakdown and Characterization of Each Sub-Module

Partitioning of Project into Smaller and Manageable Sub-Modules

As shown in figure 1.4, the circuit composes of several modules. In this section, we will name each module and describe in detail their implementations.

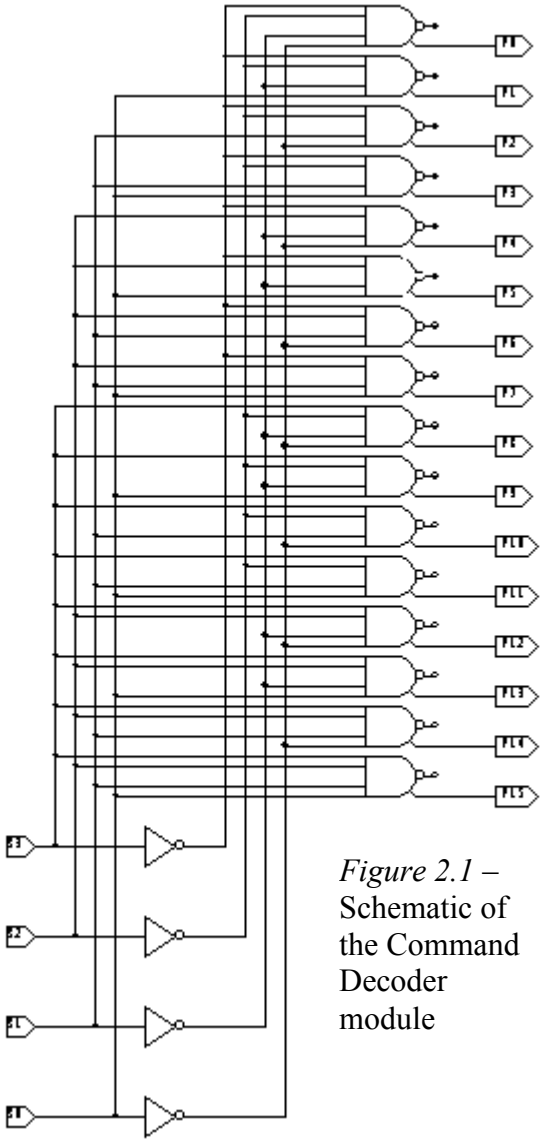


Figure 2.1 – Schematic of the Command Decoder module

To make our circuit hierarchically structural, we will divide the circuit into following modules:

- Command decoder module.
- A AND B module.
- A OR B module.
- A XOR B and Comparision module.
- Output Selector module.
- Arithmetic module.
- Carry and Overflow Logic module.

Implementation of the Command Decoder module

From our discussions so far, basically our circuit behaves like a function selector. So to facilitate this process, it is useful to have a command decoder module. The input of the module will be the 4-bit command vector S. It will have 16 outputs, each of which corresponds to a unique combination of input bits out of 16. The schematic of the command decoder is shown in figure 2.1.

At the output end, although the module has 16 outputs, there is one and only one of them being active at a time (active high). We can use the outputs of the module to control other modules accordingly so as they will give the correct data flow from our inputs and outputs. The functionality of the module can be described using the following truth table (table 2.1)

S3	S2	S1	S0	Active-High Output
0	0	0	0	P0
0	0	0	1	P1
0	0	1	0	P2
0	0	1	1	P3
0	1	0	0	P4
0	1	0	1	P5
0	1	1	0	P6
0	1	1	1	P7
1	0	0	0	P8
1	0	0	1	P9
1	0	1	0	P10
1	0	1	1	P11
1	1	0	0	P12
1	1	0	1	P13
1	1	1	0	P14
1	1	1	1	P15

Table 2.1. Truth Table describing the functionality of the Command Decoder module.

Implementation of the A OR B module

For the A OR B module the circuit is very straightforward. The module consists of 4 2-input OR gates in parallel to OR A and B bit by bit.

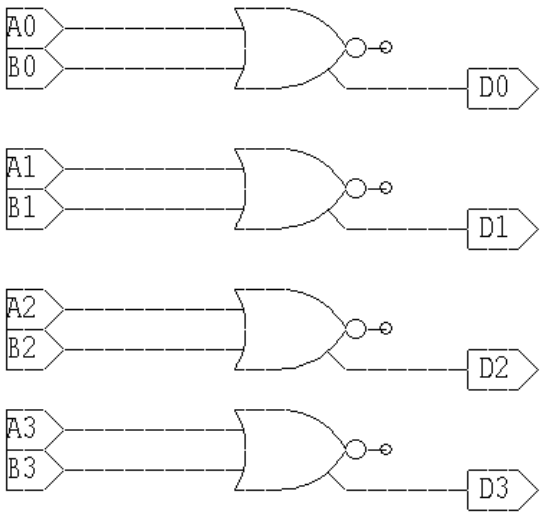


Figure 2.2- Schematic of the A OR B module

The schematic of the module is shown in Figure 2.2.

This module will be used to perform the A OR B operation. Its output will be fed to the output selector. Based on the decoding of the command inputs, if P9 is high, this output will be the output of the whole circuit (still need to be synchronize – we will describe this later).

Implementation of the A AND B module

Similar to what described in the previous section, the implementation of the A AND B module is shown in

Figure 2.3. The module consists of 4 2-input AND gates in parallel to AND A and B bit by bit.
This module will be used to perform the A AND B operation. Its output will be fed to the output selector. Based on the decoding of the current command inputs, if P11 is currently high, the output of this module will be selected to be the output of the whole circuit.

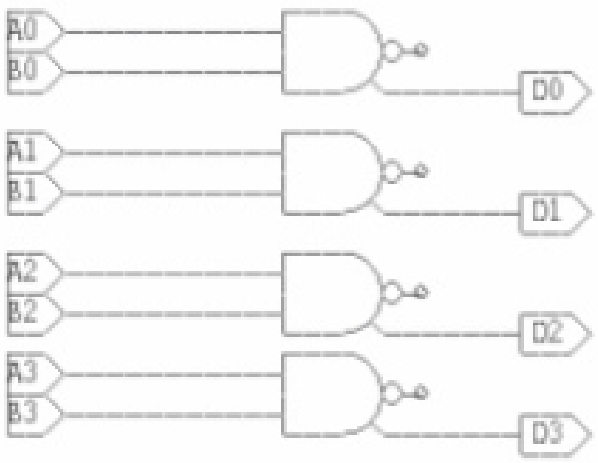


Figure 2.3 – Schematic of the A AND B module

Implementation of the A XOR B and Comparision module

This module includes two components. The first component is the 4-bit XOR logic function itself and pretty straightforward. The output of this component will be used by the second component named Half Comparator and also be fed to the output selector. In case P12 is currently active (or the current command is A XOR B), this output will be selected to be the output of the whole circuit. The schematic of the XOR component is shown in Figure 2.4.

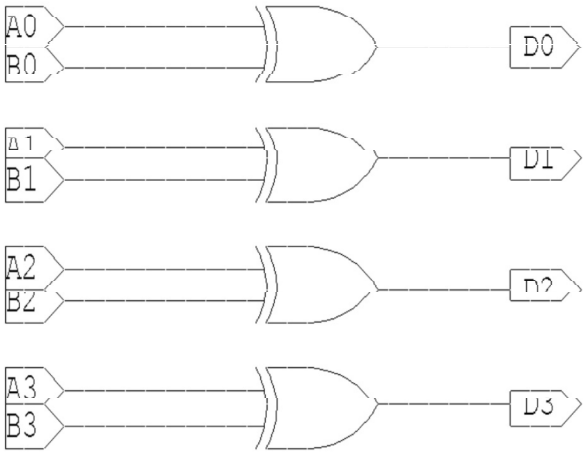


Figure 2.4 – Schematic of the XOR component of the XOR and comparision module

The second component of this module will be named Half Comparator used to compare the values of two unsigned numbers A and B.
This component will have 1-bit output called fA indicating whether A is greater than or equal to B in that case fA is 1 (and 0 if A is less than or equal to B). The schematic of this component is shown in Figure 2.5

The algorithm used to compare two unsigned number A and B is as follows:
Let $D = D_3D_2D_1D_0 = A \text{ XOR } B$.
If D_3 is 1 and $A_3 = 1$ then fA = 1 (indicating that A is greater than B). If D_3 is 1, and $A_3 = 0$ then fA = 0 (indicating A is not greater than B). If $D_3 = 0$, then we have to take a look at D_2 and A_2 and B_2 in a similar manner. In case $D_3 = D_2 = D_1 = 0$, we just look at the value of A_0. It is safe to set fA to 1

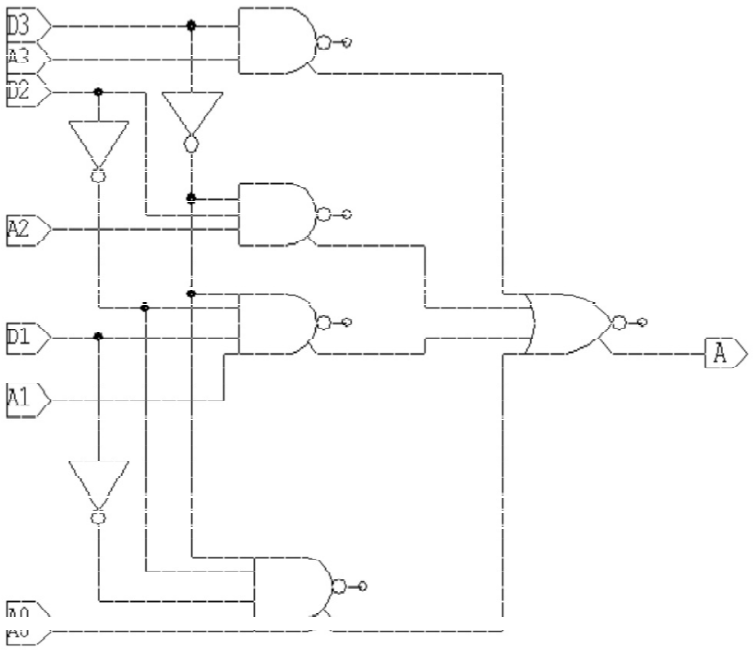


Figure 2.5 – Schematic of the Half-Comparator component of the XOR and Comparision module

if A_0 = 1 (A greater than or equal to B), and to 0 if A_0 = 0 (A is less than or equal to B). The output fA of this component will be used to control the data selector if in case the output P13 is high (current command code is D). If that is the case, A will be selected if fA is 1, otherwise B will be selected to be the output.

Implementation of the Arithmetic module

As stated above, several operations (P0, P2, P3, P4, P5, P6, P7, P8, P14, P15) will share this module to do their tasks. For this reason, this module will have extra control circuits or components to control the data flow so that the correct operation of the circuit is always guaranteed. The core of the module is the 4-bit Full Adder. For the time being, we just take a straightforward approach and use a linear full adder instead of employing a faster version. When we come to the testing stage, we may modify the circuit to meet the clock speed of 30 MHz as required. If the linear 4-bit Full Adder works well, we will keep our original design.

As far as the inputs of the FA is concerned, we can construct the following table (Table 2.2)

As shown from the third column, we will need some kind of logic combination so that it allows various inputs to be entered the adder accordingly with the active command. Besides inputs for regular operands (A and B – entered the adder at the same time for operation with command code 2 and 7, others involve only one (either A or B but not both) and a constant (either 0 or 1 but not both). Also from the table we notice that, whenever there is an operation that involves a negation (negation, subtraction, decrementing) the negation involves only one operand or constant at a time. So the logic for the inputs of the FA at least will be implemented as follows:

- Constants 0 and 1 will be fixed on the same side (because we will need only one of them at any instant).
- Each side must have at least one route for regular inputs (operands of the operation).
- One of the regular route allows negation for operations that involve negation.
- A and B should be switcheable so that the adder will work equally well with A and B without knowing in advance which route A and/or B will enter the adder.

Command Code	Active Decoder Output	Operation by the FA
0	P0	A + 0
2	P2	A - B
3	P3	B + 0
4	P4	A + 1
5	P5	A - 1
6	P6	B + 1
7	P7	A + B
8	P8	B - 1
E	P14	0 - B
F	P15	0 - A

Table 2.2 Operations performed by the FA with respect to each command

From the discussion above, we can derive the block diagram for the Arithmetic module as shown in Figure 2.6. Before going any further, we need to make it clear that the control signals S_0, S_1, S_2, and S_3 shown in the Figure have local meaning only; that is, these signals don't have anything to do with our global command input vector S. We simply take these notations for convenience.

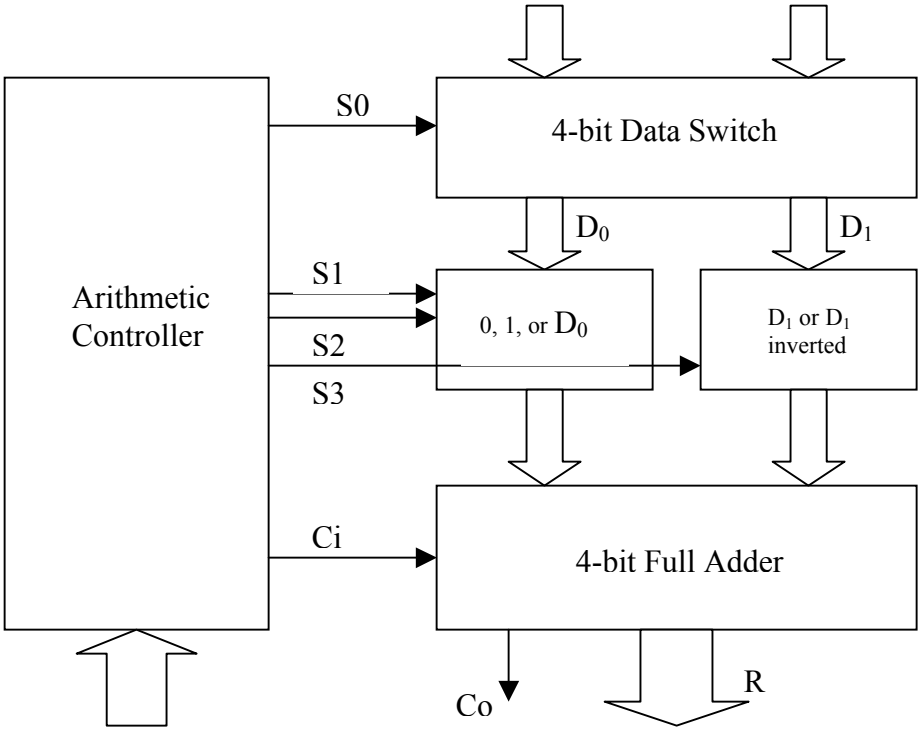


Figure 2.6 – Block Diagram for Arithmetic module

Implementation of the 4-bit Full Adder is shown in Figure 2.7.

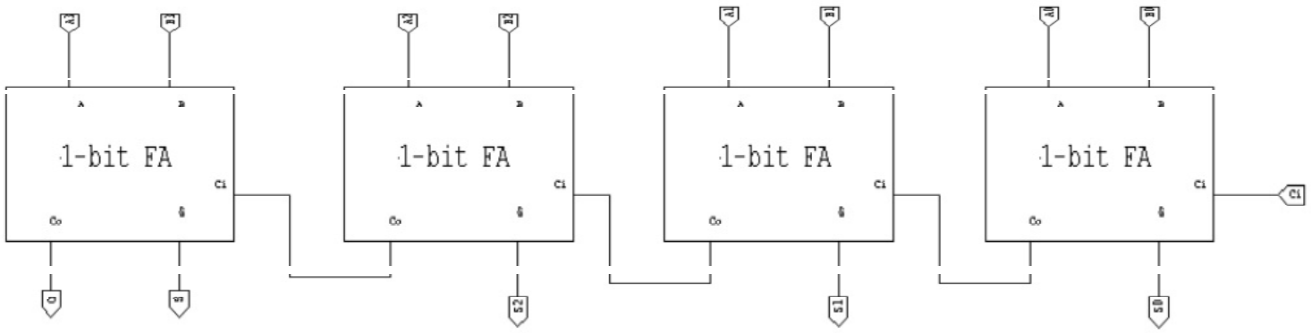


Figure 2.7 – Schematic of the 4-bit Full Adder

We can implement the first input route using the following block diagram (shown in Figure 2.8). The component will be controlled by the Arithmetic Controller component using two control signals S_1 and S_2 as shown in the figure. If we like the 4-bit D_0 to pass through the router, we will control S_1 and S_2 so that $S_1 = 1$ and $S_2 = 0$. If we want a constant 0 to appear at the FA's inputs, we will set $S_1 = 0$ and $S_2 = 0$. Similarly, if a constant 1 is desired, S_2 will be set to 1 regardless of the values of S_1 and D_0. In this case, we will choose S_1 so that it is more convenient to implement the Arithmetic Controller.

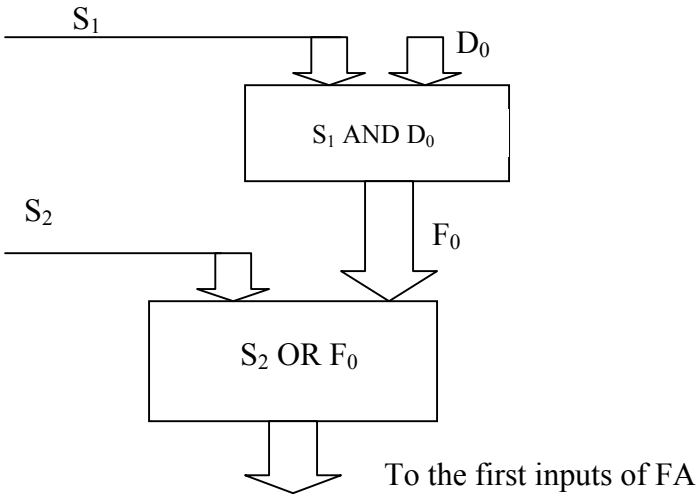


Figure 2.8 – Block diagram to implement the first input router of the 4-bit FA.

The schematics of the Arithmetic module is shown in Figure 2.9. The schematic of the second router is shown in Figure 2.10 (in figure 2.19, it is named Data Selector). The second input router will be controlled by the Arithmetic Controller using control signal S_3. If $S_3 = 1$, its input will be fed to the second inputs of the FA. Otherwise, the inverts of its inputs will pass through the router to the second inputs of the FA.

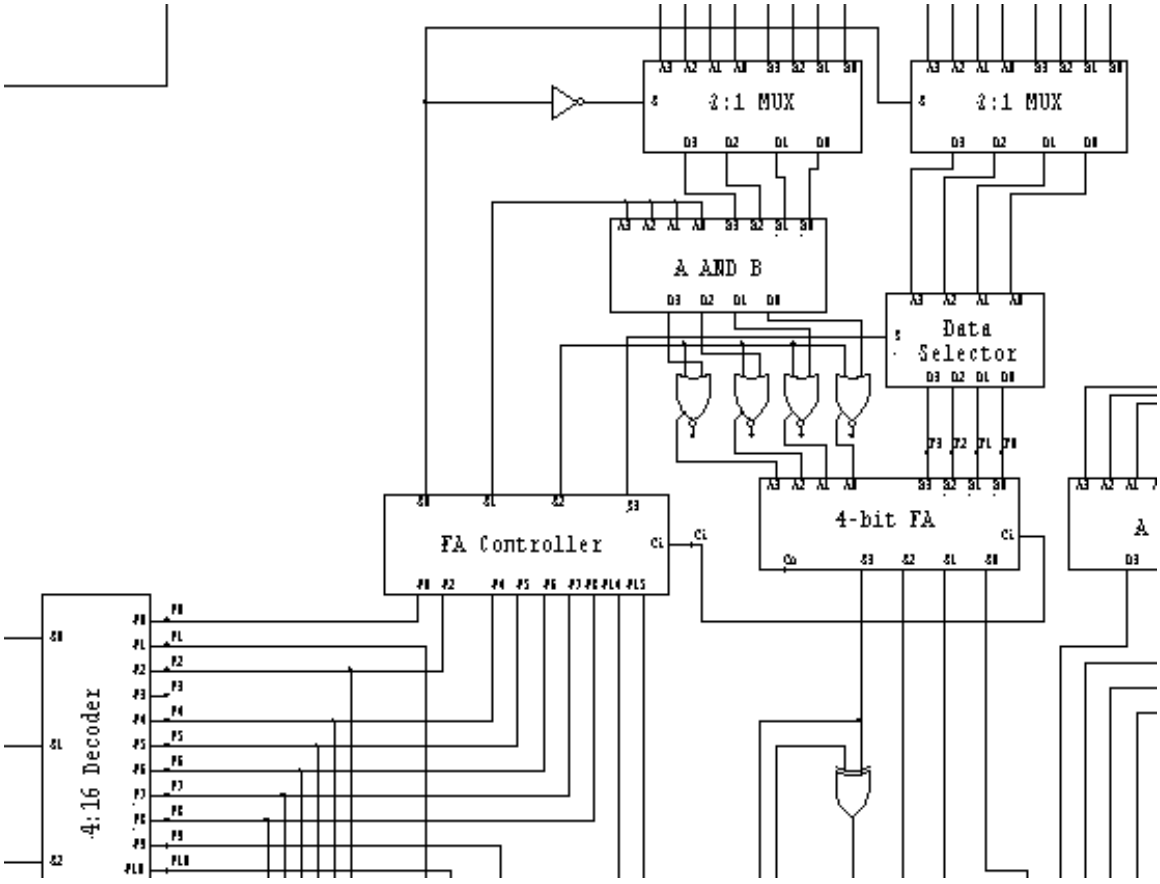


Figure 2.9 – The schematic of the Arithmetic module

To implement the 4-bit data switch, we will use two 4-bit 2:1 Multiplexers. The data switch component will be controlled by the Arithmetic Controller component using control signal S_0. The implementation of the 4-bit data switch is shown in Figure 2.11. As shown from the figure, if $S_0 = 1$, the 4-bit operand A will appear at the first 4-bit output and the 4-bit operand B will appear at the second output. If we want the operands to switch between the two outputs, we will set $S_0 = 0$. The implementation of a 4-bit 2:1 Multiplexer used by this module is shown in Figure 2.12.

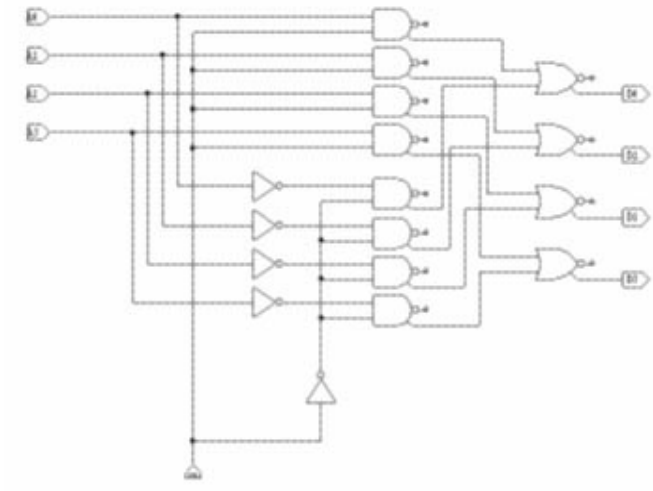


Figure 2.10 – Schematic of the second input router of the 4-bit FA

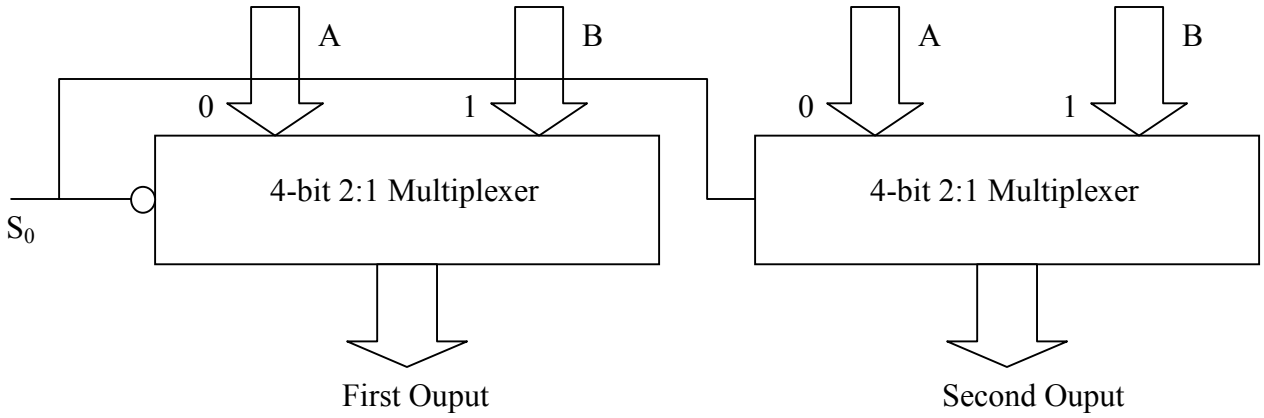


Figure 2.11- Modified form of the schematic of the 4-bit Data Switch component of the Arithmetic module

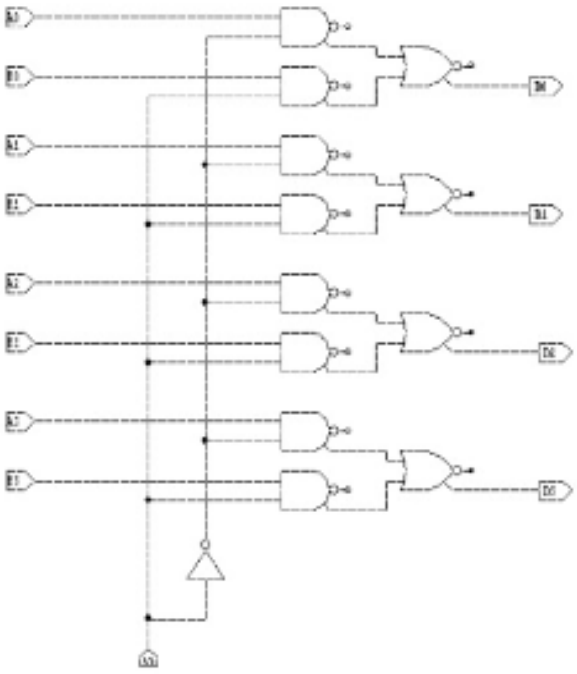


Figure 2.12 – Schematic of a 4-bit 2:1 Multiplexer used by the Data Switch component of the Arithmetic module

Implementation of the Arithmetic Controller component (in figure 2.19, it is named FA Controller)

The duty of the Arithmetic Controller component is to control the Data Switch component, the input routers to the 4-bit FA, and the FA itself so that the correct operands and inputs are applied at the inputs of the FA corresponding to a certain operation which involves arithmetics. It does so using five control signals: S_0, S_1, S_2, S_3, and C_i. The role of S_0, S_1, S_2, and S_3 have been described previously. The role of C_i is to set up the carry-in bit for the FA so that it works properly with a whole range of operations (addition, subtraction, which needs two's complement form of the subtrand). The inputs of the Arithmetic Controller component are P_0, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_{14}, P_{15}. The relationship between inputs and

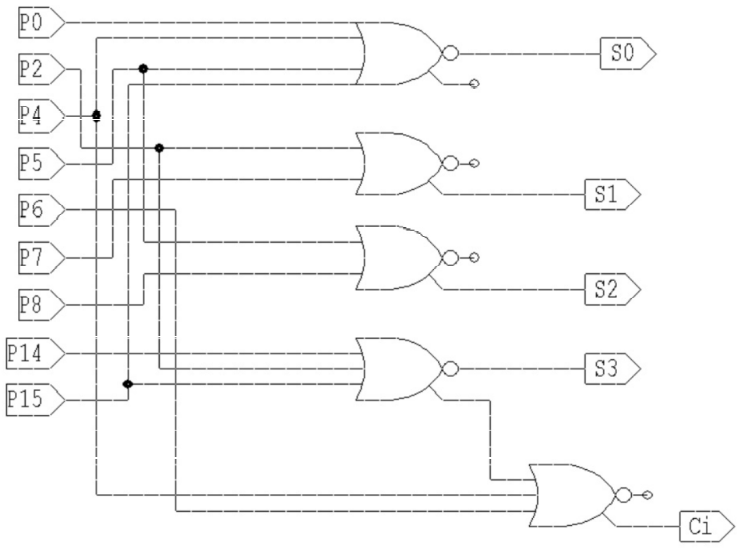
Input	Operation performed	First Input of FA	Second Input of FA	Ci	S3	S2	S1	S0
P0	A	0	A	0	1	0	0	0
P2	A - B	A	B inverted	1	0	0	1	1
P3	B	0	B	0	1	0	0	1
P4	A + 1	0	A	1	1	0	0	0
P5	A - 1	1111	A	0	1	1	X	0
P6	B + 1	0	B	1	1	0	0	1
P7	A + B	A	B	0	1	0	1	1
P8	B - 1	1111	B	0	1	1	X	1
P14	0 - B	0	B inverted	1	0	0	0	1
P15	0 - A	0	A inverted	1	0	0	0	0

Table 2.3- Truth Table to implement the Arithmetic Controller component of the Arithmetic module

outputs of the Arithmetic Controller is specified through Table 2.3.

From table 2.3, we can derive the logic expressions for C_i, S_3, S_2, S_1, S_0 as follows:
$C_i = S_3' + P_4 + P_6$
$S_3 = P_2 . P_{14} . P_{15} = (P_2 + P_{14} + P_{15})'$
$S_2 = P_5 + P_8$
$S_1 = P_2 + P_3$
$S_0 = P_0'. P_4'. P_5'. P_{15}' = (P_0 + P_4 + P_5 + P_{15})'$

Notice that for convenience, we will choose the don't-care values of S_1 (X) = 0.



Also notice that we don't use P_3 at all. This reflects the fact that when all inputs of the Arithmetic Controller are zero then either P_3 = 1 or the operation of the Arithmetic module is irrelevant (and not chosen to be the output of the ALU anyway).
The schematic of the Arithmetic Controller is shown in Figure 2.13

Figure 2.13 – The schematic of the Arithmetic Controller component of the Arithmetic module

Implementation of the Output Selector Module

The duty of the Output Selector Module is to select the one and only one output which is relevant to the current command input out of 8 different sources of outputs. The block diagram for the module is shown in Figure 2.14

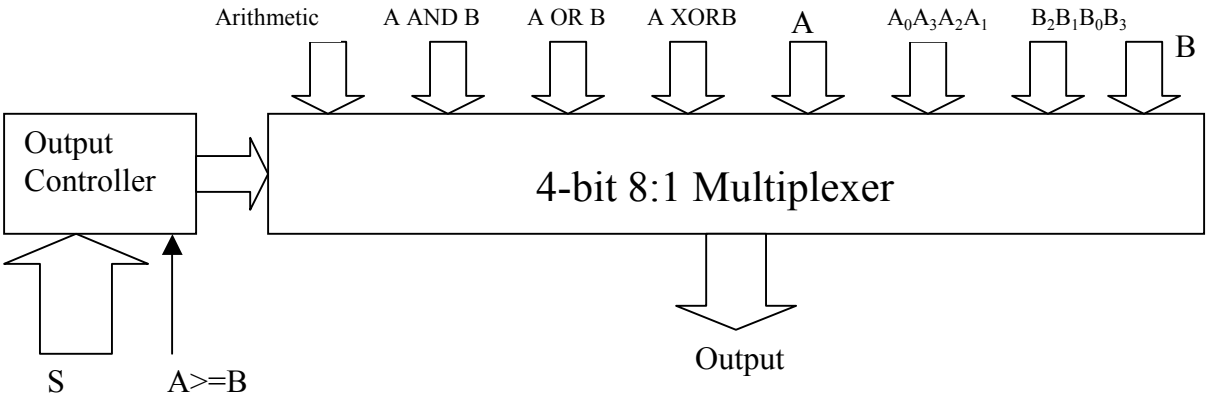


Figure 2.14 – Block diagram of the Data Selector module

We will implement the 4-bit 8:1 Multiplexer using two 4-bit 4:1 Multiplexers with Chip Select input (active-high), the schematic of which is shown in Figure 2.15. The modified schematic of the 4-bit 8:1 Multiplexer is shown in Figure 2.16

Once again, from Figure 2.16, we use the notation S_0, S_1 for local explanation. They are not directly taken from the corresponding bits of the command input vector S.

The 4-bit 8:1 Multiplexer will be controlled by the Output Controller component using control signal S_0, S_1, and CS.

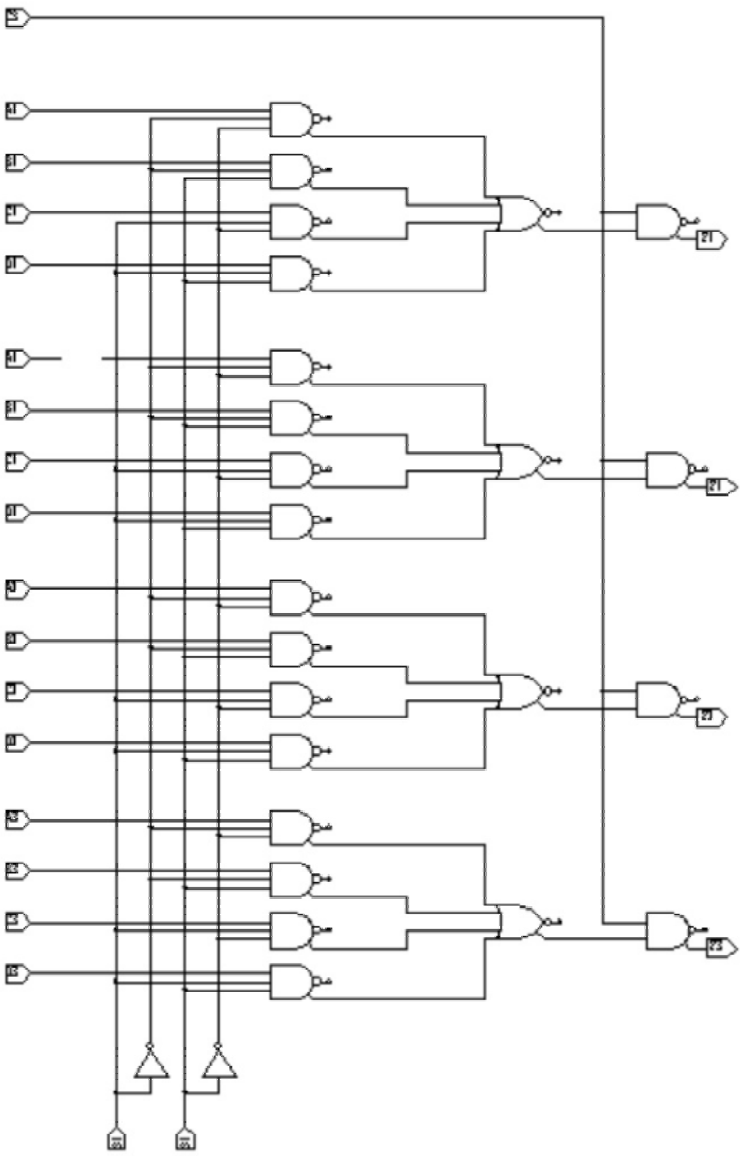


Figure 2.15 – Schematic of a 4-bit 4:1 Multiplexer used to implement the 4-bit 8:1 Multiplexer in the Output Selector module

The Output Controller itself has its inputs taken from the output of the command decoder plus one input from the output of the Half Comparator component of the **A XOR and Comparison** module.

For convenience, the outputs of the Output Controller are also locally named S_1, S_0, and CS. They are used to control the S_1, S_0, and CS inputs of the 4-bit 8:1 Multiplexer respectively.

The truth table shown in Table 2.4 is used to implement the Output Controller component.

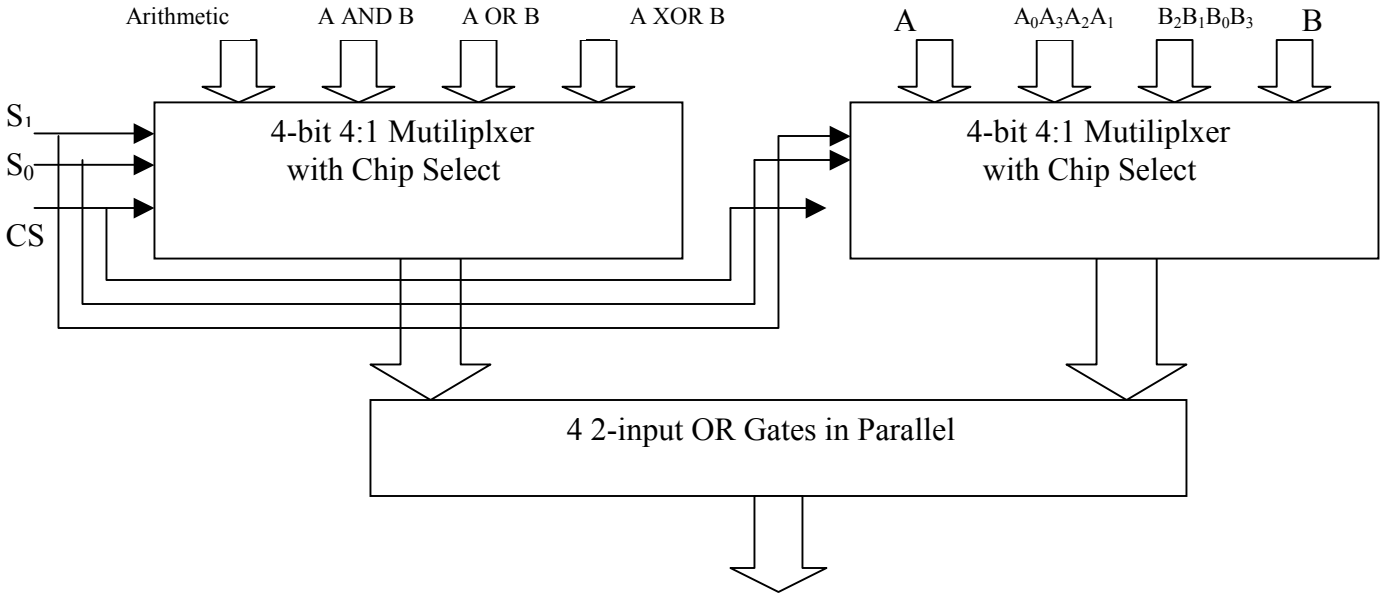


Figure 2.16 – Modified form of the schematic of the 4-bit 8:1 Multiplexer used in the Output Selector module

Operation	S1	S0	CS	Output from
P0	1	1	1	Arithmetic module
P1	0	1	0	B circular left shift
P2	1	1	1	Arithmetic module
P3	1	1	1	Arithmetic module
P4	1	1	1	Arithmetic module
P5	1	1	1	Arithmetic module
P6	1	1	1	Arithmetic module
P7	1	1	1	Arithmetic module
P8	1	1	1	Arithmetic module
P9	0	1	1	A OR B
P10	1	0	0	A circular right shift
P11	1	0	1	A AND B
P12	0	0	1	A XOR B
P13	fA	fA	0	A / B depending on fA
P14	1	1	1	Arithmetic module
P15	1	1	1	Arithmetic module

Table 2.4- Truth table used to implement the Output Controller component of the Output Selector module

From Table 2.4, we see that $CS = P_1'.\ P_{10}'.\ P_{13}' = (P_1 + P_{10} + P_{13})'$, $S_1 = P_1'.\ P_9'.\ P_{12}' = (P_1 + P_9 + P_{12})'$ if $P_{13} = 0$, $S_1 = fA$ if $P_{13} = 1$. Similary, $S_0 = P_{10}'.\ P_{11}'.\ P_{12}' = (P_{10} + P_{11} + P_{12})'$ if $P_3 = 0$, and $S_0 = fA$ if $P_{13} = 1$.

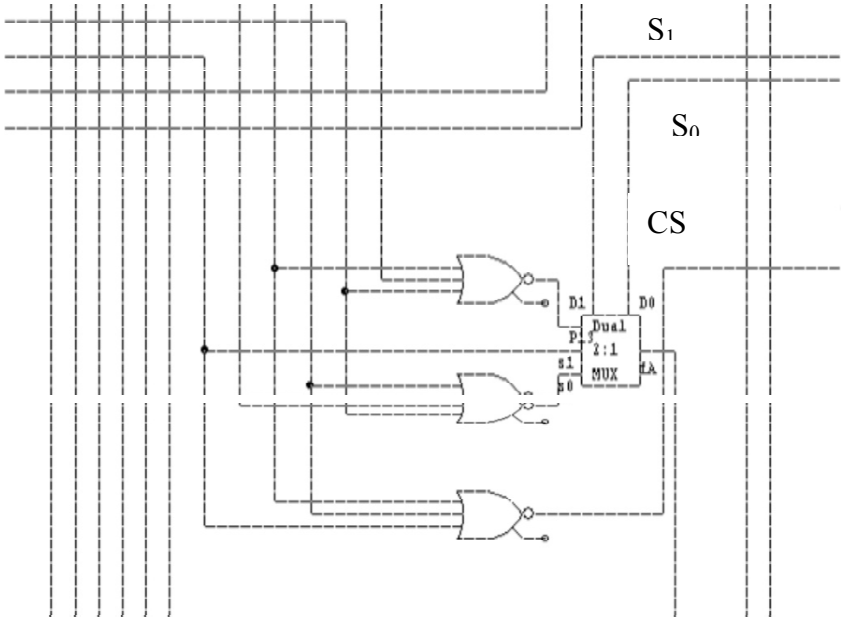


Figure 2.17- Implementation of the Output Controller

The implementation of the Output Controller is shown in Figure 2.17, in which we use a Dual 1-bit 2:1 Multiplexer controlled by P_{13}. If $P_{13} = 0$, it lets $S_1 = (P_1 + P_9 + P_{12})'$ and $S_0 = (P_{10} + P_{11} + P_{12})'$ pass to D_1 and D_0 outputs, respectively, to control the 8:1 Multiplexer.

The implementation of the Dual 1-bit 2:1 Multiplexer is shown in Figure 2.18. As shown from the

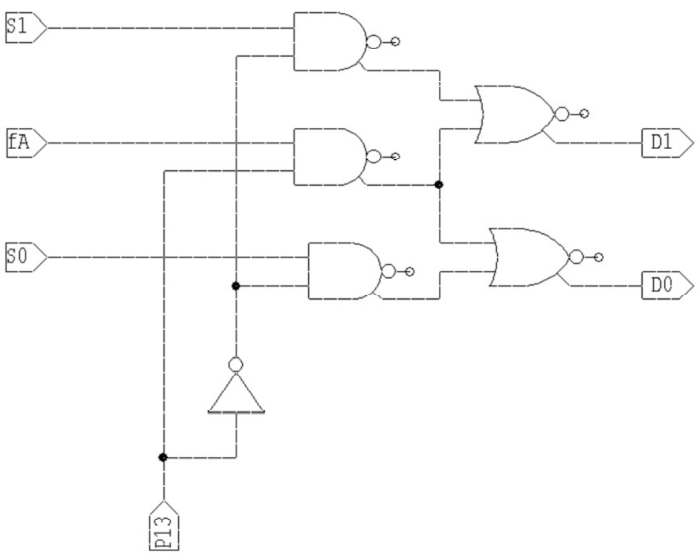


Figure 2.18- Schematic of the Dual 1-bit 2:1 Multiplexer used by the Output Controller component of the Output Selector module

figure, the input P_{13} will control the operation of the multiplexers. When P_{13} is 0, the inputs S_1 and S_0 are passed to the outputs D_1 and D_2 respectively. The inputs S_1 and S_0 will be connected to the logic gates that implement the functions $S_1 = (P_1 + P_9 + P_{12})'$, and $S_0 = (P_{10} + P_{11} + P_{12})'$ respectively. The outputs D_1 and D_0 will be used to control the control signal S_1 and S_0 from the 4-bit 8:1 Multiplexer. The CS control signal of the 4-bit 8:1 component will be controlled by a logic gate implementing the expression $CS = (P_1 + P_{10} + P_{13})'$ as shown in Figure 2.18. The whole schematic of the Output Selector is given in Figure 2.19.

Implementation of the Carry and Overflow module.

Before describing in detail the design of the Carry and Overflow module, we need to give some comments on the meaning of the Carry and Overflow bits as well as on how their values are set up.

- Because we are using complement-two form to perform arithmetic and each of operand is 4-bit number. The most significant bit will be the sign bit. So we can represent numbers in the range of +7 to –8.
- The Carry bit (C) will be set to 1 whenever we have a result of an arithmetic operation greater than +7. In that case, in addition to the Carry bit being 1, the sign bit of the result will be changed to 0.
- The Overflow bit (V) will be set to 1 whenever the result is either greater than 15 or less than –8. However, with three bits of magnitude in our particular problem, the result will never be greater 15. As a result, for this particular design, the Overflow bit will be set to 1 whenever the result is less than –8.

With that in mind, we can derive logic expressions for C bit and V bit as follows:

For the Carry (C) bit:

i) $A + B : P_7 . A_3' . B_3' . D_3$ (Addition of two positive numbers, result > 7)

ii) A + 1 : P_4. A_3'. A_2. A_1. A_0 (Incrementing A, which is already 7)
iii) B + 1: P_6. B_3'. B_2. B_1. B_0 (Incrementing B, which is already 7)
iv) A – B: P_2. A_3'. B_3. D_3 (Subtraction of a negative number from a positive number, the result is greater than 7).
v) – B: P_{14}. B_3. B_2'. B_1'. B_0' (Negation of –8)
vi) –A: P_{14}. A_3. A_2'. A_1'. A_0' (Negation of –8)

For the Overflow (V) bit:

i) A + B : P_7. A_3. B_3. D_3' (Addition of two negative numbers, result < -8)
ii) A – B : P_2. A_3. B_3'. D_3' (Subtraction of positive number from a negative number, the result is less than – 8).
iii) A – 1: P_5. A_3. A_2'. A_1'. A_0' (Decrementing A, which is currently equal to –8)
iv) B – 1: P_8. B_3. B_2'. B_1'. B_0' (Decrementing B, which is currently equal to –8)

For other arithmetic operations, neither C and V bits are set to 1 because, regardless of A and B are, the operation can't result in a number out of the range of [-8, +7]. For example, negation of a positive number less than 7 will never produce a negative number less than –8. Another example is incrementing a negative number, which is greater than –8, in which case, the result can never be less than –8 …

So the expression of C is: C = P_7. A_3'. B_3'. D_3 + P_4. A_3'. A_2. A_1. A_0 + P_6. B_3'. B_2. B_1. B_0 + P_2. A_3'. B_3. D_3 + P_{14}. B_3. B_2'. B_1'. B_0' + P_{14}. A_3. A_2'. A_1'. A_0'.

And the expression of V is: V = P_7. A_3. B_3. D_3' + P_2. A_3. B_3'. D_3' + P_5. A_3. A_2'. A_1'. A_0' + P_8. B_3. B_2'. B_1'. B_0'.

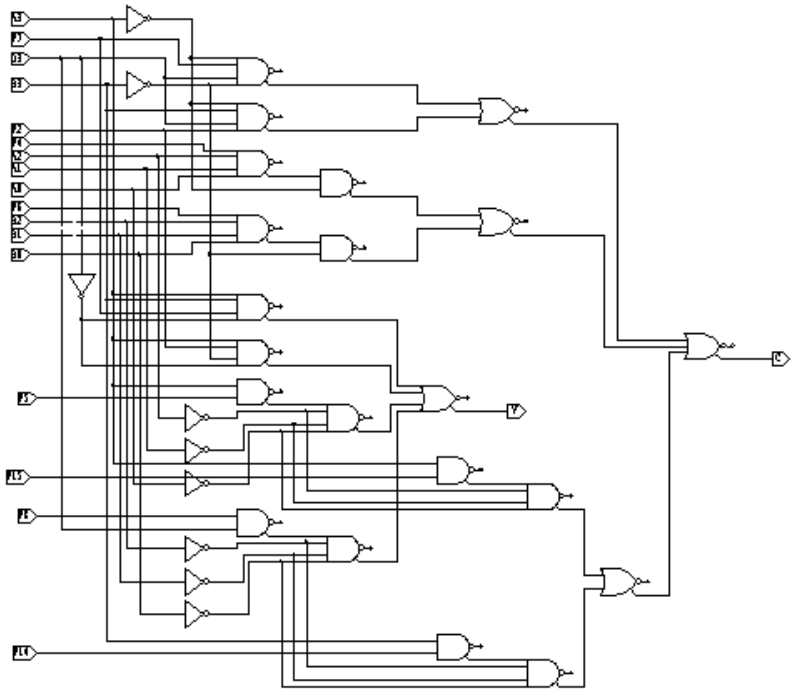


Figure 2.20 – Schematic of the Carry and Overflow module

The schematic of the Carry and Overflow module is shown in Figure 2.20.
One more thing to be mentioned about the Carry (C) bit is that: the Carry bit will be used to produce the most significant bit of the sum of the FA by just XORing them to give the final sign bit of the result.

Testing, Simulation, and Layout Results of Each Sub-Module

Due to the nature of each module's operations, some of them are very straightforward like the OR, AND, and XOR. Testing of these modules seems very trivial and prove unnecessary. Other modules, as described, are more complicated in structure because their requirements of the controller components. It turns out that simulation of them is very impractical and somehow out of context (meaning that the simulation is hardly close to the real operation of the circuit as a whole). For that reasons, we will choose to test the whole circuit as a complete simulation unit instead of testing and simulation of each individual module.
We also choose not to produce the layout for each individual module. This is to emphasize the integration of the ALU as a whole.

Implementation of the Synchronizing module

This module will include four 4-bit registers using D Flip-Flops with Edge-Trigger clock signal (to synchronize A, B, S, and the output R respectively) and two individual D Flip-Flops with Edge-Trigger clock signal to latch the values of C and V bits.
The clock inputs of the output Flip-Flops and register will be triggered using the down-edge of the clock signal used to synchronize the inputs A, B, and S. We do so by inverting the original clock signal and using that to drive the output register and Flip-Flops. Doing so, we can notice that the output should be available in half clock period since the input data are available and stay there for one clock period before the result of the next command and/or input data are sent to the ALU for a new operation. The schematic of a register used by this module is shown in Figure 2.21.

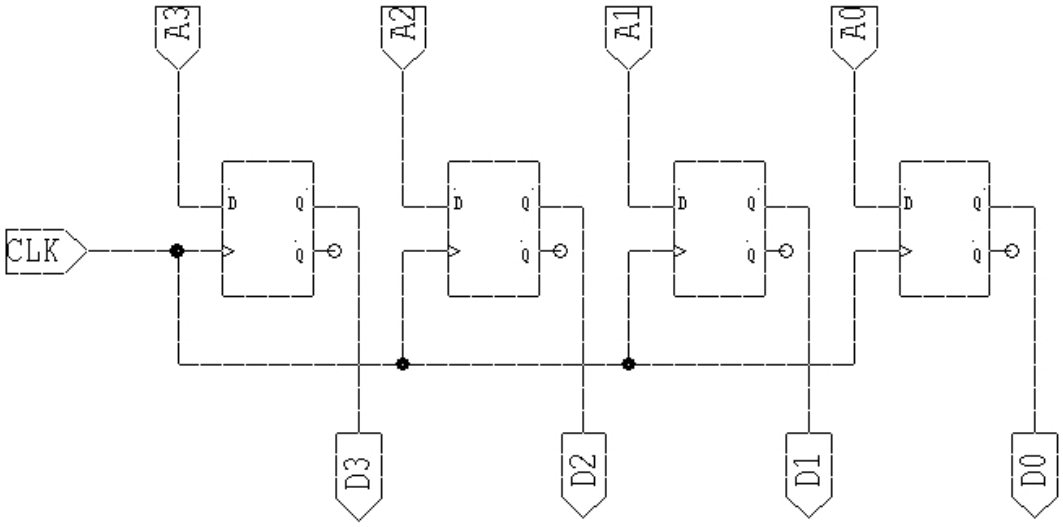


Figure 2.21- Schematic of a 4-bit register used to synchronize the data inputs and output.

III- Schematic of the Top-Level Design, Simulation, Testing, and Layout of the Top-Level Circuit

Combining all modules with all components described above, we will get the top-level design of the circuit. The whole circuit of the ALU is shown in Figure 3.1

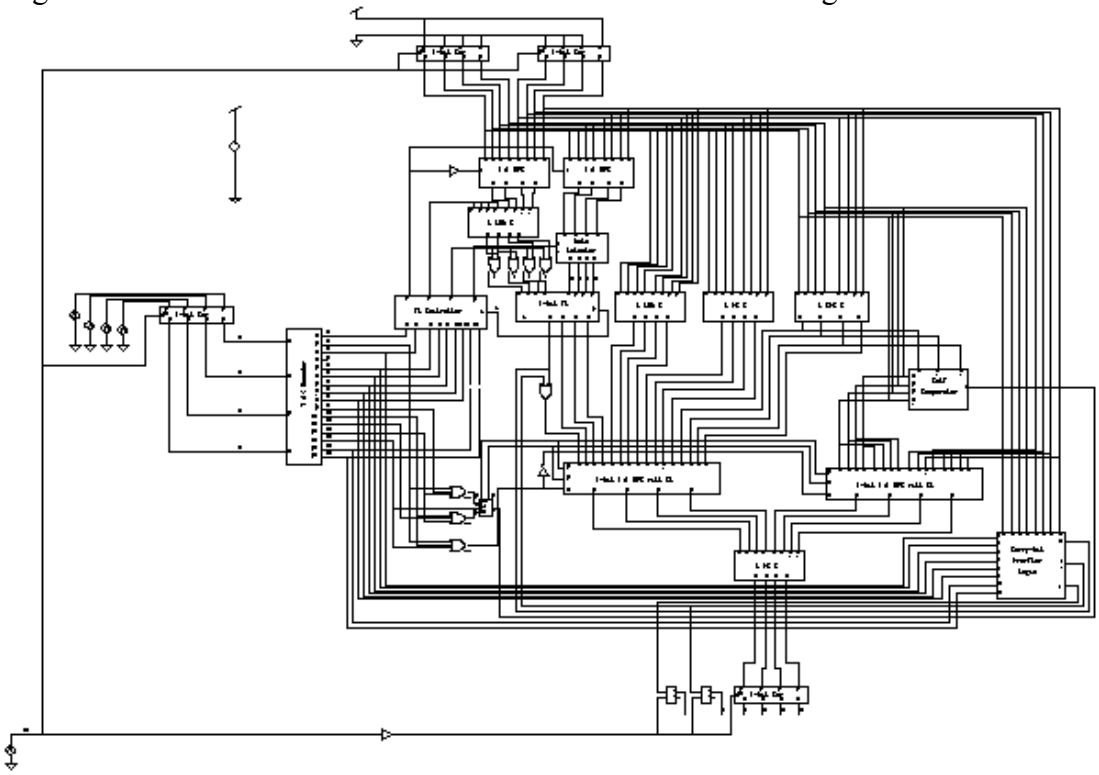


Figure 3.1- Schematic of the top-level circuit of the ALU.

Characterization and Testing of Top-Level Design

We will use two different sets of data inputs to test the circuit. For each of data set, we will apply the same command line sequence so that all 16 operations will have a chance to be done with each data set. (We will insure that one operation is done at least once by using a simulation length of at leat 16 clock signal.
We will use a clock signal with frequency of 100 MHz to test the circuit (so the period is set to 10 ns). This speed is three times faster than the required one; hence, if our circuit is successfully tested under this circumstance, it apparently meets the stated requirement.
The two sets of data input are as follows:

Set 1: A = 1001, B = 0110.
Set 2: A = 0111, B = 0101

For the first set of data, the simulation result will be compared against the data shown in Table 3.1

Operation	Decoder Output	R	C	V
A	P0	1001	0	0
B circular left shift	P1	1100	0	0
A - B	P2	0011	0	1
B	P3	0110	0	0
A + 1	P4	1010	0	0
A - 1	P5	1000	0	0
B + 1	P6	0111	0	0
A + B	P7	1111	0	0
B - 1	P8	0101	0	0
A OR B	P9	1111	0	0
A circular right shift	P10	1100	0	0
A AND B	P11	0000	0	0
A XOR B	P12	1111	0	0
Max A, B	P13	1001	0	0
-B	P14	1010	0	0
-A	P15	0111	0	0

Table 3.1 Truth Table used to compare against the simulation result for the first data set (A = 1001, B = 0110)

For the second set of data, the simulation result will be compared against the data shown in Table 3.2

Operation	Decoder Output	R	C	V
A	P0	0111	0	0
B circular left shift	P1	1010	0	0
A - B	P2	0010	0	0
B	P3	0101	0	0
A + 1	P4	0000	1	0
A - 1	P5	0110	0	0
B + 1	P6	0110	0	0
A + B	P7	0100	1	0
B - 1	P8	0100	0	0
A OR B	P9	0111	0	0
A circular right shift	P10	1011	0	0
A AND B	P11	0101	0	0
A XOR B	P12	0010	0	0
Max A, B	P13	0111	0	0
-B	P14	1011	0	0
-A	P15	1001	0	0

Table 3.1 Truth Table used to compare against the simulation result for the second data set (A = 0111, B = 0101)

Simulation and Layout Results of Top-Level Design

- We will set up our data inputs as constant voltages although still using registers to synchronize the data. (We assume we have unsynchronized data)
- We will apply 4 pulse voltage sources at S_3, S_2, S_1, S_0 so that all 16 combinations of S_3, S_2, S_1, and S_0 are available during the simulation length under the clock signal frequency of 100 MHz.
- The clock signal will have a period of 10 ns.
- The simulation will use the `ml2_125.md` model file.
- As stated by the requirement, all designs will employ the SCMOS logic gate library that is in TannerLB and the layout will employ the Hewlett Packard 0.5µm n-well technology by using `mhp_n05d.tdb` file.

From the simulation result of the synchronized command signals S_3, S_2, S_1, and S_0, we can record the time mark for each data inputs and command. The time mark of each operation is shown in the following table (Table 3.3)

Operation	Decoder Output	Time Mark (ns)	Time Mark (ns) for Output Availability
A	P0	160-170	165-175
B circular left shift	P1	150-160	155-165
A - B	P2	140-150	145-155
B	P3	130-140	135-145
A + 1	P4	120-130	125-135
A - 1	P5	110-120	115-125
B + 1	P6	100-110	105-115
A + B	P7	90-100	95-105
B - 1	P8	80-90	85-95
A OR B	P9	70-80	75-85
A circular right shift	P10	60-70	65-75
A AND B	P11	50-60	55-65
A XOR B	P12	40-50	45-55
Max A, B	P13	30-40	35-45
-B	P14	20-30	25-35
-A	P15	10-20	15-25

Table 3.3 Time mark to monitor the outputs

All simulation results can be viewed in the section entitled *"Simulation Result of the Top-Level Design"*

The layout of the chip is shown in Figure 3.2

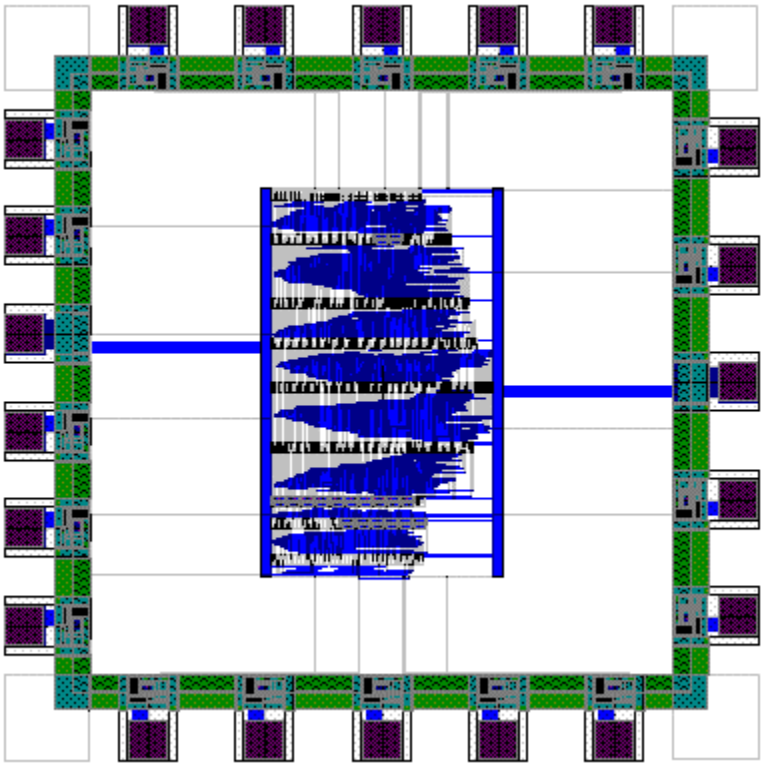


Figure 3.2 – Chip layout of the top-level design of the ALU

The layout is implemented using 1897 MOSFETs and takes a layout area of 4694760 (Internal Units).

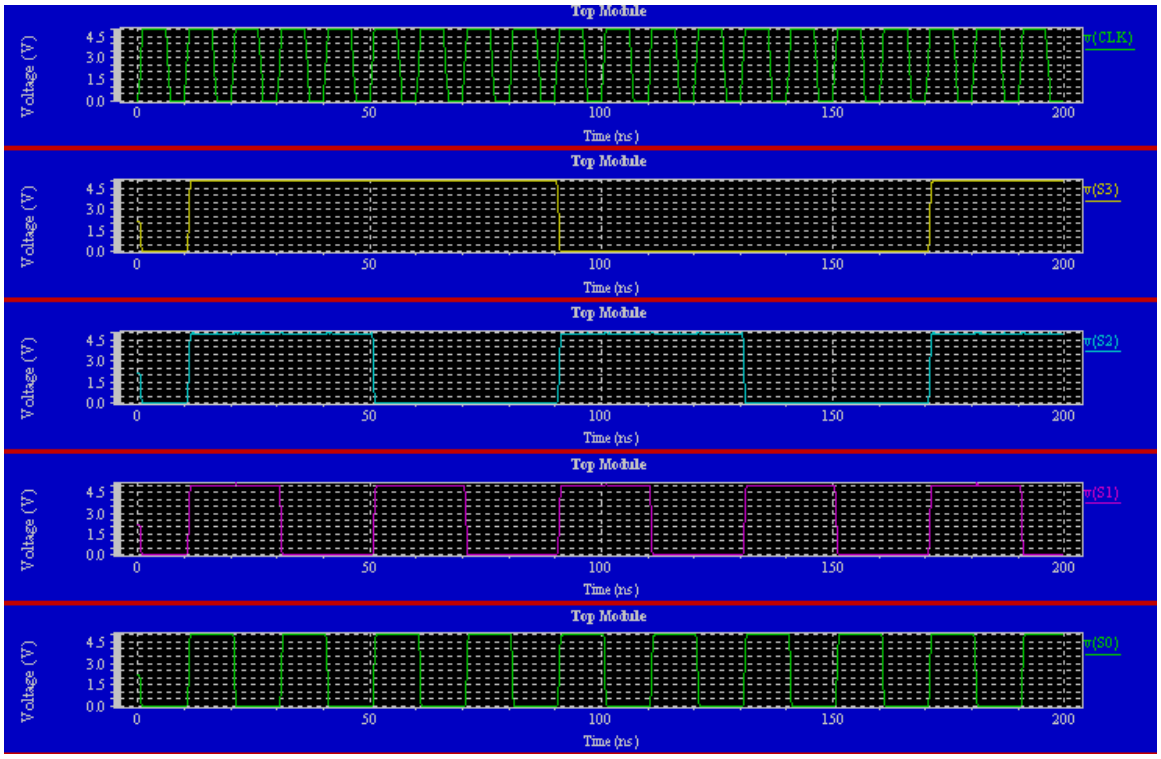
IV-Conclusions

According to the requirement of the project, we can say that all of the stated objectives have been accomplished successfully. Besides the requirements of using 5-V power sources in the designs and of employing the SCMOS logic gate library in TannerLB as well as employing the Hewlett Packard 0.5 μm n-well technology for the layout which have been properly applied in all simulation stages, all other objectives such as speed of the circuit; and most importantly the correct functionality of the ALU are also successfully met. However, there is an issue which is harder to assess in terms of a successful design. That is the goal of minimum chip area of the circuit. We have tried our best to minimize the area of the chip so that it can be more easily integrated into another chip. The assessment of this criteria is giving an estimated number of MOSFETs used in the design. This number is about 1890. We are not sure if this number reflects a good design or not. But accoring to our rule of design, that is, to share as many components as we can among various module and functions, to optimize logic functions of control lines as much as we can, and try to use as fewer gate in a certain design as we could, we are confident that that number reflects a quite good design. Under those terms, we can conclude that the design is completely and successfully done not only in correct functionality of the chip, but also in other goals aimed at optimizing the product as a whole.
In conducting this project, we also learned several things regarding to VLSI concepts and designing principles. In designing a VLSI circuit, we are dealing with various problems that are never surfaced for other type of design. We are more concerned with delay time in such a way that the functionality of the product will not be compromised when we try to integrate more and more gates into the chip. We can deal with that problem by balancing of gates between data paths. At the same time, we are equally concerned with the area of the chip. The conflicting nature of these two problems requires us to find a best way somewhere in between making their coexistence without compromising the performance of the circuit as a whole.

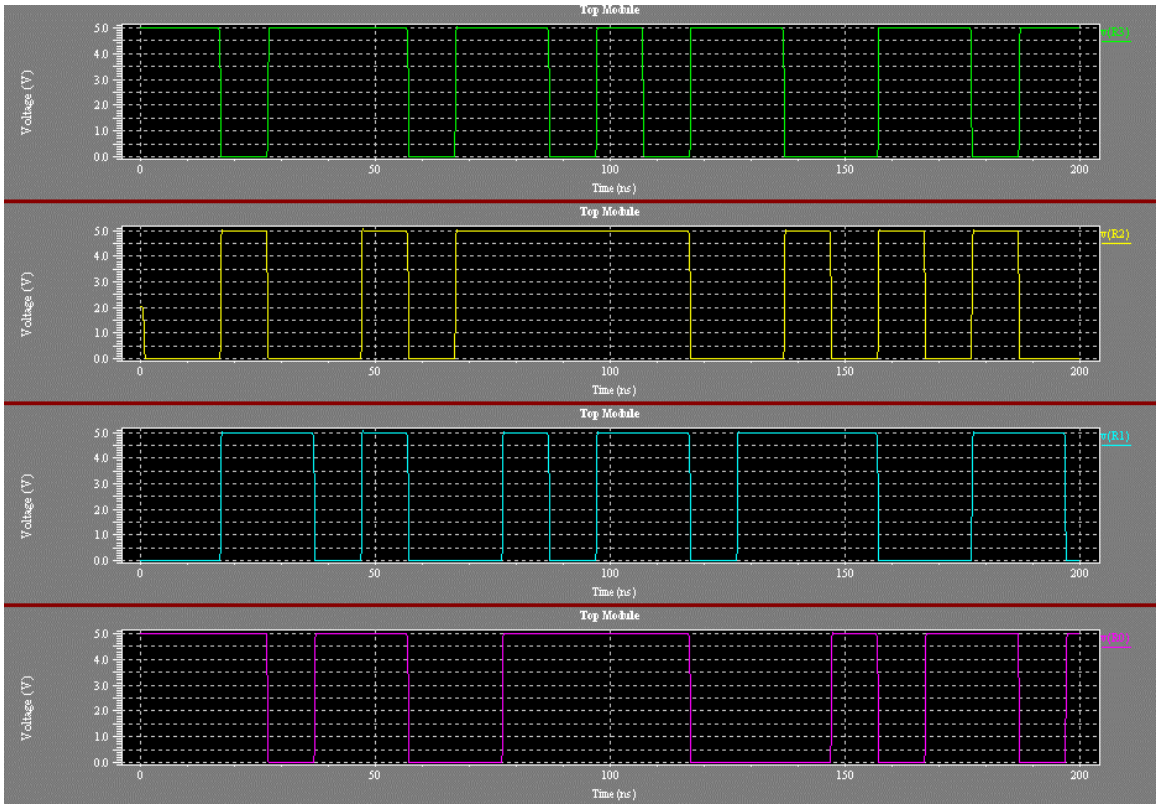
To conclude the project, we would like to say that this is an excellent but challenging project. We have accomplished all of its goals in a timely manner as well as in terms of functional quality.

Simulation of the Top-Level Design

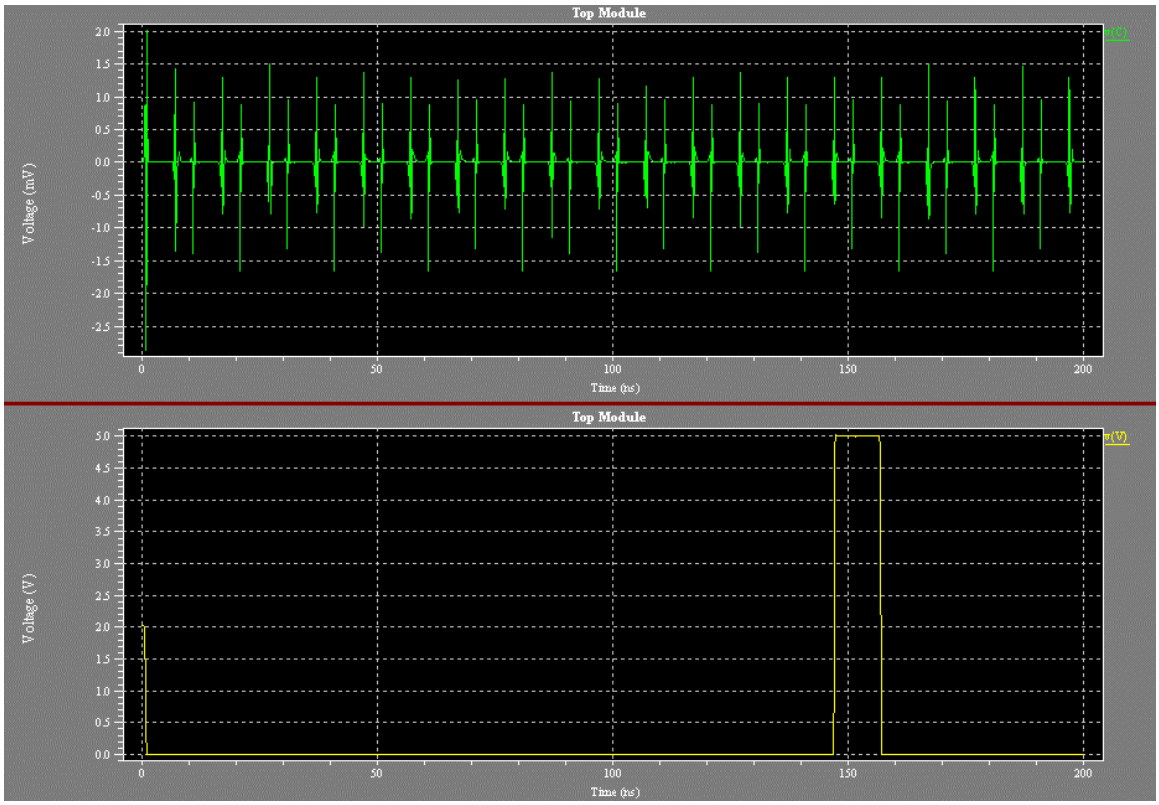
Waveforms of the Synchronized Command Inputs and the Clock
(From Top to Bottom: S3, S2, S1, S0 – all of these have been synchronized, and CLK)



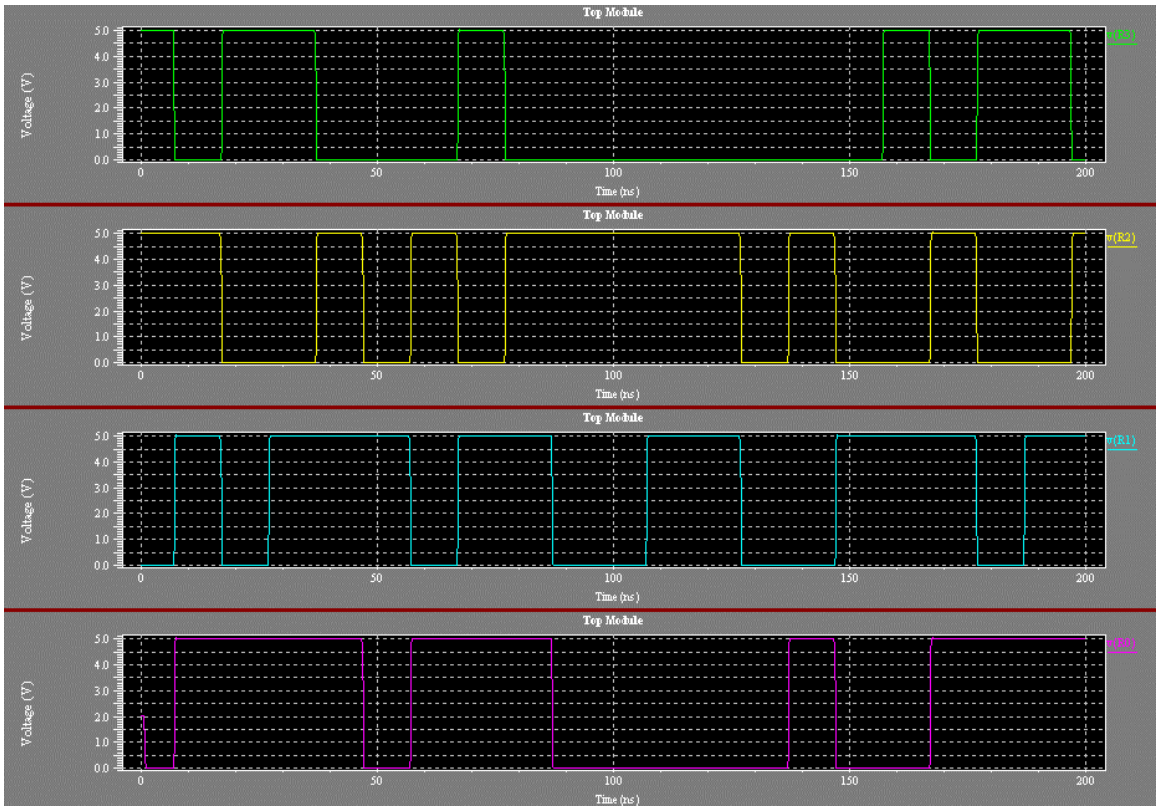
Waveforms of the Synchronized Output Data
for the First Set of Input Data (A = 1001, B = 0110)
(from Top to Bottom: R3, R2, R1, R0 – see Table 3.3 for Time Mark Reference)



Waveforms of the Synchronized Carry and Overflow bits Outputs
For the First Set of Input Data (A = 1001, B = 0110)
(from Top to Bottom: C, V – see Table 3.3 for Time Mark Reference



Waveforms of the Synchronized Output Data
for the Second Set of Input Data (A =0111, B = 0101)
(from Top to Bottom: R3, R2, R1, R0 – see Table 3.3 for Time Mark Reference)



Waveforms of the Synchronized Carry and Overflow bits Outputs
For the Second Set of Input Data (A = 0111, B = 0101)
(from Top to Bottom: C, V – see Table 3.3 for Time Mark Reference