RECONFIGURABLE SYSTEMS PRINCIPLE

EECE8020

ALU TECHNICAL DESIGN

EMBEDDED SYSTEM DEVELOPMENT

SANTHOSH NAGENDRAN

8272767

09-10-2019

CONESTOGA

Connect Life and Learning

INTRODUCTION

This document contains information about building an ALU (Arithmetic and Logic Unit) of a processor using ModelSim software using VHDL and Verilog programming. And this document tells more details about creation of the ALU from scratch with some inputs so that the reader will have a good understanding in creating an ALU, process involved to reach the outcomes and to test the ALU and with some other features it helps us to find the errors and will have idea to debug and make some improvements. After reading this document users could able to reproduce an ALU and will also have idea to upgrade the existing model.

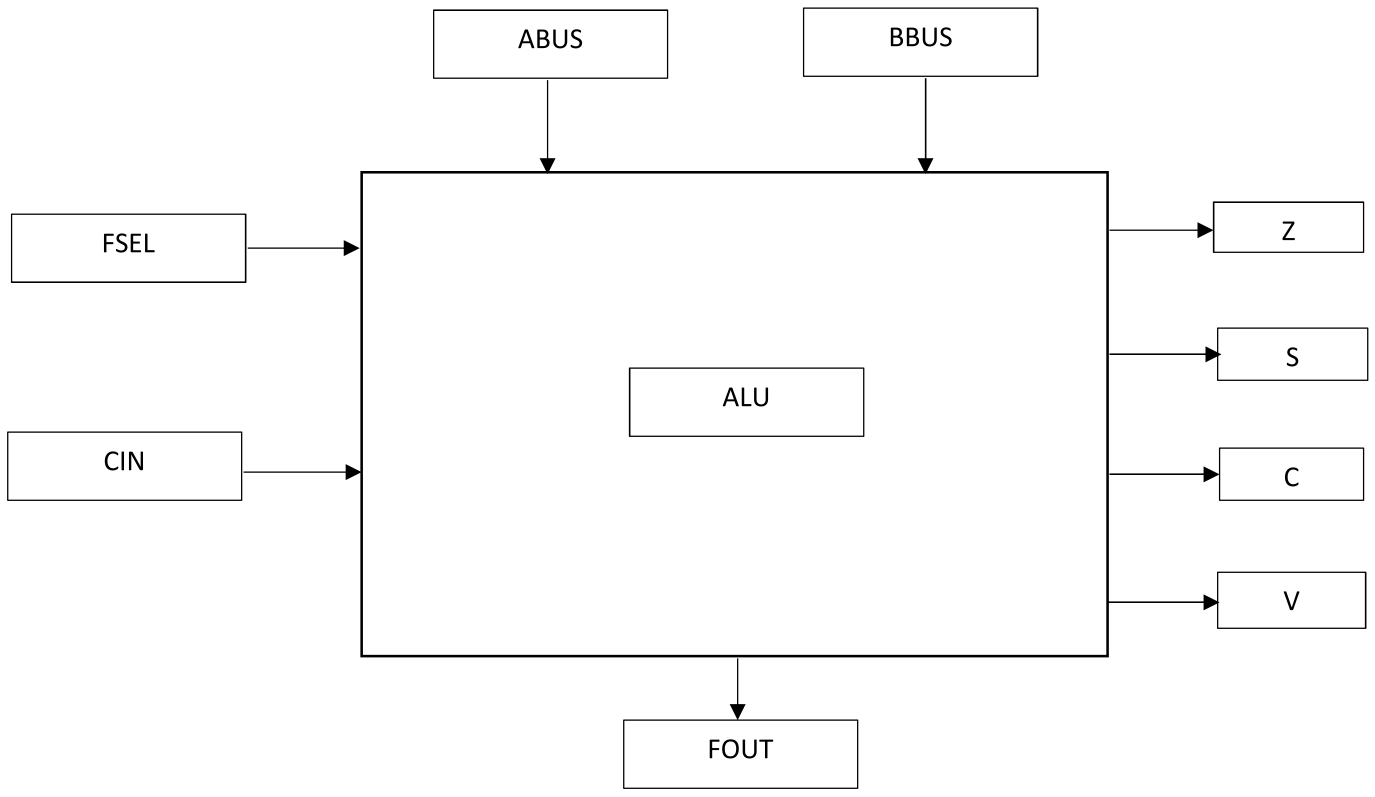
DESIGN OBJECTIVE

As the name itself says, the main objective of this design is to create an ALU which means Arithmetic and Logic, which is main part of the processor or a system. This is the place where all sort of arithmetic and logical operations of the entire processor will take place. To talk something on big picture, when we are designing a processor or a system which will be served to the users in the real time, we will be facing situations to process the data coming from the outside world. It could be either the data fed by the users to the processor or by any means of sensors and transducers. In those case to make the data as a sensible information we have to process the data, for doing so we need to have a unit called ALU. This part will be a best fit at those points.

OVERVIEW

BLOCK DIAGRAM

Ref.1.1 Mike's Slides



REQUIREMENTS AND SPECIFICATIONS

As a simple design and most common in practice and as every other system constrains, we need some inputs and outputs. Since we need to get the inputs from the users, and then process and to store the values from the variable and to select the task to perform inside the processor we need some more variables in addition to that, so the variables we need to create this design are as follows,

INPUTS: ABUS, BBUS, FSEL, CIN

OUTPUTS: FOUT, Z, C, S, V

The above mentioned are the general inputs and outputs of the design but to describe more about each variable, there are different specifications for each of them. The design developed in this part is a 16bit ALU which means the inputs and the outputs of the processor will be 16 bits wide. Only 16-bit data can be processed in this design. So, the main inputs and the outputs of the system are 16 bits wide. And the type of the data to be processed should also be looked carefully. Apart from this the status of the processor after certain process is done should be taken care, the selection of the task to be performed which will be 16 different types in our case. The types of the variables and the size used will be shown clearly below as follows,

ABUS [15:0]	– A input bus (unsigned)
BBUS [15:0]	– B input bus (unsigned)
FOUT [15:0]	– Function Output (unsigned)
FSEL [3:0]	– Function select Input (std_logic_vector)
Z, S, C, V	– Status flags output (std_logic)
CIN	– Carry In (std_logic)

Ref.1.2 Mike's Slides

While designing the processor these are all the important task which are to be taken care of. And another important point to be remembered is the name of the entity or the module to be created, it should be named only as ALU. This will not create any confusions in the future developments. The name of the design should be straight forward and easy to identify that way it will not create any problems in the future improvements and it will be easy to understand.

LOGIC IMPLEMENTATION

The logic implemented in this ALU design is nothing just to create a switch case statements for different task selection. So, when the user has to use any of the pre fixed tasks, they can choose it and with the task selection function, they can send the data to be processed. The prefixed task selection block will contain either an arithmetic or a logical expression with some assignments. When the corresponding task is selected and the user passes the data to be processed via busses (A and B) it will be processed with those pre-defined expressions and performs validation for the data sent into it. In this case I have chosen 16 different process to perform. This can be changed to something else based on the needs of the processor and the application. But these tasks involved are most common use and can be added any processors. The pre-defined tasks are as follows,

NAME	**CODE**	**FUNCTION**
TSA	0000	Transfer ABUS
INC	0001	Increment ABUS by one
DEC	0010	Decrement ABUS by one
ADD	0011	Add ABUS+BBUS+CIN
SUB	0100	Subtract ABUS-BBUS-CIN
AND	0101	Bitwise ABUS AND BBUS
OR	0110	Bitwise ABUS OR BBUS
XOR	0111	Bitwise ABUS XOR BBUS
NOT	1000	Bitwise NOT ABUS
SHL	1001	Shift ABUS left, C contains ABUS [15], FOUT [0] contains 0.
SHR	1010	Shift ABUS right, C contains ABUS [0], FOUT [15] contains 0.
ASR	1011	Arithmetic Shift A right, Bit C contains ABUS [0]
RLC	1100	Rotate Left through Carry, FOUT [0] contains CIN, C contains ABUS [15]
RRC	1101	Rotate Right through Carry, FOUT [15] contains CIN, C contains ABUS [0]
RV1	1110	Reserved 1
RV2	1111	Reserved 2

Ref.1.3 Mike's Slides

This table here is the truth table for the FSEL part. We already discussed that the variable FSEL is used to select from different pre-defined tasks. So, the 16 tasks are given above and the 4-bit wide FSEL is represented in 4 bits format for the understanding. The used define function can be used to replace the 4 bits format to a name which will make more sense for the readers and those are all the names of the tasks as well. And it is self-explanatory. While performing these tasks we have to keep close eye towards the status flags which are most important part of the process to denote some extra information about the processed data and gives us immediate result which will be used for the further processing of the data and validation. The status flag logic is given below as follows,

S – is a copy of the MSB of FOUT
Z – is a '1' when FOUT == 0
C – is a '1' when there is a carry out from any operation, or it contains the bit shifted out from the input.
V – is a '1' when there is an overflow condition present after an operation. (Note: This also includes shifting operations.)

Ref.1.4 Mike's Slides

The ALU unit that is developed here is a combinational design, it doesn't matter about the clock cycles. So, there should be more attention on the sensitivity list while performing the process or the task.

IMPLEMENTATION METHOD

This part here is the most important part of all the above discussed parts. After considering all the above discussed topics, the implementation process should be done. The tool or the software that is done to implement this was “ModelSim” software with which either a VHDL or Verilog code can be written to make this work. The special thing about the software is, it contains the simulator with it which becomes handy to the testers and the developers to do something form the scratch. We can write the code and the compiler in it says whether the code written is right or wrong with some error messages. Once the code compiles good it can be taken further to make it work. When all the things work well it could be given to the manufacturers to make it as a real working chip and can be implemented in any of the processor or a real time system where these operations are needed.

ALGORITHM

- First create a file named ALU and select the language to be developed and name the module or entity as ALE as well.
- Create a define function to declare the name and the bit values of the FSEL variable to make it easy and simple.
- Declare the variable type, size and name as mentioned earlier.
- Create some temporary registers or signals to hold the results inside the blocks.
- Assign the temporary variables with the original variables.
- Create a simple switch case statement for the FSEL with the discussed truth table on the top.
- Inside all the cases include the expression and some assignment statements to perform the specified task.
- Also make some assignments for the status flag.
- Make sure to test each and every single case of the block works by applying several test functions which holds different combinations and values of the busses and the selection lines.
- Compare the result with the actual or calculated result.

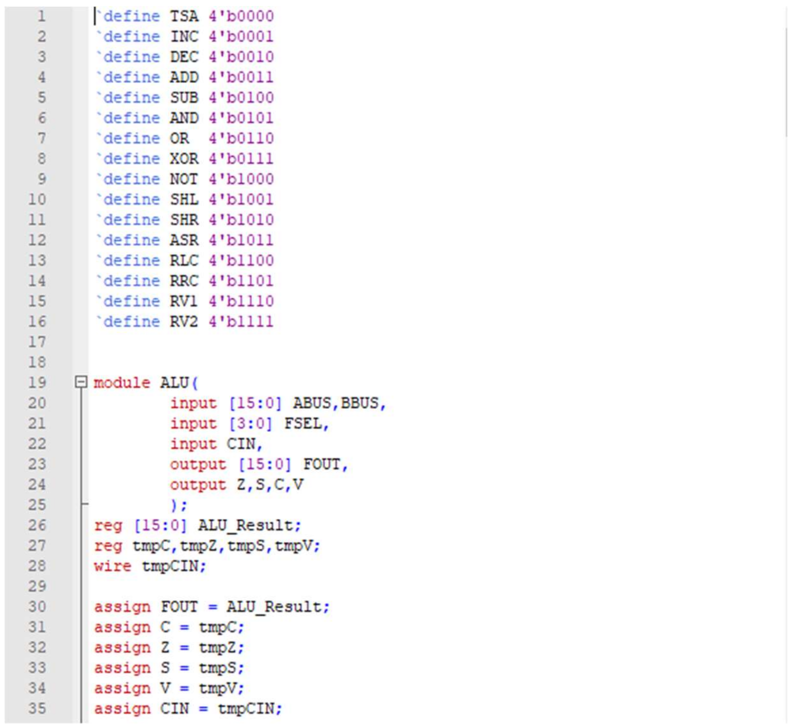
VERIFICATION AND VALIDATION

After creating the actual ALU itself, we will be creating a test bench to test the work. Verifying the work will take time and most important thing to do. Once when we develop the processor, when it is given to the manufacturer it will be converted into a chip where we cannot make any changes if there are any problem in the process. So, before hand we have to create a test bench to test every single logic and possibility. Make sure to add some print statements to show where, what and when the test is happening, which will be helpful for us to debug in case.
So, in the test bench file, we will be creating a task where it has some assignment statements to send the data from the testers to the real busses. This task should pass the parameters as a function call in C programming where the parameters sent should be FSEL, CIN, ABUS and BBUS along with the results (FOUT, Z, C, S, V).

ALGORITHM

- Create the definition function for the FSEL with the name and bits easy to read.
- Create the module or entity named as ALU Test file.
- Make some declaration of the variables, name and size.
- Instantiate the ALU here inside the test bench.
- Create the task and inside that include some assignment statements.
- Put some time domain function to make it run inside the simulator with one step higher for testing the ALU.
- Create some comparison statement to check the real output of the process with the calculated output.
- Add some print statements showing pass or fail.
- Add as much as test vectors in the rest of the file.
- Compile and run it to see it in the simulator for the inference and development.

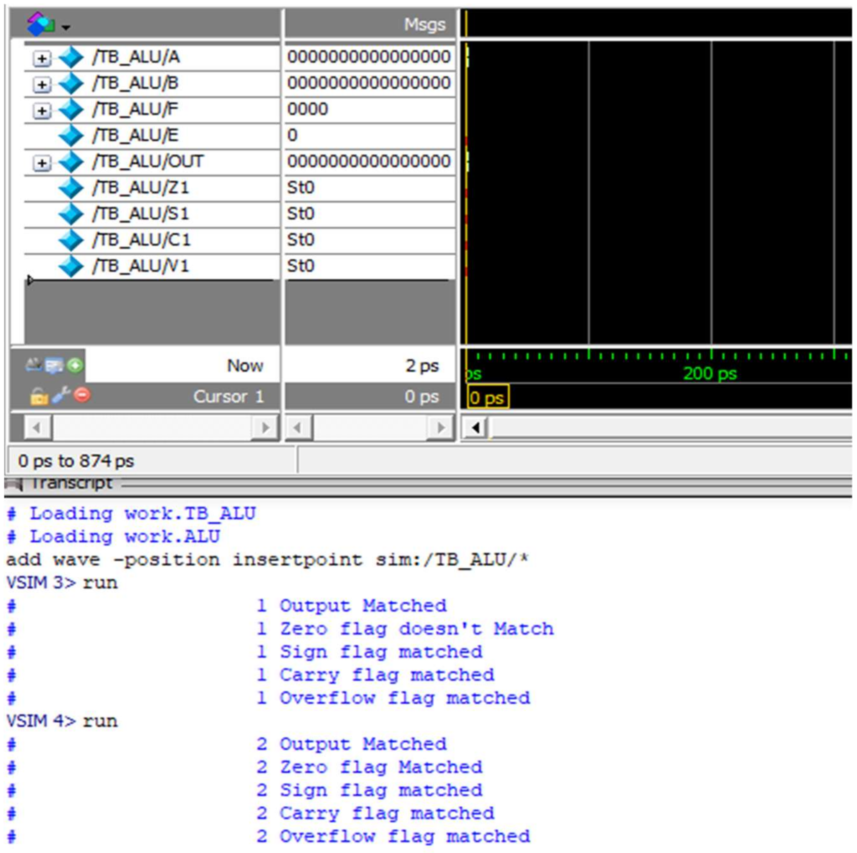
Make a note to see how many possibilities passes with all the outputs of the FOUT and the status flag. Improve the logic expression inside the ALU to make it work or pass in all the test vectors.



```
`define TSA 4'b0000
`define INC 4'b0001
`define DEC 4'b0010
`define ADD 4'b0011
`define SUB 4'b0100
`define AND 4'b0101
`define OR  4'b0110
`define XOR 4'b0111
`define NOT 4'b1000
`define SHL 4'b1001
`define SHR 4'b1010
`define ASR 4'b1011
`define RLC 4'b1100
`define RRC 4'b1101
`define RV1 4'b1110
`define RV2 4'b1111


module ALU(
        input [15:0] ABUS,BBUS,
        input [3:0] FSEL,
        input CIN,
        output [15:0] FOUT,
        output Z,S,C,V
        );
reg [15:0] ALU_Result;
reg tmpC,tmpZ,tmpS,tmpV;
wire tmpCIN;

assign FOUT = ALU_Result;
assign C = tmpC;
assign Z = tmpZ;
assign S = tmpS;
assign V = tmpV;
assign CIN = tmpCIN;
```



```
# Loading work.TB_ALU
# Loading work.ALU
add wave -position insertpoint sim:/TB_ALU/*
VSIM 3> run
#                     1 Output Matched
#                     1 Zero flag doesn't Match
#                     1 Sign flag matched
#                     1 Carry flag matched
#                     1 Overflow flag matched
VSIM 4> run
#                     2 Output Matched
#                     2 Zero flag Matched
#                     2 Sign flag matched
#                     2 Carry flag matched
#                     2 Overflow flag matched
```

BENEFIT ASSUMPTIONS RISKS/ISSUES

Common challenge that anyone will face while developing this processor is that, the error while compiling the code at some point which is most common as a starter to a language and it will be alright over the time. These are the challenges that I faced when I was creating the processor and it is discussed in the below as follows,

- After creating the ALU file and the test bench file, at the time of simulation it dint work even after including the time point in the test bench, the reason was I dint add the variables in the sensitivity list.
- There is no need to declare the variable type as unsigned in the Verilog because it will show warning as missing some brackets or semi colon.
- When we are writing a code inside any cases or inside the decision-making statements, it is advisable to include begin and end which is most equivalent to curly braces in C programming. Failed to include this will not allow you to write something inside the statements.
- There is one more way to do the test bench test by itself, it is function in Verilog instead of tasks. But once bad thing in function in Verilog is, it could not be used without the return type. The function is Verilog comes only with the return value, which will not be used in our case. So, it is not that we cannot use function but we will be wasting the memory and time of the processor.

REFERENCE SECTION AND RESOURCES

The resources and the reference are made in the Mike's Slides. The slides of Mike clearly explained the objective, block diagram and the description along with the variables used in the processor, size and type.

Ref.1.1 – Mike's Slides Block Diagram
Ref.1.2 – Mike's Slides Variable name, size and type
Ref.1.3 – Mike's Slides Define functions with the data
Ref.1.4 – Mike's Slides Status Flag explanations