Design of a 4-bit ALU Using VHDL CENG 3511

Mike Moore

December 5, 2013

Date Performed: November 17 2013
Instructor: Dr. George Collins

Abstract

The objective of the project is to implement a four-bit arithmetic logic unit for a processor. The basic requirements include the following fundamental operations: addition, increment, decrement, transfer, and the basic logical operations (AND, OR, NOT, XOR). A behavioral approach is taken to implement this ALU. The VHDL defines inputs for two four bit registers, a four bit op-code field, and a single carry in bit. The outputs are the four bit data out, and the single carry out bit. From a block diagram of the ALU design, a VHDL module is developed to satisfy the design requirements. This VHDL module is then put under test and the results are examined.

Contents

1 Introduction

An Arithmetic Logic Unit (ALU) is a fundamental component within the central processing unit (CPU). It is responsible for both the arithmetic and logical operations of the CPU, and as such, it is responsible for a large part of the CPU work load. Modern ALU designs can be quite complex, and often they are not actually a single unit within the CPU. For simplicity, this report focuses on a simplified four bit ALU design.

2 Preliminary Design

The design begins with an evaluation of a list of design requirements. The requirements include a table of operations that the ALU must implement. From this table, a set of validation criteria can be derived. This validation criteria directly leads to a set of tests that can be used to validate that the ALU performs all of the desired functions.

2.1 ALU Requirements

The ALU was designed to meet the following list of requirements:

- The unit has two 4-bit inputs, register A and register B.
- The unit must be able to add, increment, decrement, transfer and also perform basic logic operations AND, OR, NOT, XOR.
- The unit will have 4-bit select line called Operation Select, which would direct the unit as to which operation to perform.
- In addition the unit must also be able to do shift operations (right shift, left shift etc).
- The unit has a Carry-in and also has Carry-out.

We see from the above requirements list that we will need to design an ALU that has a total of thirteen input bits. Eight will be used for the two four bit operand registers, and another four will be used for the op-code. Finally, a single carry in bit is also required. The outputs are a little bit simpler. We will only need four bits for the data out, and a single bit for the carry out.

2.2 ALU Block Diagram

With this understanding of our requirements, the next step is to depict the ALU in a functional block diagram form. This will later allow for us to more easily translate the preliminary design to a VHDL module. The block diagram description of our ALU is provided below. Note how the inputs and outputs directly map to statements in the list of requirements.

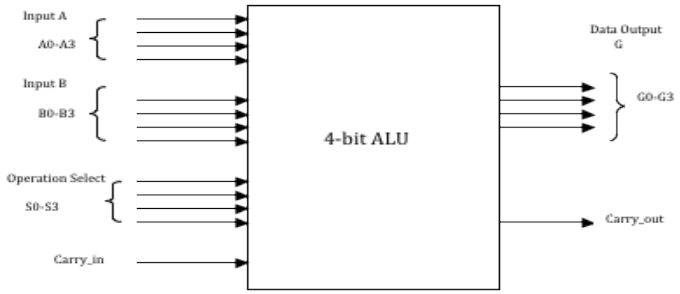

Figure 1: Four Bit ALU Block Diagram

2.3 ALU Operations

The required operations for the ALU are listed below.

Operation Select				Cin	Operation	Function
S3	S2	S1	S0			
0	0	0	0	0	G=A	Transfer A
0	0	0	1	1	G=A+1	Increment A
0	0	1	0	0	G=A+B	Addition
0	0	1	1	1	G=A+B+1	Add with Carry of 1
0	1	1	0	0	G=A-1	Decrement A
0	1	1	1	1	G=A	Transfer A
1	0	0	1	x	G= A.B	A and B
1	0	1	0	x	G= A+B	A or B
1	0	1	1	x	G=A(+)B	A xor B
1	1	0	x	x	Shift A by 1-bit right (padding 0's after shifting)	Right Shift
1	1	1	x	x	Shift A by 1-bit Left (padding 0's after shifting)	Left Shift

Figure 2: Four Bit ALU Required Operations

3 The ALU VHDL Module

The next step in the ALU design process was to actually write out the VHDL to perform the desired functions. A behavioral model is used to implement the ALU. This makes the code more readable at the expense of obfuscating the underlying hardware implementation. A structural model would be more revealing in that case, but for the sake of code readability and ease of implementation, a behavioral model was decided upon for this design.

3.1 ALU VHDL Code

The most noteworthy aspect of the VHDL module that was developed is the switch case statement that implements the ALU operations table in section 2.3. This switch case evaluates the four input op-code bits and performs the desired operation on the two register inputs. Another thing to take notice of is the use of standard logical vectors for the four bit operands and op-code. The data out also utilizes a standard logical vector. This simplifies the code and makes it much more readable. However, it does require a careful interpretation of the test bench waveform displayed in the results section.

Listing 1: Four Bit ALU VHDL Code

```
----------------------------------------------------
-- Engineer: Mike Moore
--
-- Create Date:    13:24:51 10/31/2013
-- Module Name:    FourBitALU - Behavioral
-- Project Name:   UHCL CENG 3511 Final Lab Project
-- Description:    VHDL Code for a four bit ALU.
--
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity FourBitALU is
    Port ( registerA : in std_logic_vector(3 downto 0)
        ;
           registerB : in std_logic_vector(3 downto 0)
              ;
           opCode : in std_logic_vector(3 downto 0);
           carryIn : in  STD_LOGIC;
           dataOut : out std_logic_vector(3 downto 0);
           carryOut : out  STD_LOGIC);
end FourBitALU;

architecture Behavioral of FourBitALU is
```

```
    signal Temp: std_logic_vector(4 downto 0);
begin
    process(registerA, registerB, opCode, Temp, carryIn
        ) is
    begin
        carryOut <= '0';
        case opCode is
          when "0000" => --dataOut = A (Cin = 0)
            dataOut <= registerA;
          when "0001" => --dataOut = A + 1
            dataOut <= std_logic_vector(unsigned(
                registerA) + 1);
          when "0010" => --dataOut = A + B
            Temp   <= std_logic_vector((unsigned("0" &
                registerA) + unsigned(registerB)));
            dataOut <= Temp(3 downto 0);
            carryOut   <= Temp(4);
          when "0011" => --dataOut = A + B + 1
            Temp   <= std_logic_vector((unsigned("0" &
                registerA) + unsigned(registerB)) + 1);
            dataOut <= Temp(3 downto 0);
            carryOut   <= Temp(4);
          when "0110" => --dataOut = A - 1
            dataOut <= std_logic_vector(unsigned(
                registerA) - 1);
          when "0111" => --dataOut = A (Cin = 1)
            dataOut <= registerA;
          when "1001" => --dataOut = A and B
            dataOut <= registerA and registerB;
          when "1010" => --dataOut = A or B
            dataOut <= registerA or registerB;
          when "1011" => --dataOut = A xor B
            dataOut <= registerA xor registerB;
          when "1100" => --dataOut = A shift logical
             right one bit
            dataOut <= std_logic_vector(unsigned(
                registerA) srl 1);
          when "1101" => --dataOut = A shift logical
             right one bit
            dataOut <= std_logic_vector(unsigned(
                registerA) srl 1);
          when "1110" => --dataOut = A shift logical
             left one bit
            dataOut <= std_logic_vector(unsigned(
                registerA) sll 1);
```

```
        when "1111" => --dataOut = A shift logical
            left one bit
          dataOut <= std_logic_vector(unsigned(
              registerA) sll 1);
        when others =>
          dataOut <= "0000";
      end case;
    end process;
end Behavioral;
```

3.2 ALU VHDL Test Code

With the primary ALU VHDL module developed, we turn to creating a test to verify the ALU implementation against the requirements listed in section 2. This test will be implemented as a VHDL module so that we can use the Xilinx ISE Design Suite Simulation Test Bench to run a simulated test of our ALU. The test will perform one or two checks for each operation in the ALU operations table defined in section 2.3. Test inputs and expected outputs are clearly listed in the comments of the test code.

Listing 2: Four Bit ALU Test Code

```
--------------------------------------------------
-- Engineer: Mike Moore
--
-- Create Date:   13:43:31 10/31/2013
-- Design Name:
-- Module Name:   FourBitALU/TestFourBitALU.vhd
-- Project Name:  UHCL CENG 3511 Four Bit ALU
--
--------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
ENTITY TestFourBitALU IS
END TestFourBitALU;

ARCHITECTURE behavior OF TestFourBitALU IS
    -- Component Declaration for the Unit Under Test (
       UUT)
    COMPONENT FourBitALU
    PORT(
         registerA : IN  std_logic_vector(3 downto 0);
         registerB : IN  std_logic_vector(3 downto 0);
         opCode : IN  std_logic_vector(3 downto 0);
         carryIn : IN  std_logic;
```

```
         dataOut : OUT  std_logic_vector(3 downto 0);
         carryOut : OUT  std_logic
        );
    END COMPONENT;

   --Inputs
   signal registerA : std_logic_vector(3 downto 0) :=
      (others => '0');
   signal registerB : std_logic_vector(3 downto 0) :=
      (others => '0');
   signal opCode : std_logic_vector(3 downto 0) := (
      others => '0');
   signal carryIn : std_logic := '0';
  --Outputs
   signal dataOut : std_logic_vector(3 downto 0);
   signal carryOut : std_logic;
BEGIN
  -- Instantiate the Unit Under Test (UUT)
   uut: FourBitALU PORT MAP (
          registerA => registerA,
          registerB => registerB,
          opCode => opCode,
          carryIn => carryIn,
          dataOut => dataOut,
          carryOut => carryOut
        );

   -- Stimulus process
   stim_proc: process
   begin
      wait for 100 ns; --Expected Output: 0101
      registerA <= "0101";
      registerB <= "0000";
      carryIn <= '0';
      opCode <=  "0000"; --G=A
      wait for 100 ns; --Expected Output: 0010
      registerA <= "0001";
      registerB <= "0000";
      carryIn <= '1';
      opCode <= "0001"; --G=A+1
    wait for 100 ns; --Expected Output: 1111
      registerA <= "1010";
      registerB <= "0101";
      carryIn <= '0';
      opCode <= "0010"; --G=A+B
    wait for 100 ns; --Expected Output: 1111
```

```
        registerA <= "1010";
        registerB <= "0100";
        carryIn <= '1';
        opCode <= "0011"; --G=A+B+1
      wait for 100 ns; --Expected Output: 0000
        registerA <= "0001";
        registerB <= "0000";
        carryIn <= '0';
        opCode <= "0110"; --G=A-1
      wait for 100 ns; --Expected Output: 1111
        registerA <= "1111";
        registerB <= "0000";
        carryIn <= '1';
        opCode <= "0111"; --G=A
      wait for 100 ns; --Expected Output: 1001
        registerA <= "1001";
        registerB <= "1111";
        carryIn <= '0';
        opCode <= "1001"; --G=A and B
      wait for 100 ns; --Expected Output: 1100
        registerA <= "1000";
        registerB <= "1100";
        carryIn <= '0';
        opCode <= "1010"; --G=A or B
      wait for 100 ns; --Expected Output: 0100
        registerA <= "1001";
        registerB <= "1101";
        carryIn <= '0';
        opCode <= "1011"; --G=A xor B
      wait for 100 ns; --Expected Output: 0111
        registerA <= "1111";
        registerB <= "0000";
        carryIn <= '0';
        opCode <= "1100"; --shift right logical one bit
      wait for 100 ns; --Expected Output: 1110
        registerA <= "1111";
        registerB <= "0000";
        carryIn <= '0';
        opCode <= "1111"; --shift left logical one bit
      wait for 100 ns;
        wait;
    end process;
END;
```

4 Results

The Xilinx ISE Design Suite Simulation Test Bench was used to run the tests against the four bit ALU VHDL module. The results of the test are depicted below as a time history of the input and output waveform.

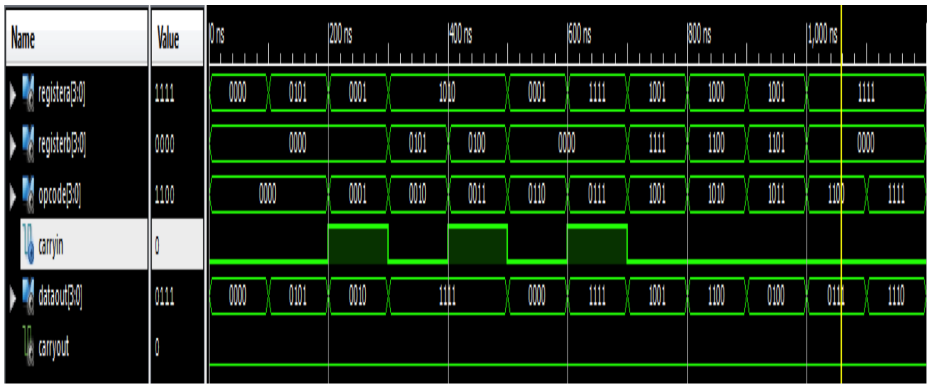

Figure 3: Four Bit ALU Required Operations

4.1 Verification of Waveform Plot

As an example, let's use this plot to verify our test of the addition operation of the ALU. Use the plot above to examine the input/output waveform at time 300ns. We can see that the desired op-code is 0010. This corresponds to the addition operation in the ALU operations table in section 2.3. The register inputs can be seen in the plot above. Register A is 1010, and register B is 0101. The expected sum of these two is 1111 with no carry out. If you examine the output waveform, you will see exactly that result. The remaining operations and their correctness can be evaluated in the same way.

5 Conclusions

The design and implementation of the simplified four bit ALU was carried out utilizing a logical progression that starts with requirement definition, moves on to preliminary design, VHDL implementation, and finally verification and validation using simulation. This process produces a design for a four bit ALU that can be taken to a hardware implementation on a CPLD. The verification and validation steps use simulation to bolster confidence that the design will work upon implementation on a CPLD. A hardware implementation was not listed in the project requirements, and as such, it is left as a follow on exercise. The ALU is a conceptually simple but nevertheless critical component in the CPU, and this exercise provided valuable insight into the CPU and computer architecture in general.