1 Project 5: Verilog Code Development – MIPS ALU Design

10 points

In the previous project (Project 4), you were introduced to the Verilog code development using the Cadence Xcelium tool. Next, we will use Verilog to modify the basic MIPS ALU of Project 4 and simulate this modification.

1. Create a copy of the **S23_MIPS_ALU_basic.v** file from Project 4. Similar to the MARS assignment, name your file using the first initial of your first name and the first 4 letters of your last name. For example, my ALU file name would be **tmani.v**.
2. **Add** the following functions to the ALU:
   a. Function code 0 to implement bitwise AND: A and B
   b. Function code 2 to implement ADD (A + B)
   c. Function code 6 to implement SUB (A - B)
   d. Function code 12 to implement bitwise NOR (A nor B)
   e. Function code 14 to implement bitwise XOR ($A \oplus B$)
3. Please download the following Verilog file:
   a. **S23_MIPS_ALU_soln_tb.v** - the testbench for testing your updated ALU
4. **Test your modified ALU using this testbench**
5. **Please include the following for your homework submission:**
   a. Your modified MIPS ALU Verilog file – **submit the actual *.v file so that the grader can run it.**
   b. Your testbench results – this can be a copy of the results in a Word document.
   c. PLEASE MAKE SURE THAT YOUR NAME APPEARS ON ALL SUBMITTED ITEMS FOR PROPER CREDIT

2 SOLUTION

2.1 Verilog code: modified MIPS ALU

An example of the Verilog code solution is the following:

```
// ===============================================================
// tmani.v
// T. Manikas    2022 Dec 27
//
//
// modified ALU (Arithmetic-Logic Unit) for MIPS processor
// ===============================================================

`timescale 1ns / 1ps

module MIPSALU (ALUctl, A, B, ALUOut, Zero);
      input [3:0] ALUctl;          // ALU control signals
      input [31:0] A, B;           // ALU input values (operands)
      output [31:0] ALUOut;        // result of ALU operation
      reg [31:0] ALUOut;
      output Zero;                 // ZERO flag

            // ZERO flag set if ALU result is 0

      assign Zero = (ALUOut==0);

            // do if any change in ALUctl, A, B

      always @(ALUctl or A or B)
      begin
         case (ALUctl)
           0:  ALUOut <= A & B;              // bitwise AND
           1:  ALUOut <= A | B;              // bitwise OR
           2:  ALUOut <= A + B;              // ADD
           6:  ALUOut <= A - B;              // SUB
           7:  ALUOut <= A < B ? 1 : 0;     // set if A < B
           12: ALUOut <= ~(A | B);         // bitwise NOR
           14: ALUOut <= A ^ B;            // bitwise XOR
           default:  ALUOut <= 0;
         endcase
      end
endmodule
```

2.2 Testbench results

```
 0 Zero=1 ALUctl= 0 A=00000000 B=00000000 ALUOut=00000000
 2 Zero=0 ALUctl= 0 A=0000000c B=00000004 ALUOut=00000004
 4 Zero=0 ALUctl= 0 A=0000000f B=00000006 ALUOut=00000006
 6 Zero=0 ALUctl= 1 A=0000000f B=00000006 ALUOut=0000000f
 8 Zero=0 ALUctl= 1 A=0000000c B=00000004 ALUOut=0000000c
10 Zero=0 ALUctl= 2 A=0000000c B=00000004 ALUOut=00000010
12 Zero=0 ALUctl= 2 A=00000001 B=00000004 ALUOut=00000005
14 Zero=0 ALUctl= 6 A=00000001 B=00000004 ALUOut=fffffffd
16 Zero=0 ALUctl= 6 A=0000000f B=00000004 ALUOut=0000000b
18 Zero=1 ALUctl= 7 A=0000000f B=00000004 ALUOut=00000000
20 Zero=0 ALUctl= 7 A=00000002 B=00000004 ALUOut=00000001
22 Zero=0 ALUctl=12 A=00000002 B=00000004 ALUOut=fffffff9
24 Zero=0 ALUctl=12 A=0000ffff B=00000004 ALUOut=ffff0000
26 Zero=0 ALUctl=14 A=0000ffff B=00000004 ALUOut=0000fffb
28 Zero=0 ALUctl=14 A=0000ffff B=00000000 ALUOut=0000ffff
```