Building an Integer ALU Final Step
CS 3339
Group: The Architects

Aaron Reed & Carlo Marroquin & Ryan Bieker

December 1, 2023

Instructor: Professor Klepetko

1 Objective

To introduce the process and methodology of designing a new computer circuit from scratch by coding 4-bit Binary Functions And, Nand, Or, Nor, Xor, Xnor, Not, and a Shifter. 4-bit arithmetic operations Addition, Subtraction, Multiplication all including carry in and carry out circuits. The ALU functions were then integrated into a control circuit. This circuit controls user input in hexadecimal coded in to choose which ALU function output to display..

2 Verilog Code

To design our Verilog control Circuit we decided on Operation Codes for each module. The Operation Codes in hexadecimal are:

AND	0
NAND	1
OR	2
NOR	3
XOR	4
XNOR	5
NOT	6
SHIFT	7
ADD	8
SUBTRACT	9
MULTIPLY	A

Table 1: Operation Codes for ALU Modules

3 Control Design

```
`include ".\add_4b\add_4b.v"
`include ".\and_4b\and_4b.v"
`include ".\mult_4b\mult_4b.v"
`include ".\nand_4b\nand_4b.v"
`include ".\nor_4b\nor_4b.v"
`include ".\or_4b\or_4b.v"
`include ".\shift_4b\shift_4b.v"
`include ".\sub_4b\sub_4b.v"
`include ".\xnor_4b\xnor_4b.v"
`include ".\xor_4b\xor_4b.v"
`include ".\not_4b\not_4b.v"

module control;

  output [3:0] AndResult, NandResult, OrResult, NorResult,
   ↪  XorResult, XnorResult, NotResult, ShiftResult, AddResult,
   ↪  SubResult;
  output [7:0] MultResult;
  output Cout;
  input [3:0] x, y, operation;
  input CinResult;

  wire x, y, AndResult, NandResult, OrResult, NorResult,
   ↪  XorResult, XnorResult, NotResult, ShiftResult, AddResult,
   ↪  SubResult, MultResult;

endmodule
```

4 Control Testbench

```
module control_test;

/* Make a regular pulsing clock. */
reg clk = 0;
always #20 clk = !clk;

reg [3:0] x = 0;
reg [3:0] y = 0;
reg [3:0] operation = 0;
reg [3:0] Result = 0;
reg Cout = 0;
reg CinResult = 0;

initial begin
```

```
//and
x = 4'b1101;
y = 4'b1110;
operation = 4'h0;
#10;
x = 4'b1001;
y = 4'b0101;
#10;
//nand
x = 4'b1101;
y = 4'b1110;
operation = 4'h1;
#10;
x = 4'b1001;
y = 4'b0101;
#10;
//or
x = 4'b1101;
y = 4'b1110;
operation = 4'h2;
#10;
x = 4'b1001;
y = 4'b0101;
#10;
//nor
x = 4'b1101;
y = 4'b1110;
operation = 4'h3;
#10;
x = 4'b1001;
y = 4'b0101;
#10;
//xor
x = 4'b1101;
y = 4'b1110;
operation = 4'h4;
#10;
x = 4'b1001;
y = 4'b0101;
#10;
//xnor
x = 4'b1101;
y = 4'b1110;
operation = 4'h5;
#10;
x = 4'b1001;
```

```
y = 4'b0101;
#10;
//not
x = 4'b1101;
y = 4'b1110;
operation = 4'h6;
#10;
x = 4'b1001;
y = 4'b0101;
#10;
//shift
x = 4'b1101;
y = 4'b1110;
operation = 4'h7;
#10;
x = 4'b1001;
y = 4'b0101;
#10;
//add
x = 4'b1101;
y = 4'b1110;
CinResult = 1;
operation = 4'h8;
Cout = CoutResult;
#10;
x = 4'b1001;
y = 4'b0101;
Cout = CoutResult;
#10;
//sub
x = 4'b1101;
y = 4'b1110;
CinResult = 1;
operation = 4'h9;
Cout = CoutResult;
#10;
x = 4'b1001;
y = 4'b0101;
Cout = CoutResult;
#10;
//mult
x = 4'b1101;
y = 4'b1110;
CinResult = 1;
operation = 4'hA;
Cout = CoutResult;
```

```
    #10;
    x = 4'b1001;
    y = 4'b0101;
    Cout = CoutResult;
    #10;
    $finish;
end

wire [3:0] AndResult, NandResult, OrResult, NorResult, XorResult,
 ↪  XnorResult, NotResult, ShiftResult, AddResult, SubResult;
wire [7:0] MultResult;
wire CoutResult;


//instantiations
and_4b and4(.x(x), .y(y), .out(AndResult));
nand_4b nand4(.x(x), .y(y), .out(NandResult));
or_4b or4(.x(x), .y(y), .out(OrResult));
nor_4b nor4(.x(x), .y(y), .out(NorResult));
xor_4b xor4(.x(x), .y(y), .out(XorResult));
xnor_4b xnor4(.x(x), .y(y), .out(XnorResult));
not_4b not4(.x(x), .out(NotResult));
shift_4b shift4(.x(x), .y(y), .out(ShiftResult),
 ↪  .Cout(CoutResult));
add_4b add4(.x(x), .y(y), .out(AddResult), .Cin(CinResult),
 ↪  .Cout(CoutResult));
sub_4b sub4(.x(x), .y(y), .out(SubResult), .Cin(CinResult),
 ↪  .Cout(CoutResult));
mult_4b mult4(.x(x), .y(y), .out(MultResult), .Cin(CinResult),
 ↪  .Cout(CoutResult));

//switch case
always @* begin
case (operation)
    4'h0: Result = AndResult;
    4'h1: Result = NandResult;
    4'h2: Result = OrResult;
    4'h3: Result = NorResult;
    4'h4: Result = XorResult;
    4'h5: Result = XnorResult;
    4'h6: Result = NotResult;
    4'h7: Result = ShiftResult;
    4'h8: Result = AddResult;
    4'h9: Result = SubResult;
    4'hA: Result = MultResult;
endcase
```

```
end

initial begin
   $dumpfile("control.vcd");
   $dumpvars(0,clk);
   $dumpvars(1,x);
   $dumpvars(2,y);
   $dumpvars (3, Cout);
   $dumpvars (4, CinResult);
   $dumpvars (5, Result);
end

initial
   $monitor("At time %t, x(%b) and y(%b) with CinResult(%b)=
    ↪  Result(%b) with Cout(%b)",
   $time, x, y, CinResult, Result, Cout);
endmodule
```

5 Waveforms

Waveforms with an X and Y input, with 2 test cases for each operation.

6 AND

Figure 1: The AND Waveform.

7 NAND

Figure 2: The NAND waveform.

8 OR

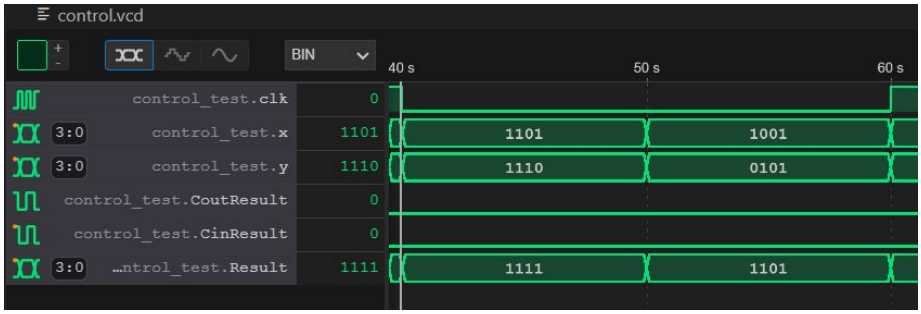

Figure 3: The OR waveform.

9 NOR

Figure 4: The NOR waveform.

10 XOR

Figure 5: The XOR waveform.

11 XNOR

Figure 6: The XNOR waveform.

12 NOT

Figure 7: The NOT waveform.

13 SHIFT

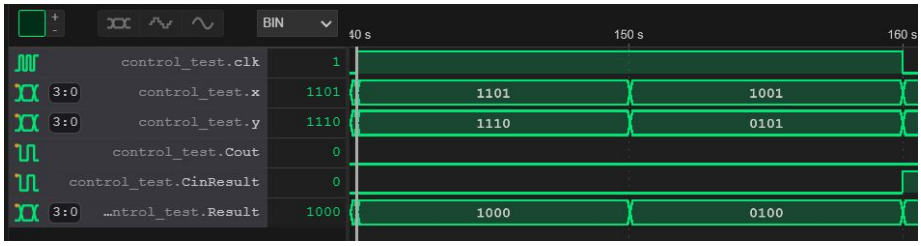

Figure 8: The SHIFT waveform.

14 ADD

Figure 9: The ADD waveform.

15 SUB

Figure 10: The SUB waveform.

16 MULT

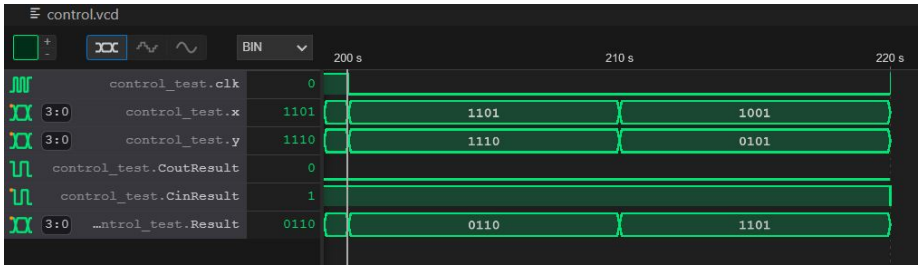

Figure 11: The MULT waveform.

17 Conclusion

In order to create our 4-bit modules in Verilog, we had to determine how to store 4-bit numbers in registers and wires. Addition was the most straightforward to code with Carry-In and Carry-Out values. In our multiplication module, we found that the product of two 4-bit numbers needed to be 8-bits in order to properly implement the Carry-Out of the module. The multiplication module gave us the most trouble as we couldn't decide what size the output product should be. The logic gates were quick to get through using 4-bit values as they are built into the Verilog code as operators. We imported each ALU function and connected them to a control circuit. From the control circuit, we were able to have a user input to choose the desired ALU function. From here, the control circuit would select the output for the user chosen function in hexadecimal coded in after running every function. Each operation was run in two test cases.