CompArch HW2

David Abrahams

September 25, 2015

1 The binary adder

The binary adder takes in three bits: A, B, and the carry-over from the previous addition.

1.1 The sum bit

It outputs a Sum bit, and a carry-over bit. $Sum = 1$ if an odd number of the inputs are true. An odd number of the inputs are true if either of these conditions are met:

- Both or neither of A and B are 1, and $CarryIn$ is 1
- One of A and B is 1, and $CarryIn$ is 0

We can express this in boolean logic:

$$Sum = (A \oplus B) \oplus CarryIn \tag{1}$$

1.2 The CarryOut bit

$CarryOut = 1$ if an two or more of the inputs are true. One of these conditions must be met met:

- A and B are 1.
- One of A and B is 1, and $CarryIn$ is 1

In boolean logic:

$$CarryOut = (A \& B) \,||\, ((A \oplus B) \,\& CarryIn) \tag{2}$$

See Figure 1.2 for the implementation in a circuit.

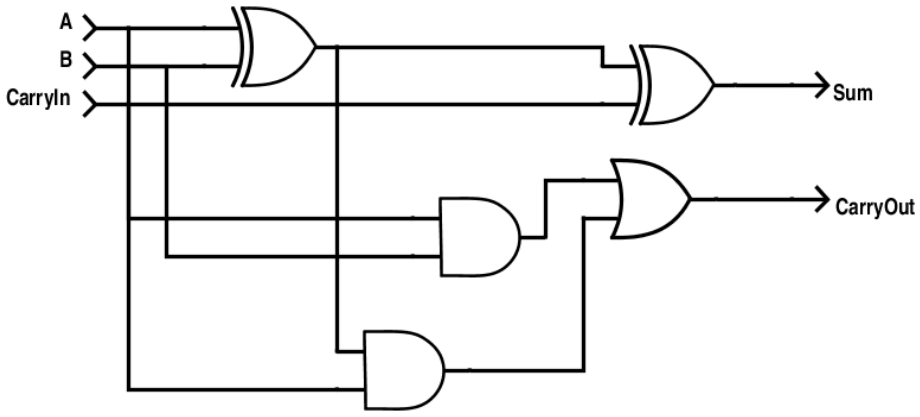

Figure 1: The circuit for a binary adder.

1.3 The generated truth table

Our circuit behaves as expected, following the previously defined rules.

Cin	A	B	Sum	Cout	Expected	Output
0	0	0	0	0	0	1
0	1	0	1	0	1	0
0	0	1	1	0	1	0
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	1	0	0	1	0	1
1	0	1	0	1	0	1
1	1	1	1	1	1	1

2 The multiplexer

The multiplexer takes in two bits: $A0$, $A1$. Using these two bits as a length-two binary number, the multiplexer selects one of its four outputs based on the inputs. It then sets this output to 1, and all the others to 0. and the carry-over from the previous addition.

The multiplexer is logically quite simple, but its circuit diagram is quite complex. To build this circuit, "and" each input with its corresponding index values from $A0$ and $A1$. Then, "or" all of these "anded" inputs together. All the wires not being indexed will be 0, and the indexed input will be 1 or 0 depending on that input. The logic is illustrated in the source code in Appendix B.

2.1 The generated truth table

Our circuit behaves as expected. The only input that matters is the one that is indexed by $A0$ and $A1$.

A0	A1	In0	In1	In2	In3	Out	Expected	Output
0	0	0	x	x	x	0	0	
0	0	1	x	x	x	1	1	
0	1	x	0	x	x	0	0	
0	1	x	1	x	x	1	1	
1	0	x	x	0	x	0	0	
1	0	x	x	1	x	1	1	
1	1	x	x	x	0	0	0	
1	1	x	x	x	1	1	1	

3 The decoder

The decoder is similar to the mux. Instead of there being only one output wire, there are four. There are only three inputs. Two indices, and an *enable* wire. The indexed output is set to the value of *enable*, and the rest to 0. This is achieved by "and"ing together the index combinations with *enable*.

The logic is illustrated in the source code in Appendix C.

3.1 The generated truth table

Our circuit behaves as expected. The indexed output is set to *enable*.

En	A0	A1	O0	O1	O2	O3	Expected	Output
0	0	0	0	0	0	0	All false	
0	1	0	0	0	0	0	All false	
0	0	1	0	0	0	0	All false	
0	1	1	0	0	0	0	All false	

1	0	0	1	0	0	0	O0 Only
1	1	0	0	1	0	0	O1 Only
1	0	1	0	0	1	0	O2 Only
1	1	1	0	0	0	1	O3 Only

A Adder code

```
// define gates with delays
`define AND and #50
`define OR or #50
`define XOR xor #50

module behavioralFullAdder(sum, carryout, a, b, carryin);
  output sum, carryout;
  input a, b, carryin;
  assign {carryout, sum}=a+b+carryin;
endmodule

module structuralFullAdder(sum, carryout, a, b, carryin);

  output sum, carryout;
  input a, b, carryin;

  wire a_xor_b;
  wire a_and_b;
  wire a_xor_b_and_cin;

  // sum is true if 1 or 3 inputs are true
  `XOR xor_gate_0(a_xor_b, a, b);
  `XOR xor_gate_1(sum, a_xor_b, carryin);

  // carryout is true if both a and be are true, or one of them is true and
  // carryin is true.
  `AND and_gate_0(a_and_b, a, b);
  `AND and_gate_1(a_xor_b_and_cin, a_xor_b, carryin);
  `OR or_gate(carryout, a_and_b, a_xor_b_and_cin);

endmodule

module testFullAdder;

  reg a, b, carryin;
  wire sum, carryout;
  structuralFullAdder adder (sum, carryout, a, b, carryin);

  initial begin

    // create the wave file
    $dumpfile("adder.vcd");
    $dumpvars(0, testFullAdder);

    // output a truth table
    $display("Adder truth table");
```

```
    $display("Cin A B | Sum Cout | Expected Output");
    carryin=0;a=0;b=0; #1000
    $display("%b   %b %b | %b   %b     | %b  %b", carryin, a, b, sum, carryout,
             1'b0, 1'b1);
    carryin=0;a=1;b=0; #1000
    $display("%b   %b %b | %b   %b     | %b  %b", carryin, a, b, sum, carryout,
             1'b1, 1'b0);
    carryin=0;a=0;b=1; #1000
    $display("%b   %b %b | %b   %b     | %b  %b", carryin, a, b, sum, carryout,
             1'b1, 1'b0);
    carryin=0;a=1;b=1; #1000
    $display("%b   %b %b | %b   %b     | %b  %b", carryin, a, b, sum, carryout,
             1'b0, 1'b1);
    carryin=1;a=0;b=0; #1000
    $display("%b   %b %b | %b   %b     | %b  %b", carryin, a, b, sum, carryout,
             1'b1, 1'b0);
    carryin=1;a=1;b=0; #1000
    $display("%b   %b %b | %b   %b     | %b  %b", carryin, a, b, sum, carryout,
             1'b0, 1'b1);
    carryin=1;a=0;b=1; #1000
    $display("%b   %b %b | %b   %b     | %b  %b", carryin, a, b, sum, carryout,
             1'b0, 1'b1);
    carryin=1;a=1;b=1; #1000
    $display("%b   %b %b | %b   %b     | %b  %b", carryin, a, b, sum, carryout,
             1'b1, 1'b1);
    $display();

  end

endmodule
```

B Multiplexer code

```
// define gates with delays
`define AND and #50
`define NOT not #50
`define OR or #50

module behavioralMultiplexer(out, address0,address1, in0,in1,in2,in3);
output out;
input address0, address1;
input in0, in1, in2, in3;
wire[3:0] inputs = {in3, in2, in1, in0};
wire[1:0] address = {address1, address0};
assign out = inputs[address];
endmodule

module structuralMultiplexer(out, address0,address1, in0,in1,in2,in3);
  output out;
  input address0, address1;
  input in0, in1, in2, in3;
  wire naddress0, naddress1;
  wire in0sel, in1sel, in2sel, in3sel;
```

```
  `NOT not_gate_0(naddress0, address0);
  `NOT not_gate_1(naddress1, address1);

  `AND and_gate_0(in0sel, naddress0, naddress1, in0);
  `AND and_gate_1(in1sel, naddress0, address1, in1);
  `AND and_gate_2(in2sel, address0, naddress1, in2);
  `AND and_gate_3(in3sel, address0, address1, in3);

  `OR or_gate(out, in0sel, in1sel, in2sel, in3sel);

endmodule


module testMultiplexer;
  reg address0, address1;
  reg in0, in1, in2, in3;
  wire out;
  structuralMultiplexer multiplexor (out, address0,address1, in0,in1,in2,in3);

  initial begin
    $display("A0 A1| In0 In1 In2 In3 | Out | Expected Output");
    address0=0;address1=0;in0=0;in1=1'bX;in2=1'bX;in3=1'bX; #1000
    $display("%b  %b | %b   %b   %b   %b   | %b   | 0", address0, address1, in0, in1, in2,
    address0=0;address1=0;in0=1;in1=1'bX;in2=1'bX;in3=1'bX; #1000
    $display("%b  %b | %b   %b   %b   %b   | %b   | 1", address0, address1, in0, in1, in2,
    address0=0;address1=1;in0=1'bX;in1=0;in2=1'bX;in3=1'bX; #1000
    $display("%b  %b | %b   %b   %b   %b   | %b   | 0", address0, address1, in0, in1, in2,
    address0=0;address1=1;in0=1'bX;in1=1;in2=1'bX;in3=1'bX; #1000
    $display("%b  %b | %b   %b   %b   %b   | %b   | 1", address0, address1, in0, in1, in2,
    address0=1;address1=0;in0=1'bX;in1=1'bX;in2=0;in3=1'bX; #1000
    $display("%b  %b | %b   %b   %b   %b   | %b   | 0", address0, address1, in0, in1, in2,
    address0=1;address1=0;in0=1'bX;in1=1'bX;in2=1;in3=1'bX; #1000
    $display("%b  %b | %b   %b   %b   %b   | %b   | 1", address0, address1, in0, in1, in2,
    address0=1;address1=1;in0=1'bX;in1=1'bX;in2=1'bX;in3=0; #1000
    $display("%b  %b | %b   %b   %b   %b   | %b   | 0", address0, address1, in0, in1, in2,
    address0=1;address1=1;in0=1'bX;in1=1'bX;in2=1'bX;in3=1; #1000
    $display("%b  %b | %b   %b   %b   %b   | %b   | 1", address0, address1, in0, in1, in2,
  end
endmodule
```

C Multiplexer code

```
// define gates with delays
`define AND and #50
`define NOT not #50

module behavioralDecoder(out0,out1,out2,out3, address0,address1, enable);

  output out0, out1, out2, out3;
  input address0, address1;
  input enable;
  assign {out3,out2,out1,out0}=enable<<{address1,address0};

endmodule
```

```
module structuralDecoder(out0,out1,out2,out3, address0,address1, enable);

  output out0, out1, out2, out3;
  input address0, address1;
  input enable;

  wire not_address0;
  wire not_address1;

  // Create the inverse wires
  `NOT not_gate_0(not_address0, address0);
  `NOT not_gate_1(not_address1, address1);

  // Set each output to its proper index.
  `AND and_gate_0(out0, enable, not_address0, not_address1);
  `AND and_gate_1(out1, enable, address0, not_address1);
  `AND and_gate_2(out2, enable, not_address0, address1);
  `AND and_gate_3(out3, enable, address0, address1);

endmodule

module testDecoder;
  reg addr0, addr1;
  reg enable;
  wire out0,out1,out2,out3;
  // behavioralDecoder decoder (out0,out1,out2,out3,addr0,addr1,enable);
  structuralDecoder decoder (out0,out1,out2,out3,addr0,addr1,enable); // Swap after testing

  initial begin

    // create the wave file
    $dumpfile("decoder.vcd");
    $dumpvars(0, testDecoder);

    // output a truth table
    $display("Decoder truth table");
    $display("En A0 A1| O0 O1 O2 O3 | Expected Output");
    enable=0;addr0=0;addr1=0; #1000
    $display("%b  %b  %b | %b  %b  %b  %b | All false", enable, addr0, addr1, out0, out1,
    enable=0;addr0=1;addr1=0; #1000
    $display("%b  %b  %b | %b  %b  %b  %b | All false", enable, addr0, addr1, out0, out1,
    enable=0;addr0=0;addr1=1; #1000
    $display("%b  %b  %b | %b  %b  %b  %b | All false", enable, addr0, addr1, out0, out1,
    enable=0;addr0=1;addr1=1; #1000
    $display("%b  %b  %b | %b  %b  %b  %b | All false", enable, addr0, addr1, out0, out1,
    enable=1;addr0=0;addr1=0; #1000
    $display("%b  %b  %b | %b  %b  %b  %b | O0 Only", enable, addr0, addr1, out0, out1, ou
    enable=1;addr0=1;addr1=0; #1000
    $display("%b  %b  %b | %b  %b  %b  %b | O1 Only", enable, addr0, addr1, out0, out1, ou
    enable=1;addr0=0;addr1=1; #1000
    $display("%b  %b  %b | %b  %b  %b  %b | O2 Only", enable, addr0, addr1, out0, out1, ou
    enable=1;addr0=1;addr1=1; #1000
    $display("%b  %b  %b | %b  %b  %b  %b | O3 Only", enable, addr0, addr1, out0, out1, ou
```

```
    $display();
  end

endmodule
```