ECEn/CS 224
Appendix B Homework Solutions

B.1 Implement a 4:1 MUX using a single dataflow assignment statement involving only concatenation, replication, and the operators for AND, OR, and NOT.

```
module mux41(out, in0, in1, in2, in3, sel);
   input in0, in1, in2, in3;
   input [1:0] sel;
   output out;

   assign out = (in0 & (~sel[1]) & (~sel[0])) |
                (in1 & (~sel[1]) &   sel[0] ) |
                (in2 &   sel[1]  & (~sel[0])) |
                (in3 &   sel[1]  &   sel[0] );
endmodule
```

B.2 Implement a 4:1 MUX using a single dataflow assignment statement and the ?: operator.

```
module mux41(out, in0, in1, in2, in3, sel);
   input in0, in1, in2, in3;
   input [1:0] sel;
   output out;

   assign out = (sel == 2'b00)? in0 :
                (sel == 2'b01)? in1 :
                (sel == 2'b10)? in2 :
                in3;
endmodule
```

B.3 Repeat the previous 2 problems but parameterize your designs for any size operands.

```
module mux41n_1(out, in0, in1, in2, in3, sel);
   parameter WID = 16;
   input [WID-1:0] in0, in1, in2, in3;
   input [1:0] sel;
   output [WID-1:0] out;

   assign out = (in0 & {WID{~sel[1]}} & {WID{~sel[0]}}) |
                (in1 & {WID{~sel[1]}} & {WID{ sel[0]}}) |
                (in2 & {WID{ sel[1]}} & {WID{~sel[0]}}) |
                (in3 & {WID{ sel[1]}} & {WID{ sel[0]}});

endmodule
```

```
module mux41n_2(out, in0, in1, in2, in3, sel);
   parameter WID = 16;
   input  [WID-1:0] in0, in1, in2, in3;
   input  [1:0] sel;
   output [WID-1:0] out;

   assign out = (sel == 2'b00)? in0:
                (sel == 2'b01)? in1:
                (sel == 2'b10)? in2:
                in3;
endmodule
```

B.4 Implement a 3:8 decoder using a single dataflow assignment statement.

```
module decoder38(out, in);
   input  [2:0] in;
   output [7:0] out;

   assign out = (in == 3'b000) ? 8'b00000001 :
                (in == 3'b001) ? 8'b00000010 :
                (in == 3'b010) ? 8'b00000100 :
                (in == 3'b011) ? 8'b00001000 :
                (in == 3'b100) ? 8'b00010000 :
                (in == 3'b101) ? 8'b00100000 :
                (in == 3'b110) ? 8'b01000000 :
                                 8'b10000000;
endmodule
```

B.5 Implement an 8-bit ALU using a single dataflow assignment statement. Your ALU should operate on 8-bit A and B inputs. It should also take in a 2-bit control code which implies the following functions: 00=PASS(A), 01=ADD, 10=AND, 11=NOT(A). Call the ALU output *Q*. Use the ?: operator.

```
module alu8(Q, A, B, ctrl);
   input  [7:0] A, B;
   input  [1:0] ctrl;
   output [7:0] Q;

   assign Q = (ctrl == 2'b00) ? A     :
              (ctrl == 2'b01) ? A + B :
              (ctrl == 2'b10) ? A & B :
                                ~A;
endmodule
```

B.6 Repeat the previous problem but make it parameterizable for any width operands.

```
module alu8(Q, A, B, ctrl);
   parameter WID = 16;
   input  [WID-1:0] A, B;
   input  [1:0] ctrl;
   output [WID-1:0] Q;

   assign Q = (ctrl == 2'b00) ? A     :
              (ctrl == 2'b01) ? A + B :
              (ctrl == 2'b10) ? A & B :
                                ~A;
endmodule
```

B.7 Design a parameterized adder/subtracter. Its A and B inputs are of width 'WID' and its Q output is of width 'WID+1'. It adds its A and B operands when MODE=0 and subtracts them when MODE=1. To avoid overflow, sign-extend each number by one bit as described in Chapter 8. Implement the sign-extensions as two separate dataflow assignment statements and then implement the adder/subtracter as a single dataflow assignment statement. Use some local wires for this design to simplify the logic.

```
module addsubn(Q, A, B, MODE);
   parameter WID = 16;
   input [WID-1:0] A, B;
   input MODE;
   output [WID:0] Q;

   wire [WID:0] extA, extB;

   assign extA = {A[WID-1], A};
   assign extB = {B[WID-1], B};

   assign Q = (MODE == 1'b0) ? A+B : A-B;
endmodule
```

B.8 Design an 8-bit adder/subtracter which has a 9-bit output. It adds its A and B operands when MODE=0 and subtracts them when MODE=1. To avoid overflow, sign-extend each number by one bit as described in Chapter 8. Implement the logic for this design in a single dataflow assignment statement.

```
module addsub8(Q, A, B, MODE);
   input [7:0] A, B;
   input MODE;
   output [8:0] Q;

   assign Q = (MODE == 1'b0) ? {1'b0,A}+{1'b0,B} :
                               {1'b0,A}-{1'b0,B};
endmodule
```