wire [15:0] decoder_output; assign decoder_output = (input_data == 4'b0000) ? 16'b0000000000000001 : (input_data == 4'b0001) ? 16'b0000000000000010 : (input_data == 4'b0010) ? 16'b0000000000000100 : (input_data == 4'b0011) ? 16'b0000000000001000 : (input_data == 4'b0100) ? 16'b0000000000010000 : (input_data == 4'b0101) ? 16'b0000000000100000 : (input_data == 4'b0110) ? 16'b0000000001000000 : (input_data == 4'b0111) ? 16'b0000000010000000 : (input_data == 4'b1000) ? 16'b0000000100000000 : (input_data == 4'b1001) ? 16'b0000001000000000 : (input_data == 4'b1010) ? 16'b0000010000000000 : (input_data == 4'b1011) ? 16'b0000100000000000 : (input_data == 4'b1100) ? 16'b0001000000000000 : (input_data == 4'b1101) ? 16'b0010000000000000 : (input_data == 4'b1110) ? 16'b0100000000000000 : (input_data == 4'b1111) ? 16'b1000000000000000 : 16'b0000000000000000; // Default case assign {a, b, c, d, e, f, g} = (decoder_output == 16'b0000000000000001) ? 7'b1000000 : (decoder_output == 16'b0000000000000010) ? 7'b1111001 : (decoder_output == 16'b0000000000000100) ? 7'b0100100 : (decoder_output == 16'b0000000000001000) ? 7'b0110000 : (decoder_output == 16'b0000000000010000) ? 7'b0011001 : (decoder_output == 16'b0000000000100000) ? 7'b0010010 : (decoder_output == 16'b0000000001000000) ? 7'b0000010 : (decoder_output == 16'b0000000010000000) ? 7'b1111000 : (decoder_output == 16'b0000000100000000) ? 7'b0000000 : (decoder_output == 16'b0000001000000000) ? 7'b0010000 : (decoder_output == 16'b0000010000000000) ? 7'b0001000 : (decoder_output == 16'b0000100000000000) ? 7'b0000011 : (decoder_output == 16'b0001000000000000) ? 7'b1000110 : (decoder_output == 16'b0010000000000000) ? 7'b0100000 : (decoder_output == 16'b0100000000000000) ? 7'b0000100 : (decoder_output == 16'b1000000000000000) ? 7'b1000000 : 7'b0000000; // Default case endmodule wire [7:0] decoder_output; assign decoder_output = (input_data == 3'b000) ? 8'b00000001 : (input_data == 3'b001) ? 8'b00000010 : (input_data == 3'b010) ? 8'b00000100 : (input_data == 3'b011) ? 8'b00001000 : (input_data == 3'b100) ? 8'b00010000 : (input_data == 3'b101) ? 8'b00100000 : (input_data == 3'b110) ? 8'b01000000 : (input_data == 3'b111) ? 8'b10000000 : 8'b00000000; assign {a, b, c, d, e, f, g} = (decoder_output == 8'b00000001) ? 7'b1000000 : (decoder_output == 8'b00000010) ? 7'b1111001 : (decoder_output == 8'b00000100) ? 7'b0100100 : (decoder_output == 8'b00001000) ? 7'b0110000 : (decoder_output == 8'b00010000) ? 7'b0011001 : (decoder_output == 8'b00100000) ? 7'b0010010 :

(decoder_output == 8'b01000000) ? 7'b0000010 : (decoder_output == 8'b10000000) ? 7'b1111000 : 7'b0000000; endmodule 2. 4:1 mux module full_adder(input wire A, B, Cin, // Inputs output wire Sum, Cout // Outputs); wire X1, X2, X3, X4; // Intermediate wires // Generate XOR gates assign X1 = A ^ B; assign X2 = X1 ^ Cin; // Generate AND gates assign X3 = A & B; assign X4 = X1 & Cin; // Use a 4:1 multiplexer to select the appropriate inputs for Sum and Cout endmodule 8:1 MUX module full_adder(input wire A, B, Cin, // Inputs output wire Sum, Cout // Outputs); wire [7:0] MuxInputs; // Array to hold 8 possible inputs for the multiplexer // Generate all possible combinations of A, B, and Cin for the 8 inputs of the multiplexer assign MuxInputs = {A & B & Cin, A & B & ~Cin, A & ~B & Cin, A & ~B & ~Cin, ~A & B & Cin, ~A & B & ~Cin, ~A & ~B & Cin, ~A & ~B & ~Cin}; wire [2:0] MuxSelect; // 3-bit signal to select one of 8 inputs // Generate the select signals based on A, B, and Cin assign MuxSelect = (Cin << 2) | (B << 1) | A; wire MuxOutput; // Output of the multiplexer // 8:1 Mux implementation for Sum assign MuxOutput = MuxInputs[MuxSelect]; // Sum output assign Sum = MuxOutput[0]; // Carry-out output assign Cout = MuxOutput[1]; endmodule 3:8 Decoder module full_adder(input wire A, B, Cin, // Inputs output wire Sum, Cout // Outputs); // 3:8 Decoder implementation assign DecoderOutputs = (A << 2'b2) + (B << 2'b1) + (Cin << 2'b0); // Generate XOR and AND gates for Sum and Carry-Out wire X1, X2, X3, X4; assign X1 = DecoderOutputs[0] ^ DecoderOutputs[1]; assign X2 = X1 ^ DecoderOutputs[2]; assign X3 = DecoderOutputs[0] & DecoderOutputs[1]; assign X4 = X1 & DecoderOutputs[2]; // Sum output assign Sum = X2; // Carry-out output assign Cout = X3 | X4; endmodule