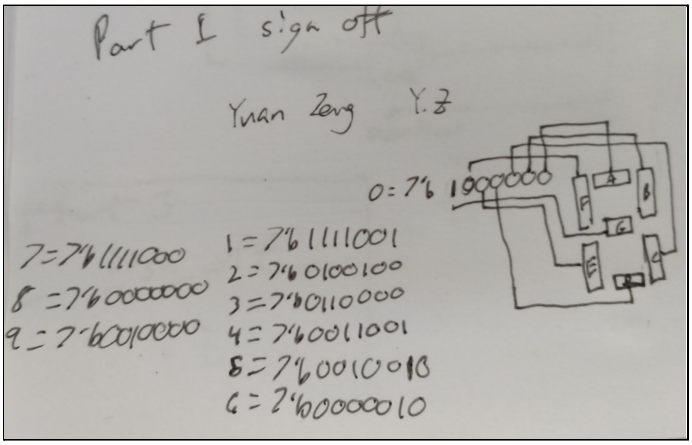
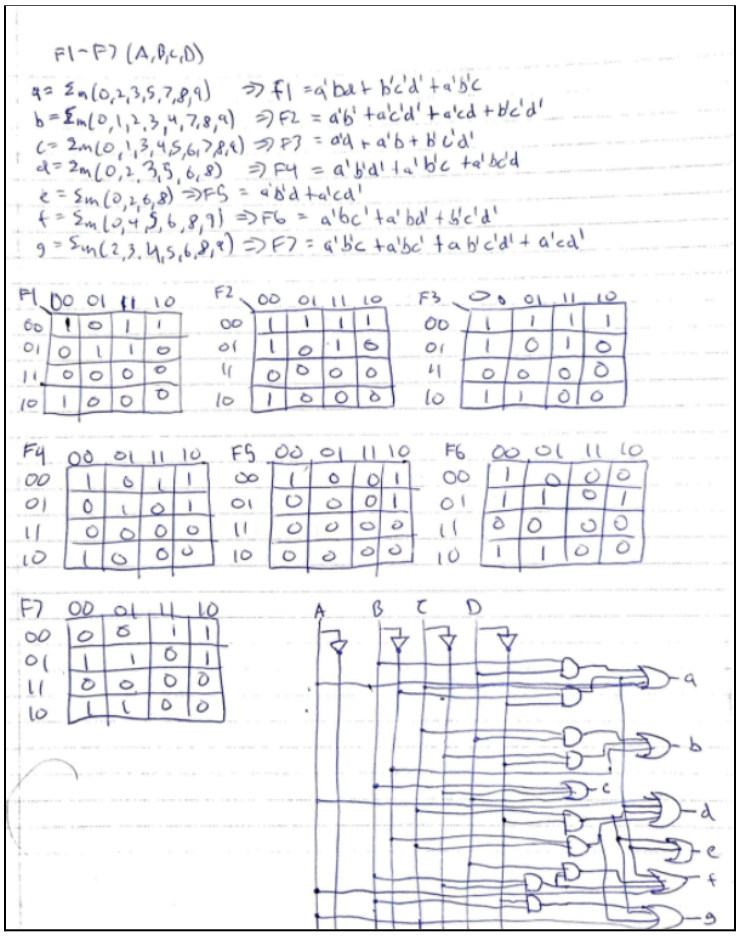
Mark Mitri and Ryan Forelli 9/6/2021
ECE 128 Lab 2

Part 1: BCD on 7-Segment Display

Design Diagram:



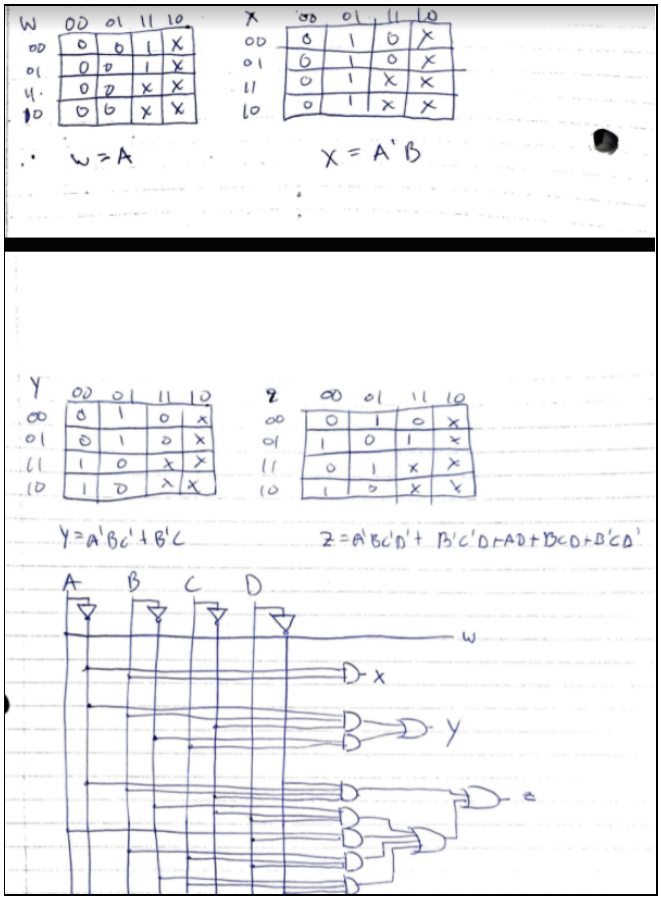


Code:

```
`timescale 1ns / 1ps

module SevenSeg(
  input[3:0] number,
  output reg[6:0] segments,
  output reg led,
  output reg[7:0] characters //Characters on the FPGA dev board. We only use character AN0
  );

  always @ (*) //event-sensitivity list
    begin
      led = 1'b0;
      characters = 8'b11111110;
      case (number) //Cases for lighting up each number
        0 : segments = 7'b1000000;
        1 : segments = 7'b1111001;
        2 : segments = 7'b0100100;
        3 : segments = 7'b0110000;
        4 : segments = 7'b0011001;
        5 : segments = 7'b0010010;
        6 : segments = 7'b0000010;
        7 : segments = 7'b1111000;
        8 : segments = 7'b0000000;
        9 : segments = 7'b0010000;
        default:
          begin
              led = 1'b1;
              segments = 7'b1111111;
            characters = 8'b11111111;
          end
      endcase
    end
Endmodule
```

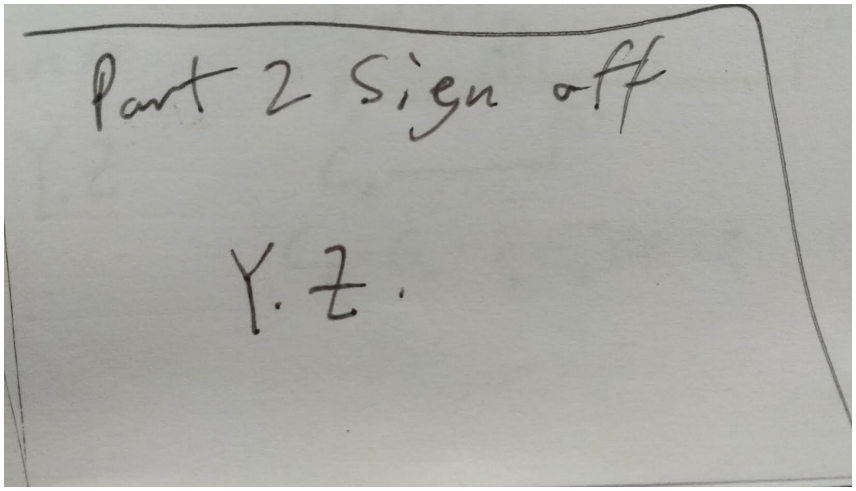
Part 2: Gray to BCD

For the conversion from Gray code to BCD, the truth table/conversion was written into the code so no K-map was used.



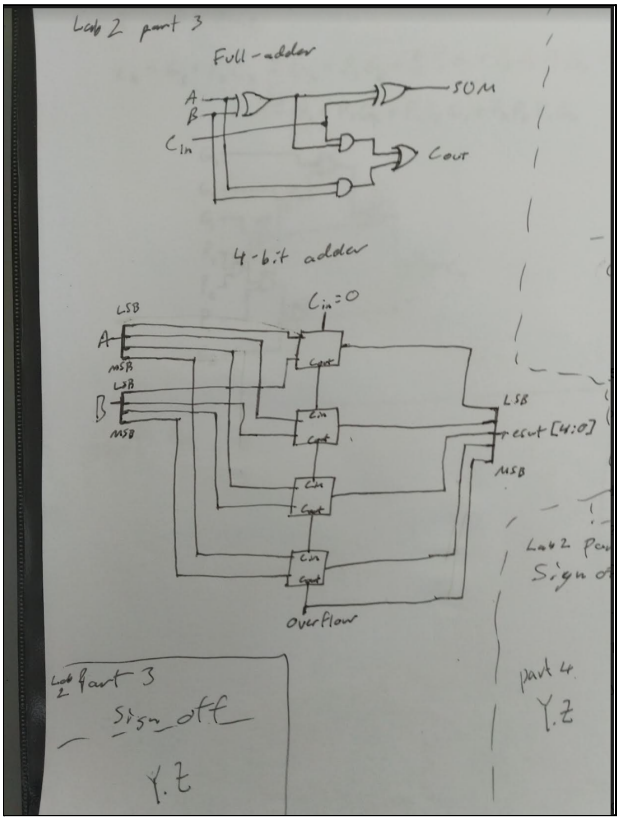

```
`timescale 1ns / 1ps

module gray_to_bcd_converter(
  input[3:0] gray,
  output reg[6:0] segments,
  output reg led,
  output reg[7:0] characters
  );
  reg[3:0] bcd;

  always @ (*) //event-sensitivity list
    begin

    led = 1'b0;
    characters = 8'b11111110;

      case (gray) //Cases for lighting up each number
        4'b0000 : bcd = 4'b0000;
        4'b0001 : bcd = 4'b0001;
        4'b0011 : bcd = 4'b0010;
        4'b0010 : bcd = 4'b0011;
        4'b0110 : bcd = 4'b0100;
        4'b1110 : bcd = 4'b0101;
        4'b1010 : bcd = 4'b0110;
        4'b1011 : bcd = 4'b0111;
        4'b1001 : bcd = 4'b1000;
```

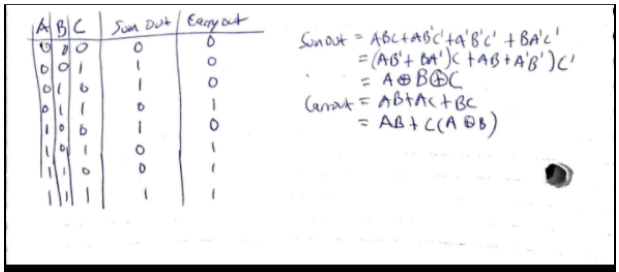
```
        4'b1000 : bcd = 4'b1001;
        default:
          begin
              led = 1'b1;
              segments = 7'b1111111;
            characters = 8'b11111111;
          end
      endcase

      case (bcd) //Cases for lighting up each number
        0 : segments = 7'b1000000;
        1 : segments = 7'b1111001;
        2 : segments = 7'b0100100;
        3 : segments = 7'b0110000;
        4 : segments = 7'b0011001;
        5 : segments = 7'b0010010;
        6 : segments = 7'b0000010;
        7 : segments = 7'b1111000;
        8 : segments = 7'b0000000;
        9 : segments = 7'b0010000;
        default:
          begin
              led = 1'b1;
              segments = 7'b1111111;
            characters = 8'b11111111;
          end
      endcase
    end
endmodule
```

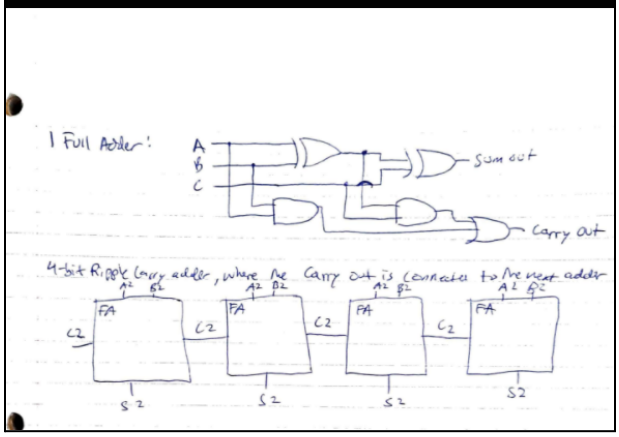


Part 3: Ripple Carry Adder

Design Diagram:







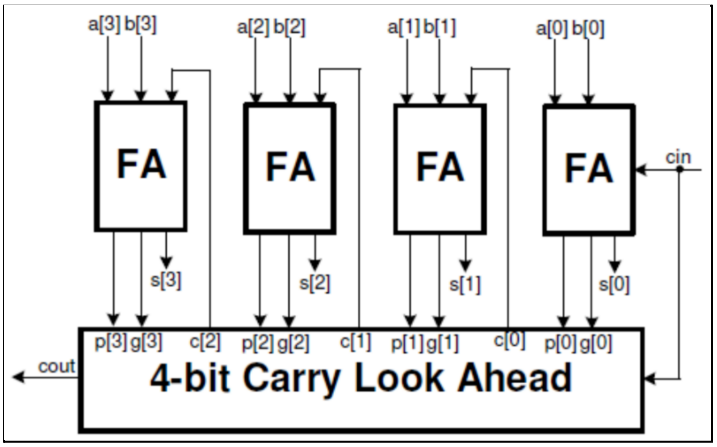
Code:

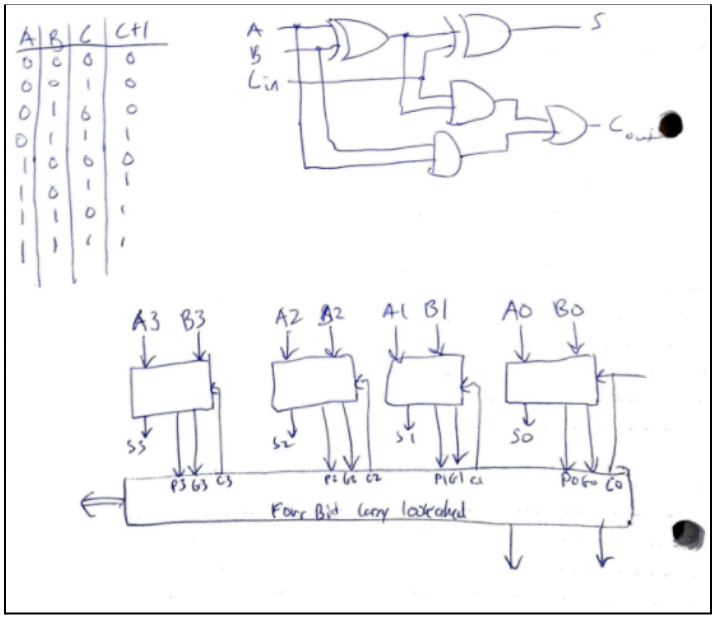
```
`timescale 1ns / 1ps

module full_adder(input A, B, Cin, output SUM, Cout);

  wire wire1, wire2, wire3, wire4;

  xor x1(wire1, A, B);
  and a1(wire3, wire1, Cin);
  and a2(wire4, A, B);
  or o1(Cout, wire3, wire4);
  xor x2(SUM, wire1, Cin);

endmodule

module ripple_adder_4bit(input[3:0] num1, num2, output[4:0] result);

  wire carry1, carry2, carry3;
```

```
  full_adder bit0(num1[0], num2[0], 1'b0, result[0], carry1);
  full_adder bit1(num1[1], num2[1], carry1, result[1], carry2);
  full_adder bit2(num1[2], num2[2], carry2, result[2], carry3);
  full_adder bit3(num1[3], num2[3], carry3, result[3], result[4]);

endmodule
```

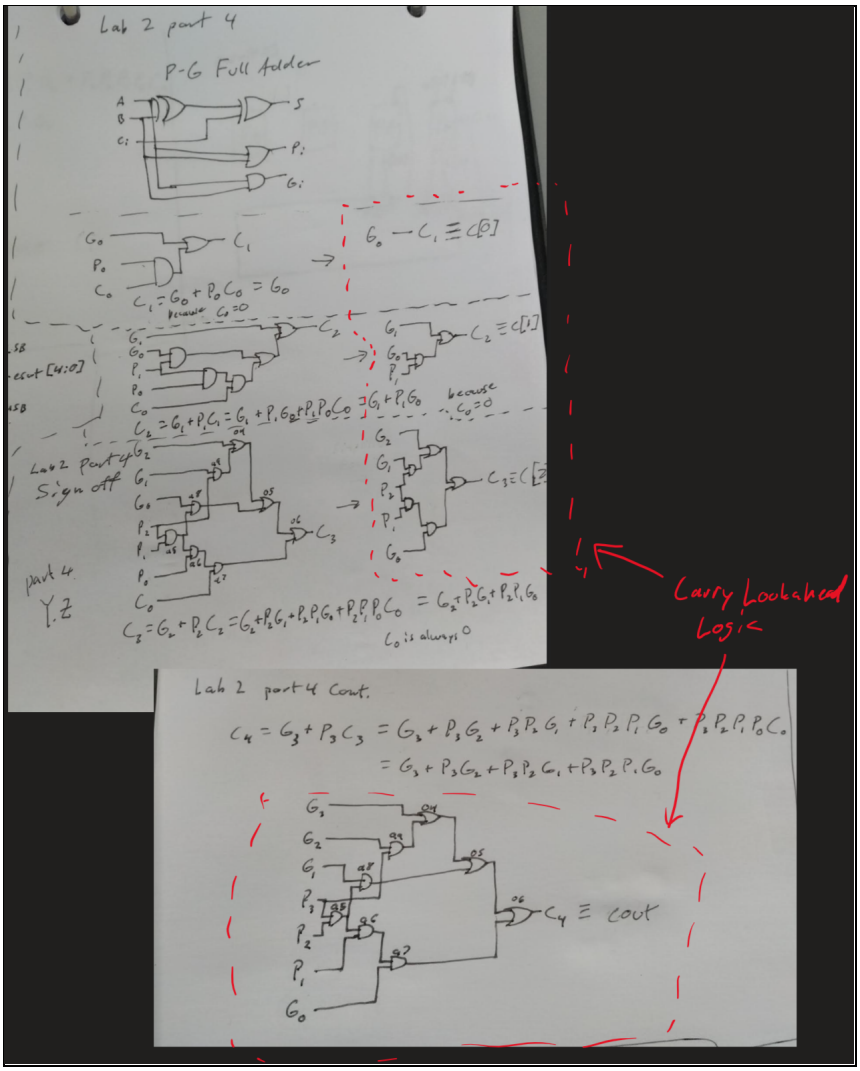
Part 4: Carry Lookahead Adder

Design Diagram:





(from slide notes)



Code:

```
`timescale 1ns / 1ps

module carry_lookahead_adder(input A, B, Cin, output SUM, Pi, Gi);

  wire wire1;

  xor x1(wire1, A, B);
  xor x2(SUM, wire1, Cin);
  or o1(Pi, A, B);
  and a1(Gi, A, B);

endmodule

module carry_lookahead_adder_4bit(input[3:0] num1, num2, output[4:0] result);

  wire c1, c2, c3;
  wire p0, p1, p2, p3, g0, g1, g2, g3;
  carry_lookahead_adder bit0(num1[0], num2[0], 1'b0, result[0], p0, g0);
  carry_lookahead_adder bit1(num1[1], num2[1], c1, result[1], p1, g1);
```

```
  carry_lookahead_adder bit2(num1[2], num2[2], c2, result[2], p2, g2);
  carry_lookahead_adder bit3(num1[3], num2[3], c3, result[3], p3, g3); //Generate function gi is last carry

  wire wire1, wire2, wire3, wire4, wire5, wire6, wire7, wire8, wire9, wire10, wire11, wire12;

//   //C1=G0+P0C0
//   and a1(wire1, p0, 1'b0);
//   or o1(c1, wire1, g0);

  //Simplified version
  //C1=G0
  buf(c1, g0);


//   //C2=G1+P1C1 = G1+P1G0+P1P0C0
//   and a2(wire2, g0, p1);
//   and a3(wire3, p1, p0);
//   and a4(wire4, wire3, 1'b0);
//   or o2(wire5, wire2, wire4);
//   or o3(c2, g1, wire5);

  //Simplified verion
  //C2=G1+P1G0
  and a1(wire1, g0, p1);
  or o1(c2, wire1, g1);

  //C3=G2+P2C2 = G2+P2G1+P2P1G0+P2P1P0C0
//   and a5(wire6, p2, p1);
//   and a6(wire7, p0, wire6);
//   and a7(wire8, 1'b0, wire7);
//   and a8(wire9, g0, wire6);
//   and a9(wire10, g1, p2);
//   or o4(wire11, g2, wire10);
//   or o5(wire12, wire11, wire9);
//   or o6(c3, wire12, wire8);

  //Simplified version
  //C3=G2+P2G1+P2P1G0
  and a2(wire2, g1, p2);
  and a3(wire3, p1, p2);
  and a4(wire4, wire3, g0);
  or o2(wire5, g2, wire2);
  or o3(c3, wire4, wire5);

   //Simplified version, find c4 (MSB of answer)
  //C4=G3+P3G2+P3P2G1+P3P2P1G0
   and a5(wire6, p3, p2);
   and a6(wire7, p1, wire6);
   and a7(wire8, g0, wire7);
   and a8(wire9, g1, wire6);
   and a9(wire10, g2, p3);
   or o4(wire11, g3, wire10);
   or o5(wire12, wire11, wire9);
   or o6(result[4], wire12, wire8);

endmodule
```