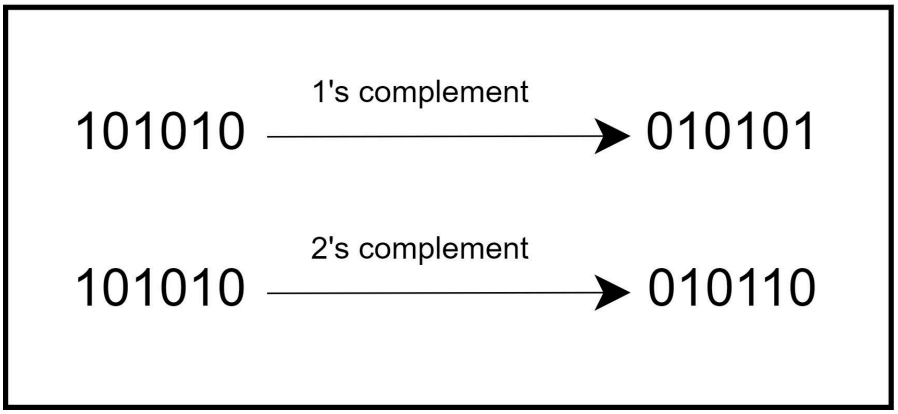
Module: R3: DLD + DSD
Section: Combination Circuits **Task:** Assignment 1

Assignment 1

Combination Circuits

---

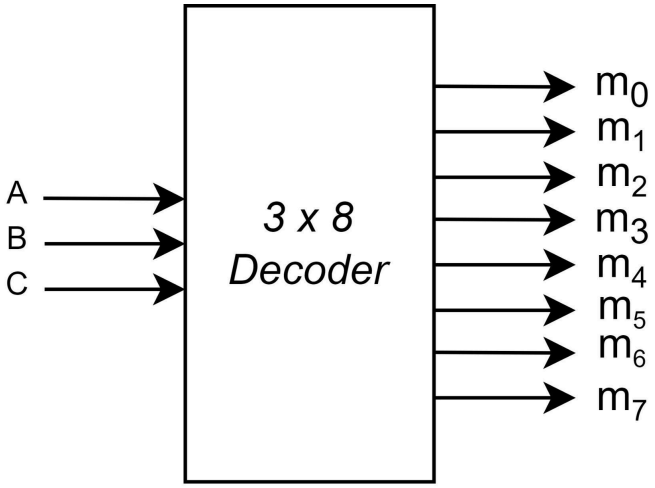
1. **1's and 2's Complement:**
   - **Solution:**



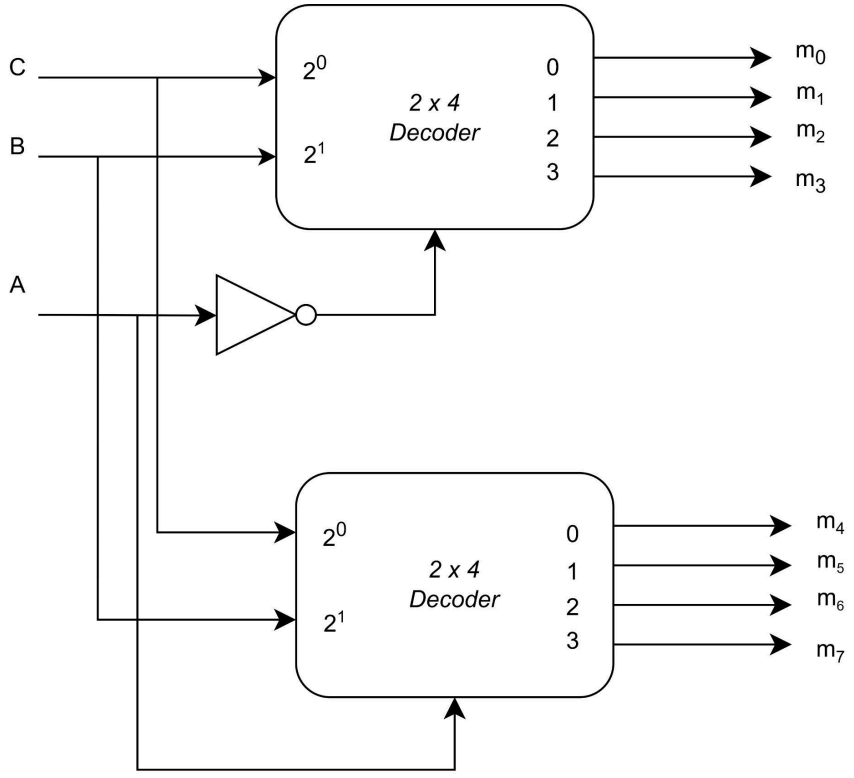
2. **3-to-8 Line Decoder using 2-to-4 Decoders:**
   - **Solution:**

Table for 3x8			
A	B	C	F
0	0	0	m0
0	0	1	m1
0	1	0	m2
0	1	1	m3
1	0	0	m4
1	0	1	m5
1	1	0	m6
1	1	1	m7

The block-level implementation will look following:

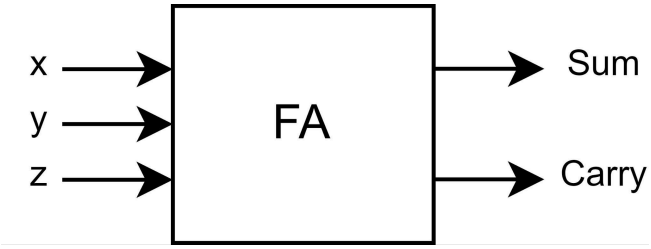


This decoder can be further broken down to two ***2 x 4*** decoders. The input A can be used to enable either of the two ***2 x 4*** decoders. The implementation is given below:



3. **Implement a full-adder with two 4 x 1 multiplexers:**
   - **Solution:**

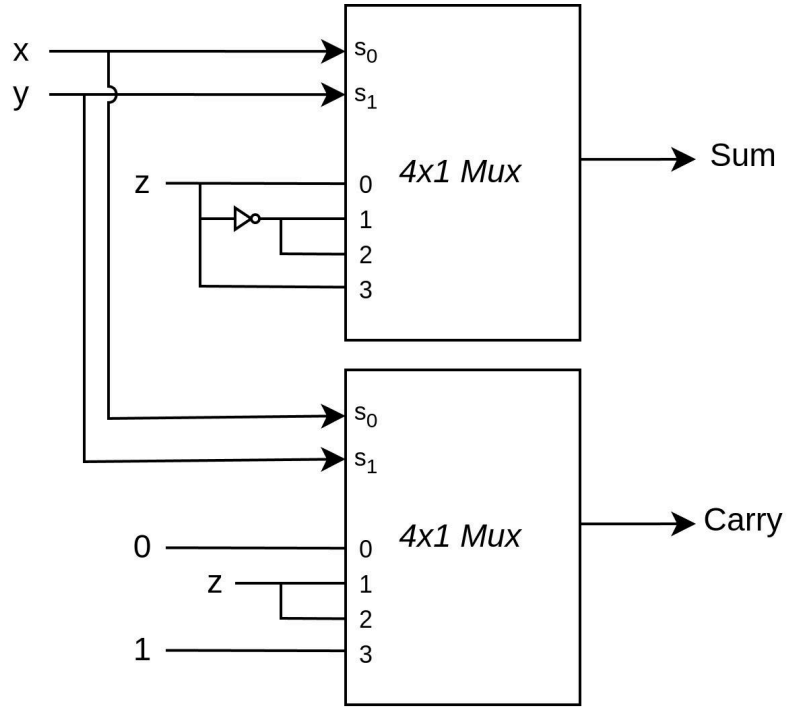
     A full adder can be implemented using two 4 x 1 multiplexers, one for each of sum and carry. A block level implementation of the full adder is as follows:



The block can be further broken into two 4 x 1 multiplexers. Using truth tables:

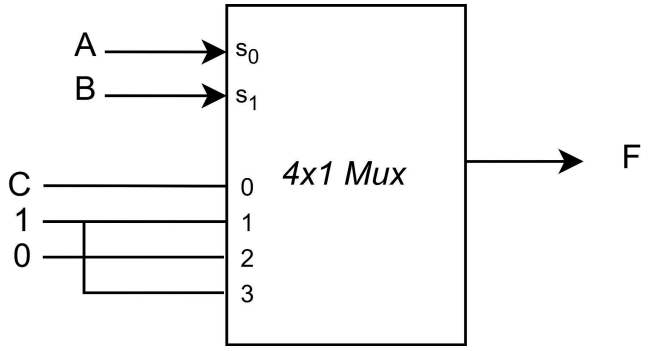
Truth table for Sum				
Select Lines				
x	y	z	Sum	F
0	0	0	0	F = z
0	0	1	1	
0	1	0	1	F = z'
0	1	1	0	
1	0	0	1	F = z'
1	0	1	0	
1	1	0	0	F = z
1	1	1	1	

Truth table for Carry				
Select Lines				
x	y	z	Carry	F
0	0	0	0	F = 0
0	0	1	0	
0	1	0	0	F = z
0	1	1	1	
1	0	0	0	F = z
1	0	1	1	
1	1	0	1	F = 1
1	1	1	1	



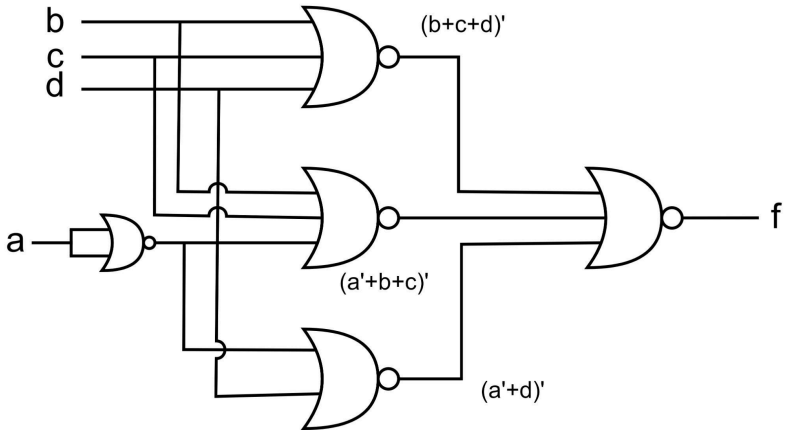
4. **Implement F=Σm(1,2,3,6,7) using 4 x 1 mux:**
   - **Solution:**

Truth table				
Select Lines				
A	**B**	**C**	**F=Σm(1,2,3,6,7)**	**F**
0	0	0	0	F = C
0	0	1	1	
0	1	0	1	F = 1
0	1	1	1	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	

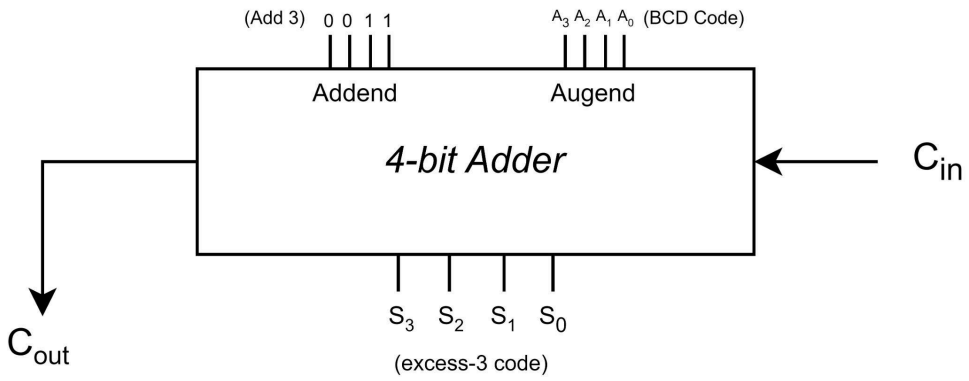


5. **NOR circuit realization of the function f = (b +c +d)(a’ + b + c)(a’ + d):**
   - **Solution:**
     Simplifying the function gives us the following equation for NOR equivalent circuit.

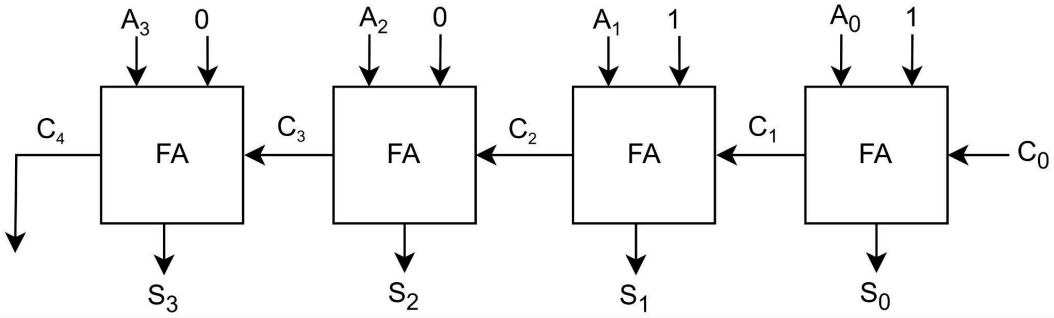
$$f = [(b+c+d)' + (a'+b+c)' + (a'+d)']'$$



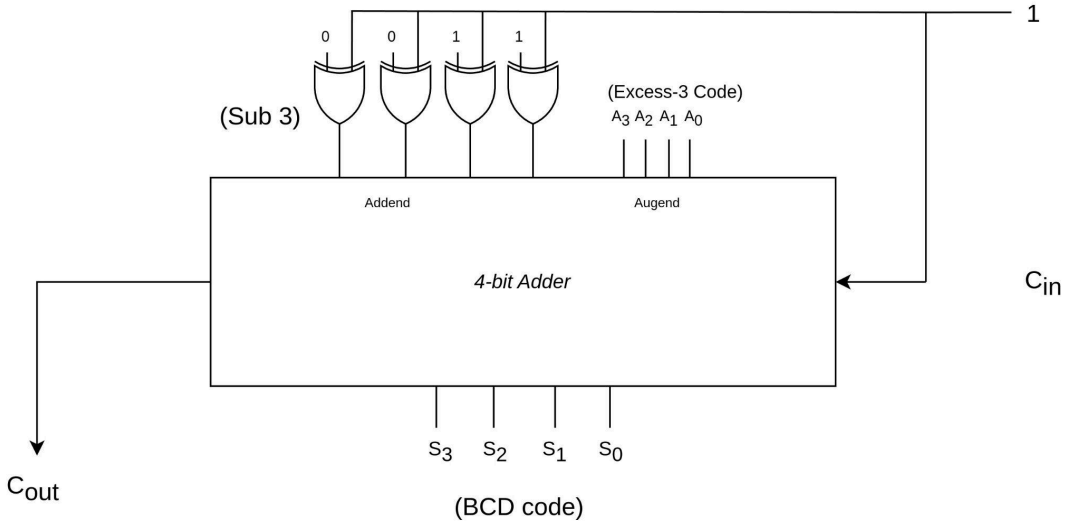
6. **BCD-to-excess-3-code converter:**
   - **Solution:**
     The block level implementation will look like the following. Here’s we will be adding 3 (0011) to each bit of the BCD code to create an excess-3 code at output using a 4-bit adder circuit.



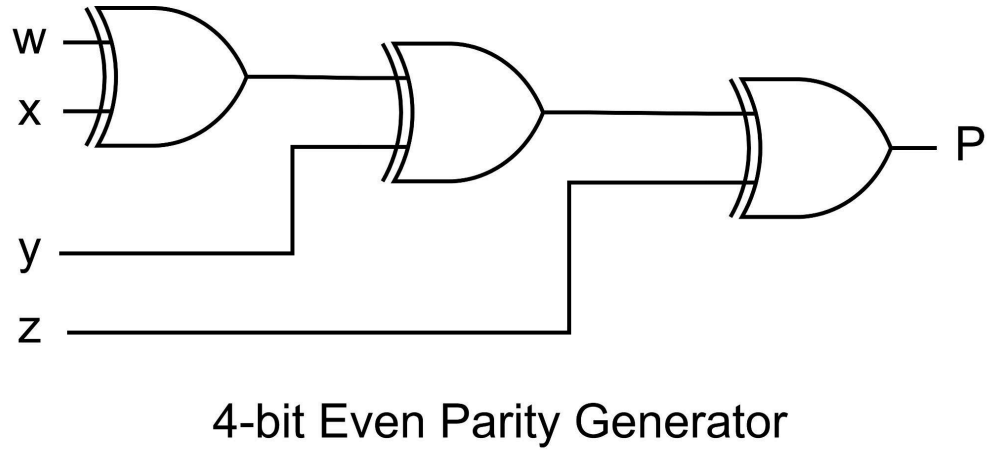
The 4-bit Adder is further broken down into 4 full-adder circuits. This is illustrated in the following:



C_0 in this case will be 0, since we are performing addition. In order to convert this circuit into excess-3-code-to-BCD converter, we will need to set **C_0** to 1 and alter our logic using xor gates to perform subtraction (Excess-3 code) - 3 = (BCD code). The circuit will look like this:



7. **4-bit Input Parity Generator:**
   - **Diagram:**



4-bit Even Parity Generator

   - **Verilog Code:**

```
module parity_generator ( input [3:0] data, output reg
parity);

always @*
     begin
          assign parity = data[0] ^ data[1] ^ data[2] ^
data[3];
     end
endmodule
```

   - **Testbench:**

```
     `timescale 1ns/1ps
     module test_tb;

     reg [3:0] data;
     wire parity;

     parity_generator dut(.data(data), .parity(parity));

     // Stimulus
     initial begin
     // Test with different inputs
     data = 4'b0000; // Even number of high bits
```

```
        #20;
        data = 4'b0001; // Odd number of high bits
        #20;
        data = 4'b1100; // Even number of high bits
        #20;
        data = 4'b1011; // Odd number of high bits
        #20;
        data = 4'b0000; // Even number of high bits
        #20;
        data = 4'b1101; // Odd number of high bits
        #20;
        end
        endmodule
```

- **Output:**