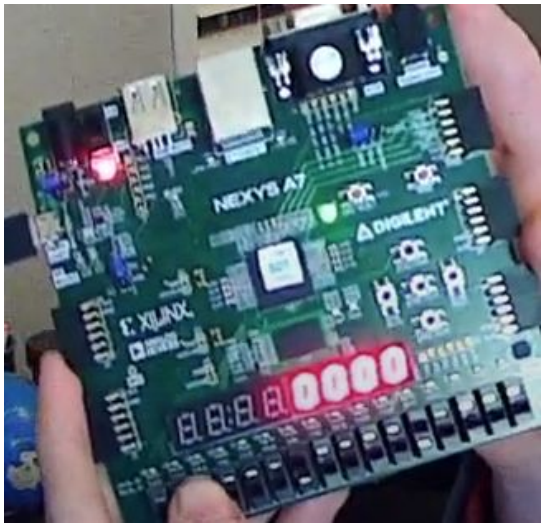
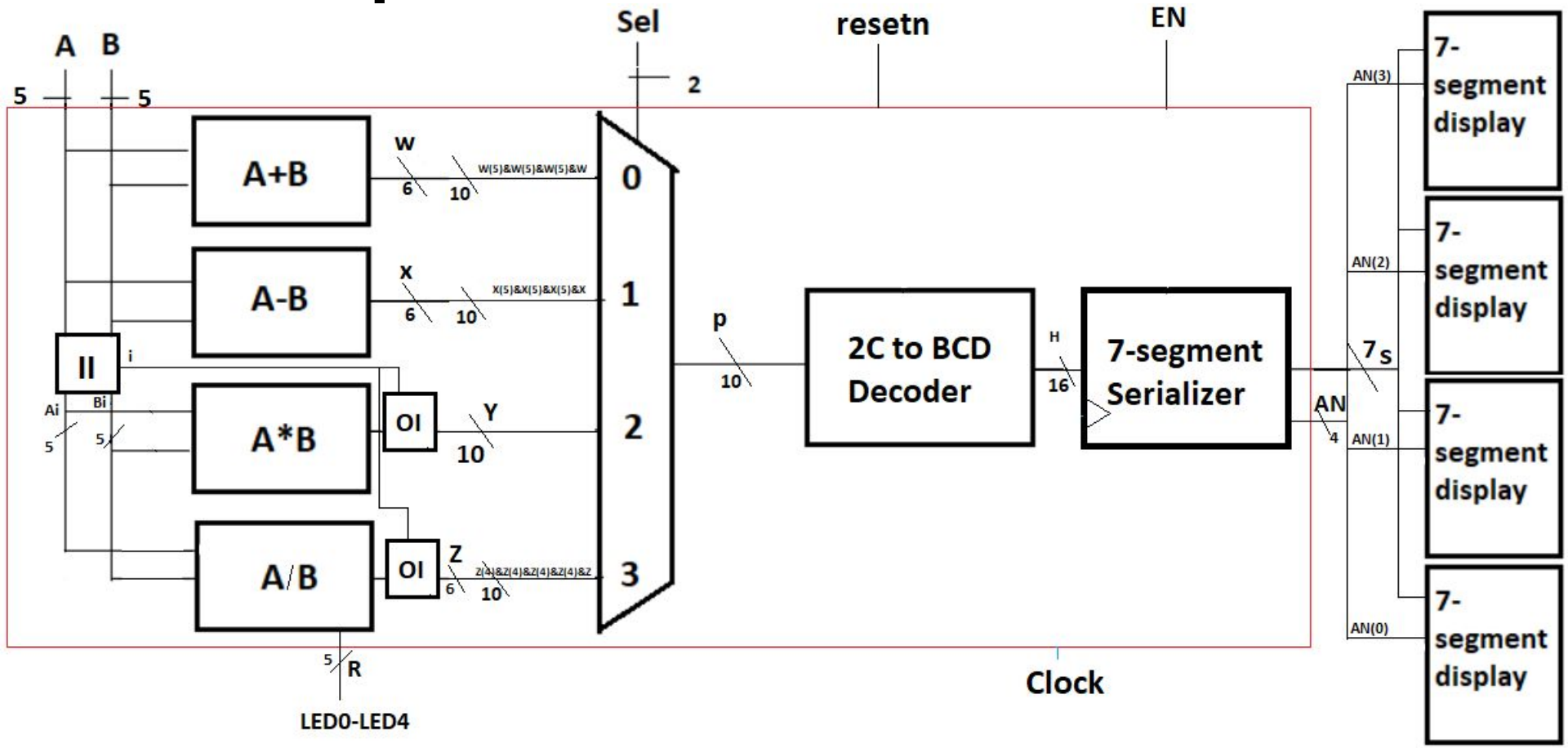
5-bit Signed Calculator with Switches

VHDL implemented calculator

Adam Kidwell, Braun Mayette, Christopher Gibson, Melad Haddad
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: bkidwell@oakland.edu, bmayette@oakland.edu, clgibson@oakland.edu, meladjoseph@oakland.edu



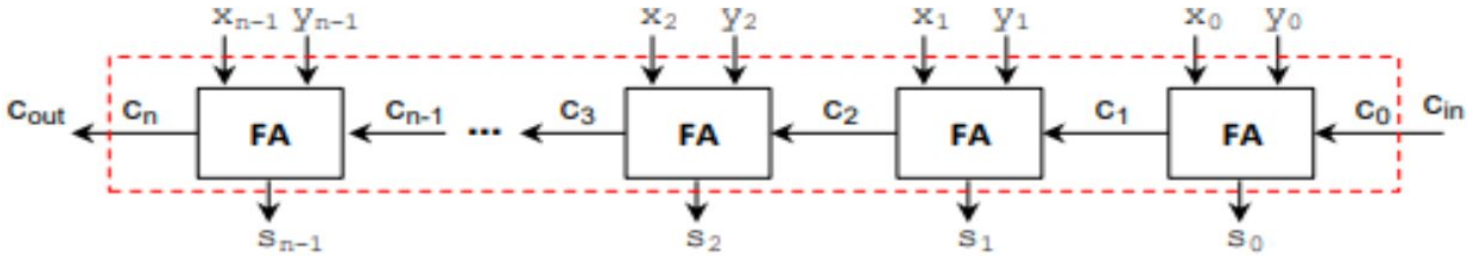
Topfile


A
B
5
5
Sel
2
resetn
EN
A+B
A-B
A*B
A/B
II
i
Ai
Bi
5
5
OI
OI
W
X
Y
Z
6
10
W(5)&W(5)&W(5)&W
X(5)&X(5)&X(5)&X
Z(4)&Z(4)&Z(4)&Z(4)&Z
0
1
2
3
p
10
2C to BCD Decoder
H
16
7-segment Serializer
AN
4
7s
AN(3)
AN(2)
AN(1)
AN(0)
7-segment display
7-segment display
7-segment display
7-segment display
5
R
LED0-LED4
Clock

6-bit Adder

Modified lab 2 for a 4-bit adder to create a 6-bit adder. 6-bit adder modified to take 6 bit input and output

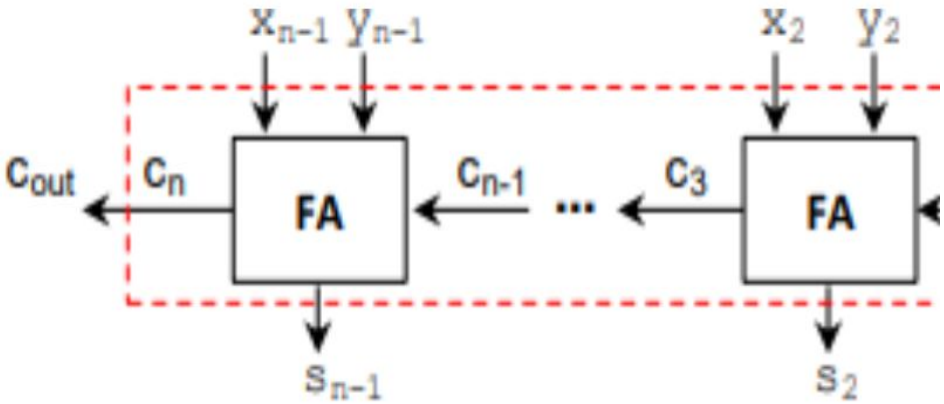
```
entity melad_6bitAdder is

Port ( X : in STD_LOGIC_VECTOR
          (5 downto 0);

 Y : in STD_LOGIC_VECTOR
           (5 downto 0);

 Cin : in STD_LOGIC;

 S : out STD_LOGIC_VECTOR
          (5 downto 0);
 Cout : out STD_LOGIC);

 end melad_6bitAdder;
```



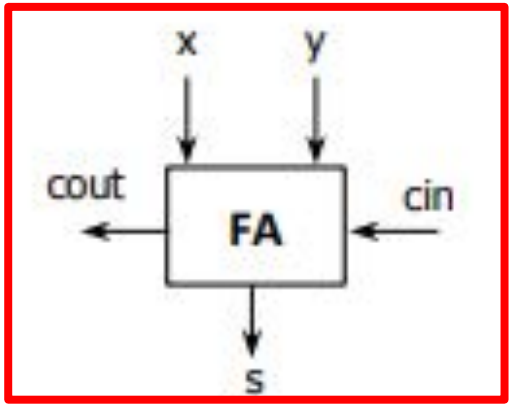
6 Full-Adder

The 6-bit signed adder is a simple combinational circuit where the inputs are A and B . By extending A and B to 6-bits sign to avoid overflow, and with the use of 6 full-adder with a Cn-1 sutractor that was added will be able to produce signal S 5 downto 0. This will result in the sum of two numbers either positive or negative.

```
component melad_fulladder
port (X : in STD_LOGIC;

      Y : in STD_LOGIC;

      Cin : in STD_LOGIC;

      S : out STD_LOGIC;

      Cout : out STD_LOGIC);
end component;
```



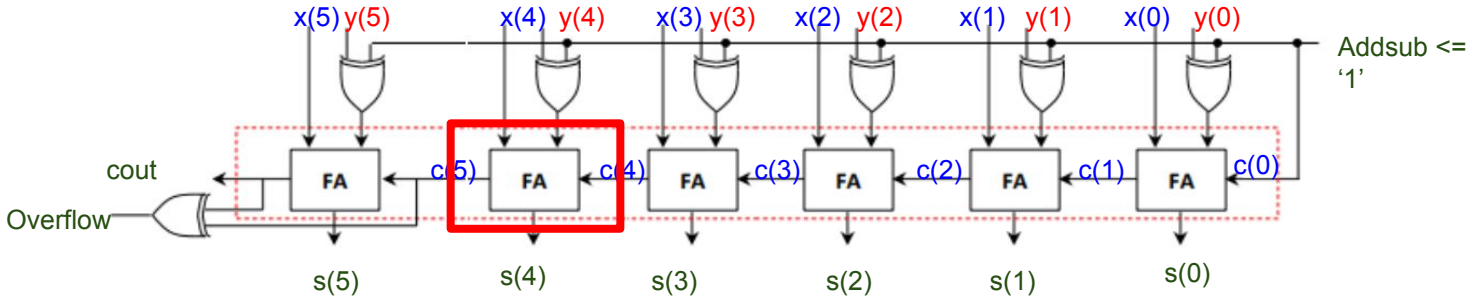
Subtractor (2's complement)



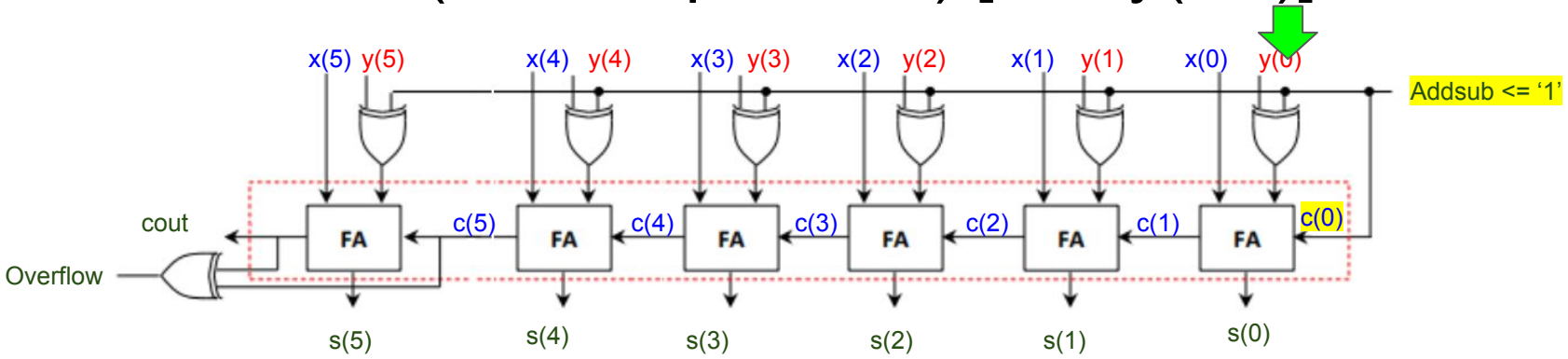
FULL ADDER

The full adder was added from a previous assignment to be used as a component in the final combinational subtracter (2C).

```
entity fulladd is

port( cin, x, y : in std_logic;
         s, cout    : ou
t std_logic);
end fulladd;

architecture structure of fulladd
is

begin

      s <= x xor y xor cin;
      cout <= (x and y) or (x and
cin) or (y and cin);

end structure;
```



Subtractor cont. (2's complement) [x + y(2C)]



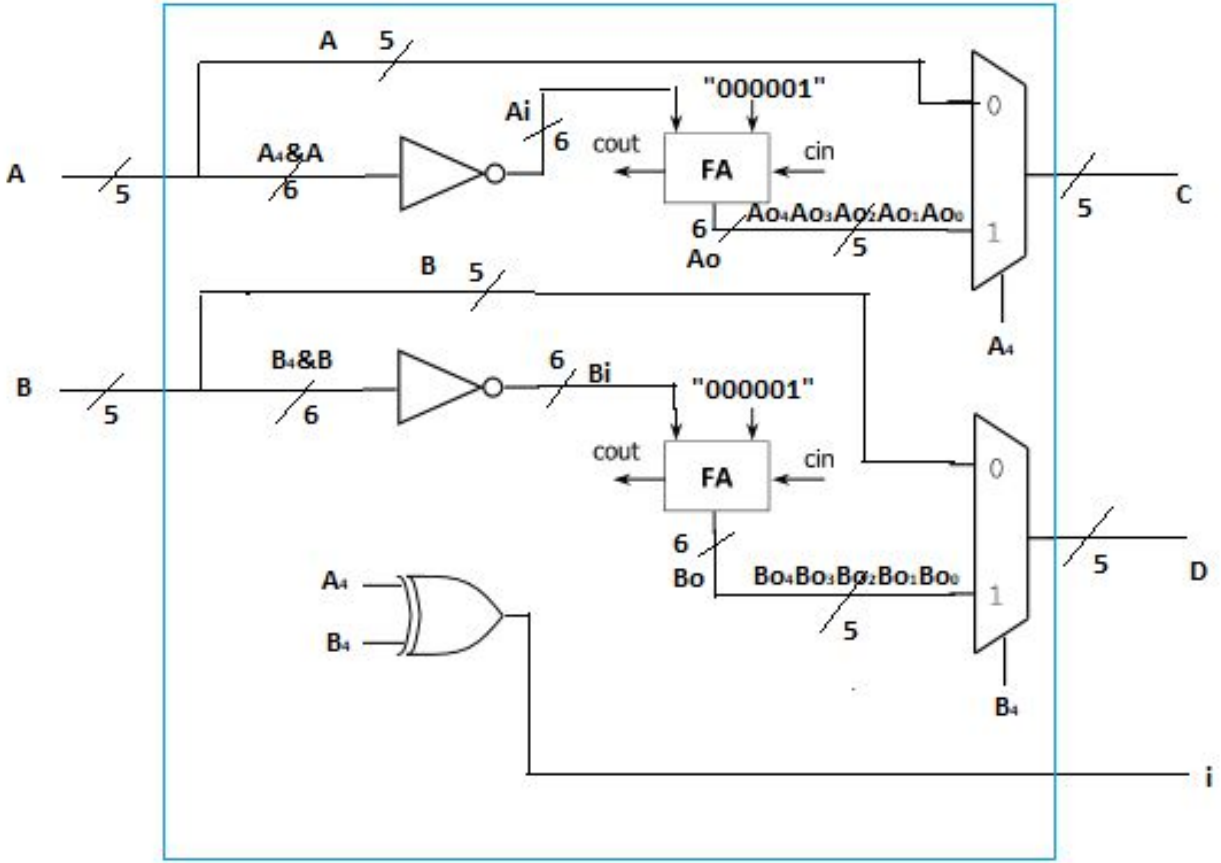

```
signal c: std_logic_vector (6 downto 0);
       signal yx: std_logic_vector (5 downto 0);

begin

   c(0) <= addsub; cout <= c(6);
       overflow <= c(6) xor c(5);

       gi: for i in 0 to 5 generate
                          yx(i) <= y(i) xor addsub;
                               fi: fulladd port map (cin => c(i), x => x(i), y => yx(i), s => s(i), cout => c(i+1));
           end generate;

end structure;
```

With the modification from a previous code, a combinational circuit was created with logic gates to form a 2's complement subtractor with the input of two five bit signed numbers.

Input Inverter

This is the component that converts negative inputs to positive. Bit “i’ tracks if the output will need to be inverted.

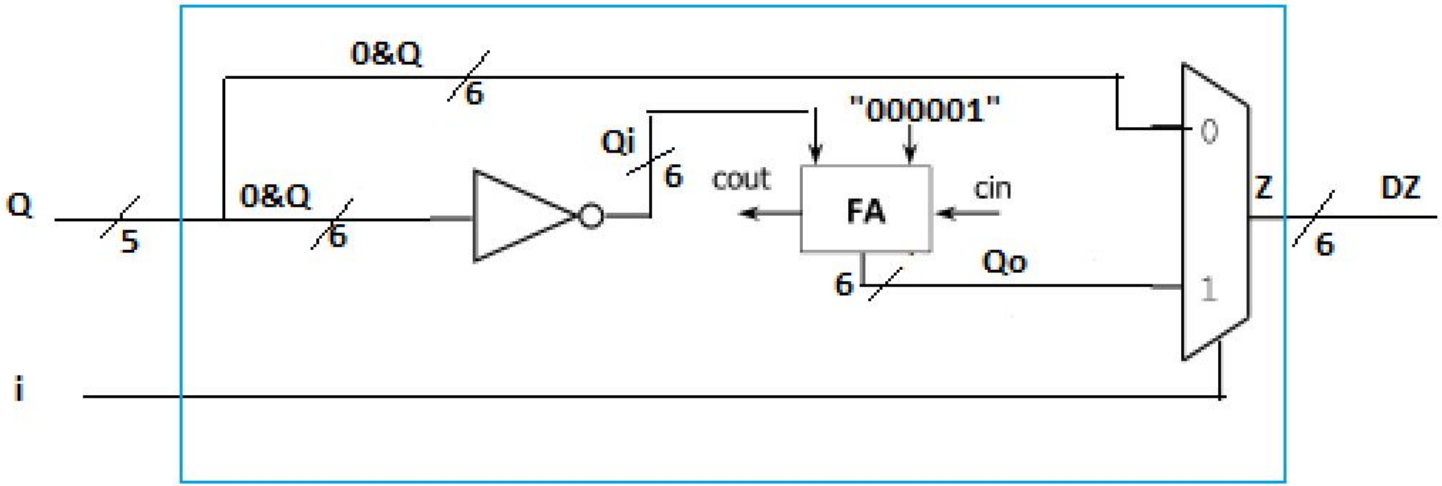
This component is necessary for our multiplier and divider, which only work with unsigned binary



Output Inverter

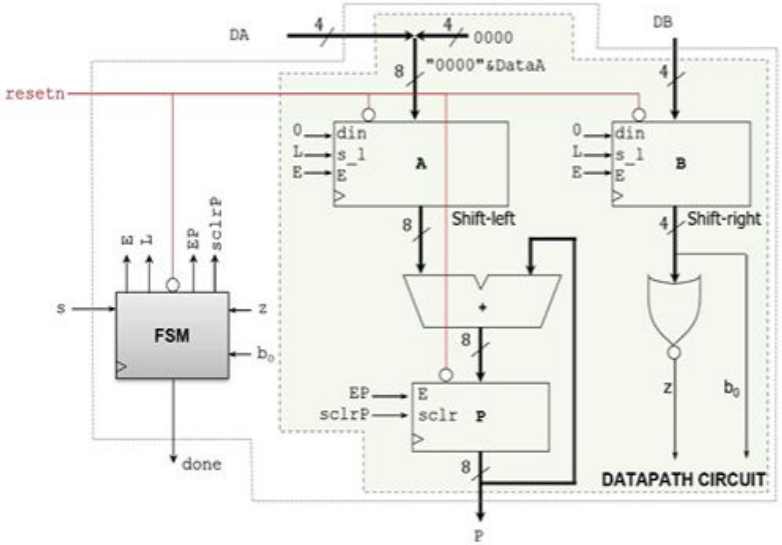
This is the component to invert the output of the divider, if necessary.

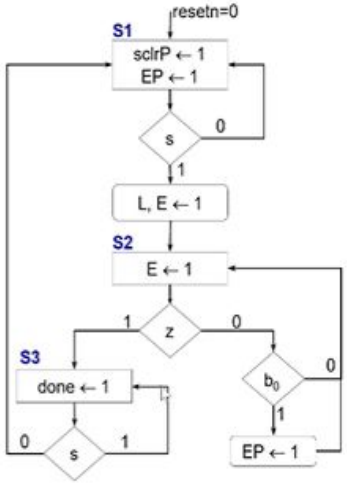
The same diagram is applicable with 10 bit inputs and outputs for the multiplier



Multiplier

Sequential multiplier modified to take two five bit inputs and output a ten bit answer.The sequential multiplier works through the use of two different types of registers, an adder and a fsm Mealy state machine



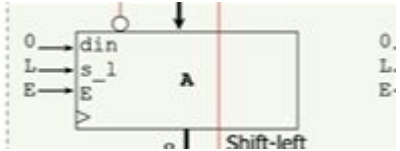


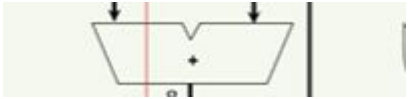
Multiplier cont.

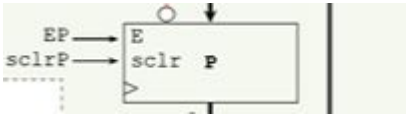
Parallel access shift register: These registers load the two five bit inputs into the adder and the xor gate.

Adder: The adder controls the multiplication of the circuit, replicating the inputs untils the correct answer is obtained.

Register: The register holds the output and transmits it to next location, being the leds and switches.

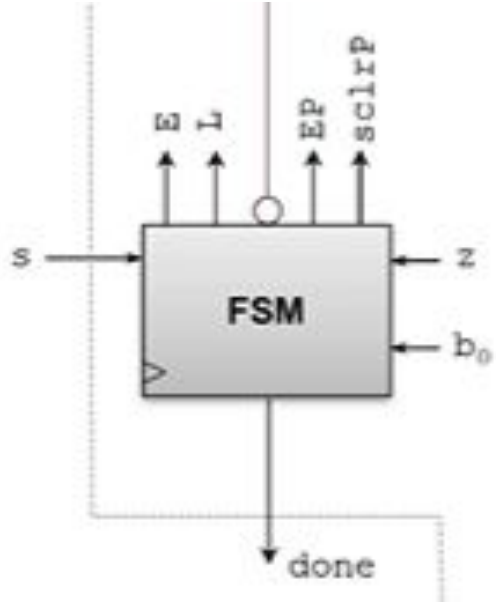


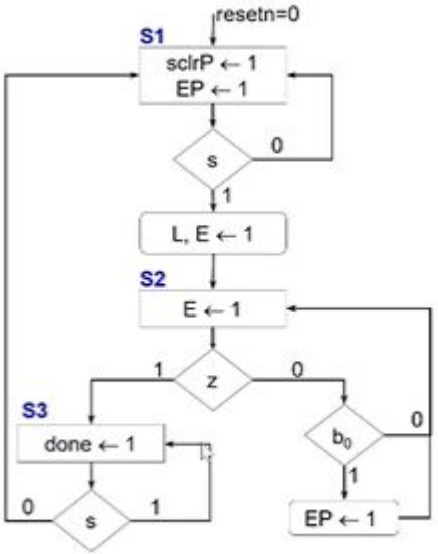




Multiplier Cont. FSM (Mealy)

FSM: The fsm state machine controls the operation of the multiplier through the use of binary numbers. State one checks whether or not an input is entered. State two ensure that the input is valid and then calculates an answer. If the input is invalid then the fsm will restart the process from state two. State three sends the output (answer) to its next location.



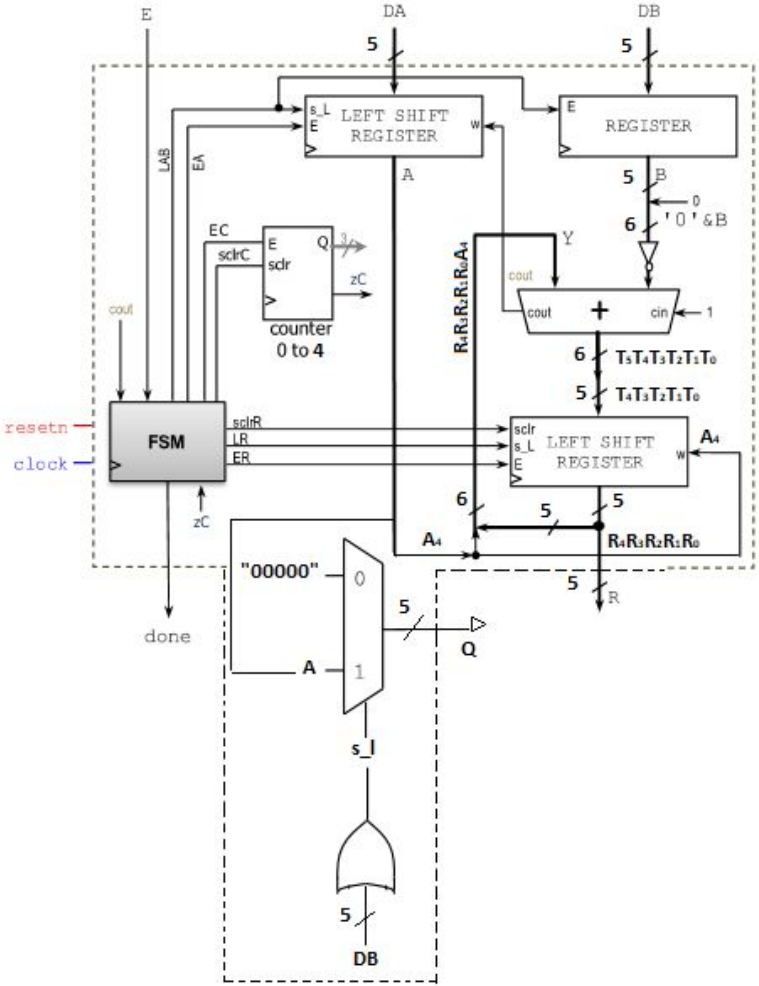


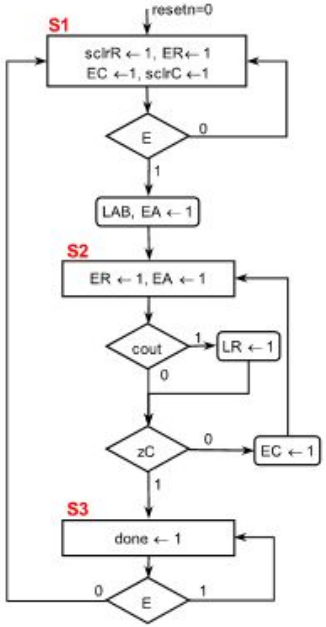
Divider

Modified Lab 6 for two 5 bit outputs. Additionally, added output MUX to prevent errors when dividing by 0.

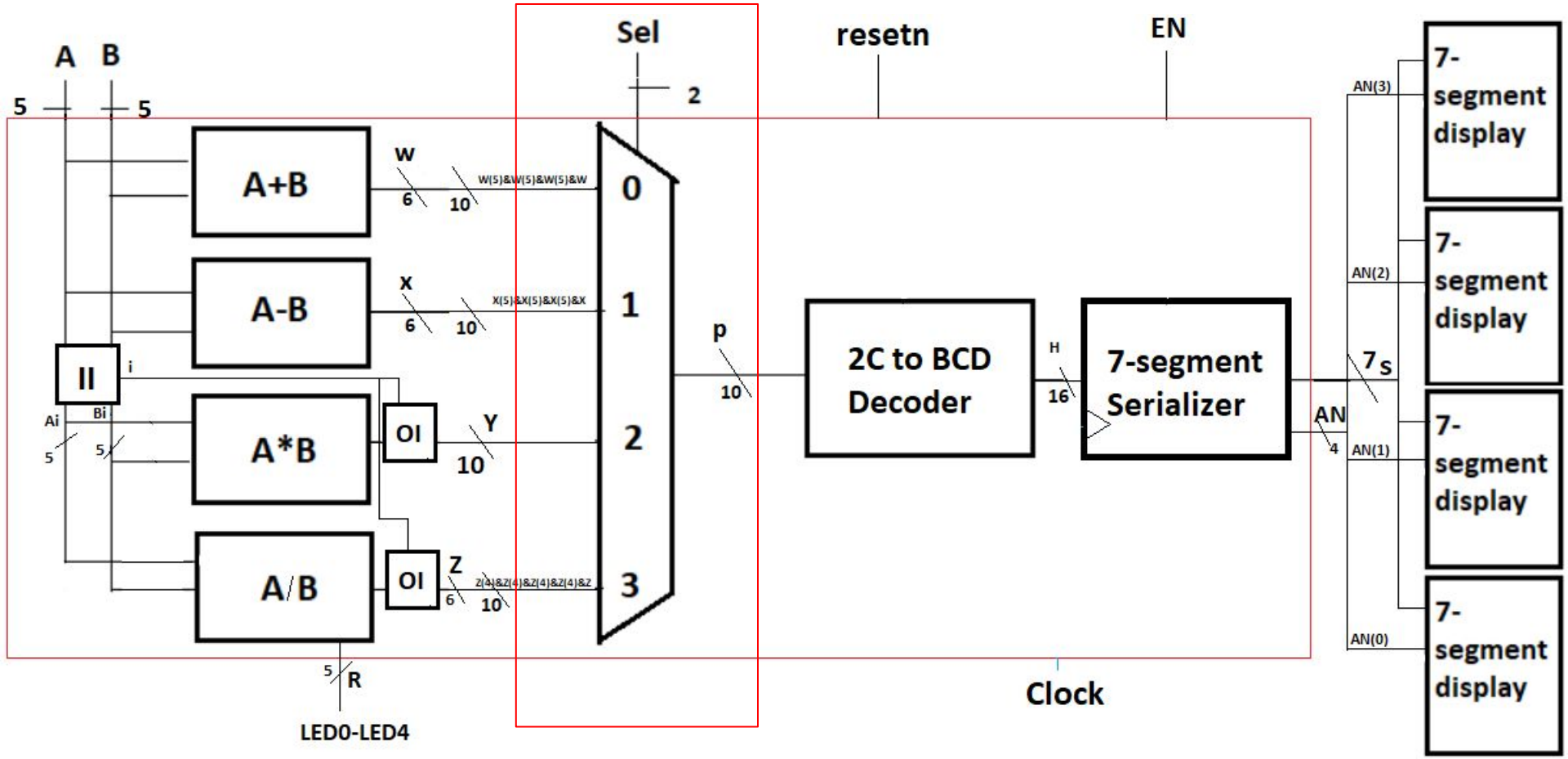
ALGORITHM

```
R = 0
for i = n-1 downto 0
    left shift R (input = a_i)
    if R ≥ B
       q_i = 1, R ← R-B
    else
       q_i = 0
    end
end
```





Operation Selecting MUX (10-bit)



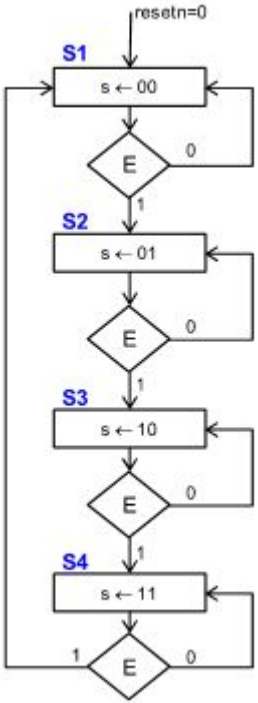
2C to BCD Decoder

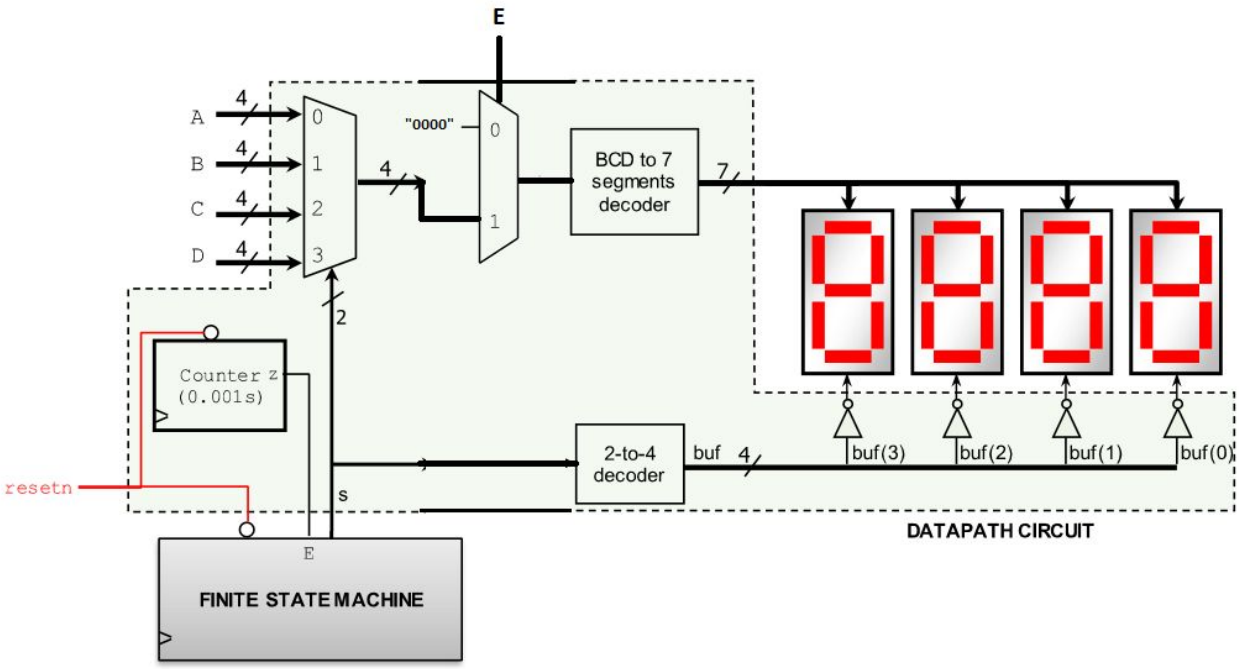
Converts the outputs of our components from 2C to BCD for the 7-segment displays

```
with ILUT select
    OLUT <=
      "0000000000000000" when "0000000000", --0
      "0000000000000001" when "0000000001", --1
      "0000000000000010" when "0000000010", --2
      "0000000000000011" when "0000000011", --3
      "0000000000000100" when "0000000100", --4
      "0000000000000101" when "0000000101", --5
      "0000000000000110" when "0000000110", --6
      "0000000000000111" when "0000000111", --7
      "0000000000001000" when "0000001000", --8
      "0000000000001001" when "0000001001", --9
      "0000000000010000" when "0000001010", --10
      "0000000000010001" when "0000001011",
      "0000000000010010" when "0000001100",
      "0000000000010011" when "0000001101",
      "0000000000010100" when "0000001110",
      "0000000000010101" when "0000001111", --15
      "0000000000010110" when "0000010000",
       ...
```

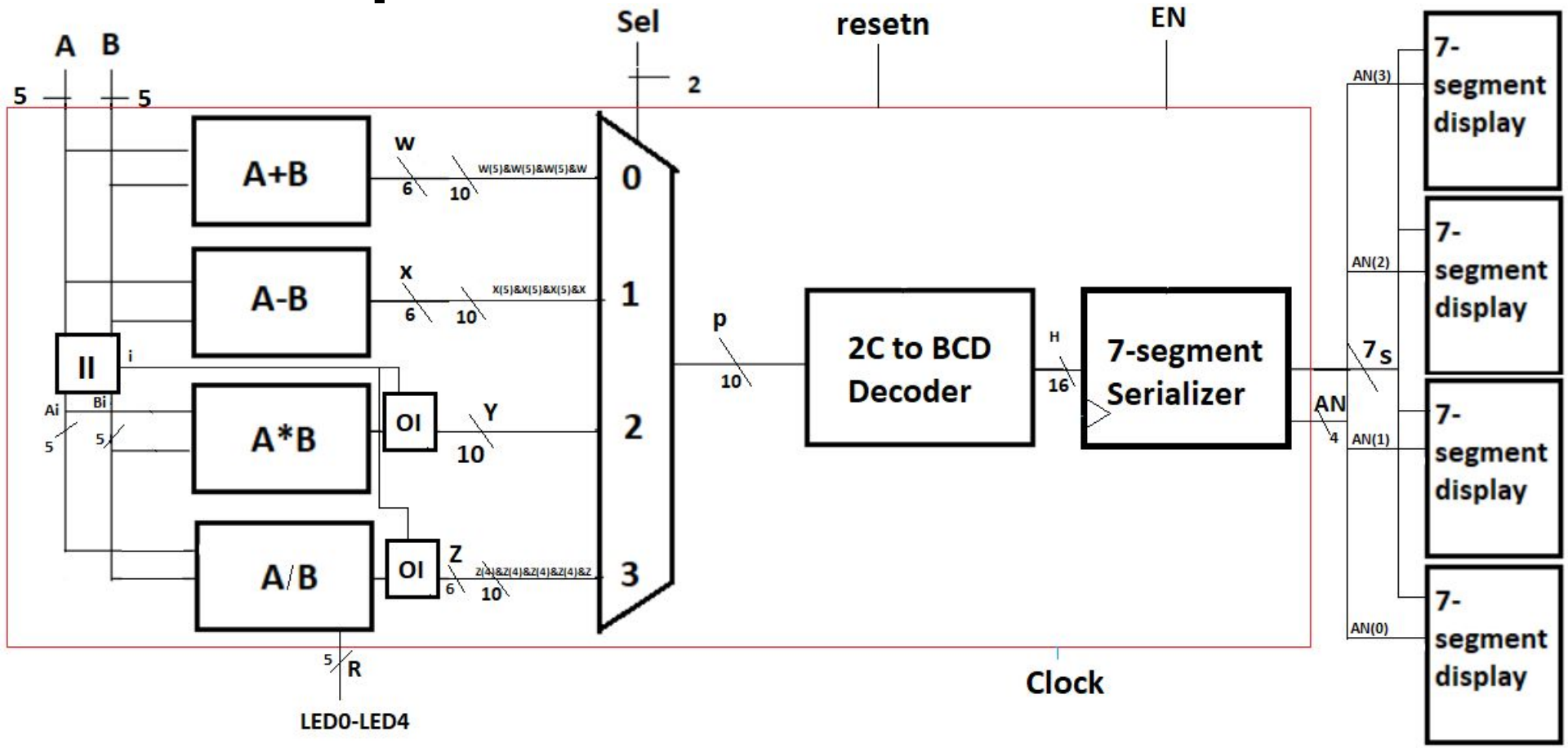
7-Segment Serializer

Slightly modified code from VHDL page. Added an enable to the output for improved user experience.





Topfile


A
B
5
5
Sel
2
resetn
EN
A+B
A-B
A*B
A/B
II
i
Ai
Bi
5
5
OI
OI
W
X
Y
Z
6
10
W(5)&W(5)&W(5)&W
X(5)&X(5)&X(5)&X
Z(4)&Z(4)&Z(4)&Z(4)&Z
0
1
2
3
p
10
2C to BCD Decoder
H
16
7-segment Serializer
AN
4
7s
AN(3)
AN(2)
AN(1)
AN(0)
7-segment display
7-segment display
7-segment display
7-segment display
5
R
LED0-LED4
Clock