Embedded 1

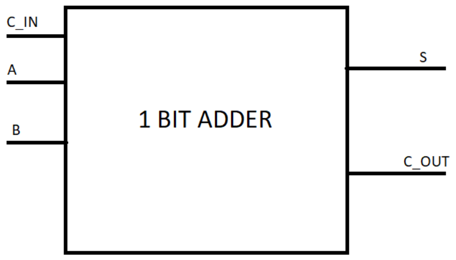
Lab Report #2

Justin Evron

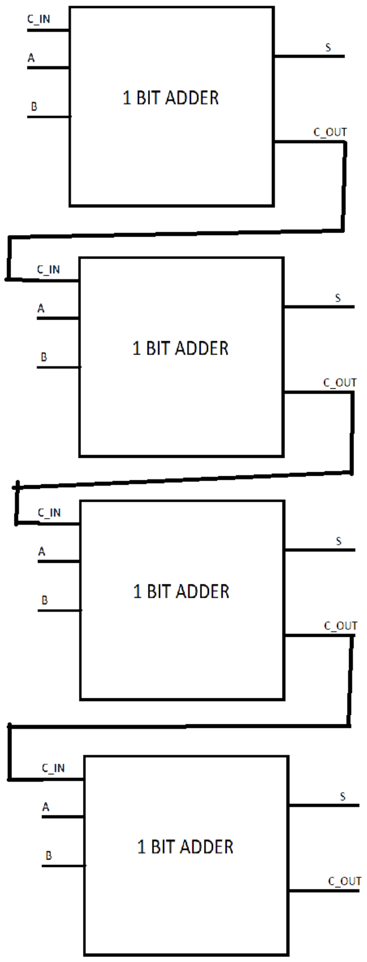
2/6/19

PRELAB

1. Y = ((A XOR B) XOR C))
   Cout = ((A XOR B) and Cin) or A and B
2.



3.

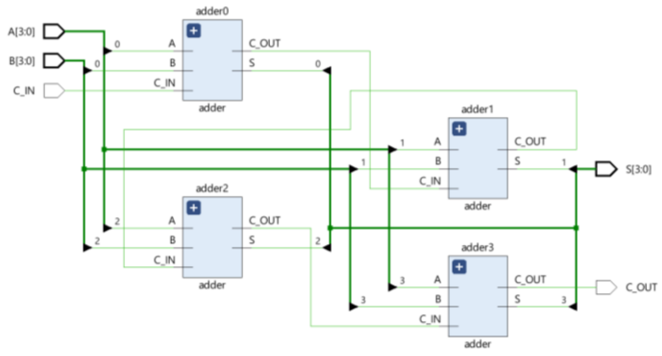


Adders

Operation

The circuit should add the A and B inputs and output them to S with C_out acting as the overflow indicator.

Schematic



Design

1 bit adder

```
entity adder is
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          C_IN : in STD_LOGIC;
          C_OUT : out STD_LOGIC;
          S : out STD_LOGIC);
end adder;
architecture Behavioral of adder is
signal AB : STD_LOGIC;
begin
   AB <= A XOR B;
   S <= AB XOR C_IN;
   C_OUT <= (AB AND C_IN) OR (A AND B);
end Behavioral;
```

Ripple Adder

```
entity ripple_adder is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          C_IN : in STD_LOGIC;
          C_OUT : out STD_LOGIC;
          S : out STD_LOGIC_VECTOR (3 downto 0));
end ripple_adder;
architecture Behavioral of ripple_adder is
   component adder is
```

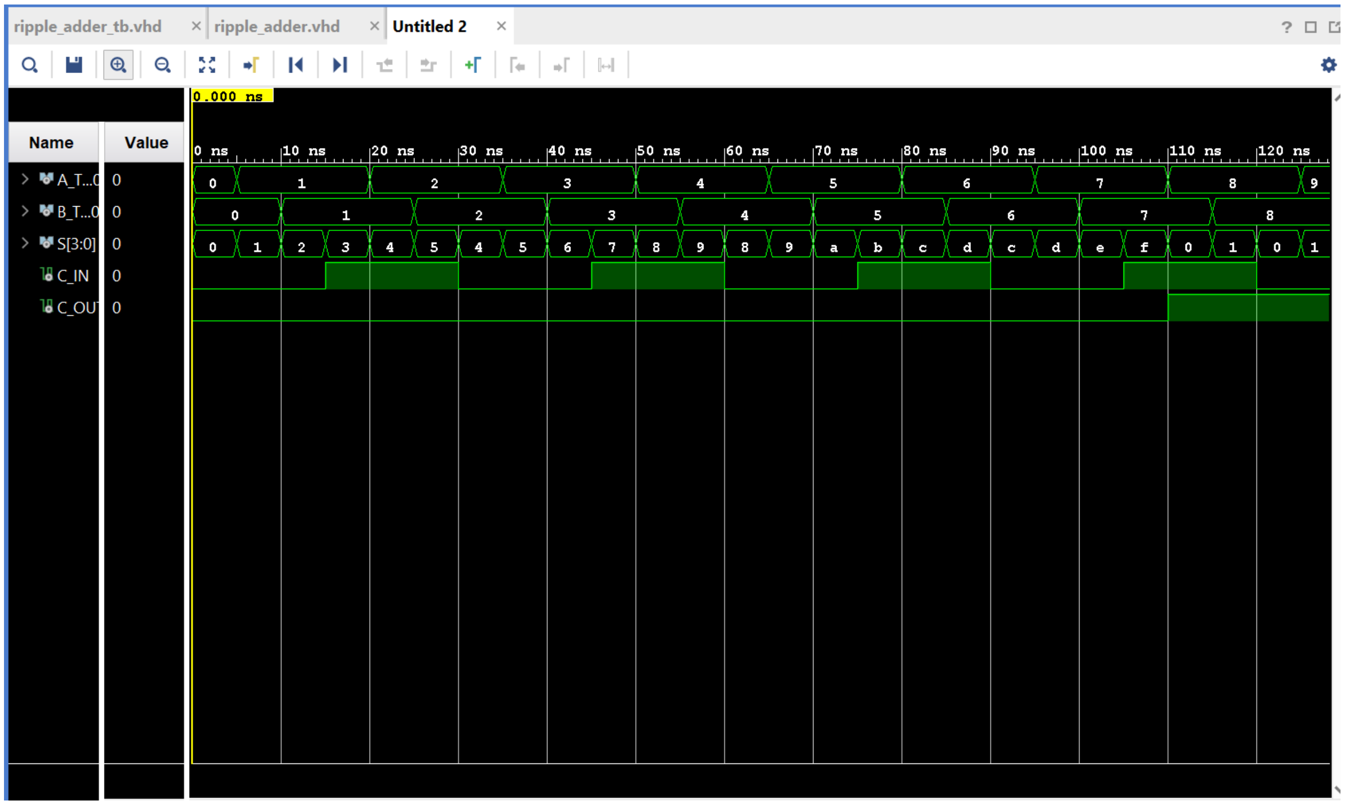
```
            Port ( A : in STDLOGIC;
                  B : in STD_LOGIC;
                  C_IN : in STD_LOGIC;
                  C_OUT : out STD_LOGIC;
                  S : out STD_LOGIC);
        end component;
        signal C0, C1, C2 : STD_LOGIC;
    begin
    adder0: adder
    port map(A => A(0),
            B => B(0),
            C_IN => C_IN,
            S => S(0),
            C_OUT => C0);
    adder1: adder
    port map(A => A(1),
            B => B(1),
            c_IN => C0,
            S => S(1),
            C_OUT => C1);
    adder2: adder
    port map(A => A(2),
            B => B(2),
            C_IN => C1,
            S => S(2),
            C_OUT => C2);
    adder3: adder
    port map(A => A(3),
            B => B(3),
            C_IN => C2,
            S => S(3),
            C_OUT => C_OUT)
    end Behavioral;
```

Testbench

```
 entity ripple_adder_tb is
--  Port ( );
end ripple_adder_tb;

architecture Behavioral of ripple_adder_tb is
component ripple_adder is
Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
      B : in STD_LOGIC_VECTOR (3 downto 0);
      C_IN : in STD_LOGIC;
      C_OUT : out STD_LOGIC;
```

```
        S : out STD_LOGIC_VECTOR (3 downto 0));
end component;
signal A_TB, B_TB, S: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
signal C_IN, C_OUT : STD_LOGIC := '0';
begin
ADDER:RIPPLE_ADDER
PORT MAP(A => A_TB,
        B => B_TB,
        C_IN => C_IN,
        C_OUT => C_OUT,
        S => S);
    process
    begin
    wait for 5 ns;
    A_TB <= STD_LOGIC_VECTOR(UNSIGNED(A_TB) + 1);
    wait for 5 ns;
    B_TB <= STD_LOGIC_VECTOR(UNSIGNED(B_TB) + 1);
    wait for 5 ns;
    C_in <= not(C_IN);
    end process;
end Behavioral;
```

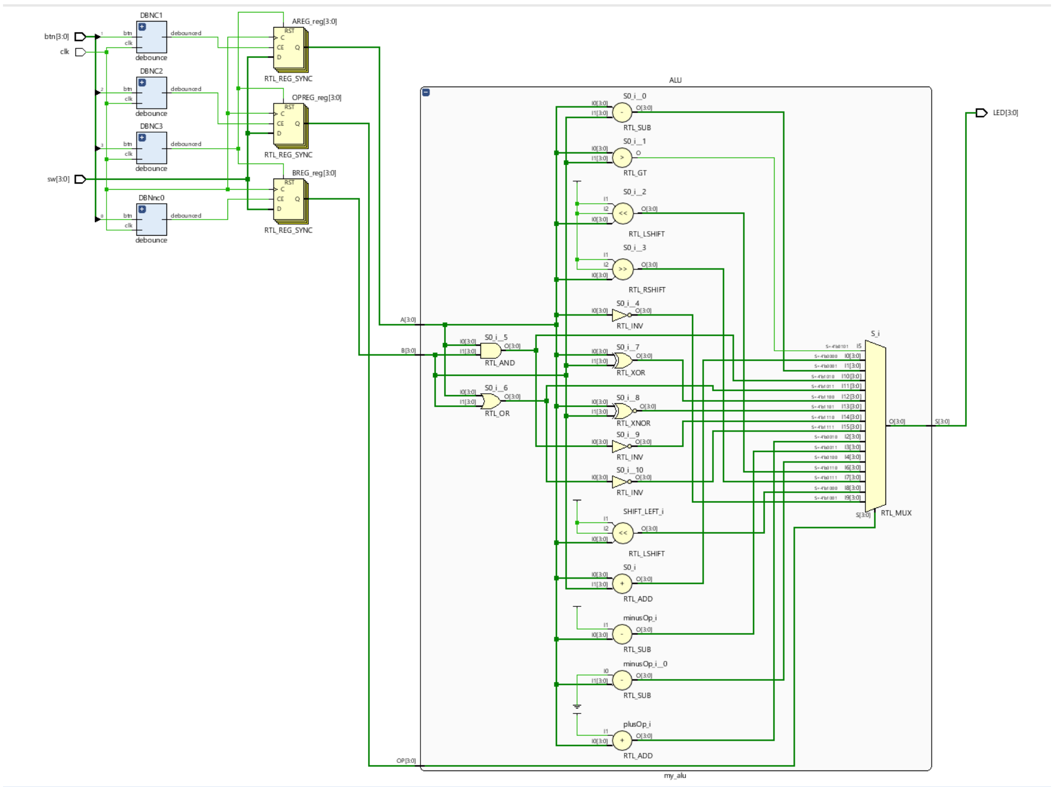


16 function 4 bit ALU

Operation

The circuit should take 3 four bit inputs (A, B, OP) and perform operations on A and B based on OP.

Schematic



Design

```
entity my_alu is
   Port ( OP : in STD_LOGIC_VECTOR (3 downto 0);
         A : in STD_LOGIC_VECTOR (3 downto 0);
         B : in STD_LOGIC_VECTOR (3 downto 0);
         S : out STD_LOGIC_VECTOR (3 downto 0));
end my_alu;
architecture Behavioral of my_alu is
begin
process
begin
case OP is
   when X"0" => S <= STD_LOGIC_VECTOR(UNSIGNED(A) + UNSIGNED(B));
   when X"1" => S <= STD_LOGIC_VECTOR(UNSIGNED(A) - UNSIGNED(B));
   when X"2" => S <= STD_LOGIC_VECTOR(UNSIGNED(A) + 1);
```

```
    when X"3" => S <= STD_LOGIC_VECTOR(UNSIGNED(A) - 1);
    when X"4" => S <= STD_LOGIC_VECTOR(0 - UNSIGNED(A));
    when X"5" => S <= x"1" when (UNSIGNED(A) > UNSIGNED(B)) else x"0";
    when X"6" => S <= STD_LOGIC_VECTOR(UNSIGNED(A) sll 1);
    when X"7" => S <= STD_LOGIC_VECTOR(UNSIGNED(A) srl 1);
    when X"8" => S <= STD_LOGIC_VECTOR(SHIFT_LEFT(UNSIGNED(A),1));
    when X"9" => s <= NOT(A);
    when X"A" => S <= A AND B;
    when X"B" => S <= A OR B;
    when X"C" => S <= A XOR B;
    when X"D" => S <= A XNOR B;
    when X"E" => S <= A NAND B;
    when X"F" => S <= A NOR B;
end case;
end process;
end Behavioral;
```

Implementation