Task 1 Design Process

Task 1 was pretty straightforward in terms of entering the necessary equations required for task 1. We decided to “hard code” our inputs,A, B, and Cin, since we figured it would be easier for us to keep track of. We only had two issues in our simulation source. The first issue was that there was an error that kept appearing at the Port(map) line. We fixed the issues by stating that each variable would be less than or each to their own variable which fixed the errors. The second issue was that our initial results appeared to have different results from the example. We realized that our “hard code” was not in the proper order as our binary numbers were mixed up. Once that was fixed we easily

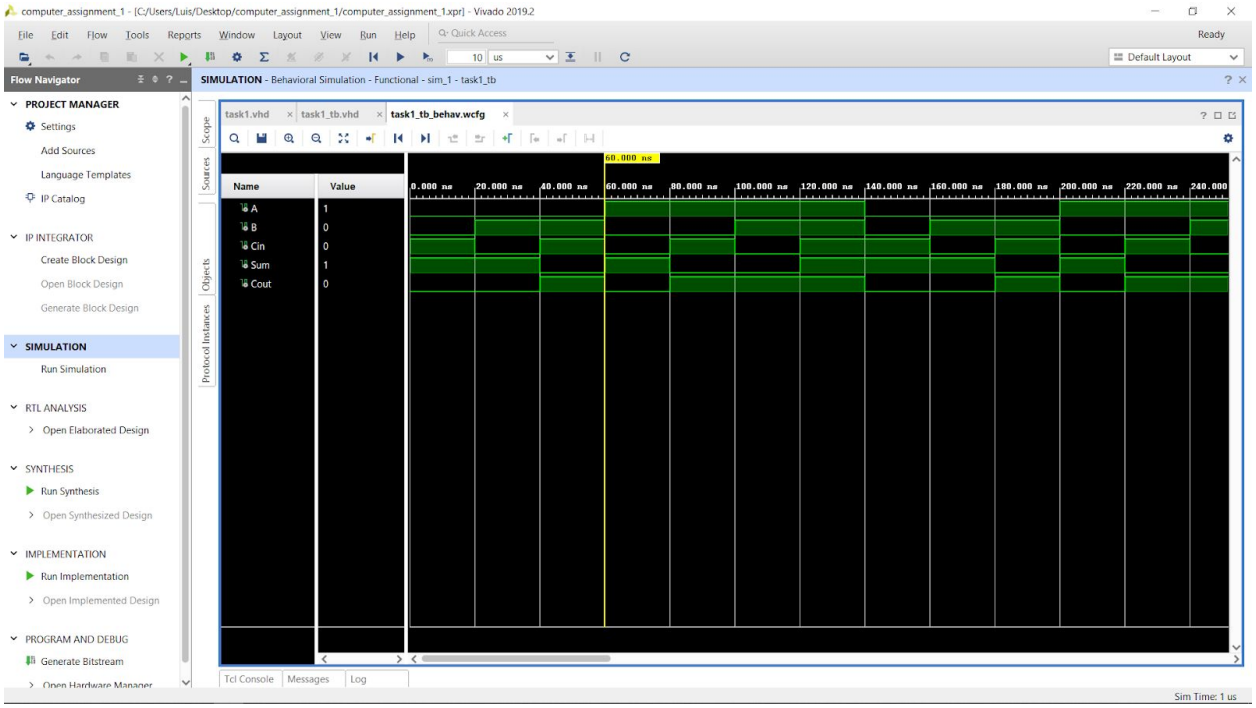
Task 1 Design Source Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity task1 is

Port ( A,B,Cin : in STD_LOGIC;

Sum,Cout : out STD_LOGIC);

end task1;

architecture Behavioral of task1 is

begin
 Sum <= A xor B xor Cin;
 Cout <= ((A and B) or (A and Cin) or (B and Cin));


end Behavioral;
```

Task 1 Simulation Source Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity task1_tb is
end task1_tb;
architecture Behavioral of task1_tb is
COMPONENT task1 is

PORT(A,B,CIN: IN std_logic;

Sum,Cout : OUT std_logic);

END COMPONENT;

Signal A, B, Cin, Sum, Cout : STD_LOGIC;

begin
uut: task1
Port Map(A => A, B => B, Cin => Cin, Sum => Sum, Cout => Cout);

test: process

begin

A <= '0'; B <= '0'; Cin <= '1'; wait for 20 ns;

A <= '0'; B <= '1'; Cin <= '0'; wait for 20 ns;

A <= '0'; B <= '1'; Cin <= '1'; wait for 20 ns;

A <= '1'; B <= '0'; Cin <= '0'; wait for 20 ns;

A <= '1'; B <= '0'; Cin <= '1'; wait for 20 ns;

A <= '1'; B <= '1'; Cin <= '0'; wait for 20 ns;

A <= '1'; B <= '1'; Cin <= '1'; wait for 20 ns;

end process test;
end Behavioral;
```

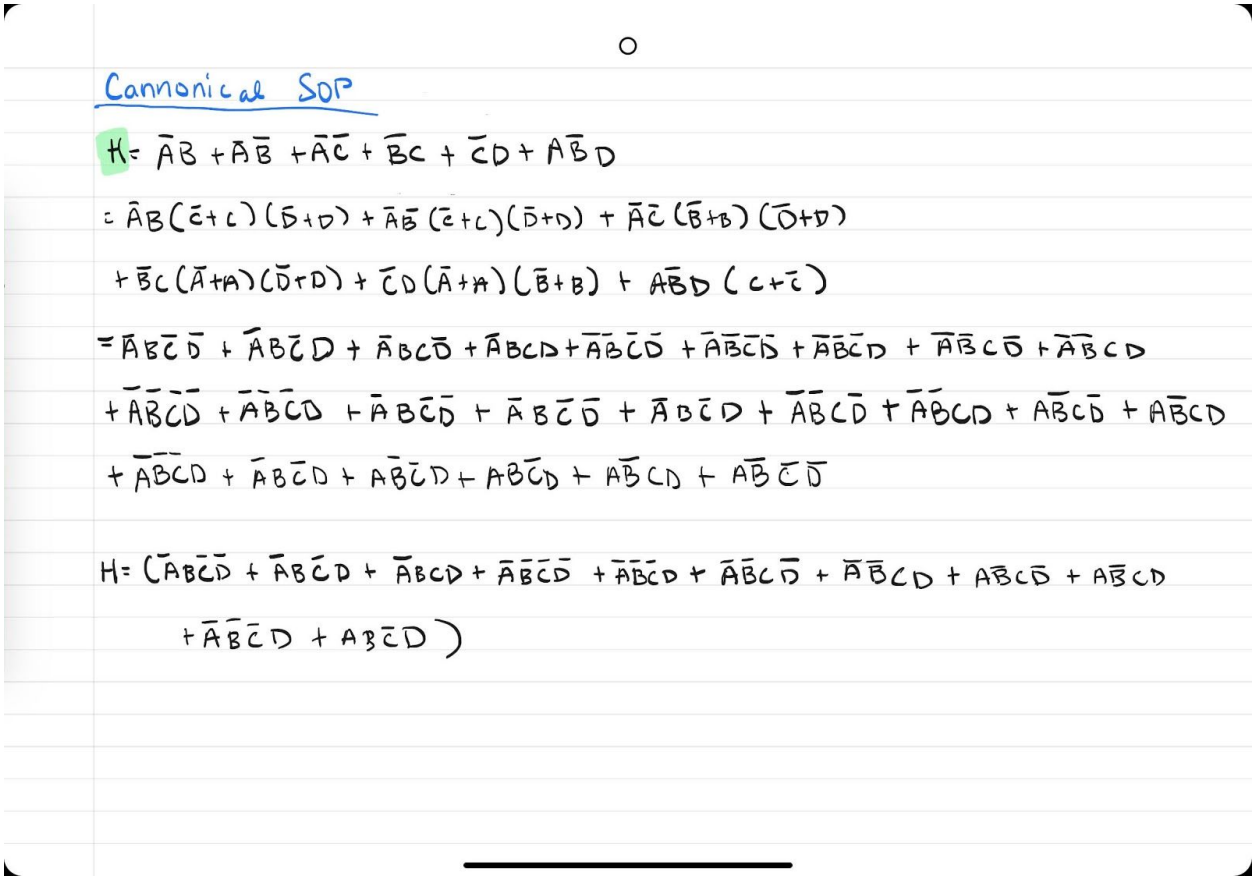
Task 1 Results



Task 2 Design Process

Task 2 was more of a headache then it should have been. We constantly had to look over and over again to make sure our F was correct so that the simulation produced the exact same results as our answers of G and H. The simulation source was roughly the same as task one in terms of hardcoding, but we had no issues with it. We had a ton of issues with our design source code since almost all of our answers did not match with F. Until we realized that one of our NOTS were covering the whole equation when it should have been only under the XOR. Once that was resolved we finally got our simulation to math G AND H.

Task 2 work for G and H



Task 2 Design Source Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity task2 is

Port ( A,B,C,D : in STD_LOGIC;

F,G,H : out STD_LOGIC);

end task2;

architecture Behavioral of task2 is

begin
 F <= NOT
```

```
 (NOT ((A AND (NOT C) AND (NOT D)) XOR
 (((NOT A) AND B) OR (NOT (B AND C))))
  OR (((NOT A) AND C AND D) AND ((B AND (NOT C))
  OR (A AND (NOT B) AND C AND (NOT D)) OR ((NOT A) AND C AND (NOT D)))));

G <= ((NOT A) AND B) OR
((NOT A) AND (NOT B))OR ((NOT A) AND (NOT C)) OR
 ((NOT B) AND C) OR
 ((NOT C) AND D) OR
 (A AND (NOT B) AND D);

H <= ((NOT A) AND B AND (NOT C) AND (NOT D)) OR
((NOT A) AND B AND (NOT C) AND D) OR
((NOT A) AND B AND C AND (NOT D)) OR
((NOT A) AND B AND C AND D) OR ((NOT A) AND (NOT B) AND (NOT C) AND (NOT D)) OR
((NOT A) AND (NOT B) AND (NOT C) AND D) OR
((NOT A) AND (NOT B) AND C AND (NOT D)) OR
((NOT A) AND (NOT B) AND C AND D) OR
((NOT A) AND B AND (NOT C) AND (NOT D)) OR
(A AND (NOT B) AND C AND (NOT D)) OR
(A AND (NOT B) AND C AND D) OR
(A AND (NOT B) AND (NOT C) AND D) OR
(A AND B AND (NOT C) AND D);
end Behavioral;
```

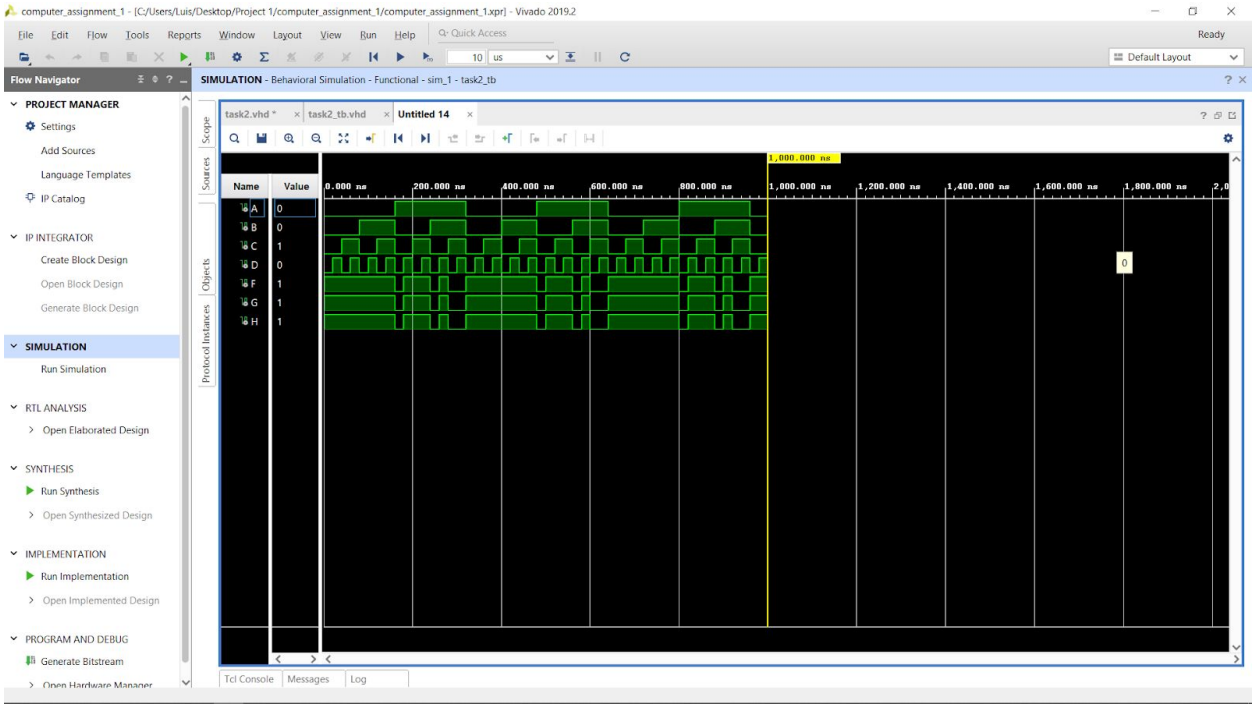
Task 2 Simulation Source Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity task2_tb is
end task2_tb;
architecture Behavioral of task2_tb is
COMPONENT task2 is

PORT(A,B,C,D: IN std_logic;

F,G,H : OUT std_logic);

END COMPONENT;

Signal A, B, C, D, F, G, H: STD_LOGIC;
```

```
begin

uut:task2

Port Map(A => A, B => B, C => C, D => D, F => F, G => G, H => H);

test: process

begin


A <= '0'; B <= '0'; C <= '0'; D <= '0'; wait for 20 ns;

A <= '0'; B <= '0'; C <= '0'; D <= '1'; wait for 20 ns;

A <= '0'; B <= '0'; C <= '1'; D <= '0'; wait for 20 ns;

A <= '0'; B <= '0'; C <= '1'; D <= '1'; wait for 20 ns;

A <= '0'; B <= '1'; C <= '0'; D <= '0'; wait for 20 ns;

A <= '0'; B <= '1'; C <= '0'; D <= '1'; wait for 20 ns;

A <= '0'; B <= '1'; C <= '1'; D <= '0'; wait for 20 ns;

A <= '0'; B <= '1'; C <= '1'; D <= '1'; wait for 20 ns;

A <= '1'; B <= '0'; C <= '0'; D <= '0'; wait for 20 ns;

A <= '1'; B <= '0'; C <= '0'; D <= '1'; wait for 20 ns;

A <= '1'; B <= '0'; C <= '1'; D <= '0'; wait for 20 ns;

A <= '1'; B <= '0'; C <= '1'; D <= '1'; wait for 20 ns;

A <= '1'; B <= '1'; C <= '0'; D <= '0'; wait for 20 ns;

A <= '1'; B <= '1'; C <= '0'; D <= '1'; wait for 20 ns;

A <= '1'; B <= '1'; C <= '1'; D <= '0'; wait for 20 ns;

A <= '1'; B <= '1'; C <= '1'; D <= '1'; wait for 20 ns;
```

```
end process test;
end Behavioral;
```

Task 2 Results

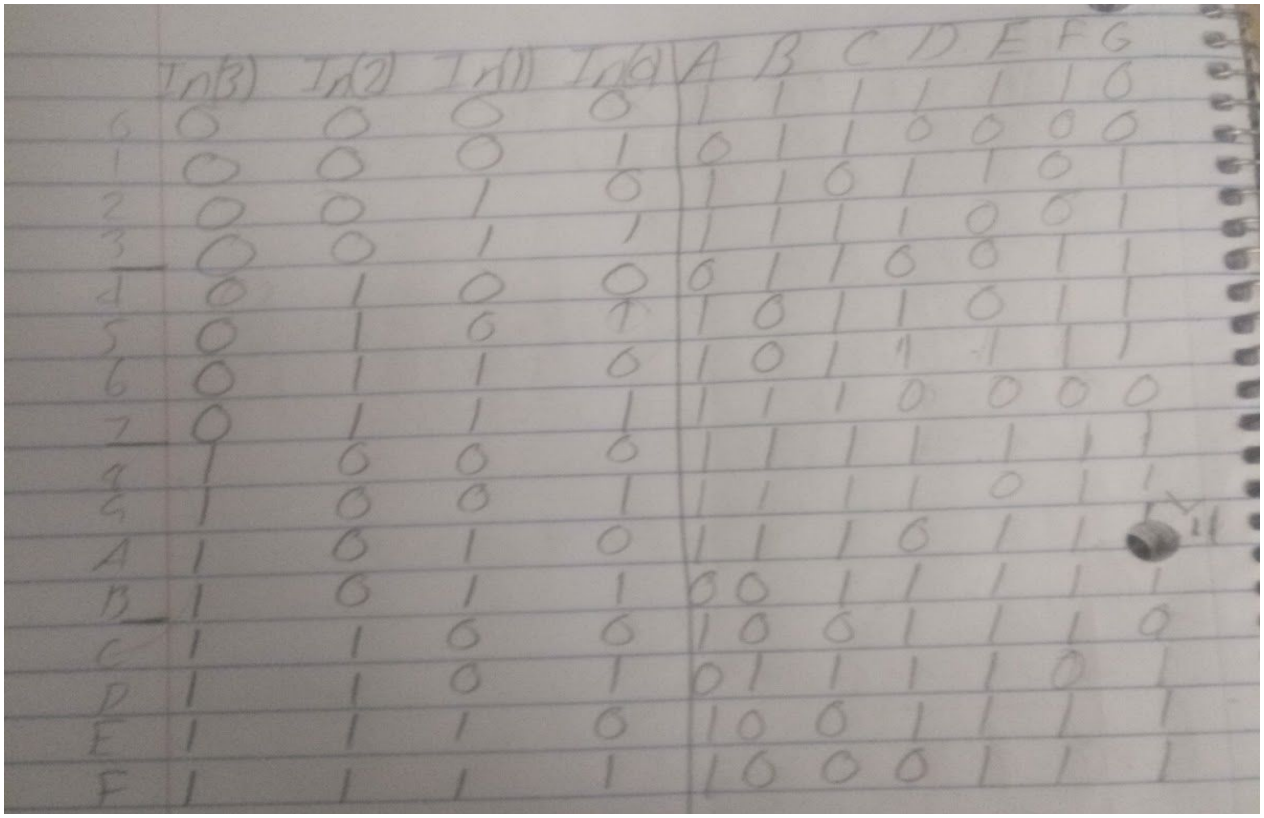


Task 3 Design Process

In our simulation code, we include hex_in(XXXX) to get the proper waveform for each of our numbers as well as a bit of code that involved vectors in order to produce A,B,C,D,E, AND F. Initially we tried to input the values we got in our truth table into the code in order to get the proper wave forms for 0-9 and A-F. However, the way we initially put in our code it did not give

us any results, whenever one error was fixed, two more would take its place, so we decided to state and order what values were expected for A-G to get what we needed.

Task 3 Truth Table



Task 3 Design Source code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;


entity task3 is
Port ( hex_in : in STD_LOGIC_VECTOR (3 downto 0);

A,B,C,D,E,F,G : out STD_LOGIC);

end task3;


architecture Behavioral of task3 is
```

```
begin
A <= '1' when hex_in = "0000" else  '1' when hex_in = "0010" else '1' when hex_in = "0011" else
'1' when hex_in = "0101" else '1' when hex_in = "0110" else
    '1' when hex_in = "0111" else  '1' when hex_in = "1000" else '1' when hex_in = "1001" else '1'
when hex_in = "1010" else '1' when hex_in = "1100" else
    '1' when hex_in = "1110" else  '1' when hex_in = "1111" else '0';

B <= '1' when hex_in = "0000" else '1' when hex_in = "0001" else '1' when hex_in = "0010" else
'1' when hex_in = "0011" else '1' when hex_in = "0100" else
    '1' when hex_in = "0111" else '1' when hex_in = "1000" else '1' when hex_in = "1001" else '1'
when hex_in = "1010" else '1' when hex_in = "1101" else '0';

C <= '1' when hex_in = "0000" else '1' when hex_in = "0001" else '1' when hex_in = "0011" else
'1' when hex_in = "0100" else  '1' when hex_in = "0101" else
    '1' when hex_in = "0110" else '1' when hex_in = "0111" else '1' when hex_in = "1000" else '1'
when hex_in = "1001" else '1' when hex_in = "1010" else
    '1' when hex_in = "1011" else '1' when hex_in = "1101" else '0';

D <= '1' when hex_in = "0000" else '1' when hex_in = "0010" else '1' when hex_in = "0011" else
'1' when hex_in = "0101" else '1' when hex_in = "0110" else
    '1' when hex_in = "1000" else '1' when hex_in = "1001" else '1' when hex_in = "1011" else '1'
when hex_in = "1100" else '1' when hex_in = "1101" else
    '1' when hex_in = "1110" else '0';

E <= '1' when hex_in = "0000" else '1' when hex_in = "0010" else '1' when hex_in = "0110" else
'1' when hex_in = "1000" else '1' when hex_in = "1010" else
    '1' when hex_in = "1011" else '1' when hex_in = "1100" else '1' when hex_in = "1101" else '1'
when hex_in = "1110" else
    '1' when hex_in = "1111" else '0';

F <= '1' when hex_in = "0000" else '1' when hex_in = "0100" else '1' when hex_in = "0101" else
'1' when hex_in = "0110" else '1' when hex_in = "1000" else
    '1' when hex_in = "1001" else '1' when hex_in = "1010" else '1' when hex_in = "1011" else '1'
when hex_in = "1100" else '1' when hex_in = "1110" else
    '1' when hex_in = "1111" else '0';

 G <= '1' when hex_in = "0010" else '1' when hex_in = "0011" else '1' when hex_in = "0100" else
'1' when hex_in = "0101" else '1' when hex_in = "0110" else
     '1' when hex_in = "1000" else '1' when hex_in = "1001" else '1' when hex_in = "1011" else '1'
when hex_in = "1101" else '1' when hex_in = "1110" else
     '1' when hex_in = "1111" else '0';

end Behavioral;
```

Task 3 Simulation Source code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity encoder_tb is
end encoder_tb;

architecture Behavioral of encoder_tb is
component task3
port (
  hex_in : in std_logic_vector(3 downto 0);
  A, B, C, D, E, F, G : out std_logic);

end component;

signal hex_in: std_logic_vector(3 downto 0);
signal A, B, C, D, F, E, G : std_logic ;

signal counter: unsigned(3 downto 0) := "0000";

begin
uut: Task3 port map(hex_in => hex_in, A => A, B => B, C => C, D => D,  E => E, F => F, G =>
G);

hex_in <= std_logic_vector(counter);

test : Process
        Begin
   hex_in <= "0000"; wait For 20ns;
   hex_in <= "0001"; wait For 20ns;
   hex_in <= "0010"; wait For 20ns;
   hex_in <= "0011"; wait For 20ns;
   hex_in <= "0100"; wait For 20ns;
   hex_in <= "0101"; wait For 20ns;
   hex_in <= "0110"; wait For 20ns;
   hex_in <= "0111"; wait For 20ns;
   hex_in <= "1000"; wait For 20ns;
   hex_in <= "1001"; wait For 20ns;
   hex_in <= "1010"; wait For 20ns;
   hex_in <= "1011"; wait For 20ns;
```

```
   hex_in <= "1100"; wait For 20ns;
   hex_in <= "1101"; wait For 20ns;
   hex_in <= "1110"; wait For 20ns;
   hex_in <= "1111"; wait For 20ns;
 End Process test;

  tb: process
    begin
    wait for 20ns;
    counter <= counter + 1;
  end process tb;

end Behavioral;
```

Task 3 Results