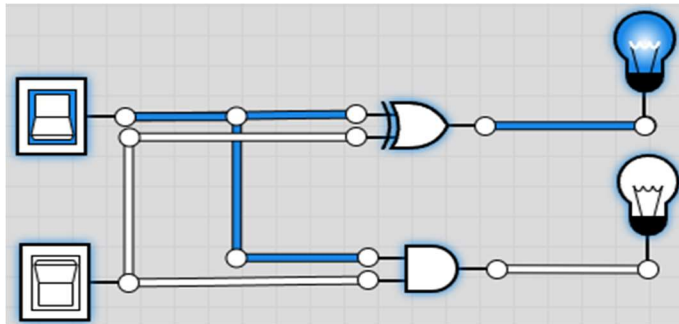


Assignment #3

## Half Adder



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

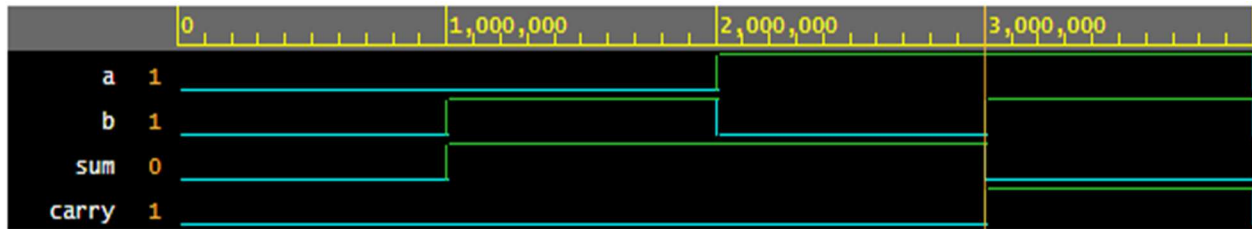
Design:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 -- ENTITY
5 entity HalfAdder is
6 port(
7     a: in std_logic;
8     b: in std_logic;
9     sum: out std_logic;
10    carry: out std_logic);
11 end HalfAdder;
12
13 -- ARCHITECTURE
14 architecture dataflow of HalfAdder is
15 begin
16     sum <= a xor b;
17     carry <= a and b;
18 end dataflow;
```

Testbench:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 -- TESTBENCH ENTITY
5 entity testbench is
6 --empty
7 end testbench;
8
9 architecture tb of testbench is
10 -- DUT COMPONENT
11 component HalfAdder is
12 port(
13     a: in std_logic;
14     b: in std_logic;
15     sum: out std_logic;
16     carry: out std_logic);
17 end component;
18
19 signal aIN, bIN, SUM, CARRY: std_logic;
20
21 begin
22
23 -- CONNECT DUT
24 DUT:HalfAdder port map(aIN, bIN, SUM, CARRY);
25
26 process
27 begin
28     aIN <='0';
29     bIN <='0';
30     wait for 1 ns;
31
32     aIN <='0';
33     bIN <='1';
34     wait for 1 ns;
35
36     aIN <='1';
37     bIN <='0';
38     wait for 1 ns;
39
40     aIN <='1';
41     bIN <='1';
42     wait for 1 ns;
43
44 -- CLEAR INPUTS
45     aIN <='0';
46     bIN <='0';
47     wait;
48 end process;
49 end tb;
```

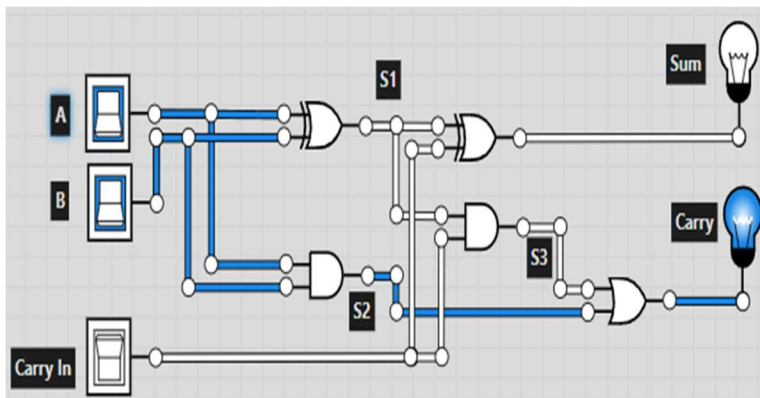
Wave:



Observation:

Wave matches truth table, simple enough easy to design and test.

## Full Adder



A	B	CarryIN	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

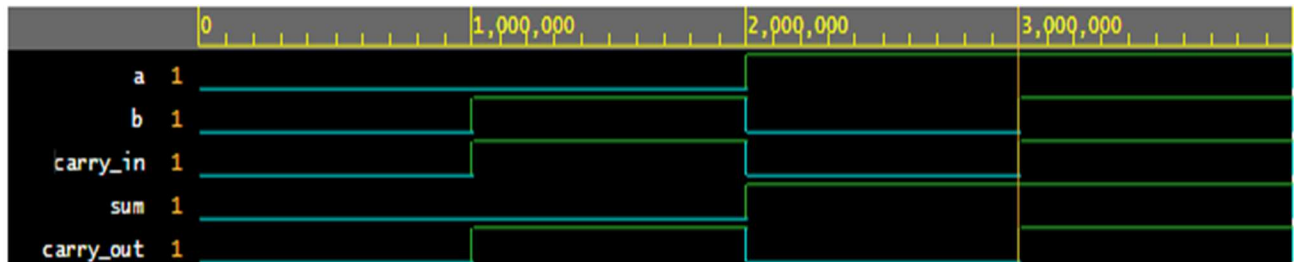
Design:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 -- ENTITY
5 entity fullAdder is
6 port(
7     a, b, CarryIn: in std_logic;
8     sum, CarryOut: out std_logic);
9 end fullAdder;
10
11 -- ARCHITECTURE
12 architecture dataflow of fullAdder is
13 signal S1, S2, S3: std_logic;
14 begin
15     S1 <= a xor b;
16     S2 <= a and b;
17     S3 <= S1 and CarryIn;
18     sum <= S1 xor CarryIn;
19     CarryOut <= S3 or S2;
20 end dataflow;
```

Testbench:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 -- TESTBENCH ENTITY
5 entity testbench is
6 --empty
7 end testbench;
8
9 architecture tb of testbench is
10 -- DUT COMPONENT
11 component fullAdder is
12 port(
13     a, b, CarryIn: in std_logic;
14     sum, CarryOut: out std_logic);
15 end component;
16
17 signal aIN, bIN, CARRY_IN, SUM, CARRY_OUT : std_logic;
18
19 begin
20
21 -- CONNECT DUT
22 DUT: fullAdder port map(aIN, bIN, CARRY_IN, SUM, CARRY_OUT);
23 process
24 begin
25     aIN <='0';
26     bIN <='0';
27     CARRY_IN <='0';
28     wait for 1 ns;
29
30     aIN <='0';
31     bIN <='1';
32     CARRY_IN <='1';
33     wait for 1 ns;
34
35     aIN <='1';
36     bIN <='0';
37     CARRY_IN <='0';
38     wait for 1 ns;
39
40     aIN <='1';
41     bIN <='1';
42     CARRY_IN <='1';
43     wait for 1 ns;
44
45 -- CLEAR INPUTS
46 aIN <='0';
47 bIN <='0';
48 CARRY_IN <='0';
49 wait;
50 end process;
51 end tb;
52
```

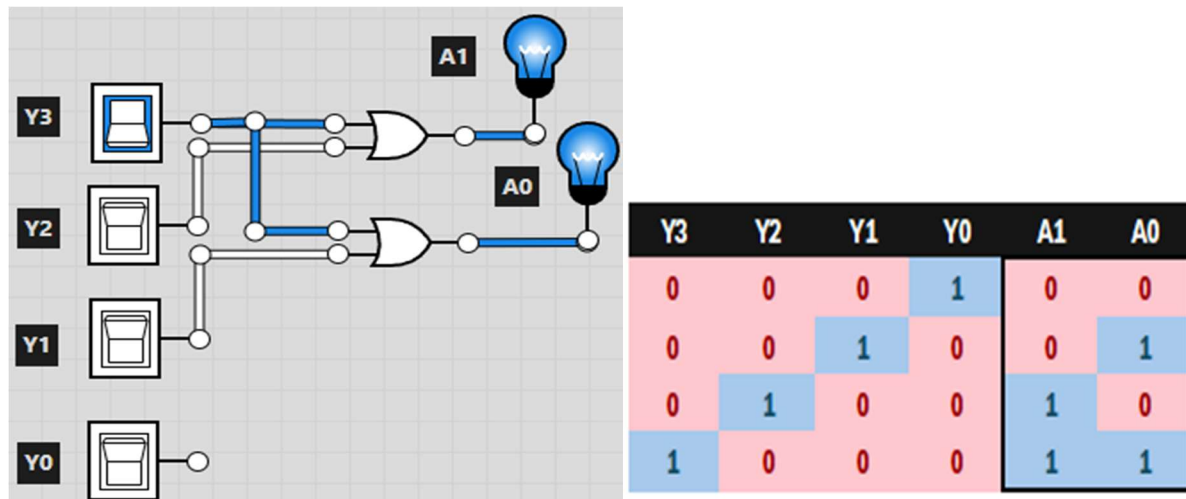
Wave:



Observation:

Wave and truth table match, success. More intricate than the half adder but easy to see how it is built upon.

## 4:2 Encoder



Design:

```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 -- ENTITY
5 entity Encoder is
6 port(
7     y3, y2, y1: in std_logic;
8     a1, a0: out std_logic);
9 end Encoder;
10
11 -- ARCHITECTURE
12 architecture dataflow of Encoder is
13 begin
14     a1 <= y3 or y2;
15     a0 <= y3 or y1;
16 end dataflow;

```

Testbench:

```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 -- TESTBENCH ENTITY
5 entity testbench is
6 --empty
7 end testbench;
8
9 architecture tb of testbench is
10 -- DUT COMPONENT
11 component Encoder is
12 port(
13     y3, y2, y1: in std_logic;
14     a1, a0: out std_logic);
15 end component;
16
17 signal Y3_IN, Y2_IN, Y1_IN, A1_OUT, A0_OUT : std_logic;
18
19 begin
20 -- CONNECT DUT
21 DUT: Encoder port map(Y3_IN, Y2_IN, Y1_IN, A1_OUT, A0_OUT);
22
23 process
24 begin
25
26     Y3_IN <='0';
27     Y2_IN <='0';
28     Y1_IN <='0';
29     wait for 1 ns;
30
31     Y3_IN <='0';
32     Y2_IN <='0';
33     Y1_IN <='1';
34     wait for 1 ns;
35
36     Y3_IN <='0';
37     Y2_IN <='1';
38     Y1_IN <='0';
39     wait for 1 ns;
40
41     Y3_IN <='1';
42     Y2_IN <='0';
43     Y1_IN <='0';
44     wait for 1 ns;
45
46 -- CLEAR INPUTS
47 Y3_IN <='0';
48 Y2_IN <='0';
49 Y1_IN <='0';
50 wait;
51 end process;
52 end tb;

```

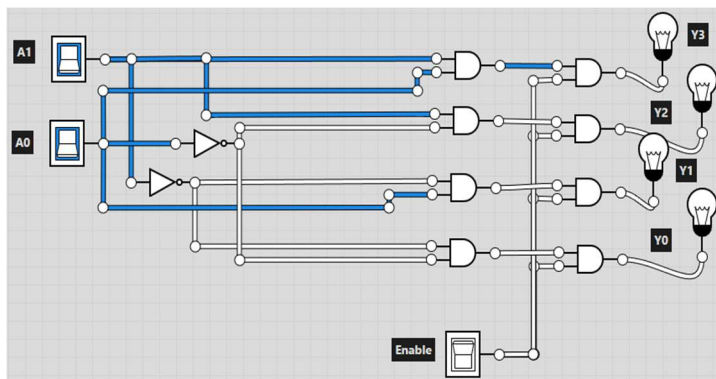
Wave:



Observations:

Truth table and wave match. Simple enough to implement, however I don't understand the point of leaving the y0 input out of the circuit isn't the data from that line just useless then?

## 2:4 Decoder



E	A1	A0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Design:

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 -- ENTITY
5 entity Decoder is
6 port(
7     a1, a0, e: in std_logic;
8     y3, y2, y1, y0: out std_logic);
9 end Decoder;
10
11 -- ARCHITECTURE
12 architecture dataflow of Decoder is
13 begin
14     y3 <= a1 and a0 and e;
15     y2 <= a1 and not a0 and e;
16     y1 <= not a1 and a0 and e;
17     y0 <= not a1 and not a0 and e;
18 end dataflow;
```

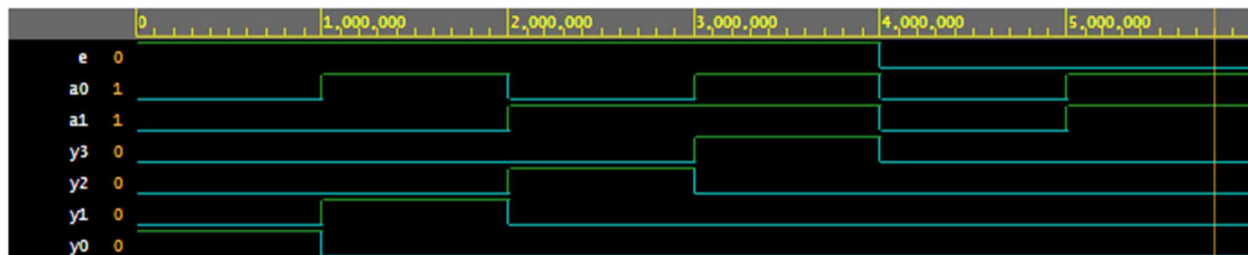
Testbench:

```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 -- TESTBENCH ENTITY
5 entity testbench is
6 --empty
7 end testbench;
8
9 architecture tb of testbench is
10 -- DUT COMPONENT
11 component Decoder is
12 port(
13     a1, a0, e: in std_logic;
14     y3, y2, y1, y0: out std_logic);
15 end component;
16
17 signal A1_IN, AO_IN, ENABLE, Y3_OUT, Y2_OUT, Y1_OUT, Y0_OUT : std_logic;
18
19 begin
20
21 -- CONNECT DUT
22 DUT: Decoder port map(A1_IN, AO_IN, ENABLE, Y3_OUT, Y2_OUT, Y1_OUT, Y0_OUT);
23
24 process
25 begin
26
27     A1_IN <='0';
28     AO_IN <='0';
29     ENABLE <='1';
30     wait for 1 ns;
31
32     A1_IN <='0';
33     AO_IN <='1';
34     ENABLE <='1';
35     wait for 1 ns;
36
37     A1_IN <='1';
38     AO_IN <='0';
39     ENABLE <='1';
40     wait for 1 ns;
41
42     A1_IN <='1';
43     AO_IN <='1';
44     ENABLE <='1';
45     wait for 1 ns;
46
47     A1_IN <='0';
48     AO_IN <='0';
49     ENABLE <='0';
50     wait for 1 ns;
51
52     A1_IN <='1';
53     AO_IN <='1';
54     ENABLE <='0';
55     wait for 1 ns;
56
57     -- CLEAR INPUTS
58     A1_IN <='0';
59     AO_IN <='0';
60     ENABLE <='0';
61     wait;
62 end process;
63 end tb;

```

Wave:



Observation:

Wave and truth table match. This one is far more intuitive to me than the Encoder, very easy to put together and easy to understand the concept and function.