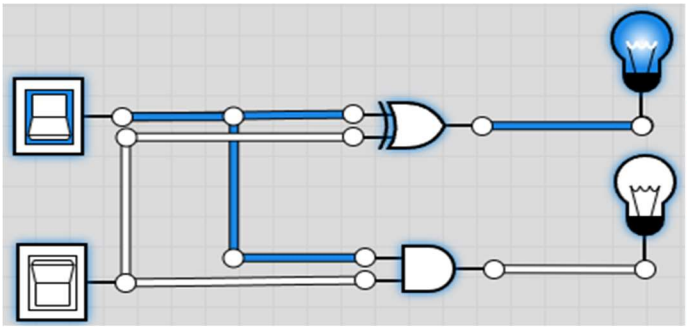
Dylan Liesenfelt
14 June 2024
CDA3203

Assignment #3

Half Adder



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

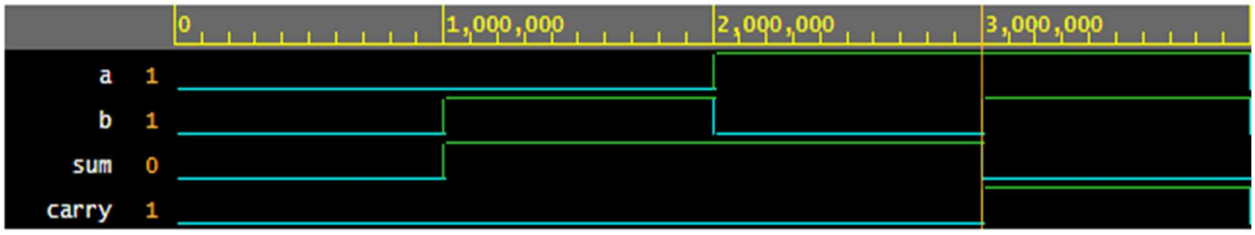
Design:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- ENTITY
entity HalfAdder is
port(
    a: in std_logic;
    b: in std_logic;
    sum: out std_logic;
    carry: out std_logic);
end HalfAdder;

-- ARCHITECTURE
architecture dataflow of HalfAdder is
begin
    sum <= a xor b;
    carry <= a and b;
end dataflow;
```

Testbench:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- TESTBENCH ENTITY
entity testbench is
--empty
end testbench;

architecture tb of testbench is
-- DUT COMPONENT
component HalfAdder is
port(
    a: in std_logic;
    b: in std_logic;
    sum: out std_logic;
    carry: out std_logic);
end component;

signal aIN, bIN, SUM, CARRY: std_logic;

begin

-- CONNECT DUT
DUT:HalfAdder port map(aIN, bIN, SUM, CARRY);

    process
    begin
        aIN <='0';
        bIN <='0';
        wait for 1 ns;

        aIN <='0';
        bIN <='1';
        wait for 1 ns;

        aIN <='1';
        bIN <='0';
        wait for 1 ns;

        aIN <='1';
        bIN <='1';
        wait for 1 ns;

        -- CLEAR INPUTS
        aIN <='0';
        bIN <='0';
        wait;
    end process;
end tb;
```

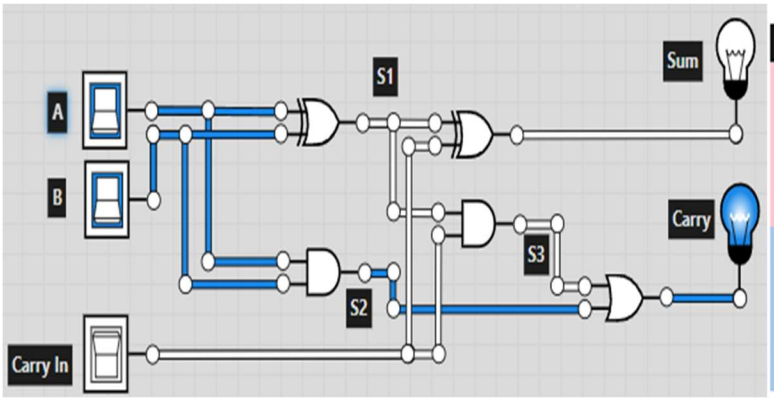
Wave:



Observation:

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

Full Adder



A	B	Carry IN	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

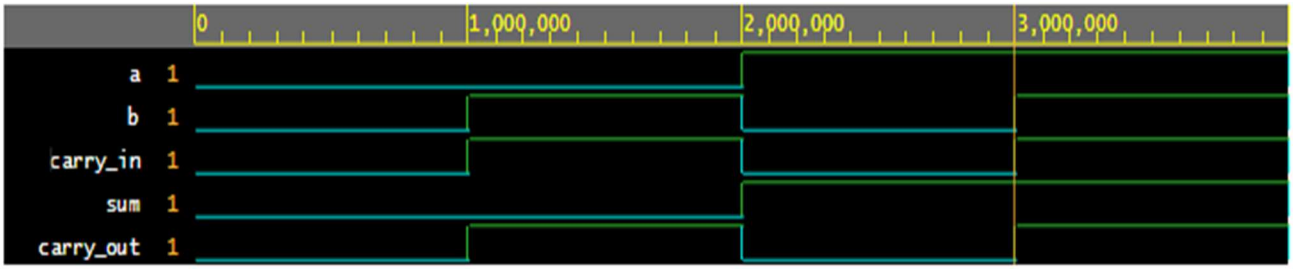
Design:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- ENTITY
entity fullAdder is
port(
    a, b, CarryIn: in std_logic;
    sum, CarryOut: out std_logic);
end fullAdder;

-- ARCHITECTURE
architecture dataflow of fullAdder is
signal S1, S2, S3: std_logic;
begin
    S1 <= a xor b;
    S2 <= a and b;
    S3 <= S1 and CarryIn;
    sum <= S1 xor CarryIn;
    CarryOut <= S3 or S2;
end dataflow;
```

Testbench:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- TESTBENCH ENTITY
entity testbench is
--empty
end testbench;

architecture tb of testbench is
-- DUT COMPONENT
component fullAdder is
port(
    a, b, CarryIn: in std_logic;
    sum, CarryOut: out std_logic);
end component;

signal aIN, bIN, CARRY_IN, SUM, CARRY_OUT : std_logic;

begin

-- CONNECT DUT
DUT: fullAdder port map(aIN, bIN, CARRY_IN, SUM, CARRY_OUT);
process
begin
    aIN <='0';
    bIN <='0';
    CARRY_IN <='0';
    wait for 1 ns;

    aIN <='0';
    bIN <='1';
    CARRY_IN <='1';
    wait for 1 ns;

    aIN <='1';
    bIN <='0';
    CARRY_IN <='0';
    wait for 1 ns;

    aIN <='1';
    bIN <='1';
    CARRY_IN <='1';
    wait for 1 ns;

    -- CLEAR INPUTS
    aIN <='0';
    bIN <='0';
    CARRY_IN <='0';
    wait;
end process;
end tb;
```

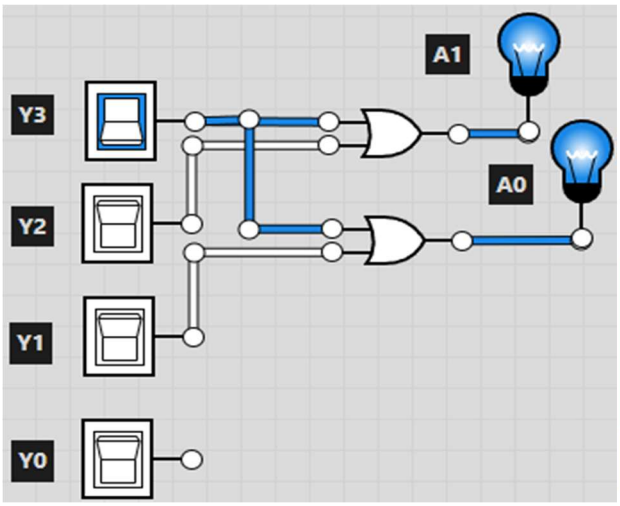
Wave:



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



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

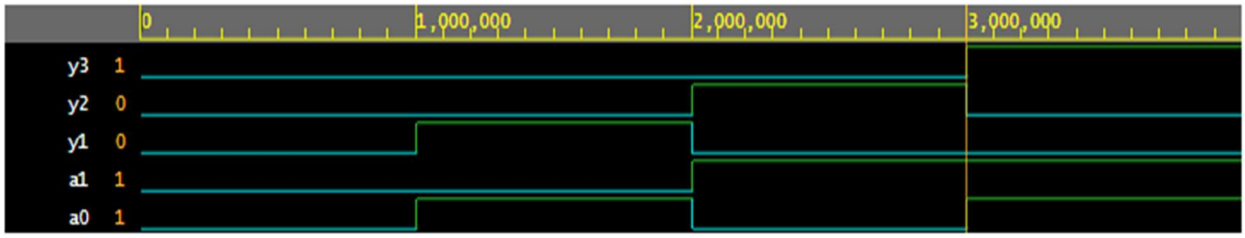
Design:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- ENTITY
entity Encoder is
port(
    y3, y2, y1: in std_logic;
    a1, a0: out std_logic);
end Encoder;

-- ARCHITECTURE
architecture dataflow of Encoder is
begin
    a1 <= y3 or y2;
    a0 <= y3 or y1;
end dataflow;
```

Testbench:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- TESTBENCH ENTITY
entity testbench is
--empty
end testbench;

architecture tb of testbench is
-- DUT COMPONENT
component Encoder is
port(
    y3, y2, y1: in std_logic;
    a1, a0: out std_logic);
end component;

signal Y3_IN, Y2_IN, Y1_IN, A1_OUT, A0_OUT : std_logic;

begin

-- CONNECT DUT
DUT: Encoder port map(Y3_IN, Y2_IN, Y1_IN, A1_OUT, A0_OUT);

    process
    begin
        Y3_IN <='0';
        Y2_IN <='0';
        Y1_IN <='0';
        wait for 1 ns;

        Y3_IN <='0';
        Y2_IN <='0';
        Y1_IN <='1';
        wait for 1 ns;

        Y3_IN <='0';
        Y2_IN <='1';
        Y1_IN <='0';
        wait for 1 ns;

        Y3_IN <='1';
        Y2_IN <='0';
        Y1_IN<='0';
        wait for 1 ns;

        -- CLEAR INPUTS
        Y3_IN <='0';
        Y2_IN <='0';
        Y1_IN <='0';
        wait;
    end process;
end tb;
```

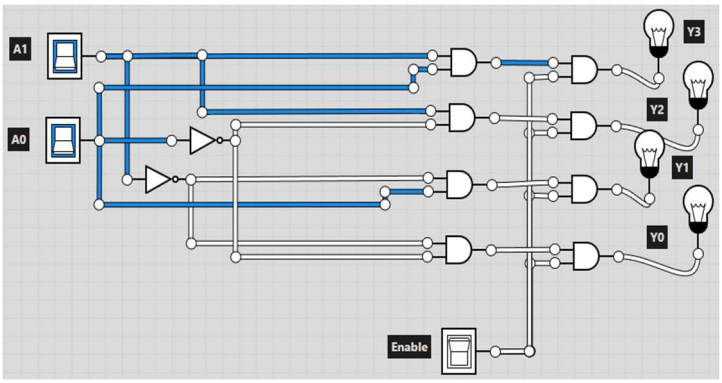
Wave:



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

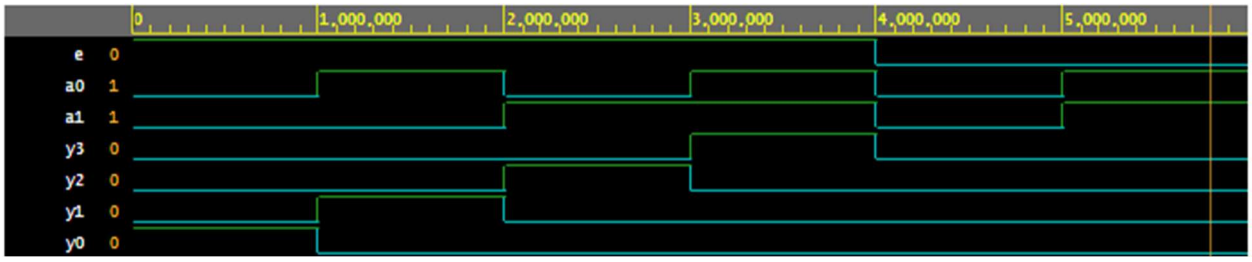
Design:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- ENTITY
entity Decoder is
port(
    a1, a0, e: in std_logic;
    y3, y2, y1, y0: out std_logic);
end Decoder;

-- ARCHITECTURE
architecture dataflow of Decoder is
begin
    y3 <= a1 and a0 and e;
    y2 <= a1 and not a0 and e;
    y1 <= not a1 and a0 and e;
    y0 <= not a1 and not a0 and e;
end dataflow;
```

Testbench:

```
library IEEE;
use IEEE.std_logic_1164.all;

-- TESTBENCH ENTITY
entity testbench is
--empty
end testbench;

architecture tb of testbench is
-- DUT COMPONENT
component Decoder is
port(
    a1, a0, e: in std_logic;
    y3, y2, y1, y0: out std_logic);
end component;

signal A1_IN, A0_IN, ENABLE, Y3_OUT, Y2_OUT, Y1_OUT, Y0_OUT : std_logic;

begin

-- CONNECT DUT
DUT: Decoder port map(A1_IN, A0_IN, ENABLE, Y3_OUT, Y2_OUT, Y1_OUT, Y0_OUT);

    process
    begin

        A1_IN <='0';
        A0_IN <='0';
        ENABLE <='1';
        wait for 1 ns;

        A1_IN <='0';
        A0_IN <='1';
        ENABLE <='1';
        wait for 1 ns;

        A1_IN <='1';
        A0_IN <='0';
        ENABLE <='1';
        wait for 1 ns;

        A1_IN <='1';
        A0_IN <='1';
        ENABLE <='1';
        wait for 1 ns;

        A1_IN <='0';
        A0_IN <='0';
        ENABLE <='0';
        wait for 1 ns;

        A1_IN <='1';
        A0_IN <='1';
        ENABLE <='0';
        wait for 1 ns;

        -- CLEAR INPUTS
        A1_IN <='0';
        A0_IN <='0';
        ENABLE <='0';
        wait;
    end process;
end tb;
```

Wave:



Observation:

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