SCHOOL OF INFORMATION, COMPUTER AND COMMUNICATION TECHNOLOGY
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY

LAB REPORT

EES 370 DIGITAL CIRCUIT LABORATORY

Lab 04 Introduction to VHDL in LogicWorks

By

Mr. Makhapoom Euaumpon **ID: 6222780833**

Mr. Paphana Yiwsiw **ID: 6222780379**

Group No. 12 Section 2

Date: 8 Feb 2021, Time: 13:00-16:00

Objectives

1. To know how to create half adder and full adder by using VHDL code and implement the circuit in LogicWorks.
2. To know how to create 4-bit Binary circuit by using half adder and full adder in LogicWorks.
3. To know how to create 3-Bit BCD to 7-Segment Display Encoder by using VHDL and implement the circuit in LogicWorks.

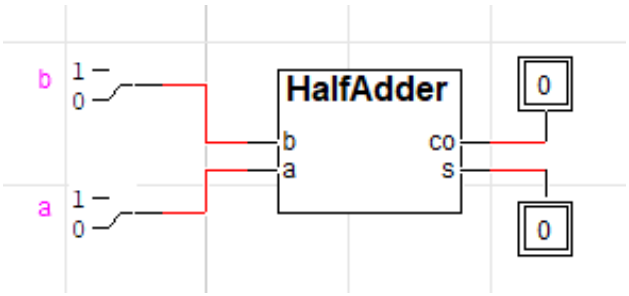
Lab Result

Part A-1: Implement a half adder circuit.

LogicWorks 5 - [HalfAdder.dwv]

File Edit View VHDL Window Help

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


entity HalfAdder is

 port(
        a    : in    std_logic;
        b    : in    std_logic;
        s    : out   std_logic;
        co   : out   std_logic
    );

end HalfAdder;


architecture arch1 of HalfAdder is

begin

  -- Your VHDL code defining the model goes here
  s <= a xor b;
  co <= a and b;

end arch1;
```

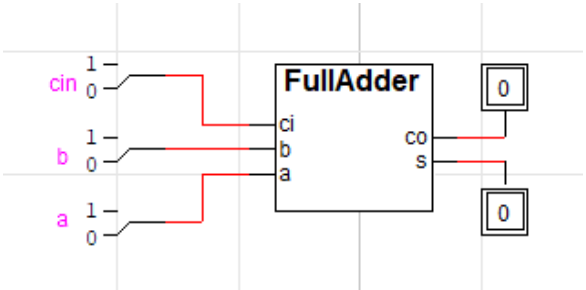


Part A-2: Implement a full adder circuit.

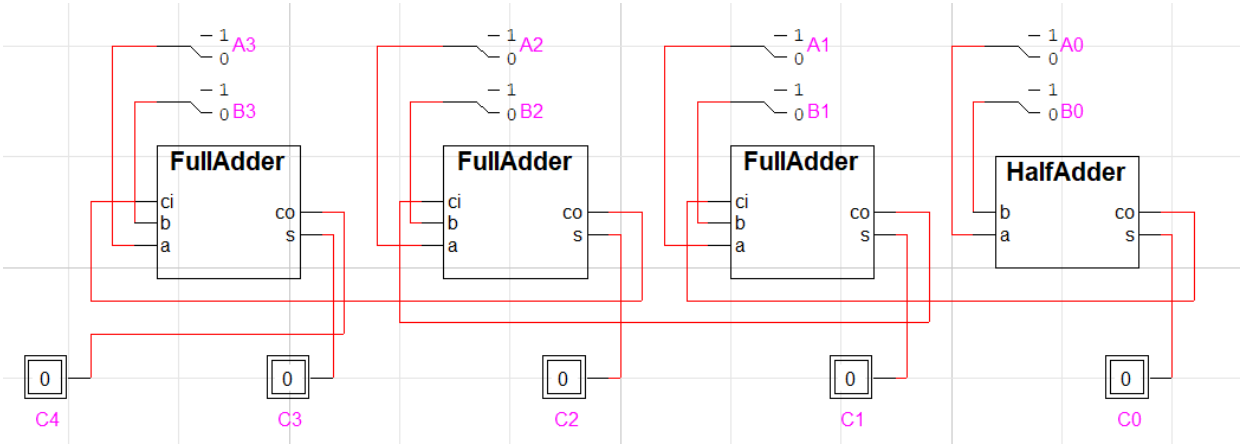
LogicWorks 5 - [FullAdder.dwv]

File Edit View VHDL Window Help

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


entity FullAdder is

 port(
        a    : in    std_logic;
        b    : in    std_logic;
        ci   : in    std_logic;
        s    : out   std_logic;
        co   : out   std_logic
    );

end FullAdder;


architecture arch1 of FullAdder is

begin

  -- Your VHDL code defining the model goes here
  s <= (a xor b) xor ci;
  co <= (a and b) or (b and ci) or (a and ci);

end arch1;
```



Part A-3: Develop 4-bit Binary Adder.

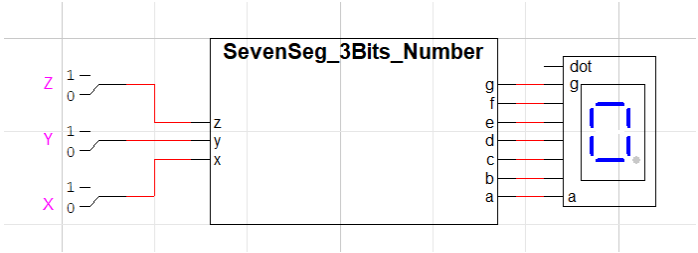


Part A-4: Develop 3-Bit Encoder for 7-Segment Display 3-Bit Number (0-7).

LogicWorks 5 - [SevenSeg_3Bits_Number.dwv]

File Edit View VHDL Window Help

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

entity SevenSeg_3Bits_N is

 port(
        x   : in    std_logic;
        y   : in    std_logic;
        z   : in    std_logic;
        a   : out   std_logic;
        b   : out   std_logic;
        c   : out   std_logic;
        d   : out   std_logic;
        e   : out   std_logic;
        f   : out   std_logic;
        g   : out   std_logic
    );

end SevenSeg_3Bits_N;


architecture arch1 of SevenSeg_3Bits_N is

begin

  -- Your VHDL code defining the model goes here
  a <= (x xnor z) or y;
  b <= (x xnor y) or (not z);
  c <= x or (not y) or z;
  d <= ((not x) and (not z)) or (y and (not z))
       or ((not x) and y) or (x and (not y) and z);
  e <= ((not x) and (not z)) or ((not x) and y);
  f <= ((not x) and (not y)) or ((not y) and z)
       or ((not x) and z);
  g <= ((not x) and z) or (y xor z);

end arch1;
```



Discussion

From part A-1, we created the half adder model by using VHDL. First, we have to design the circuit of half adder. Half adder will receive 2 inputs: A and B, as operands in addition and generate 2 outputs: S as sum and C_O as carry out. Then, we design truth table and derive the Boolean expression of both output of half adder as shown:

A	B	S	C_O
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 1: Truth table of half adder

The reason of last combination (A = 1 and B = 1) to give outputs as S = 0 and C_O = 1 is because 1+1 = 2 but 2 in binary is represented as 10 which means it produce the carry of 1 to be send over to the next bit. Thus, from the truth table, the Boolean expression of half adder are as follows: $S = \bar{A}B + A\bar{B} = A \oplus B$ and $C_O = AB$

After that, we can go to the model wizard in LogicWorks to create this half adder. We select type as VHDL and name the model *HalfAdder*. Then, define the input and output of this model as we designed, check the pin location, save the model to the symbol library, and drag the symbol from the part panel to the circuit editor to finish the model setup. To edit VHDL code, double clicked the *HalfAdder* model on the circuit editor then the code editor will open up. VHDL code consist of 2 parts: entity and architecture. The entity part defines the pin of the model such as input and output. This part we already finished setup from the model wizard menu. Next, the architecture part. This part defines the behavior of the model such as what is the result of this output corresponding from the input in the Boolean expression. We can type the VHDL code into the architecture part under the comment "-- *Your VHDL code defining the model goes here*". We can use the Boolean operator (and, or, not etc.) by typing it down. The output will be calculated from left to right. Noted that the VHDL code is case-insensitive which means A and a is basically the same and ending each command or line with semicolon. The VHDL code of the half adder circuit are `s <= a xor b; co <= a and b;` (also shown in the lab result part).

As same as other programming, before running, you need to compile the code first to make sure that anything is working properly in correct syntax and no-error. To compile the code, press Ctrl+Q or click VHDL->Compile. If the message shown to be no error, you can save the code and click File->Revert to update the model to the latest code. Test circuit by connecting it to the binary switch and observe output with binary probe.

In part A-2, we created full adder by using VHDL and implement the circuit. The main different between half adder and full adder is full adder accept one more input which is carry in (C_i). This is to calculate the addition for every next bit on the left. The least significant bit (LSB) or the rightmost bit will be calculated by using half adder. We apply the same process as part A-1 which is design the truth table and derived the Boolean expression of the output. The truth table and Boolean expression are as follow:

A	B	C_i	S	C_O
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

Table 2: Truth table of full adder

$$
\begin{aligned}
S &= \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + ABC_i + A\bar{B}\bar{C}_i \\
&= (A\bar{B} + \bar{A}B)\bar{C}_i + (\bar{A}\bar{B} + AB)C_i \\
&= (A \oplus B)\bar{C}_i + (A \bar{\oplus} B)C_i \\
&= (A \oplus B) \oplus C_i \\
C_o &= AB + BC_i + AC_i
\end{aligned}
$$

After we derived Boolean expression of both outputs, we can type the VHDL code to the model as follows (also shown in the lab result part):

```
s <= (a xor b) xor ci;
co <= (a and b) or (b and ci) or (a and ci);
```

After we type the code, compile the code, and observe the output to verify what we designed in the first place.

In part A-3, we designed 4-bit binary addition circuit. We are going to use half adder and full adder model that we designed in part A-1 and A-2, respectively. The input of this circuit are 2 4-bit binary numbers A ($A_3A_2A_1A_0$) and B ($B_3B_2B_1B_0$) and will produce output C in 5-bit binary number ($C_4C_3C_2C_1C_0$). The reason of why the output is 5-bit binary number is because when we observe the all-possible combination of input, we saw that some result obtained from the addition cannot be represented in 4-bit binary number. For example, give A and B to be maximum number, 15 or 1111 in binary. The result, C, is equal to 30 and will be represented as 11110 in binary. Thus, the output is needed to be in 5-bit binary number.

To achieve the output, we will implement the circuit by *cascading* the adder module by using half adder for addition operation in the rightmost or the least significant bit and using full adder for addition in the remaining bit until the most significant bit. Each full adder' carry in (C_i) input will be connected by its previous adder' carry out (C_o) output (a.k.a. send the carry out to the next bit). The sum (s) output of each adder will be the result of the output in corresponding bit. For the most significant bit, the carry out that it produced will be the most significant bit in the output C. Thus, we can obtain the circuit as shown in the lab result.

For part A-4, we designed 3-bit BCD number to 7-segment Display encoder by using VHDL. In this case, Input are 3-bit BCD number (zyx) and produce 7 output corresponding to the 7 pins used to connect to 7-segment display. Applying the same process as the previous part (design the truth table and derive the Boolean expression of the output by using K-map), we can obtain the truth table, K-map and Boolean expression as follows:

Z	Y	X	a	b	c	d	e	f	g
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0

Table 3: Truth table of 3-bit BCD to 7-segment encoder

a

zy \ x	0	1
00	1	0
01	1	1
11	1	1
10	0	1

b

zy \ x	0	1
00	1	1
01	1	1
11	0	1
10	1	0

c

zy \ x	0	1
00	1	1
01	0	1
11	1	1
10	1	1

d

zy \ x	0	1
00	1	0
01	1	1
11	1	0
10	0	1

e

zy \ x	0	1
00	1	0
01	1	0
11	1	0
10	0	0

f

zy \ x	0	1
00	1	0
01	0	0
11	1	0
10	1	1

g

zy \ x	0	1
00	0	0
01	1	1
11	1	0
10	1	1

Figure 4: K-map of all output from 3-bit BCD to 7-segment encoder

$$a = \overline{X \oplus Z} + Y$$
$$b = \overline{X \oplus Y} + \overline{Z}$$
$$c = X + \overline{Y} + Z$$
$$d = \overline{X}\,\overline{Z} + Y\overline{Z} + \overline{X}Y + X\overline{Y}\,Z$$
$$e = \overline{X}\,\overline{Z} + \overline{X}Y$$
$$f = \overline{X}\,\overline{Y} + \overline{Y}\,Z + \overline{X}Z$$
$$g = \overline{X}Z + (Y \oplus Z)$$

In another way, we can obtain the Boolean expression from the truth table by using Logisim program that we used in Lab 2. We add the input, output, and desire truth table into that program. Then, the Logisim will produce the Boolean expression that we want. After we obtain Boolean expression in any way, we setup the encoder model by using the same step as we used in part A-1. Then, we can type down the VHDL code of Boolean expression as follows (also shown in the lab result part).

```
a <= (x xnor z) or y;
b <= (x xnor y) or (not z);
c <= x or (not y) or z;
d <= ((not x) and (not z)) or (y and (not z)) or ((not x) and y)
     or (x and (not y) and z);
e <= ((not x) and (not z)) or ((not x) and y);
f <= ((not x) and (not y)) or ((not y) and z) or ((not x) and z);
g <= ((not x) and z) or (y xor z);
```

After that, we can connect the encoder to receive 3 inputs from binary switch and connect the output pin to the corresponding pin of 7-segment display (Of course, with display color of choice). Lastly, we verify the output and check with what we designed.

Conclusion

From this experiment, we studied and know how VHDL works and how to create half adder and full adder by using VHDL in LogicWorks. Also, we know how to use half adder and full adder in order to create and implement in a 4-bit binary adder circuit by cascading the full adder and half adder. Lastly, we know and understand to create 3-bit BCD to 7-segment encoder in LogicWorks by using VHDL and previous lab experiments knowledge in the creation of encoder such as K-Map and obtain the minimize expression from Logisim.