Switching Circuit and Logic Design Laboratory Report-3

by

Group-11

Devendra Palod (*20CS10024*)

Subhajyoti Halder (*20CS10064*)

Anuj Kakde (*20CS30005*)

Deepiha S. (*20CS30015*)

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2021-22

Contents

Logic file url:

`https://github.com/subhu-darkknight72/SCLD_Lab-3_20CS10064.git`

1 Part-1 (BCD to 7-segment display)

1.1 Problem Statement

A seven segment display is pictured.

1. For the BCD number given as $\langle x_3, x_2, x_1, x_0 \rangle$ form the truth tables and the corresponding circuits to light up the LEDs $a..g$ of the 7-segment display to indicate the decimal number properly
2. Test that it works by applying appropriate inputs and checking the outputs – test with bit displays in lieu of the LEDs
3. Label the terminals to reflect their roles
4. Save it as a regular circuit (logic file), reopen and retest
5. Save it as a component (cmp file), reopen and retest – only the logic to drive the segments will be effectively encapsulated; when the module is used bit displays will again have to be connected to the outputs

1.2 Solution Description

7-Segment display contains 7 LED segments arranged in a shape given in figure. Generally, there are 8 input pins in a 7-Segment display. 7 input pins for each of the 7 LEDs and one pin for the common terminal.

In such type of 7-segment display, all the cathodes of the 7 LEDs are connected together to form a common terminal. It should be connected to GND or logic '0' during its operation. To illuminate any LED of the display, you need to supply logic '1' to its corresponding input pin.

Suppose we want to display digit '0', in order to display 0, we need to turn on "a", "b", "c", "d", "e", "f". turn-off the "g". To display the digit 1 we need to turn on the segments b, c. and turn off the LED segments a, d, e, f, and g.

And similarly we illuminate necessary LEDs for other digits.

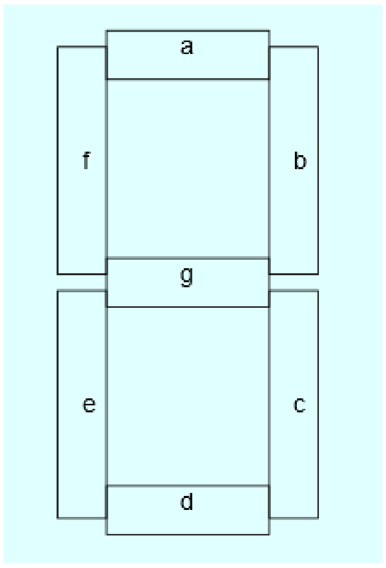


Figure 1: 7 Segment Display

1.3 Logic Expression

The logical expression for each variable are as follows:

$$\mathbf{a} = A + C + BD + \bar{B}\bar{D}$$

$$\mathbf{b} = \bar{B} + \bar{C}\bar{D} + CD$$

$$\mathbf{c} = B + \bar{C} + D$$

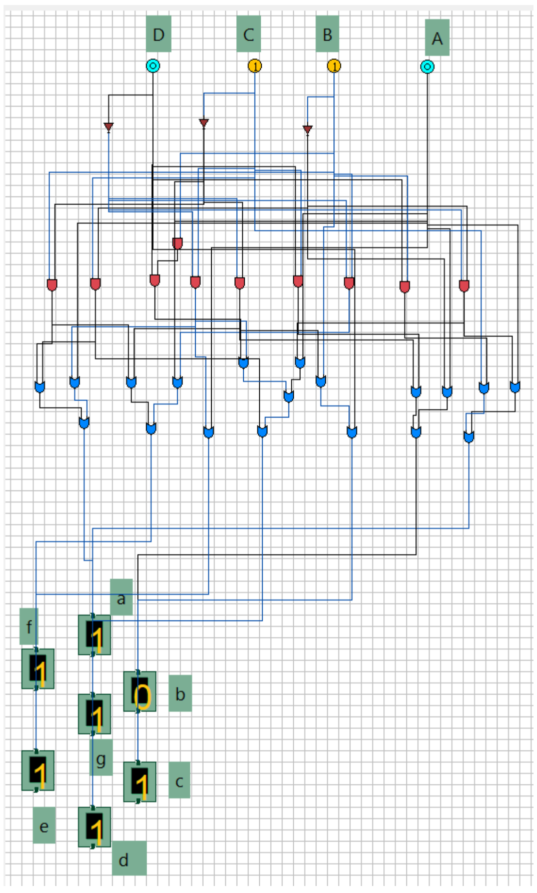
$$\mathbf{d} = A + \bar{B}\bar{D} + \bar{B}C + C\bar{D} + B\bar{C}D$$

$$\mathbf{e} = \bar{B}\bar{D} + C\bar{D}$$

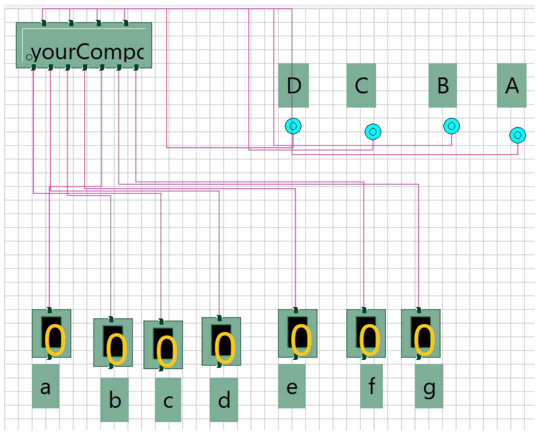
$$\mathbf{f} = A + B\bar{C} + B\bar{D} + \bar{C}\bar{D}$$

$$\mathbf{g} = A + B\bar{C} + \bar{B}C + C\bar{D}$$

1.4 Circuit Diagram



(a) BCD to 7-segment Display Circuit



(b) BCD to 7-segment Display Circuit (with component)

2 Part-2 (3-bit decoder)

2.1 Problem Statement

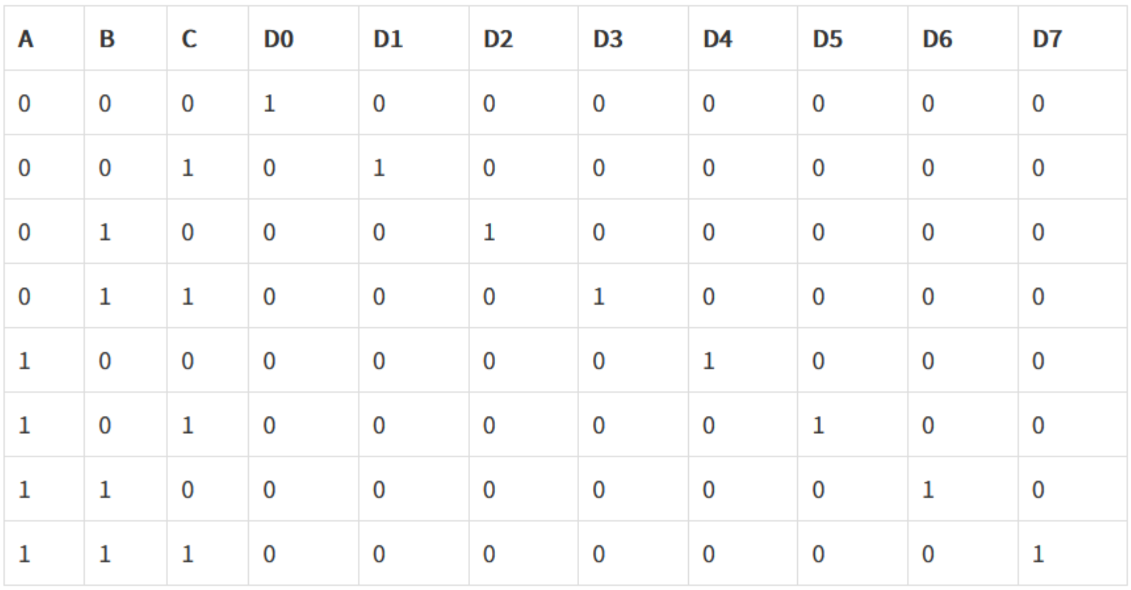
A k-bit decoder has three input lines, an enable line and 2^k output lines indexed 0 to $2^k - 1$; all output lines take the value 0 in the enable line is 0, otherwise if n represents the binary value of the input bits, output line n takes the value 1 while all other output lines take the value 0

1. Design a 3-bit decoder (it can be conveniently designed by connecting a number of 1-bit decoder in a tree structure).
2. Test that the decoders work by applying appropriate inputs and checking the outputs
3. Label the terminals to reflect their roles
4. Save decoders as regular circuits (logic file), reopen and retest
5. Save the decoders as components (cmp file), reopen and retest

2.2 Solution Description

A 3-bit decoder or 3 to 8 decoder has three inputs (A, B, C) and eight outputs (D_0 to D_7). If n represents the binary value of the 3 input bits, output line n takes the value 1 while all other output lines take the value 0. Based on the 3 inputs one of the eight outputs is selected. Using the expressions given below, the circuit of a 3 to 8 decoder can be implemented using three NOT gates and eight 3-input AND gates as shown in the circuit diagram.

2.3 Logic Expression



A	B	C	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Figure 3: Truth Table for 3-bit decoder

The logical expression for each variable are as follows:

$$D_0 = \bar{A}\bar{B}\bar{C}$$
$$D_1 = \bar{A}\bar{B}C$$
$$D_2 = \bar{A}B\bar{C}$$
$$D_3 = \bar{A}BC$$
$$D_4 = A\bar{B}\bar{C}$$
$$D_5 = A\bar{B}C$$
$$D_6 = AB\bar{C}$$
$$D_7 = ABC$$

2.4 Circuit Diagram

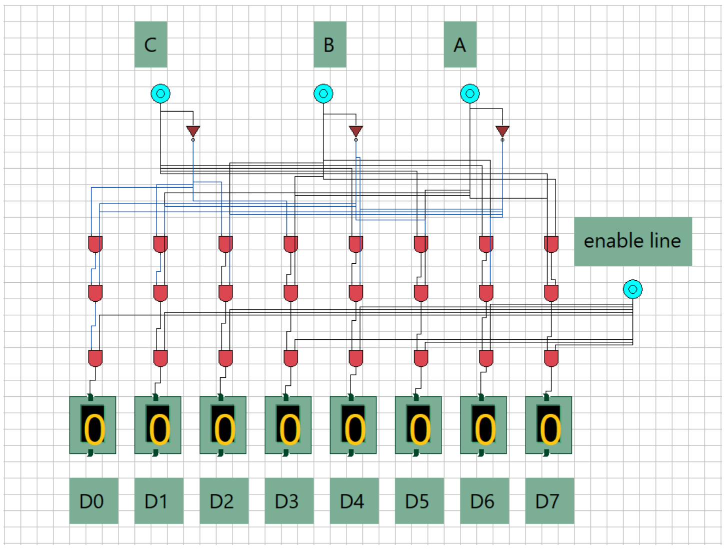


Figure 4: 3-bit Decoder Circuit

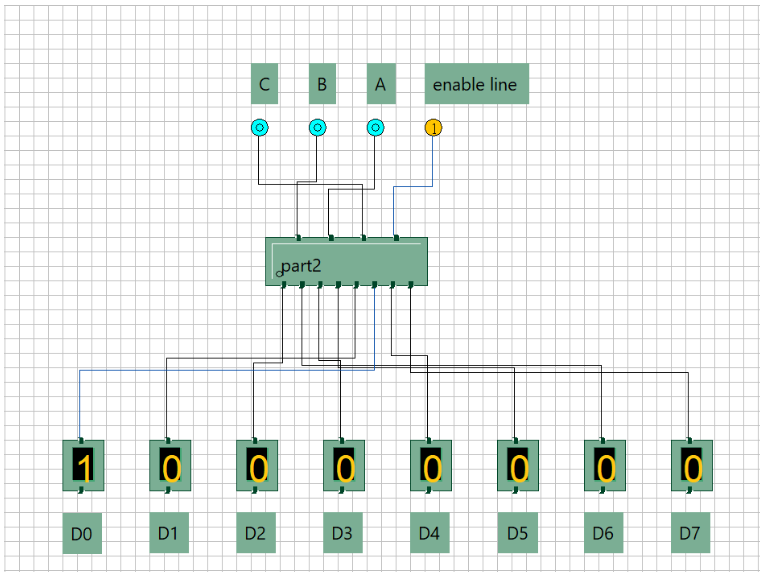


Figure 5: 3-bit Decoder Circuit (with component)

3 Part-3 (3-bit ECC parity generator)

3.1 Problem Statement

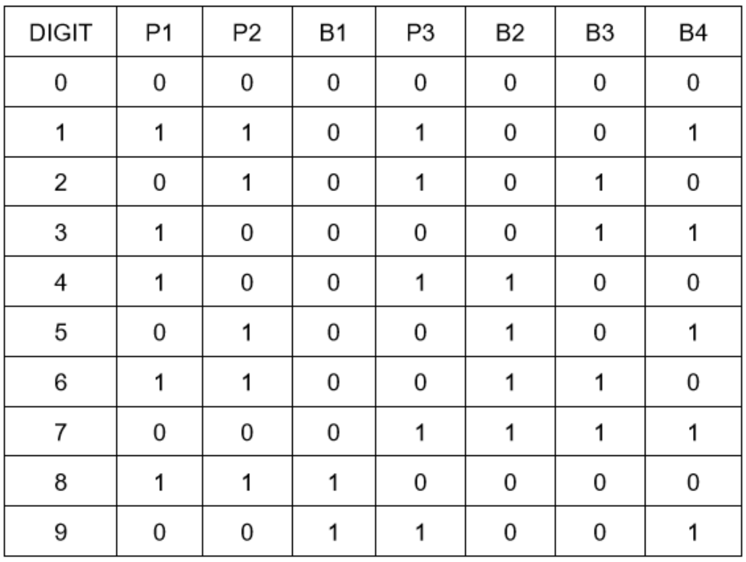
1. For the BCD number given as $\langle x_3, x_2, x_1, x_0 \rangle$ form the truth tables and the corresponding circuits for the three parity bits required for 1-bit Hamming ECC.
2. Test that it works by applying appropriate inputs and checking the outputs.
3. Label the terminals to reflect their roles.
4. Save it as a regular circuit (logic file), reopen and retest.
5. Save it as a component (cmp file), reopen and retest.

3.2 Solution Description

The $(7, 4)$ binary Hamming block encoder accepts blocks of 4-bit of information, adds 3 parity bits to each such block and produces 7-bits wide Hamming coded blocks.

Codeword belonging to $(7, 4)$ Hamming code be represented by $[P_1, P_2, B1_1, P_3, B_2, B_3, B_4]$, where B represents information bits (Binary digit of BCD codes) and P represents parity bits at respective bit positions which can be computed using the logic expressions given below.

3.3 Logical Expression



DIGIT	P1	P2	B1	P3	B2	B3	B4
0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1
2	0	1	0	1	0	1	0
3	1	0	0	0	0	1	1
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	1
6	1	1	0	0	1	1	0
7	0	0	0	1	1	1	1
8	1	1	1	0	0	0	0
9	0	0	1	1	0	0	1

Figure 6: Truth Table for 3-bit ECC parity generator

The logical expression for each variable are as follows:

$$P_1 = B_1 \oplus B_2 \oplus B_4$$
$$P_2 = B_3 \oplus B_4 \oplus B_1$$
$$P_3 = B_3 \oplus B_4 \oplus B_2$$

3.4 Circuit Diagram

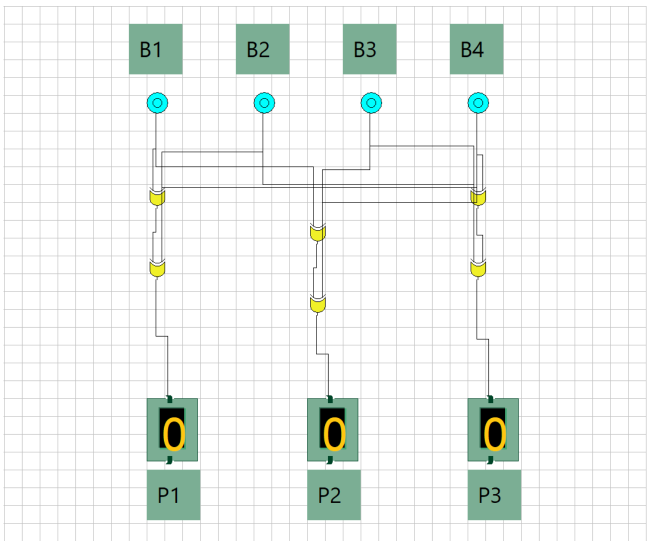


Figure 7: 3-bit ECC parity generator

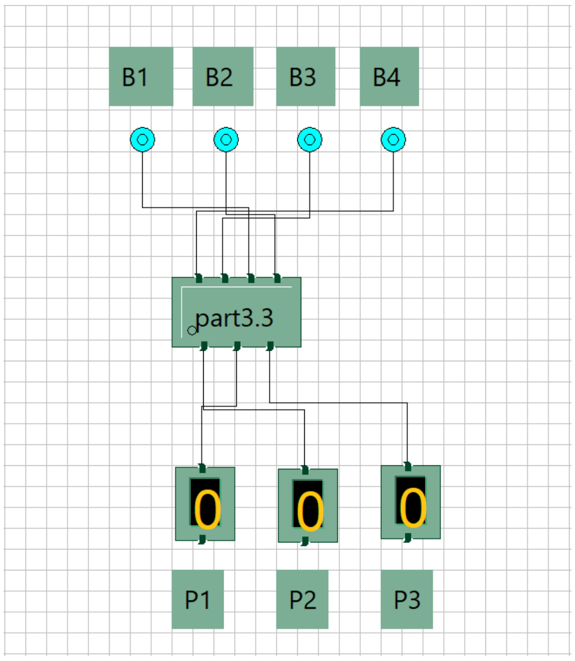


Figure 8: 3-bit ECC parity generator (with component)

4 Part-4 (1-bit ECC)

4.1 Problem Statement

1. FAs indicated in the figure construct the ECC module that takes the 4-bit received data $\langle x_3, x_2, x_1, x_0 \rangle$, 3-bit received parity $\langle p_2, p_1, p_0 \rangle$ and 3-bit recomputed parity $\langle p_2', p_1', p_0' \rangle$ to generate the 4-bit restored data

2. Test that it works by applying appropriate inputs and checking the outputs.

3. Label the terminals to reflect their roles

4. Save it as a regular circuit (logic file), reopen and retest.

5. Save it as a component (cmp file), reopen and retest

4.2 Circuit Diagram

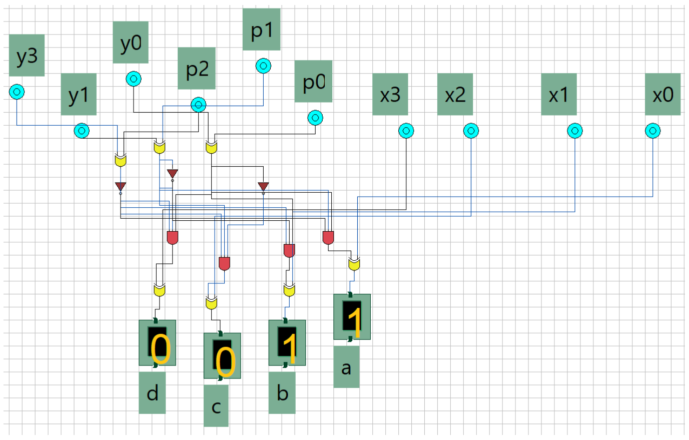


Figure 9: 1-bit ECC Circuit

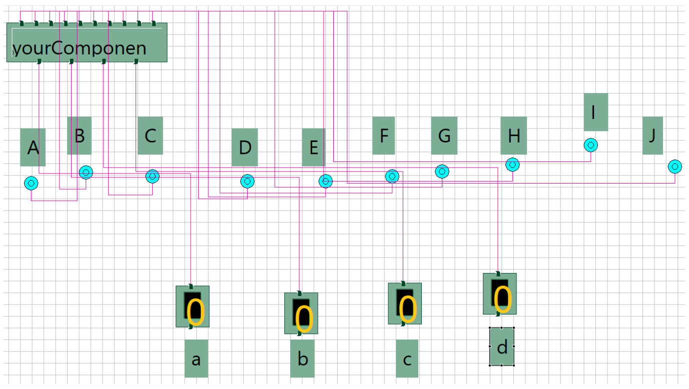


Figure 10: 1-bit ECC Circuit (with component)

5 Part-5 (1-bit ECC test setup)

5.1 Problem Statement

1. Wire up the designed components as indicated in the figure with the provision to introduce a single bit error in any of the seven lines using the EXOR gates in conjunction with the 3-bit decoder, displaying the restored data using the 7-segment display.

2. Test that it works by applying appropriate inputs and checking the outputs.

3. Label the terminals to reflect their roles.

4. Save it as a regular circuit (logic file), reopen and retest.

5. How would you extend this to allow error detection for more than 1-bit (without correction)? You need not realise this in the tool, only report the paper design in your PDF report.

5.2 Solution Description

Using parity logic, a hamming code (as seen in parts 1- 4) can be used to correct just one bit. If we correct more bits like two more bits, then the parity will change, and we'll have to change a correct bit. As a result, changing the code is required to correct a two-bit error. It is possible to accomplish this by adding one more parity bit, which is the XOR of all the parity bits. As a result, when a double error occurs, the XOR parity of all bits is unaffected, but the other parities are.

5.3 Circuit Diagram

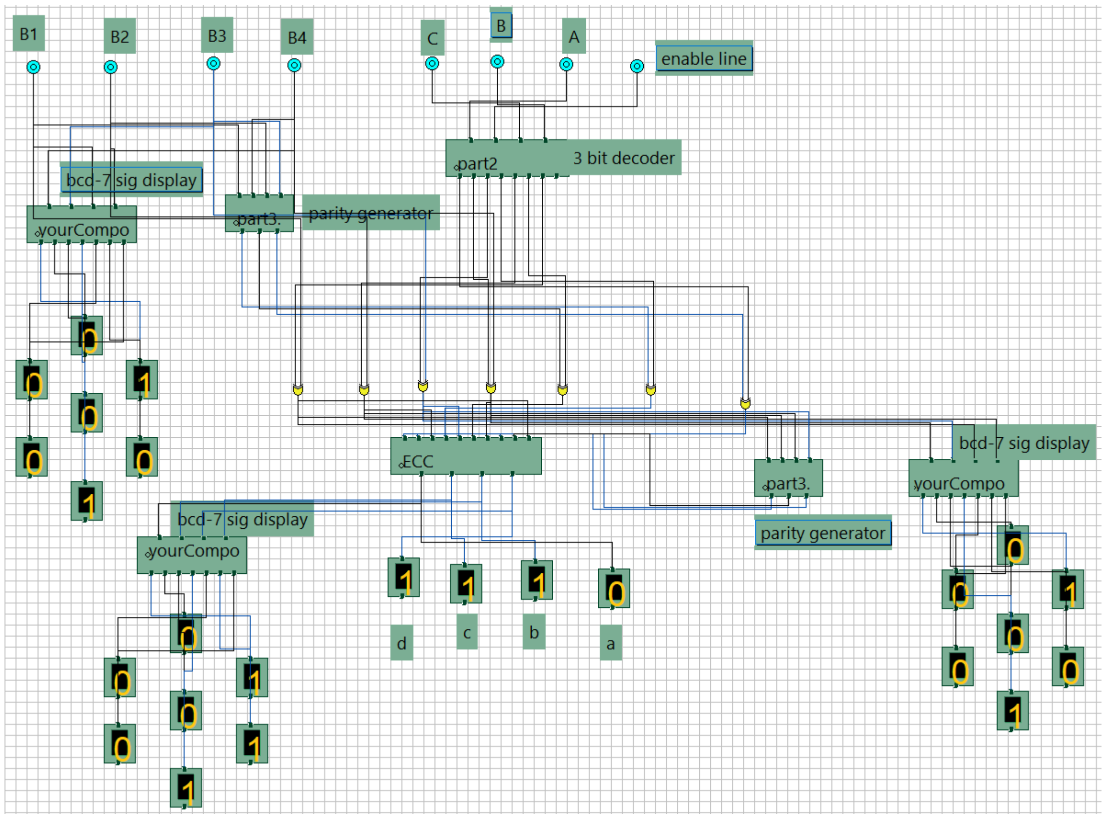


Figure 11: 1-bit ECC Test Setup