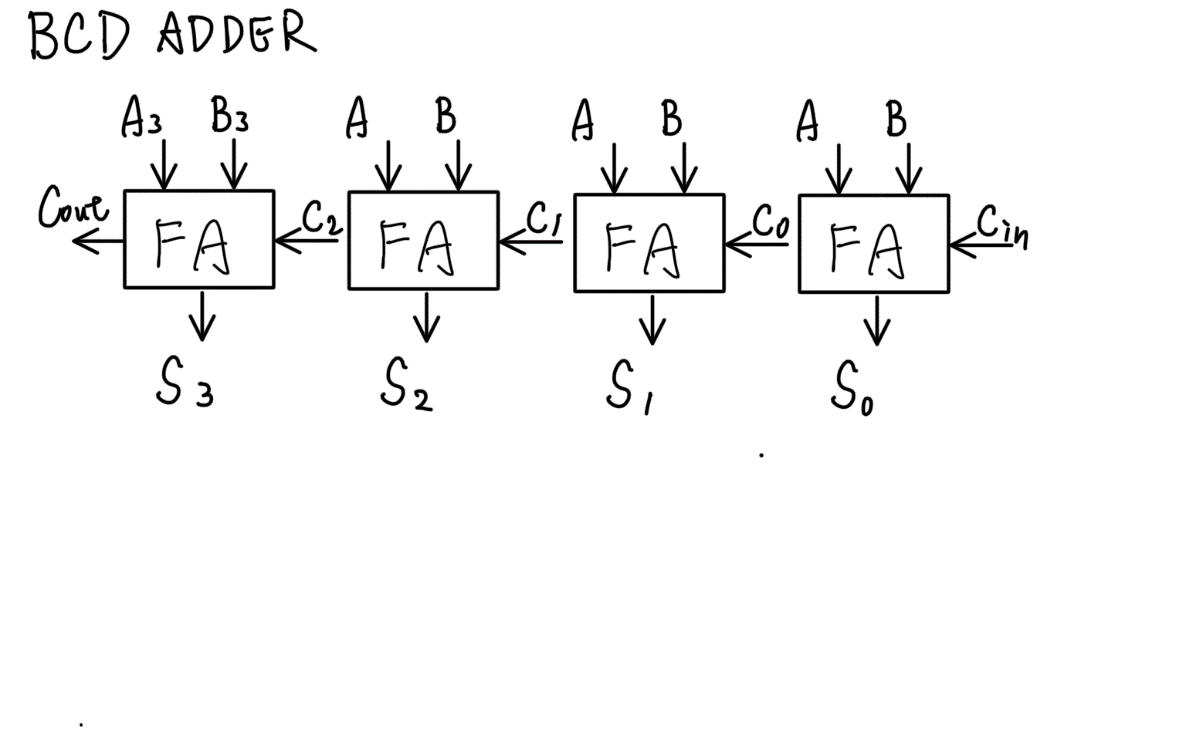
班級：　四電子一乙　　　　　　　 姓名：李家睿　　　　　 學號：B11102112

# ­Binary to Decimal converter

1. Design

A picture containing shape

Description automatically generated



A picture containing diagram

Description automatically generated

1. Schematic entry
2. full adder:
   * 1. Block diagram

Diagram, schematic

Description automatically generated

* + 1. Behavioral

Graphical user interface, application, table

Description automatically generated

* + 1. Timing verification

Graphical user interface, application

Description automatically generated

1. Geq5Plus3
   * 1. Block diagram

Diagram, schematic

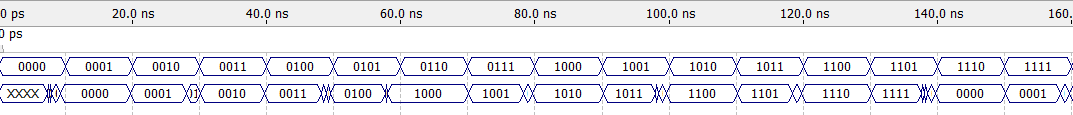
Description automatically generated

* + 1. Behavioral

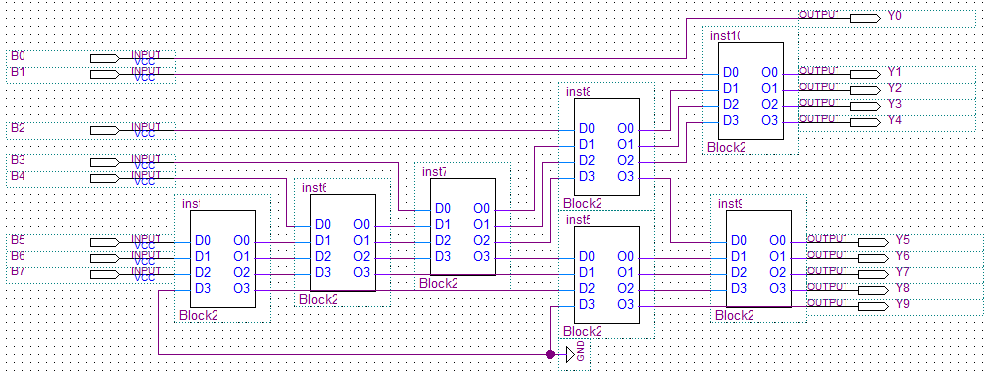
Table

Description automatically generated

* + 1. Timing verification



1. Binary-to-BCD converter :
   * 1. Block diagram



* + 1. Behavioral

Table

Description automatically generated

* + 1. A picture containing table

       Description automatically generatedTiming verification

1. Verilog-HDL-entry
2. Verilog entry

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | **module** binarytobcd(  **input** clk,  **input** [7:0] binary\_value,  **output** reg [3:0] one = 0,  **output** reg [3:0] ten = 0,  **output** reg [3:0] hundred = 0  );  reg [3:0] i=0;  reg [19:0] shift = 0;  reg [3:0] temp\_one =0;  reg [3:0] temp\_ten =0;  reg [3:0] temp\_hundred =0;  reg [7:0] old\_binary\_value = 0;  **always**@(**posedge** clk)  **begin**  **if** (i==0 & old\_binary\_value != binary\_value)  **begin**  shift = 20'd0;  old\_binary\_value = binary\_value;  shift[7:0] = binary\_value;  temp\_hundred = shift[19:16];  temp\_ten = shift[15:12];  temp\_one = shift[11:8];  i = i + 1;  **end**  **if**(i < 9 & i > 0)  **begin**  **if**(temp\_hundred >= 5)  temp\_hundred = temp\_hundred + 3;  **if**(temp\_ten >= 5)  temp\_ten = temp\_ten + 3;  **if**(temp\_one >= 5)  temp\_one = temp\_one + 3;  shift [19:8] = {temp\_hundred , temp\_ten , temp\_one};  shift = shift << 1;  temp\_hundred = shift[19:16];  temp\_ten = shift[15:12];  temp\_one = shift[11:8];  i = i + 1;  **end**  **if**(i == 9)  **begin**  i = 0;  hundred = temp\_hundred;  ten = temp\_ten;  one = temp\_one;  **end**  **end**  **endmodule** |

1. RTL verification

Chart

Description automatically generated

1. Chart

   Description automatically generated with medium confidenceGate-level verification
2. Report

To convert the 8-bit binary input to BCD, we will first need an input clock and three outputs for the BCD digits (hundreds, tens, and ones). We will also create a register called "i" to count the eight cycles in the code, as well as five more registers: a 20-bit shift register to shift the input number, and registers for the hundreds, tens, and ones digits to keep the original value unchanged. Additionally, we will have one more register to keep the old binary input unchanged.

Starting with the code, we will set up three "if" statements. The first "if" statement checks if "i" is equal to 0 and if the old binary value is not equal to the new one. If this is the case, we will initialize the shift register to zero and set the old binary value to the new one. We will then set the shift register's [7:0] to the new binary value and set temporary hundreds to shift register [19:16], temporary tens to shift register [15:12], and temporary ones to shift register [11:8]. Finally, we will increment "i" by one.

The second "if" statement checks if "i" is less than nine and greater than zero. If this is the case, we will check if the hundreds digit is greater than four. If it is, we will add three to the hundred digits, and do the same for the tens and one digits. After adding three to each digit, we will store the result in the shift register.

Finally, the third "if" statement checks if "i" is equal to nine. If this is the case, we will set "i" back to zero, and assign temporary hundreds, tens, and ones to the corresponding registers.

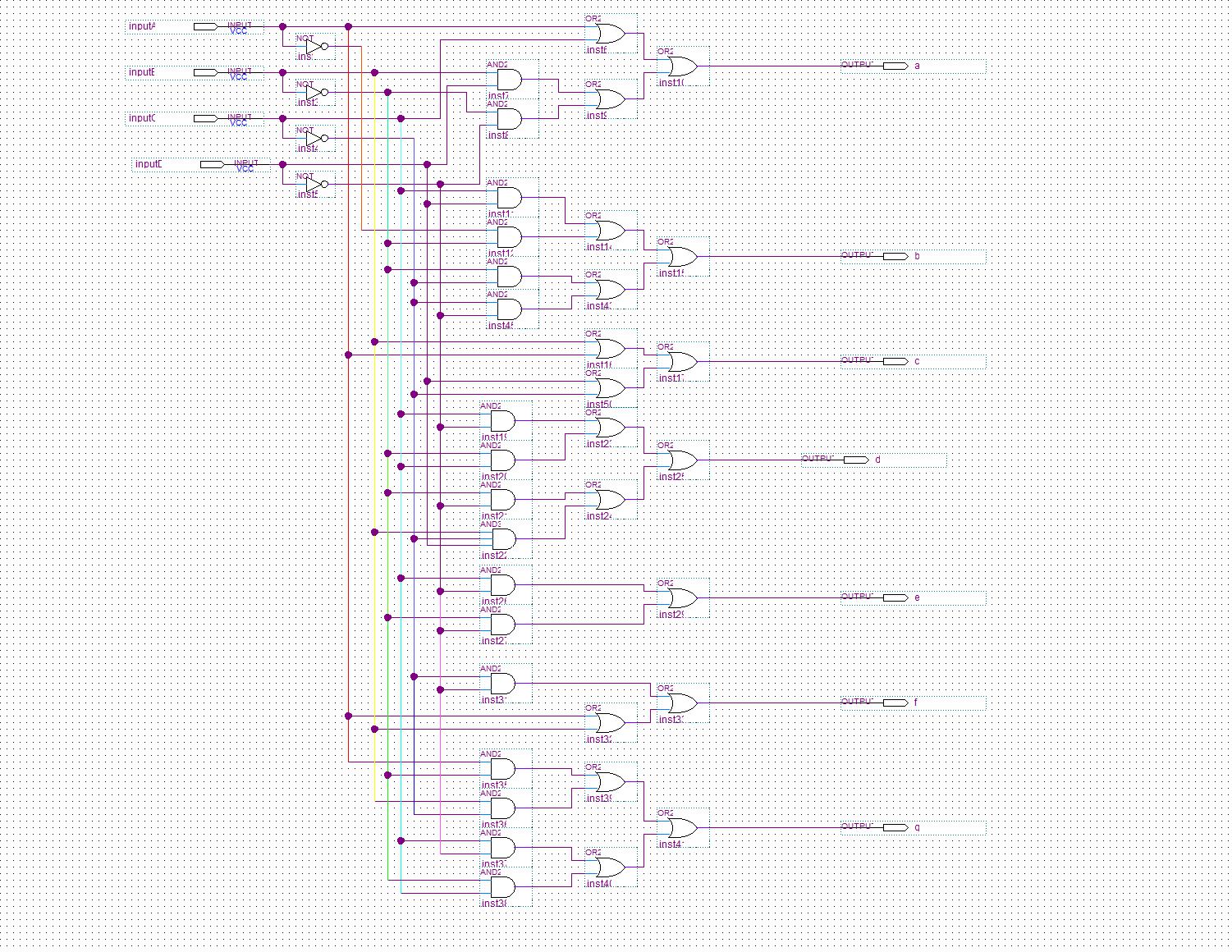
# BCD to seven segment display code converter

1. Design

A picture containing text, receipt

Description automatically generated

1. Schematic entry
2. Block diagram



1. Behavioral

Diagram

Description automatically generated

1. Diagram

   Description automatically generatedTiming Verification

1. Verilog-HDL entry
2. Verilog entry

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | **module** segement7decoderverilog(  **input** [3:0] bcd\_in,  **output** reg [7:0] seg\_out  );  **always** @ (bcd\_in)  **begin**  **case** (bcd\_in)  4'b0000: seg\_out = 8'b0000001;*//0*  4'b0001: seg\_out = 8'b1001111;*//1*  4'b0010: seg\_out = 8'b0010010;*//2*  4'b0011: seg\_out = 8'b0000110;*//3*  4'b0100: seg\_out = 8'b1001100;*//4*  4'b0101: seg\_out = 8'b0100100;*//5*  4'b0110: seg\_out = 8'b0100000;*//6*  4'b0111: seg\_out = 8'b0001111;*//7*  4'b1000: seg\_out = 8'b0000000;*//8*  4'b1001: seg\_out = 8'b0000100;*//9*  **default**: seg\_out = 8'b0011111;*//n*  **endcase**  **end**  **endmodule** |

1. Verilog-HDL entry
2. Diagram, table

   Description automatically generatedRTL verification
3. Graphical user interface, application, Word

   Description automatically generatedGate-level verification

1. Report

To convert BCD to a seven-segment display, we first need an input for the BCD code and an output for the seven-segment display code. Next, because it needs to continuously receive signals, we use "always" to receive the input. Finally, we need to transfer the BCD code to the seven-segment display code. To do this, we can use a case statement to list the BCD codes and their corresponding seven-segment display codes. Here is what the case statement looks like:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **always** @(bcd\_in) **begin**  **case** (bcd\_in)  4'b0000: seg\_out = 8'b0000001; *// 0*  4'b0001: seg\_out = 8'b1001111; *// 1*  4'b0010: seg\_out = 8'b0010010; *// 2*  4'b0011: seg\_out = 8'b0000110; *// 3*  4'b0100: seg\_out = 8'b1001100; *// 4*  4'b0101: seg\_out = 8'b0100100; *// 5*  4'b0110: seg\_out = 8'b0100000; *// 6*  4'b0111: seg\_out = 8'b0001111; *// 7*  4'b1000: seg\_out = 8'b0000000; *// 8*  4'b1001: seg\_out = 8'b0000100; *// 9*  **default**: seg\_out = 8'b0011111; *// blank*  **endcase**  **end** |

# Hardware costs

* + - 1. BCD to seven segment converter

|  |  |
| --- | --- |
| Logic gate | Number of costs(Schematic) |
| AND | 17 |
| OR | 18 |
| NOT | 4 |

* + - 1. Four bits full adder

|  |  |
| --- | --- |
| Logic gate | Number of costs(Schematic) |
| AND | 21 |
| NOT | 6 |
| NAND | 4 |
| NOR | 15 |
| XOR | 4 |

* + - 1. Geq5Plus3

|  |  |
| --- | --- |
| Logic gate | Number of costs(Schematic) |
| AND | 23 |
| OR | 2 |
| NOT | 6 |
| NAND | 4 |
| NOR | 15 |
| XOR | 4 |

* + - 1. Binary to BCD converter

|  |  |
| --- | --- |
| Logic gate | Number of costs(Schematic) |
| AND | 115 |
| OR | 10 |
| NOT | 30 |
| NAND | 20 |
| NOR | 75 |
| XOR | 20 |