DIGITAL DESIGN

ASSIGNMENT REPORT

ASSIGNMENT ID : 1

Student Name: 张嘉浩

Student ID: 12010423

PART 1: DIGITAL DESIGN THEORY

Provide your answers here:

1. (a) 16384 bytes

   (b) 33554432 bytes

   (c) 3435973837 bytes

2. 1111,1111,1111; 4095; FFF (unsigned)

   1111,1111,1111; 2047; 7FF (signed)

3. (a) Divide 248 by 2 iteratively and get its remaining, 1111,1000

   (b) Divide 248 by 16 iteratively and get its remaining, then considering every four digits, transform 8E directly to 1111,1000

   “b” is faster (requires fewer operations), and the answer is 1111,1000.

4. (a) 9’s: 7472,6963

   10’s: 7472,6964

   (b) 9’s: 3567,7389

   10’s: 3567,7390

5. (a) 3941

   (b) 1100,0110,1011,1111

   (c) 0011,1001,0100,0001

   (d) 3941

6. (a) 10111.1001

   (b) 1.1010,1010; 1.6640625; 5 / 3 = 1.66666667, the difference is 0.0026

   (c) 1.AA; 1.6640625; the answer is the same, because it is not a repeating decimal which will not lose accuracy during conversion.

7. (a) 975

   (b) 642

(c) 713

(d) 754

(e) $(2421)_{10}$

8. (a) 4A

(b) DE

(c) 94

(d) 25

(e) B1

(f) B5

(g) 21

PART 2: DIGITAL DESIGN LAB (TASK1)

DESIGN

Describe the design of your system by providing the following information:

- *Verilog design (provide the Verilog code)*

// 1bit

```
1  `timescale 1ns / 1ps
2
3  module UnsignedAddition(addend, augend, sum_led);
4
5  parameter WIDTH = 1;
6
7     input[WIDTH-1:0] addend;
8     input[WIDTH-1:0] augend;
9     output[WIDTH:0] sum_led;
10    assign sum_led = addend + augend;
11
12 endmodule
13
```

// 2bit

```
1  `timescale 1ns / 1ps
2
3  module UnsignedAddition(addend, augend, sum_led);
4
5  parameter WIDTH = 2;
6
7     input[WIDTH-1:0] addend;
8     input[WIDTH-1:0] augend;
9     output[WIDTH:0] sum_led;
10    assign sum_led = addend + augend;
11
12 endmodule
13
```

- *Truth-table*

addend	augend	sum_led
0	0	0
0	1	1
1	0	1
1	1	$(2)_{10}/(10)_2$
00	00	00
00	01	01
00	10	10
00	11	11
01	00	01
01	01	10
01	10	11
01	11	100
10	00	10
10	01	11
10	10	100
10	11	101
11	00	11
11	01	100
11	10	101
11	11	110

SIMULATION

Describe how you build the test bench and do the simulation.

- *Using Verilog (provide the Verilog code)*

// 1 bit:

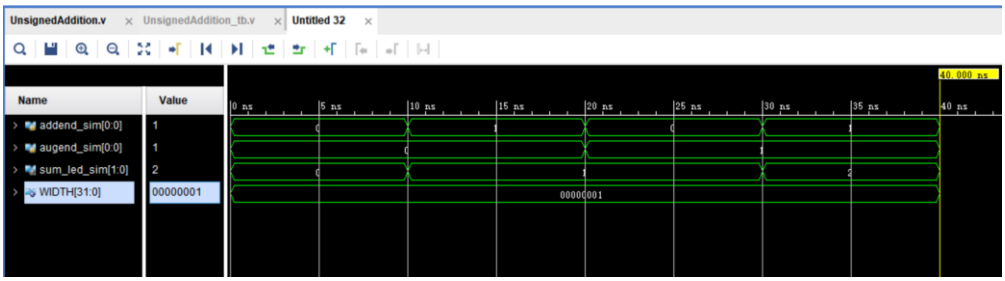
```
1   `timescale 1ns / 1ps
2
3   module UnsignedAddition_tb( );
4
5       parameter WIDTH = 1;
6       reg [WIDTH-1:0] addend_sim, augend_sim;
7       wire [WIDTH:0] sum_led_sim;
8
9       UnsignedAddition #(WIDTH) ua(
10          .addend(addend_sim),
11          .augend(augend_sim),
12          .sum_led(sum_led_sim)
13      );
14
15      initial begin
16          addend_sim = 1'b0; augend_sim = 1'b0;
17          #10
18          addend_sim = 1'b1; augend_sim = 1'b0;
19          #10
20          addend_sim = 1'b0; augend_sim = 1'b1;
21          #10
22          addend_sim = 1'b1; augend_sim = 1'b1;
23          #10
24          $finish();
25      end
26  endmodule
```

// 2bit:

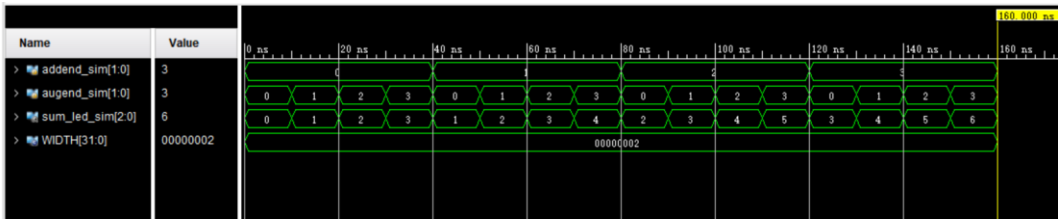
```
28  `timescale 1ns / 1ps
29
30  module UnsignedAddition_tb( );
31      parameter WIDTH = 2;
32      reg [WIDTH-1:0] addend_sim, augend_sim;
33      wire [WIDTH:0] sum_led_sim;
34
35      UnsignedAddition #(WIDTH) ua(
36          .addend(addend_sim),
37          .augend(augend_sim),
38          .sum_led(sum_led_sim)
39      );
40
41      initial begin
42          addend_sim = 2'b00; augend_sim = 2'b00;
43          #10
44          addend_sim = 2'b00; augend_sim = 2'b01;
45          #10
46          addend_sim = 2'b00; augend_sim = 2'b10;
47          #10
48          addend_sim = 2'b00; augend_sim = 2'b11;
49          #10
50          addend_sim = 2'b01; augend_sim = 2'b00;
51          #10
52          addend_sim = 2'b01; augend_sim = 2'b01;
53          #10
54          addend_sim = 2'b01; augend_sim = 2'b10;
55          #10
56          addend_sim = 2'b01; augend_sim = 2'b11;
57          #10
58          addend_sim = 2'b10; augend_sim = 2'b00;
59          #10
60          addend_sim = 2'b10; augend_sim = 2'b01;
61          #10
62          addend_sim = 2'b10; augend_sim = 2'b10;
63          #10
64          addend_sim = 2'b10; augend_sim = 2'b11;
65          #10
66          addend_sim = 2'b11; augend_sim = 2'b00;
67          #10
68          addend_sim = 2'b11; augend_sim = 2'b01;
69          #10
70          addend_sim = 2'b11; augend_sim = 2'b10;
71          #10
72          addend_sim = 2'b11; augend_sim = 2'b11;
73          #10
74          $finish();
75          end
76
77  endmodule
78
```

- *Wave form of simulation result (provide screen shots)*

  //1bit



  //2bit



- *The description on whether the simulation result is same as the truth-table, is the function of the design meet the expectation.*

  The result in the wave form is the same as the value in the truth-table if converted from decimal number system to binary system.

  For example,

  Addend = 0, augend = 0, sum_led = 0(truth-table), sum_led = 0(wave form)

  Addend = 11, augend = 11, sum_led = 110(truth-table), sum_led = 6(wave form)

THE DESCRIPTION OF OPERATION

Describe the problem occurred while in the lab and your solution. Any suggestions are welcomed.

- *Problems and solutions*

1. **Using the parameter for the first time is a little bit difficult for me. Nevertheless, after referring to the discussion in the group and the Q&A in the document, I'm able to use parameter properly.**
2. **Even the last simulation needs to have a "#10" behind when reaching the "$finish( )" statement, or it will not be displayed in the behavioral simulation.**
3. **Being not familiar enough with Verilog, when Vivado display the message that "Syntax error near…" is sometimes not enough for me (not as intelligent as IDEA). Requires more careful speculation into the syntax error.**

PART 2: DIGITAL DESIGN LAB (TASK2)

DESIGN

Describe the design of your system by providing the following information:

- *Verilog design while using data flow (provide the Verilog code)*

  // Use data flow to prove the 1st equation

  //1bit

```
1  `timescale 1ns / 1ps
2
3
4  module distributive1bit_df(
5      input a,
6      input b,
7      input c,
8      output q1,
9      output q2
10     );
11
12     assign q1 = a & (b | c);
13     assign q2 = (a & b) | (a & c);
14
15 endmodule
16
```

//2bit

```
1  `timescale 1ns / 1ps
2
3  module distributive2bit_df #(parameter width = 2)(
4      input [width-1:0] a,
5      input [width-1:0] b,
6      input [width-1:0] c,
7      output [width-1:0] q1,
8      output [width-1:0] q2
9      );
10
11     assign q1 = a & (b | c);
12     assign q2 = (a & b) | (a & c);
13
14 endmodule
15
```

- *Verilog design while using structured design (provide the Verilog code)*

  // Use structured design to prove the 2nd equation

//1bit

```
1  `timescale 1ns / 1ps
2
3
4  module distributive1bit_sd(A, B, C, q1, q2);
5  input A,B,C;
6  output q1,q2;
7  wire o_BandC, o_AorB, o_AorC;
8
9  and and1(o_BandC, B, C);
10 or or1(o_AorB, A, B);
11 or or2(o_AorC, A, C);
12
13 //final
14 or or3(q1,A,o_BandC);
15 and and2(q2,o_AorB,o_AorC);
16
17 endmodule
18
```

//2bit

```
1  `timescale 1ns / 1ps
2
3  module distributive2bit_sd
4  #(parameter width=2) (
5    input [width-1:0] A,
6    input [width-1:0] B,
7    input [width-1:0] C,
8    output [width-1:0] q1,
9    output [width-1:0] q2
10 );
11   wire [width-1:0] o_AorC;
12   wire [width-1:0] o_AorB;
13   wire [width-1:0] o_BandC;
14
15   and and1_0(o_BandC[0], B[0], C[0]);
16   and and1_1(o_BandC[1], B[1], C[1]);
17   or or1_0(o_AorB[0], A[0], B[0]);
18   or or1_1(o_AorB[1], A[1], B[1]);
19   or or2_0(o_AorC[0], A[0], C[0]);
20   or or2_1(o_AorC[1], A[1], C[1]);
21
22   //final
23   or or3_0(q1[0], A[0], o_BandC[0]);
24   or or3_1(q1[1], A[1], o_BandC[1]);
25   and and2_0(q2[0], o_AorB[0], o_AorC[0]);
26   and and2_1(q2[1], o_AorB[1], o_AorC[1]);
27
28 endmodule
29
```

- *Truth-table*

A	B	C	q1	q2	q3	q4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1
00	00	00	00	00	00	00
00	00	01	00	00	00	00
00	00	10	00	00	00	00
00	00	11	00	00	00	00
00	01	00	00	00	00	00
00	01	01	00	00	01	01
00	01	10	00	00	00	00
00	01	11	00	00	01	01
00	10	00	00	00	00	00
00	10	01	00	00	00	00
00	10	10	00	00	10	10
00	10	11	00	00	10	10
00	11	00	00	00	00	00
00	11	01	00	00	01	01
00	11	10	00	00	10	10
00	11	11	00	00	11	11
01	00	00	00	00	00	00
01	00	01	01	01	01	01
01	00	10	00	00	01	01
01	00	11	01	01	01	01
01	01	00	01	01	01	01
01	01	01	01	01	01	01
01	01	10	01	01	01	01
01	01	11	01	01	01	01
01	10	00	00	00	01	01
01	10	01	01	01	01	01
01	10	10	00	00	11	11
01	10	11	01	01	11	11
01	11	00	01	01	01	01
01	11	01	01	01	01	01
01	11	10	01	01	11	11
01	11	11	01	01	11	11
10	00	00	00	00	10	10
10	00	01	00	00	10	10
10	00	10	10	10	10	10
10	00	11	10	10	10	10
10	01	00	00	00	10	10
10	01	01	00	00	11	11
10	01	10	10	10	10	10
10	01	11	10	10	11	11
10	10	00	10	10	10	10
10	10	01	10	10	10	10
10	10	10	10	10	10	10
10	10	11	10	10	10	10
10	11	00	10	10	10	10
10	11	01	10	10	11	11
10	11	10	10	10	10	10
10	11	11	10	10	11	11
11	00	00	00	00	11	11
11	00	01	01	01	11	11
11	00	10	10	10	11	11
11	00	11	11	11	11	11
11	01	00	01	01	11	11
11	01	01	01	01	11	11
11	01	10	11	11	11	11
11	01	11	11	11	11	11
11	10	00	10	10	11	11
11	10	01	11	11	11	11
11	10	10	10	10	11	11
11	10	11	11	11	11	11
11	11	00	11	11	11	11
11	11	01	11	11	11	11
11	11	10	11	11	11	11
11	11	11	11	11	11	11

-

SIMULATION

Describe how you build the test bench and do the simulation.

- *Using Verilog (provide the Verilog code)*

  // distributive1bit_dataflow (1st case)

```
1   `timescale 1ns / 1ps
2
3   module distributive1bit_df_sim( );
4     reg sima, simb, simc;
5     wire simq1, simq2;
6     distributive1bit_df u_df(
7     .a(sima), .b(simb), .c(simc), .q1(simq1), .q2(simq2) );
8
9     initial
10    begin
11    sima = 0;
12    simb = 0;
13    simc = 0;
14    #10
15    sima = 0;
16    simb = 0;
17    simc = 1;
18    #10
19    sima = 0;
20    simb = 1;
21    simc = 0;
22    #10
23    sima = 0;
24    simb = 1;
25    simc = 1;
26    #10
27    sima = 1;
28    simb = 0;
29    simc = 0;
30    #10
31    sima = 1;
32    simb = 0;
33    simc = 1;
34    #10
35    sima = 1;
36    simb = 1;
37    simc = 0;
38    #10
39    sima = 1;
40    simb = 1;
41    simc = 1;
42    #10
43    $finish();
44    end
45  endmodule
46
```

//distributive2bit_dataflow (1st case)

```
`timescale 1ns / 1ps

module distributive2bit_df_sim( );
  reg [1:0] sima, simb, simc;
  wire [1:0] simq1, simq2;
  distributive2bit_df u_df(
  .a(sima), .b(simb), .c(simc), .q1(simq1), .q2(simq2) );
  initial
    begin
      {sima, simb, simc} = 6'b000000;
      #10
      {sima, simb, simc} = 6'b000001;
      #10
      {sima, simb, simc} = 6'b000010;
      #10
      {sima, simb, simc} = 6'b000011;
      #10
      {sima, simb, simc} = 6'b000100;
      #10
      {sima, simb, simc} = 6'b000101;
      #10
      {sima, simb, simc} = 6'b000110;
      #10
      {sima, simb, simc} = 6'b000111;
      #10
      {sima, simb, simc} = 6'b001000;
      #10
      {sima, simb, simc} = 6'b001001;
      #10
      {sima, simb, simc} = 6'b001010;
      #10
      {sima, simb, simc} = 6'b001011;
      #10
      {sima, simb, simc} = 6'b001100;
      #10
      {sima, simb, simc} = 6'b001101;
      #10
      {sima, simb, simc} = 6'b001110;
      #10
      {sima, simb, simc} = 6'b001111;
      #10
      {sima, simb, simc} = 6'b010000;
      #10
      {sima, simb, simc} = 6'b010001;
      #10
      {sima, simb, simc} = 6'b010010;
      #10
      {sima, simb, simc} = 6'b010011;
      #10
      {sima, simb, simc} = 6'b010100;
      #10
      {sima, simb, simc} = 6'b010101;
      #10
      {sima, simb, simc} = 6'b010110;
      #10
      {sima, simb, simc} = 6'b010111;
      #10
      {sima, simb, simc} = 6'b011000;
      #10
      {sima, simb, simc} = 6'b011001;
      #10
      {sima, simb, simc} = 6'b011010;
      #10
      {sima, simb, simc} = 6'b011011;
      #10
      {sima, simb, simc} = 6'b011100;
      #10
      {sima, simb, simc} = 6'b011101;
      #10
      {sima, simb, simc} = 6'b011110;
      #10
      {sima, simb, simc} = 6'b011111;
      #10
      {sima, simb, simc} = 6'b100000;
      #10
      {sima, simb, simc} = 6'b100001;
      #10
      {sima, simb, simc} = 6'b100010;
      #10
      {sima, simb, simc} = 6'b100011;
      #10
      {sima, simb, simc} = 6'b100100;
      #10
      {sima, simb, simc} = 6'b100101;
      #10
      {sima, simb, simc} = 6'b100110;
      #10
      {sima, simb, simc} = 6'b100111;
      #10
      {sima, simb, simc} = 6'b101000;
      #10
      {sima, simb, simc} = 6'b101001;
      #10
      {sima, simb, simc} = 6'b101010;
      #10
      {sima, simb, simc} = 6'b101011;
      #10
      {sima, simb, simc} = 6'b101100;
      #10
      {sima, simb, simc} = 6'b101101;
      #10
      {sima, simb, simc} = 6'b101110;
      #10
      {sima, simb, simc} = 6'b101111;
      #10
      {sima, simb, simc} = 6'b110000;
      #10
      {sima, simb, simc} = 6'b110001;
      #10
      {sima, simb, simc} = 6'b110010;
      #10
      {sima, simb, simc} = 6'b110011;
      #10
      {sima, simb, simc} = 6'b110100;
      #10
      {sima, simb, simc} = 6'b110101;
      #10
      {sima, simb, simc} = 6'b110110;
      #10
      {sima, simb, simc} = 6'b110111;
      #10
      {sima, simb, simc} = 6'b111000;
      #10
      {sima, simb, simc} = 6'b111001;
      #10
      {sima, simb, simc} = 6'b111010;
      #10
      {sima, simb, simc} = 6'b111011;
      #10
      {sima, simb, simc} = 6'b111100;
      #10
      {sima, simb, simc} = 6'b111101;
      #10
      {sima, simb, simc} = 6'b111110;
      #10
      {sima, simb, simc} = 6'b111111;
      #10
      $finish();
    end
endmodule
```

//distributive1bit_StructureDesign (2nd equation)

```
1  `timescale 1ns / 1ps
2
3  module distributive1bit_sd_sim( );
4  reg A,B,C;
5  wire q1,q2;
6  distributive1bit_sd usd(A,B,C,q1,q2);
7
8  initial
9     begin
10       {A,B,C} = 3'b000;
11       #10
12       {A,B,C} = 3'b001;
13       #10
14       {A,B,C} = 3'b010;
15       #10
16       {A,B,C} = 3'b011;
17       #10
18       {A,B,C} = 3'b100;
19       #10
20       {A,B,C} = 3'b101;
21       #10
22       {A,B,C} = 3'b110;
23       #10
24       {A,B,C} = 3'b111;
25       #10
26       $finish(0);
27    end
28 endmodule
29
```

//distributive2bit_StructureDesign (2nd equation)

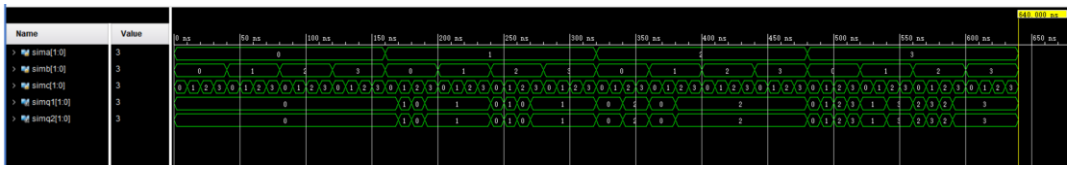
```
`timescale 1ns / 1ps

module distributive2bit_sd_sim ( );
reg [1:0] A,B,C;
wire [1:0] q1,q2;
distributive2bit_sd usd(A,B,C,q1,q2);

initial
begin
    {A, B, C} = 6'b000000;
    #10
    {A, B, C} = 6'b000001;
    #10
    {A, B, C} = 6'b000010;
    #10
    {A, B, C} = 6'b000011;
    #10
    {A, B, C} = 6'b000100;
    #10
    {A, B, C} = 6'b000101;
    #10
    {A, B, C} = 6'b000110;
    #10
    {A, B, C} = 6'b000111;
    #10
    {A, B, C} = 6'b001000;
    #10
    {A, B, C} = 6'b001001;
    #10
    {A, B, C} = 6'b001010;
    #10
    {A, B, C} = 6'b001011;
    #10
    {A, B, C} = 6'b001100;
    #10
    {A, B, C} = 6'b001101;
    #10
    {A, B, C} = 6'b001110;
    #10
    {A, B, C} = 6'b001111;
    #10
    {A, B, C} = 6'b010000;
    #10
    {A, B, C} = 6'b010001;
    #10
    {A, B, C} = 6'b010010;
    #10
    {A, B, C} = 6'b010011;
    #10
    {A, B, C} = 6'b010100;
    #10
    {A, B, C} = 6'b010101;
    #10
    {A, B, C} = 6'b010110;
    #10
    {A, B, C} = 6'b010111;
    #10
    {A, B, C} = 6'b011000;
    #10
    {A, B, C} = 6'b011001;
    #10
    {A, B, C} = 6'b011010;
    #10
    {A, B, C} = 6'b011011;
    #10
    {A, B, C} = 6'b011100;
    #10
    {A, B, C} = 6'b011101;
    #10
    {A, B, C} = 6'b011110;
    #10
    {A, B, C} = 6'b011111;
    #10
    {A, B, C} = 6'b100000;
    #10
    {A, B, C} = 6'b100001;
    #10
    {A, B, C} = 6'b100010;
    #10
    {A, B, C} = 6'b100011;
    #10
    {A, B, C} = 6'b100100;
    #10
    {A, B, C} = 6'b100101;
    #10
    {A, B, C} = 6'b100110;
    #10
    {A, B, C} = 6'b100111;
    #10
    {A, B, C} = 6'b101000;
    #10
    {A, B, C} = 6'b101001;
    #10
    {A, B, C} = 6'b101010;
    #10
    {A, B, C} = 6'b101011;
    #10
    {A, B, C} = 6'b101100;
    #10
    {A, B, C} = 6'b101101;
    #10
    {A, B, C} = 6'b101110;
    #10
    {A, B, C} = 6'b101111;
    #10
    {A, B, C} = 6'b110000;
    #10
    {A, B, C} = 6'b110001;
    #10
    {A, B, C} = 6'b110010;
    #10
    {A, B, C} = 6'b110011;
    #10
    {A, B, C} = 6'b110100;
    #10
    {A, B, C} = 6'b110101;
    #10
    {A, B, C} = 6'b110110;
    #10
    {A, B, C} = 6'b110111;
    #10
    {A, B, C} = 6'b111000;
    #10
    {A, B, C} = 6'b111001;
    #10
    {A, B, C} = 6'b111010;
    #10
    {A, B, C} = 6'b111011;
    #10
    {A, B, C} = 6'b111100;
    #10
    {A, B, C} = 6'b111101;
    #10
    {A, B, C} = 6'b111110;
    #10
    {A, B, C} = 6'b111111;
    #10
    $finish();
  end
endmodule
```

- *Wave form of simulation result (provide screen shots)*

//Data flow

//1bit

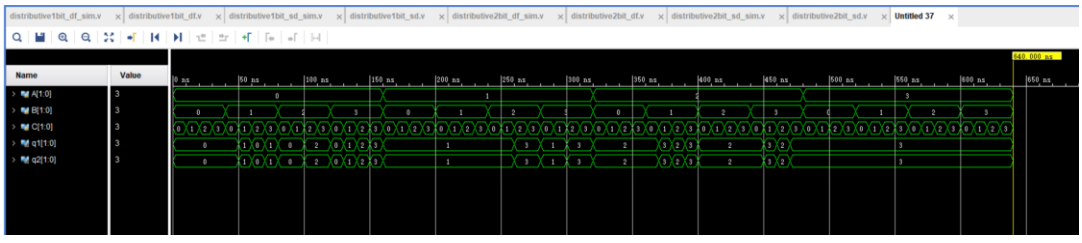
//2bit



//Structure Design

//1bit

//2bit



- *The description on whether the simulation result is same as the truth-table, is the function of the design meet the expectation*

Focusing on A, B, C, q1, q2, on data flow, the value in the truth table is literally the value of the simulation result, which change from the decimal number system to the binary number system.

For example,

A=1,B=1,C=1,q1=1,q2=1(truth table) q1=1,q2=1(wave form)

A=11,B=11,C=11,q1=11,q2=11(truth table) q1=3,q2=3(wave form)

Focusing on A, B, C, q3, q4, on structure design, the value in the truth table is literally the value of the simulation result, which change from the decimal number system to the binary number system.

For example,

A=1, B=1, C=1, q1=1, q2=1(truth table) q1=1, q2=1(wave form)

A=11, B=11, C=11, q1=11, q2=11(truth table) q1=3, q2=3(wave form)

THE DESCRIPTION OF OPERATION

Describe the problem occurred while in the lab and your solution. Any suggestions are welcomed.

- *Problems and solutions*

1. **Due to the lack of familiarity of Verilog grammar, I sometimes need to refer to the lab slides when coding. Such as exemplifying and binding the variable in design sources to simulation sources.**
2. **The lack of familiarity when dealing with multi-bits. I review the lecture the watch the lab video on blackboard which finally dispel my uncertainty.**
3. **I encounter the problem that there's no syntax error but when I attempt to run behavioral simulation, the error exists. I watch the video that the instructor record**

on blackboard, learnt to find the error in “log” file and “show message below”, which made me soon solve the problem.