Name: Zhijie Lan

Number: 201990309

Q1.

a. i.

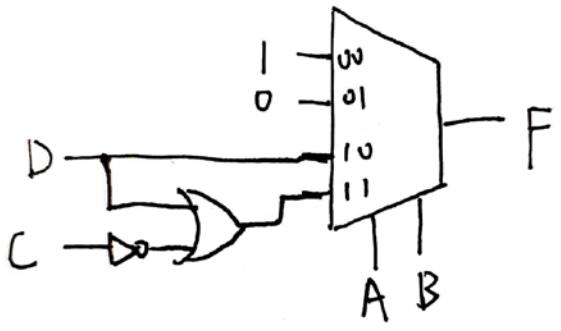
Q1. a,

i. $F(A,B,C,D) = \sum(0,1,2,3,9,11,12,13,15)$

	A	B	C	D
a_0	0	0	0	0
a_1	0	0	0	1
a_2	0	0	1	0
a_3	0	0	1	1
a_9	1	0	0	1
a_{11}	1	0	1	1
a_{12}	1	1	0	0
a_{13}	1	1	0	1
a_{15}	1	1	1	1

sel A	sel B	Input
0	0	$D_0 = \bar{C}\bar{D} + \bar{C}D + C\bar{D} + CD = 1$
0	1	$D_1 = 0$
1	0	$D_2 = \bar{C}D + CD = D$
1	1	$D_3 = \bar{C}\bar{D} + \bar{C}D + CD = \bar{C} + D$

ii.



iii: 4x1 multiplexer code:

```
library ieee;
use ieee.std_logic_1164.all;

entity mux4 is
	port (s0, s1		:	in	std_logic;
		d0, d1, d2, d3	:	in	std_logic;
		F_mux4		:	out std_logic );
end mux4;

architecture design_mux4 of mux4 is
signal s0s1 : std_logic_vector(1 downto 0);
begin
	s0s1  <= s0 & s1;
	F_mux4 <= d0 when s0s1 = "00" else
		  d1 when s0s1 = "01" else
				d2 when s0s1 = "10" else
				d3;
end design_mux4;
```

Final entity code:

```
library ieee;
use ieee.std_logic_1164.all;

entity Q1 is
	port (A, B, C, D	:	in	std_logic;
		F_Q1		:	out std_logic );
end Q1;

architecture design_Q1 of Q1 is

component mux4 is
	port (s0, s1		:	in	std_logic;
		d0, d1, d2, d3	:	in	std_logic;
		F_mux4		:	out std_logic );
end component;

signal cbar_d : std_logic;
begin

cbar_d <= (not C) or D;

m1: mux4 port map (A, B, '1', '0', D, cbar_d, F_Q1);

end design_Q1
```

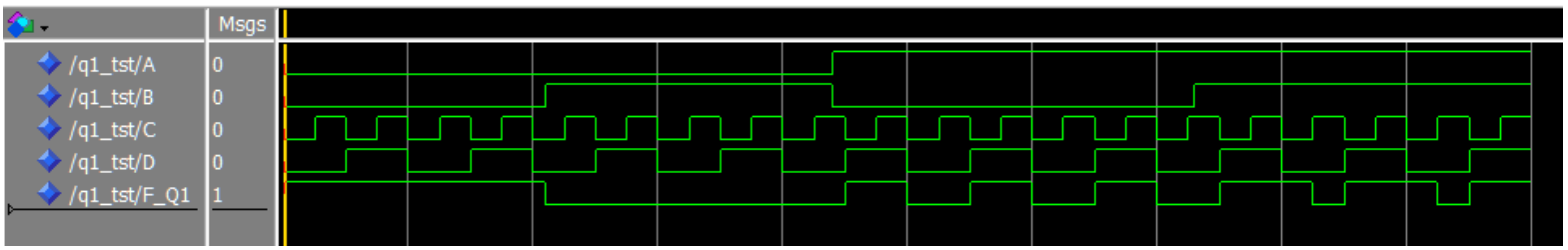
iV:

Testbench code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Q1_tst IS
END Q1_tst;
ARCHITECTURE Q1_arch OF Q1_tst IS

SIGNAL A : STD_LOGIC;
SIGNAL B : STD_LOGIC;
SIGNAL C : STD_LOGIC;
SIGNAL D : STD_LOGIC;
SIGNAL F_Q1 : STD_LOGIC;
COMPONENT Q1
	PORT (
	A : IN STD_LOGIC;
	B : IN STD_LOGIC;
	C : IN STD_LOGIC;
	D : IN STD_LOGIC;
	F_Q1 : OUT STD_LOGIC
	);
END COMPONENT;
BEGIN
	i1 : Q1
	PORT MAP (

	A => A,
	B => B,
	C => C,
	D => D,
	F_Q1 => F_Q1
	);


t_A: PROCESS
BEGIN
	A <= '0';
	WAIT FOR 440000 ps;
	A <= '1';
WAIT;
END PROCESS t_A;


t_B: PROCESS
BEGIN
	B <= '0';
```

```
        WAIT FOR 210000 ps;
        B <= '1';
        WAIT FOR 230000 ps;
        B <= '0';
        WAIT FOR 290000 ps;
        B <= '1';
WAIT;
END PROCESS t_B;


t_C: PROCESS
BEGIN
LOOP
        C <= '0';
        WAIT FOR 25000 ps;
        C <= '1';
        WAIT FOR 25000 ps;
        IF (NOW >= 1000000 ps) THEN WAIT; END IF;
END LOOP;
END PROCESS t_C;


t_D: PROCESS
BEGIN
LOOP
        D <= '0';
        WAIT FOR 50000 ps;
        D <= '1';
        WAIT FOR 50000 ps;
        IF (NOW >= 1000000 ps) THEN WAIT; END IF;
END LOOP;
END PROCESS t_D;
END Q1_arch;
```

Test wave:



The input AB starts from “00” to “01”, “10”, and “11”.
When AB = “00”, the output F_Q1 is always ‘1’.
When AB = “01”, the output F_Q1 is always ‘0’.
When AB = “10”, we can see that the output wave is always same with the wave of D, which means the output F_Q1 is always ‘D’.
When AB = “11”, we can see that the output wave is always same with the wave of D plus a small length when C is 0. Thus, F_Q1 is always ‘not C’ or ‘D’.

b. i:

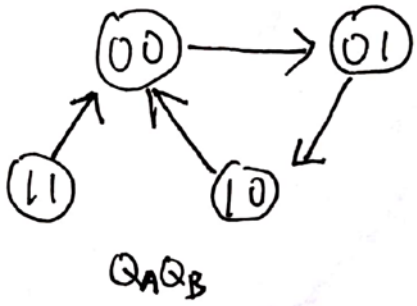
equation for FF_A and FF_B

$$Q_A^{n+1} = T_A \oplus Q_A^n$$
$$= (Q_A^n + Q_B^n) \oplus Q_A^n$$
$$= (Q_A^n + Q_B^n) \cdot \overline{Q_A^n} + \overline{Q_A^n + Q_B^n} \cdot Q_A^n$$
$$= \overline{Q_A^n} \cdot Q_B^n + \overline{Q_A^n} \cdot Q_B^n \cdot Q_A^n$$
$$= \overline{Q_A^n} \cdot Q_B^n$$

$$Q_B^{n+1} = T_B \oplus Q_B^n$$
$$= (\overline{Q_A^n} + Q_B^n) \oplus Q_B^n$$
$$= (\overline{Q_A^n} + Q_B^n) \cdot \overline{Q_B^n} + \overline{\overline{Q_A^n} + Q_B^n} \cdot Q_B^n$$
$$= \overline{Q_A^n} \cdot \overline{Q_B^n} + Q_A^n \cdot \overline{Q_B^n} \cdot Q_B^n$$
$$= \overline{Q_A^n} \cdot \overline{Q_B^n}$$

State table, and state diagram:

Q_A^n	Q_B^n	Q_A^{n+1}	Q_B^{n+1}
0	0	0	1
0	1	1	0
1	0	0	0
1	1	0	1



ii: Function: This is a counter that can count 3. And it has the ability to automatically start.

iii: structural description code:

Inverter:

```
library ieee;
use ieee.std_logic_1164.all;

entity inv is
	port (i1		:	in		std_logic;
		F_inv	:	out	std_logic );
end inv;

architecture design_inv of inv is
begin
	F_inv <= not i1;
end design_inv;
```

Or gate:

```
library ieee;
use ieee.std_logic_1164.all;

entity or_2 is
	port ( o1, o2		: in	std_logic;
		F_or_2		: out	std_logic );
end or_2;

architecture design_or_2 of or_2 is
begin
	F_or_2 <= o1 or o2;
end design_or_2;
```

T Flip Flop:

```
library ieee;
use ieee.std_logic_1164.all;

entity T_FF is
	port ( clk, T		: in	std_logic;
		Q, Q_bar	: out	std_logic );
end T_FF;

architecture design_T_FF of T_FF is

component inv is
	port (i1		:	in		std_logic;
		F_inv	:	out	std_logic );
end component;

signal temp_Q, temp_Q_bar : std_logic:='0';
```

```
begin

	process(clk)
	begin
		if clk'event and clk='1' then
			if T='0' then
				temp_Q <= temp_Q;
			elsif T='1' then
				temp_Q <= temp_Q_bar;
			end if;
		end if;
	end process;

iv1: inv port map (temp_Q, temp_Q_bar);
	Q <= temp_Q;
	Q_bar <= temp_Q_bar;
end design_T_FF;
```

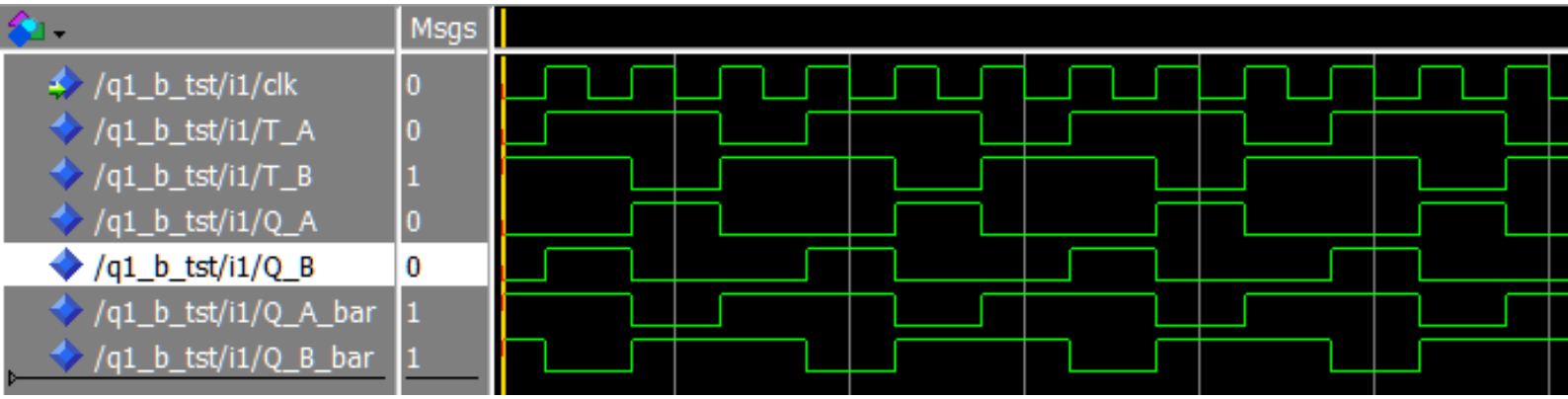
Final entity:

```
library ieee;
use ieee.std_logic_1164.all;

entity Q1_b is
	port(clk: in std_logic);
end Q1_b;

architecture design_Q1_b of Q1_b is

component or_2 is
	port ( o1, o2		: in	std_logic;
		F_or_2		: out	std_logic );
end component;

component T_FF is
	port ( clk, T		: in	std_logic;
		Q, Q_bar	: out	std_logic );
end component;

signal T_A, T_B, Q_A, Q_B, Q_A_bar, Q_B_bar : std_logic;

begin

T1: T_FF port map (clk, T_A, Q_A, Q_A_bar);
T2: T_FF port map (clk, T_B, Q_B, Q_B_bar);
or_A: or_2 port map (Q_A, Q_B, T_A);
or_B: or_2 port map (Q_A_bar, Q_B, T_B);

end design_Q1_b;
```

Testbench code:

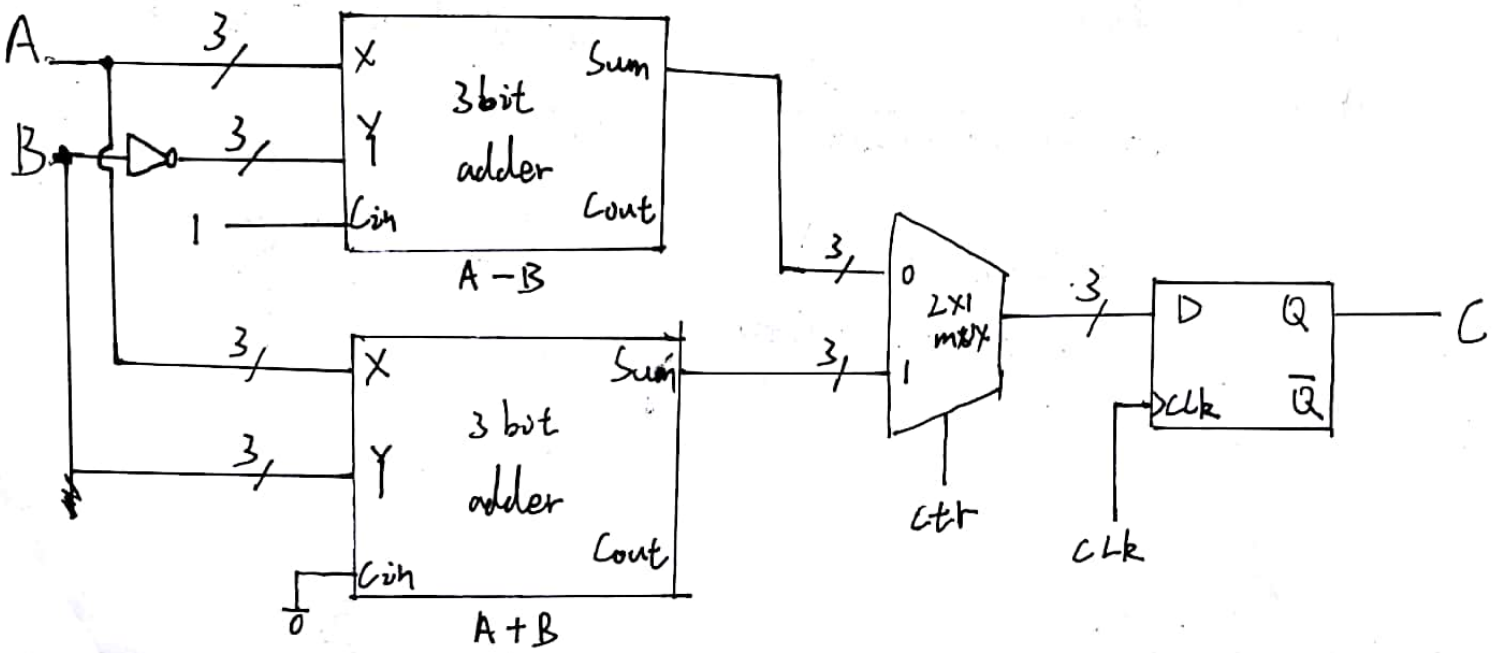
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Q1_b_tst IS
END Q1_b_tst;
ARCHITECTURE Q1_b_arch OF Q1_b_tst IS

SIGNAL clk : STD_LOGIC;
COMPONENT Q1_b
	PORT (
	clk : IN STD_LOGIC
	);
END COMPONENT;
BEGIN
	i1 : Q1_b
	PORT MAP (
	clk => clk
	);

t_clk: PROCESS
BEGIN
LOOP
	clk <= '0';
	WAIT FOR 25000 ps;
	clk <= '1';
	WAIT FOR 25000 ps;
	IF (NOW >= 1000000 ps) THEN WAIT; END IF;
END LOOP;
END PROCESS t_clk;
END Q1_b_arch
```

Result wave:



From the wave, we can see that the state (Q_A, Q_B) starts from “00” going to “01”, then”10”, then back to “00”, just like the state diagram given above.

c. i



ii. The (A-B) adder inputs A and B', '1' as Cin. The (A+B) adder inputs A and B, '0' as Cin.

The 2x1 mux has (A-B) and (A+B) as inputs, ctr as the selection bit. When ctr = '0', the mux outputs (A-B). When ctr='1', the mux outputs (A+B).

The D flip flop has the output of 2x1 mux as the D input, clk as the clock signal. When clk at its rising edges, it outputs corresponding results to C.

Q2.

a. Form the article, we can get the steps for the Booth's algorithm:

1) Get the two's compliment for multiplicand and multiplier as *M* and *Q*

2) Initialize the *A* and Q_{-1} as '0', where the Q_{-1} is the right bit of the multiplier's 0 bit (Q_0).
Initialize the count as 'n', where 'n' is the number of bits used in multiplicand and multiplier

3) Start a loop for 'n' times.
In each loop, operate the A following the value of Q_0Q_{-1}.

Q_0Q_{-1}	A
00	A=A
01	A=A+M
10	A=A-M
11	A=A

Then, concatenate *A*, *Q*, and Q_{-1} as a whole and right shift it.
Then, n=n-1, continue loop until n = 0.

4) After the loop, get the concatenation of *AQ*, then the multiplication result is the two's compliment of that concatenation of *AQ*.

b. My student number is 201990309
So my multiplicand is -2; my multiplier is 9.
Set the 5th bit as the sign bit.
So -2 is 10010, 9 is 01001
The two's compliment of -2 is 11110

Thus, M(-2) = 11110, Q(9) = 01001

A	Q	Q_{-1}	M	Operations
00000	01001	0	11110	Initial Values
00010	01001	0	11110	A=A-M
00001	00100	1	11110	Shift
11111	00100	1	11110	A=A+M
11111	10010	0	11110	Shift
11111	11001	0	11110	Shift
00001	11001	0	11110	A=A-M
00000	11100	1	11110	Shift
11110	11100	1	11110	A=A+M
11111	01110	0	11110	Shift

So the result is 11,1110,1110. The MSB is '1' so the number is negative. Its one's compliment = 10,0001,0001.
Its two's compliment = 10,0001,0010 where the MSB is the sign bit. So the number is -18.

c. Code:

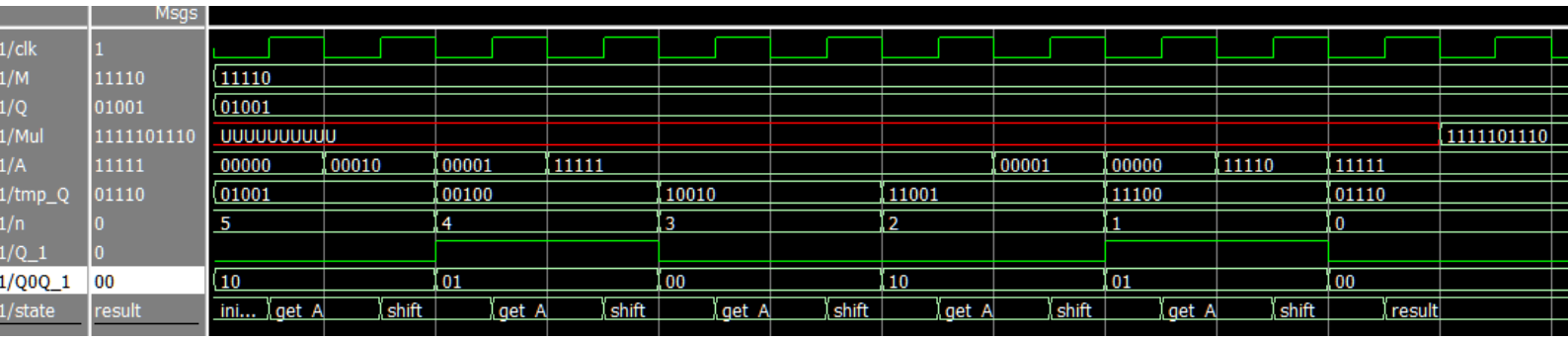
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Q2 is
	port (clk	: in std_logic;
		M, Q	:	in	std_logic_vector(4 downto 0);
		Mul	:	out	std_logic_vector(9 downto 0));
end Q2;

architecture design_Q2 of Q2 is

signal A : std_logic_vector(4 downto 0) := "00000";
signal tmp_Q : std_logic_vector(4 downto 0);

signal n : integer := 5;
signal Q_1 : std_logic := '0';
signal Q0Q_1 : std_logic_vector(1 downto 0);

type state_type is (initial, get_A, shift, result);
signal state: state_type := initial ;

begin

process ( state, clk)
```

```
begin
if (clk'event and clk='1') then
	case state is
		when initial =>
			state <= get_A;

		when get_A =>
			state <= shift;

		when shift =>
			if n/=0 then
				state <= get_A;
			else
				state <= result;
			end if;

		when result =>
			state <= result;
	end case ;
elsif (clk'event and clk='0') then
		case state is
			when initial =>
				tmp_Q <= Q;
			when get_A =>
				if Q0Q_1="01" then
					A <= A + M ;
				elsif Q0Q_1="10" then
					A <= A - M ;
				end if;
			when shift =>
				--shift A,Q,Q_1
				Q_1 <= tmp_Q(0);
				tmp_Q(3 downto 0) <= tmp_Q(4 downto 1);
				tmp_Q(4) <= A(0);
				A(3 downto 0) <= A(4 downto 1);
				n<=n-1;
			when result =>
				Mul <= A & tmp_Q;
		end case;
end if;
end process ;

Q0Q_1 <= tmp_Q(0) & Q_1;

end design_Q2;
```

Test bench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY Q2_tst IS
END Q2_tst;
ARCHITECTURE Q2_arch OF Q2_tst IS
SIGNAL clk : STD_LOGIC;
SIGNAL M : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL Mul : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL Q : STD_LOGIC_VECTOR(4 DOWNTO 0);
COMPONENT Q2
	PORT (
	clk : IN STD_LOGIC;
	M : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
	Mul : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
	Q : IN STD_LOGIC_VECTOR(4 DOWNTO 0)
	);
END COMPONENT;
BEGIN
	i1 : Q2
	PORT MAP (
	clk => clk,
	M => M,
	Mul => Mul,
	Q => Q
	);

t_clk: PROCESS
BEGIN
LOOP
	clk <= '0';
	WAIT FOR 25000 ps;
	clk <= '1';
	WAIT FOR 25000 ps;
	IF (NOW >= 1000000 ps) THEN WAIT; END IF;
END LOOP;
END PROCESS t_clk;

t_M: PROCESS
BEGIN
	M <= "11110";
WAIT;
END PROCESS t_M;
t_Q: PROCESS
BEGIN
	Q <= "01001";
WAIT;
END PROCESS t_Q;
END Q2_arch;
```

Test wave:

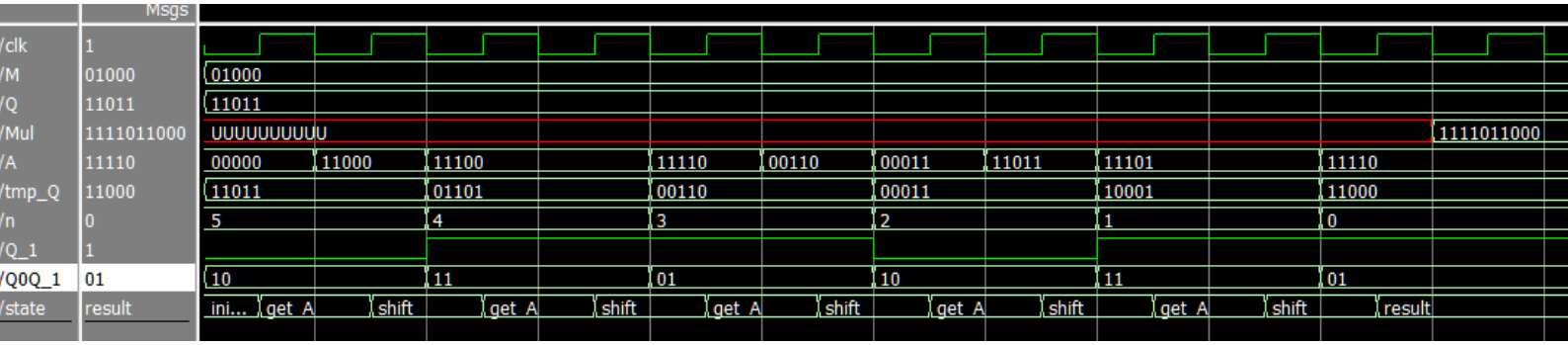


From the wave we can see

Q = 11110 = -2

M = 01001 = 9

The result is 11,1110,1110 which is the two's compliment of -18.



From this wave we can see

M = 01000 = 8

Q = 11011 = -5

The result is 11,1101,1000 which is the two's compliment of -40.

Q3.

Design 1

a. My student number is 201990309. So the last two bits are "09"

BCD of 09 : 00001001

7 bits code : 0001001

My code has errors. Since my D4, D3, D2, D1 = 0, my P3, P2, P1 should be 0.

The right code: 0000000

b. For odd parity:

$$P_3 = \overline{D_2 \oplus D_3 \oplus D_4}$$

$$P_2 = \overline{D_1 \oplus D_3 \oplus D_4}$$

$$P_1 = \overline{D_1 \oplus D_2 \oplus D_4}$$

$$S_3 = \overline{P_3 \oplus D_2 \oplus D_3 \oplus D_4}$$

$$S_2 = \overline{P_2 \oplus D_1 \oplus D_3 \oplus D_4}$$

$$S_1 = \overline{P_1 \oplus D_1 \oplus D_2 \oplus D_4}$$

c. Code:

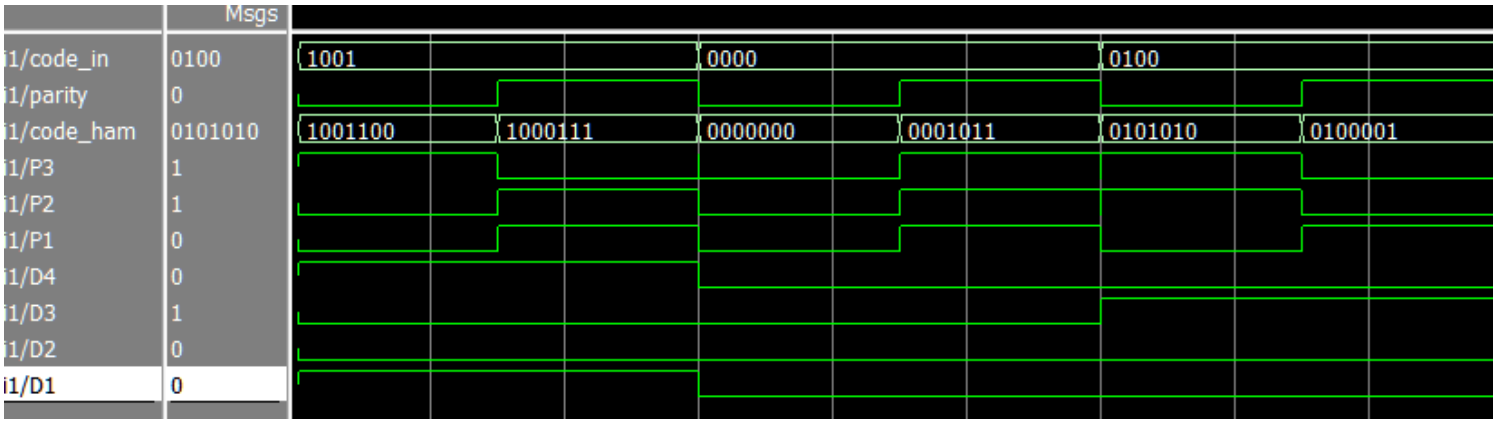
```
library ieee;
use ieee.std_logic_1164.all;

entity Q3_D1_encoder is
       port (code_in :      in std_logic_vector(3 downto 0);
             parity :       in     std_logic;
            code_ham :      out    std_logic_vector(6 downto 0));
end Q3_D1_encoder;

architecture design of Q3_D1_encoder is
signal P3, P2, P1 : std_logic;
signal D4, D3, D2, D1 : std_logic;

begin

D4 <= code_in(3);
D3 <= code_in(2);
D2 <= code_in(1);
D1 <= code_in(0);

P3 <= D2 xor D3 xor D4 when parity = '0' else
       not (D2 xor D3 xor D4);

P2 <= D1 xor D3 xor D4 when parity = '0' else
       not (D1 xor D3 xor D4);

P1 <= D1 xor D2 xor D4 when parity = '0' else
       not (D1 xor D2 xor D4);

code_ham(6) <= D4;
code_ham(5) <= D3;
code_ham(4) <= D2;
code_ham(3) <= P3;
code_ham(2) <= D1;
code_ham(1) <= P2;
code_ham(0) <= P1;

end design;
```

Test bench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Q3_D1_encoder_tst IS
END Q3_D1_encoder_tst;
ARCHITECTURE Q3_D1_encoder_arch OF Q3_D1_encoder_tst IS

SIGNAL code_ham : STD_LOGIC_VECTOR(6 DOWNTO 0);
```

```
SIGNAL code_in : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL parity : STD_LOGIC;
COMPONENT Q3_D1_encoder
	PORT (
	code_ham : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
	code_in : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
	parity : IN STD_LOGIC
	);
END COMPONENT;
BEGIN
	i1 : Q3_D1_encoder
	PORT MAP (
	code_ham => code_ham,
	code_in => code_in,
	parity => parity
	);
t_code_in: PROCESS
BEGIN
	code_in <= "1001";
	WAIT FOR 300000 ps;
	code_in <= "0000";
	WAIT FOR 300000 ps;
	code_in <= "0100";
WAIT;
END PROCESS t_code_in;
t_parity: PROCESS
BEGIN
	FOR i IN 1 TO 3
	LOOP
		parity <= '0';
		WAIT FOR 150000 ps;
		parity <= '1';
		WAIT FOR 150000 ps;
	END LOOP;
	parity <= '0';
WAIT;
END PROCESS t_parity;
END Q3_D1_encoder_arch;
```

Test wave:



d. Code:

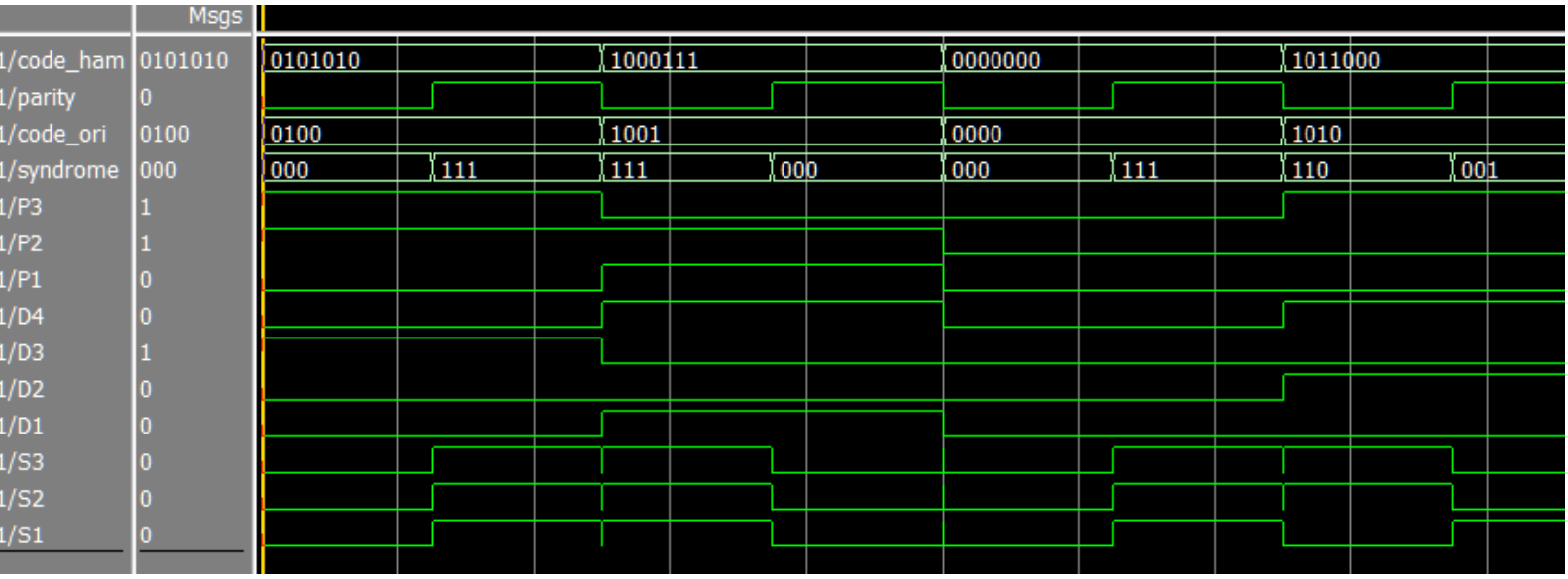
```
library ieee;
use ieee.std_logic_1164.all;

entity Q3_D1_decoder is
       port (code_ham       :      in std_logic_vector(6 downto 0);
                parity      :      in     std_logic;
              code_ori      :      out    std_logic_vector(3 downto 0);
              syndrome      : out  std_logic_vector(2 downto 0));
end Q3_D1_decoder;

architecture design of Q3_D1_decoder is
signal P3, P2, P1 : std_logic;
signal D4, D3, D2, D1 : std_logic;
signal S3, S2, S1 : std_logic;

begin

D4 <= code_ham(6);
D3 <= code_ham(5);
D2 <= code_ham(4);
P3 <= code_ham(3);
D1 <= code_ham(2);
P2 <= code_ham(1);
P1 <= code_ham(0);

S3 <= P3 xor D2 xor D3 xor D4 when parity = '0' else
       not (P3 xor D2 xor D3 xor D4);

S2 <= P2 xor D1 xor D3 xor D4 when parity = '0' else
       not (P2 xor D1 xor D3 xor D4);

S1 <= P1 xor D1 xor D2 xor D4 when parity = '0' else
       not (P1 xor D1 xor D2 xor D4);

code_ori <= D4 & D3 & D2 & D1;
syndrome <= S3 & S2 & S1;

end design;
```

Test bench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Q3_D1_decoder_tst IS
END Q3_D1_decoder_tst;
ARCHITECTURE Q3_D1_decoder_arch OF Q3_D1_decoder_tst IS
```

```
SIGNAL code_ham : STD_LOGIC_VECTOR(6 DOWNTO 0);
SIGNAL code_ori : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL parity : STD_LOGIC;
SIGNAL syndrome : STD_LOGIC_VECTOR(2 DOWNTO 0);
COMPONENT Q3_D1_decoder
	PORT (
	code_ham : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
	code_ori : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
	parity : IN STD_LOGIC;
	syndrome : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
	);
END COMPONENT;
BEGIN
	i1 : Q3_D1_decoder
	PORT MAP (
	code_ham => code_ham,
	code_ori => code_ori,
	parity => parity,
	syndrome => syndrome
	);

t_code_ham: PROCESS
BEGIN
	code_ham <= "0101010";
	WAIT FOR 250000 ps;
	code_ham <= "1000111";
	WAIT FOR 250000 ps;
	code_ham <= "0000000";
	WAIT FOR 250000 ps;
	code_ham <= "1011000";
WAIT;
END PROCESS t_code_ham;

t_parity: PROCESS
BEGIN
LOOP
	parity <= '0';
	WAIT FOR 125000 ps;
	parity <= '1';
	WAIT FOR 125000 ps;
	IF (NOW >= 1000000 ps) THEN WAIT; END IF;
END LOOP;
END PROCESS t_parity;
END Q3_D1_decoder_arch;
```

Test wave:



Design 2:

a.

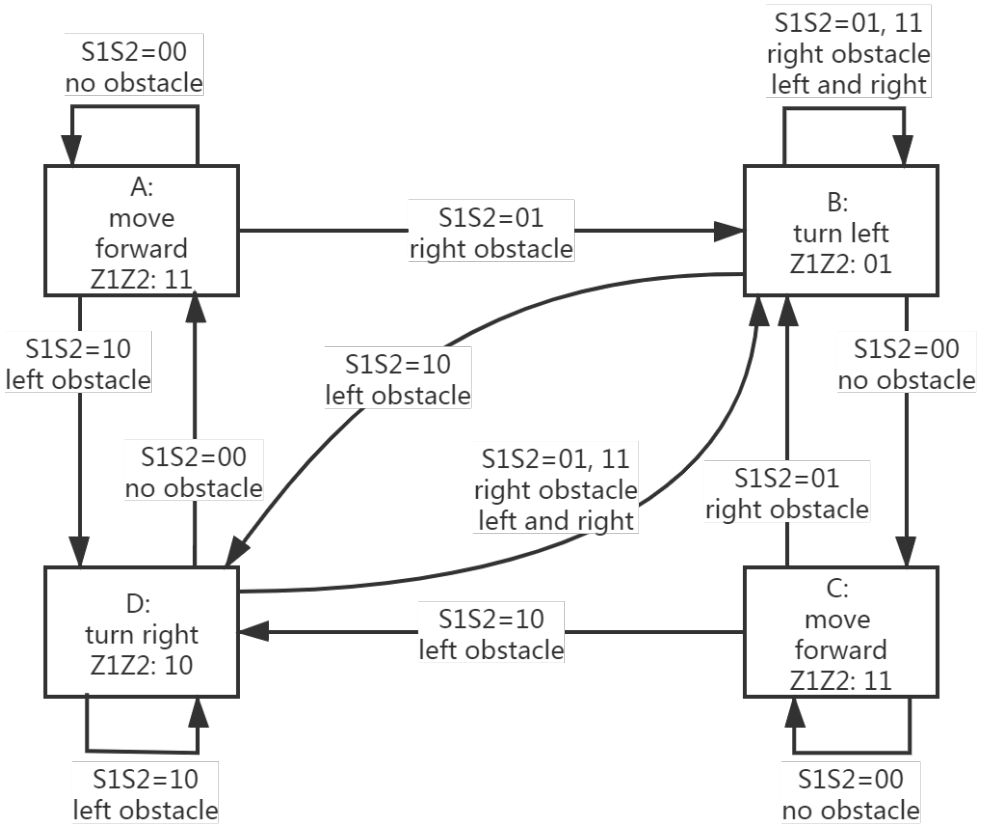
Table for S1S2:

S1S2	meaning
00	No obstacle
01	Right obstacle
10	Left obstacle
11	Both left and right

Table for Z1Z2:

Z1Z2 (state)	meaning
00 (none)	Stop
01 (B)	Turn left
10 (D)	Turn right
11 (A, C)	Move forward

Diagram:



b. code:

```
library ieee;
use ieee.std_logic_1164.all;

entity Robot is
	port (clk	:	in std_logic;
		S1S2	:	in std_logic_vector(1 downto 0);
		Z1Z2	:	out std_logic_vector(1 downto 0));
end Robot;

architecture design of Robot is

type state_type is (A, B, C, D);
signal state: state_type := A;

begin
process ( S1S2, clk)
begin
if (clk'event and clk='1') then
	case state is
		when A =>
			if S1S2 = "00" then
				state <= A;
			elsif S1S2 = "01" then
				state <= B;
			elsif S1S2 = "10" then
				state <= D;
			end if;

		when B =>
			if S1S2 = "00" then
				state <= C;
			elsif S1S2 = "01" then
				state <= B;
			elsif S1S2 = "10" then
				state <= D;
			elsif S1S2 = "11" then
				state <= B;
			end if;

		when C =>
			if S1S2 = "00" then
				state <= C;
			elsif S1S2 = "01" then
				state <= B;
			elsif S1S2 = "10" then
				state <= D;
			end if;
```

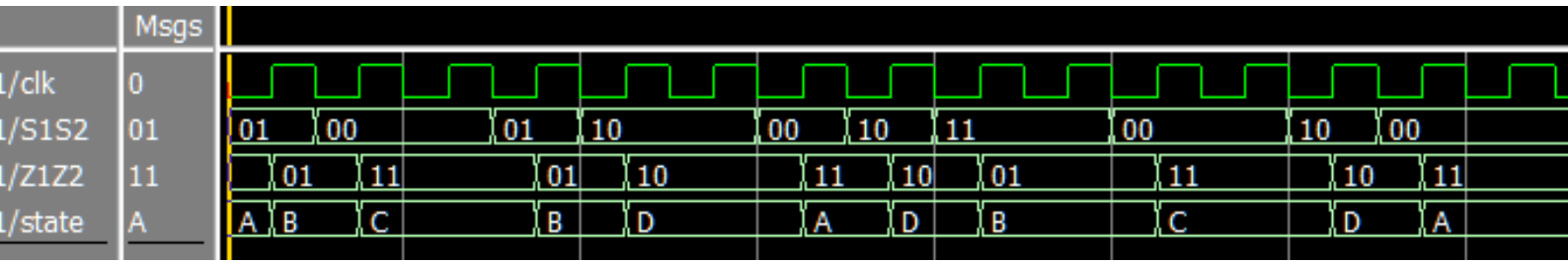
```
                when D =>
                        if S1S2 = "00" then
                                state <= A;
                        elsif S1S2 = "10" then
                                state <= D;
                        elsif S1S2 = "11" then
                                state <= B;
                        end if;
        end case ;
end if;
end process ;

Z1Z2 <= "01" when state = B else
        "10" when state = D else
        "11" when state = A or state = C else
        "00";

end design;
```

Test bench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Robot_vhd_vec_tst IS
END Robot_vhd_vec_tst;
ARCHITECTURE Robot_arch OF Robot_vhd_vec_tst IS

SIGNAL clk : STD_LOGIC;
SIGNAL S1S2 : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL Z1Z2 : STD_LOGIC_VECTOR(1 DOWNTO 0);
COMPONENT Robot
        PORT (
        clk : IN STD_LOGIC;
        S1S2 : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        Z1Z2 : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
        );
END COMPONENT;
BEGIN
        i1 : Robot
        PORT MAP (
        clk => clk,
        S1S2 => S1S2,
        Z1Z2 => Z1Z2
        );

t_clk: PROCESS
BEGIN
LOOP
        clk <= '0';
```

```
        WAIT FOR 25000 ps;
        clk <= '1';
        WAIT FOR 25000 ps;
        IF (NOW >= 1000000 ps) THEN WAIT; END IF;
END LOOP;
END PROCESS t_clk;

t_S1S2: PROCESS
BEGIN
        S1S2 <= "01";
        WAIT FOR 50000 ps;
        S1S2 <= "00";
        WAIT FOR 100000 ps;
        S1S2 <= "01";
        WAIT FOR 50000 ps;
        S1S2 <= "10";
        WAIT FOR 100000 ps;
        S1S2 <= "00";
        WAIT FOR 50000 ps;
        S1S2 <= "10";
        WAIT FOR 50000 ps;
        S1S2 <= "11";
        WAIT FOR 100000 ps;
        S1S2 <= "00";
        WAIT FOR 100000 ps;
        S1S2 <= "10";
        WAIT FOR 50000 ps;
        S1S2 <= "00";
WAIT;
END PROCESS t_S1S2;

END Robot_arch;
```

Test wave:



Q4.

a. A testability design is very important. Although the task of testing one logic gate at a time seems very simple, for today's extremely complex circuit design, most of inner gates are hard to be tested since most gates are deeply embedded whereas the test equipment is only able to connect to the main I/O or some physical test points. If the testability of the circuit is not designed well in advance, with the expansion of the design scale, the test difficulty will increase exponentially, which may eventually lead to unsolvable problems.

b.

Test Vectors				Normal Gate Inputs											Faults Tested
A	B	C	D	e	f	g	h	i	j	k	l	m	p	q	r	
1	1	0	0	1	1	1	1	0	0	0	1	0	1	0	0	e0, f0, g0 p0
1	0	1	X	1	0	0	1	1	1	0	0	X	0	1	0	h0, i0, j0 q0
0	1	1	1	0	1	0	0	0	1	1	1	1	0	0	1	k0, l0, m0 r0
0	0	1	1	0	0	0	0	1	1	1	0	1	0	0	0	h1, l1 p1, q1, r1
1	1	1	1	1	1	0	1	0	1	0	1	1	0	0	0	g1, i1, k1 p1, q1, r1
1	0	0	X	1	0	1	1	1	0	0	0	X	0	0	0	f1, j1 p1, q1, r1
0	1	0	0	0	1	1	0	0	0	1	1	0	0	0	0	e1, m1 p1, q1, r1