DSDL HW4

資管三 朱紹綸 B06705028

1. 001 → 011 → 010 → 110 → 111 → 101 → 100^{5}

(1) Truth table

CBA	C^+	B^+	A^+
000	X	×	×
001	0	1	1
010	1	1	0
011	0	1	0
100	0	0	1
101	1	0	0
110	1	1	1
111	1	0	1

C^+

BA \ C	0	1
00	X	0
01	0	1
11	0	1
10	1	1

B^+

BA \ C	0	1
00	X	0
01	1	0
11	1	0
10	1	1

A^+

BA \ C	0	1
00	X	1
01	1	0
11	0	1
10	0	1

(2) D flip-flops

BA \ C	0	1
00	X	0
01	0	1
11	0	1
10	1	1

$D_C = A'B + AC$

(3) T flip-flop

BA \ C	0	1
00	X	1
01	0	0
11	0	0
10	1	0

$T_C = A'C' + A'B'$

(4) S-R flip-flop

S_B

BA \ C	0	1
00	X	0
01	1	0
11	X	0
10	X	X

R_B

BA \ C	0	1
00	X	X
01	0	X
11	0	1
10	0	0

$S_B = B'C'$

$R_B = AC$

(5) J-K flip-flop

J_A:

BA \ C	0	1
00	X	1
01	X	X
11	X	X
10	0	1

K_A

BA \ C	0	1
00	X	X
01	0	1
11	1	0
10	X	X

$J_A = C$

$K_A = B'C + BC'$

2. (1) $J_1 = X$

$K_1 = (XQ_2')'$

$J_2 = X$

$K_2 = (XQ_1)'$

$Z = X \oplus Q_2'$

$Q_1^+ = J_1 Q_1' + K_1' Q_1$

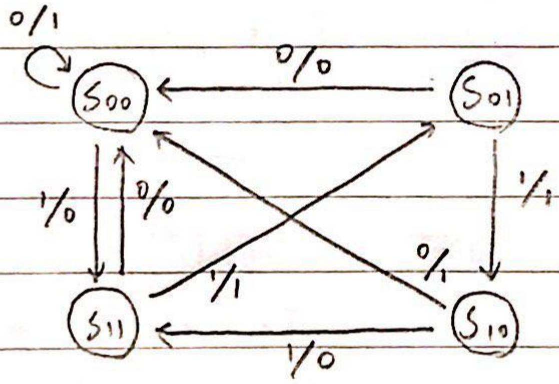
$= XQ_1' + XQ_1Q_2'$

$Q_2^+ = J_2 Q_2' + K_2' Q_2$

$= XQ_2' + XQ_1Q_2$

	$Q_1^+ Q_2^+$		Z	
Q_1Q_2	$X=0$	$X=1$	$X=0$	$X=1$
00	00	11	1	0
01	00	10	0	1
10	00	11	1	0
11	00	01	0	1

(2)



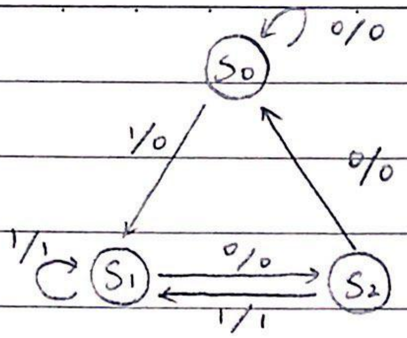
3. (1) invalid BCD encoding : 1010, 1011, 1100, 1101, 1110, 1111

→ 11xx , 101x

S_0 : reset

S_1 : 1xxx

S_2 : 01xx

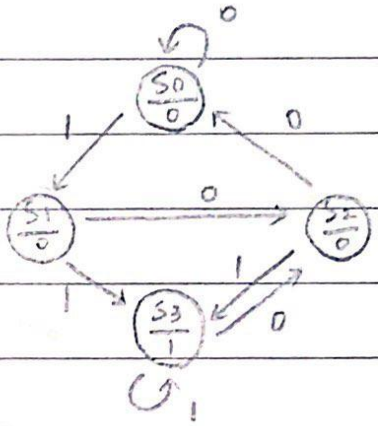


(2) S0 : 00XX

S1 : 100X

S2 : 01XX

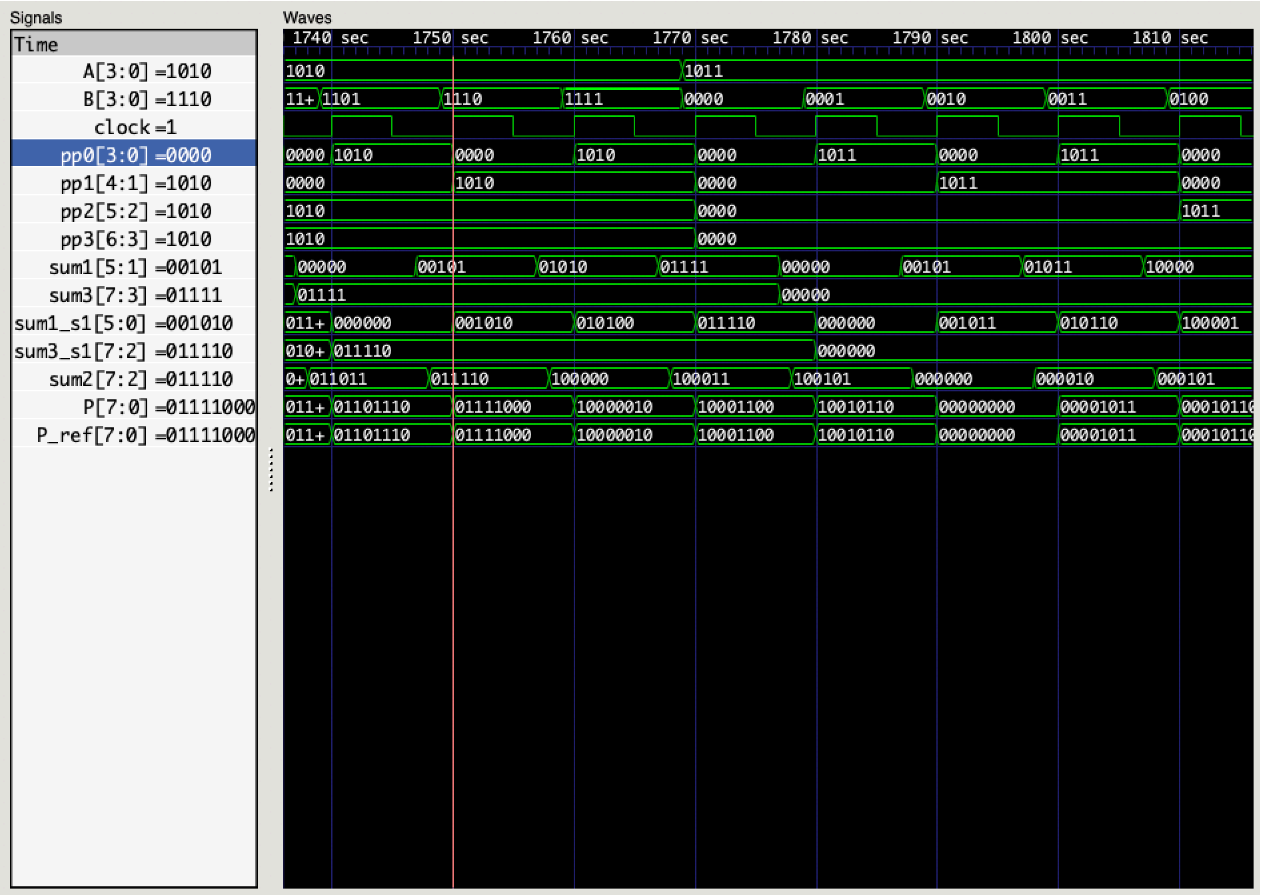
S3 : 11XX or 101X



4 (1)

```
module mult_fast(
  output reg[7:0] P,  // product
  input[3:0] A, B,    // multiplicand and multiplier
  input clk       // clock (posedge)
  );
  // stage 0 (input)
  reg[3:0] a_s0, b_s0;
  always @(posedge clk) begin
    a_s0 <= A;
    b_s0 <= B;
  end
  // stage 1
  wire[3:0] pp0 = a_s0 & {4{b_s0[0]}}; // ignore the delays of AND gates
  wire[4:1] pp1 = a_s0 & {4{b_s0[1]}}; // ignore the delays of AND gates
  wire[5:2] pp2 = a_s0 & {4{b_s0[2]}}; // ignore the delays of AND gates
  wire[6:3] pp3 = a_s0 & {4{b_s0[3]}}; // ignore the delays of AND gates
  reg[5:1] sum1;
  always @(pp0, pp1)
    sum1[5:1] <= #7 pp0[3:1] + pp1[4:1]; // delay of the 4-bit adder
  reg[7:3] sum3;
  always @(pp2, pp3)
    sum3[7:3] <= #7 pp2[5:3] + pp3[6:3]; // delay of the 4-bit adder
  reg[5:0] sum1_s1;
  reg[7:2] sum3_s1;
  always @(posedge clk) begin
    sum1_s1 <= {sum1, pp0[0]};
    sum3_s1 <= {sum3, pp2[2]};
  end
  // stage 2 (outout)
  reg[7:2] sum2;
  always @(sum1_s1, sum3_s1)
    sum2[7:2] <= #8 sum1_s1[5:2] + sum3_s1[7:2]; // delay of the 6-bit adder
  always @(posedge clk) begin
    P <= {sum2, sum1_s1[1:0]};
  end
endmodule
```

4 (2)



Signals

Time

A[3:0] =1010

B[3:0] =1110

clock =1

pp0[3:0] =0000

pp1[4:1] =1010

pp2[5:2] =1010

pp3[6:3] =1010

sum1[5:1] =00101

sum3[7:3] =01111

sum1_s1[5:0] =001010

sum3_s1[7:2] =011110

sum2[7:2] =011110

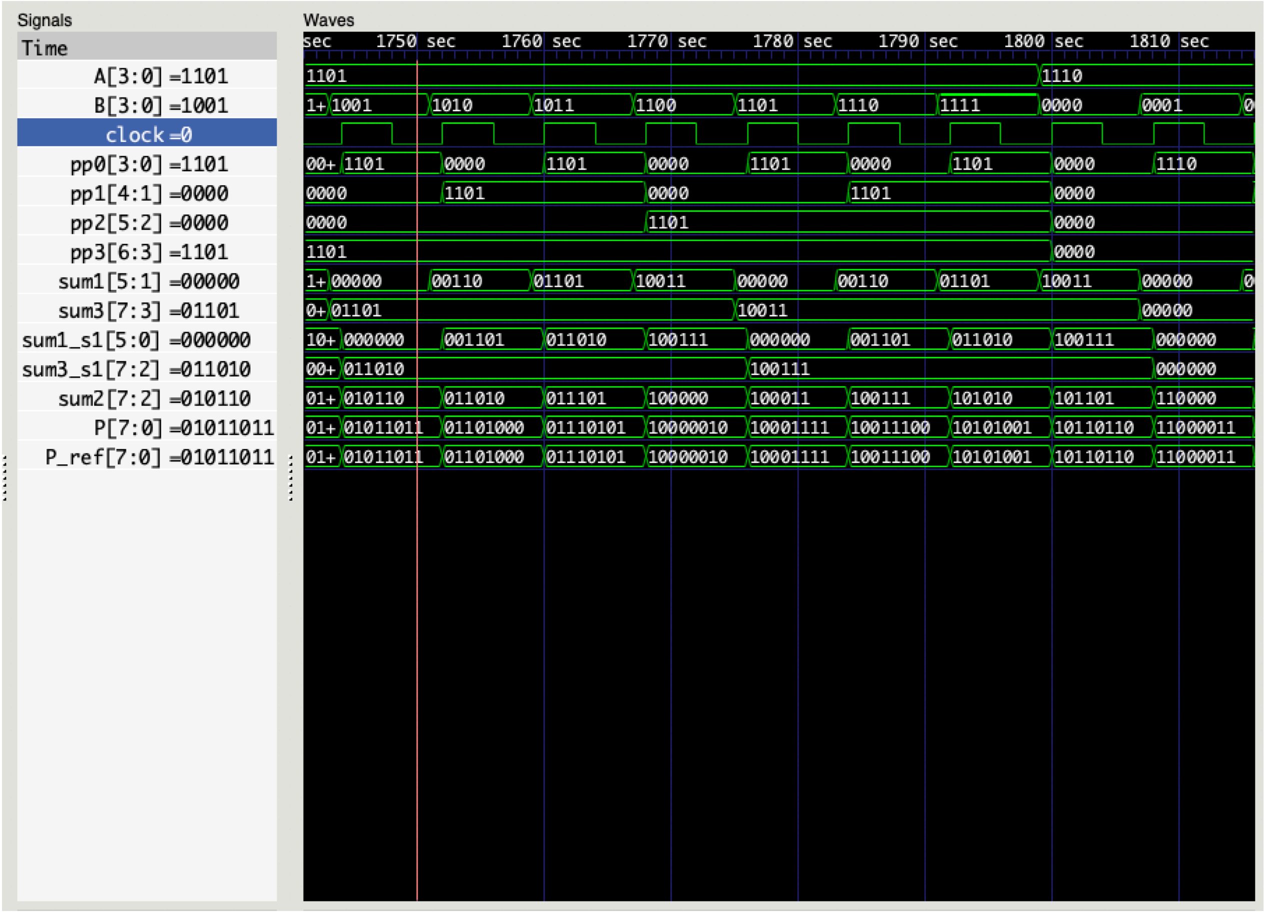
P[7:0] =01111000

P_ref[7:0] =01111000

5(1)

The minimum clock cycle is 8 ticks.

5(2)



Signals

Time

A[3:0] =1101

B[3:0] =1001

clock =0

pp0[3:0] =1101

pp1[4:1] =0000

pp2[5:2] =0000

pp3[6:3] =1101

sum1[5:1] =00000

sum3[7:3] =01101

sum1_s1[5:0] =000000

sum3_s1[7:2] =011010

sum2[7:2] =010110

P[7:0] =01011011

P_ref[7:0] =01011011

6(1)

Assume the clock cycle is 10 microseconds.

A logic operation can be done within 10 microseconds. That is to say $\frac{1}{10\times10^{-6}} = 10^5$ logic operations can be done per second.

Therefore, the throughput is 10^5.

6(2)

With the implementation of `mult_fast`, stage 0 can be done with no delay, stage 1 can be done with a delay of 7 microseconds, and stage 2 can be done with a delay of 8 microseconds.

Though stage 1 can be done with a delay of 7 microseconds, it still takes a clock cycle to move on to stage 2.

Therefore, the latency is $10 + 8 = 18$ microseconds.