1)

ab=00

cd/ef	00	01	11	10
00	0	4 1	12	8 1
01	1	5	13 1	9
11	3	7	15	11
10	2 1	6 1	14 1	10

ab=01

cd/ef	00	01	11	10
00	16	20	28	24
01	17	21	29	25
11	19 1	23	31	27
10	18	22	30	26

ab=10

cd/ef	00	01	11	10
00	32 1	36	44	40 1
01	33	37	45 1	41
11	35	39	47	43
10	34	38	46 1	142

ab=11

cd/ef	00	01	11	10
00	48	52	60	56
01	49	53	61	57
11	51	55	63 1	59
10	50	54	62	58

!a!b!cd!f, !bc!d!e!f, !bcd!ef, !a!b!ce!f, !a!bde!f, !bcde!f, a!b!d!e!f, a!b!c!def, abcdef

Min SoP=!a!b!cd!f+!bc!d!e!f+!bcd!ef+a!b!ce!f+!bcde!f+a!b!d!e!f+a!b!c!de!f+abcdef

2,4,6,8,13,14,19,32,40,45,46,63

2	000010	X
4	000100	X
8	001000	X
32	100000	X
6	000110	X
40	101000	X
13	001101	x
14	001110	x
19	001011	
45	101101	x
46	101110	x
63	111111	

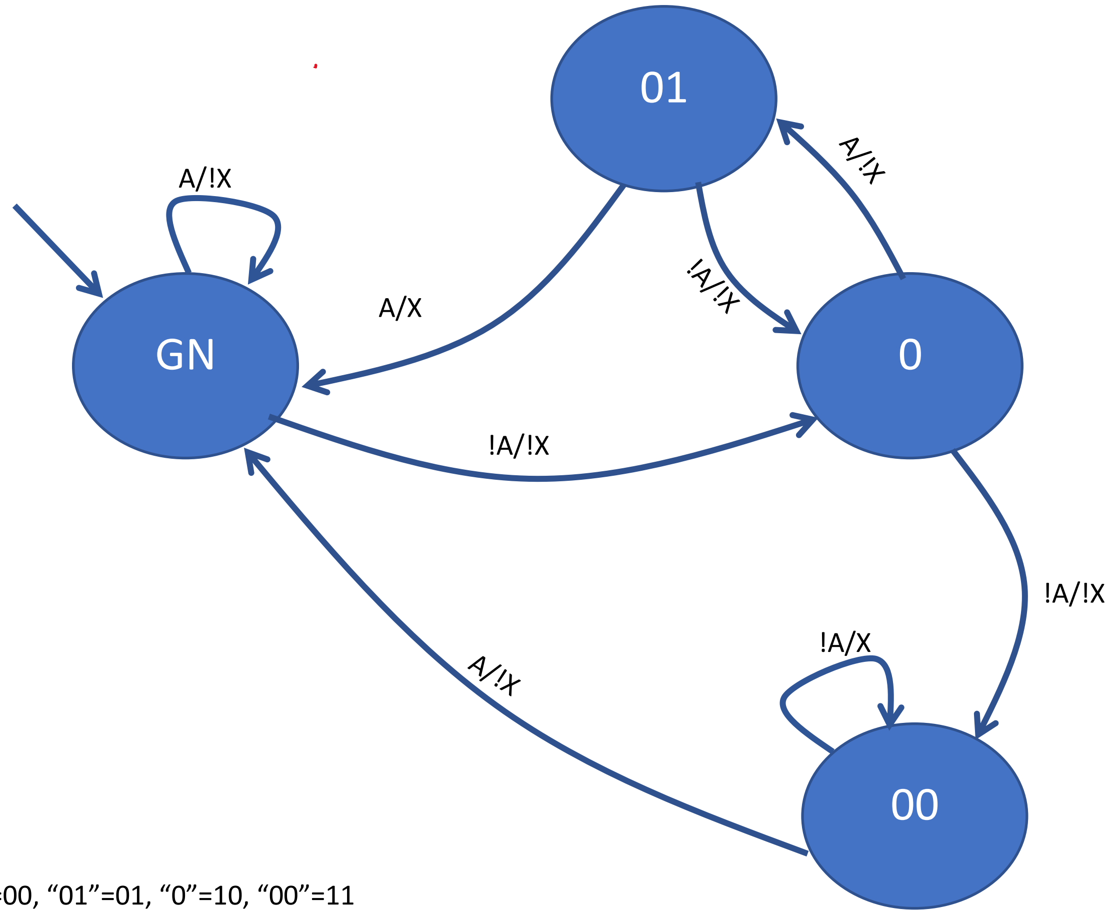
2,6	000x10	
4,6	0001x0	
8,40	X01000	
32,40	10x000	
6,14	00x110	
13,45	X01101	
14,46	X01110	

term	19	63	2,6	4,6	8,40	32,40	6,14	13,45	14,46
2			X*						
4				X*					
8					X*				
32						X*			
6				X			X		
40					X	X			
13								X*	
14							X		X
19	X*								
45								X*	
46									X*
63		X*							

Almost every prime implicant is an essential prime implicant. And the one that isn't isn't in the SoP.

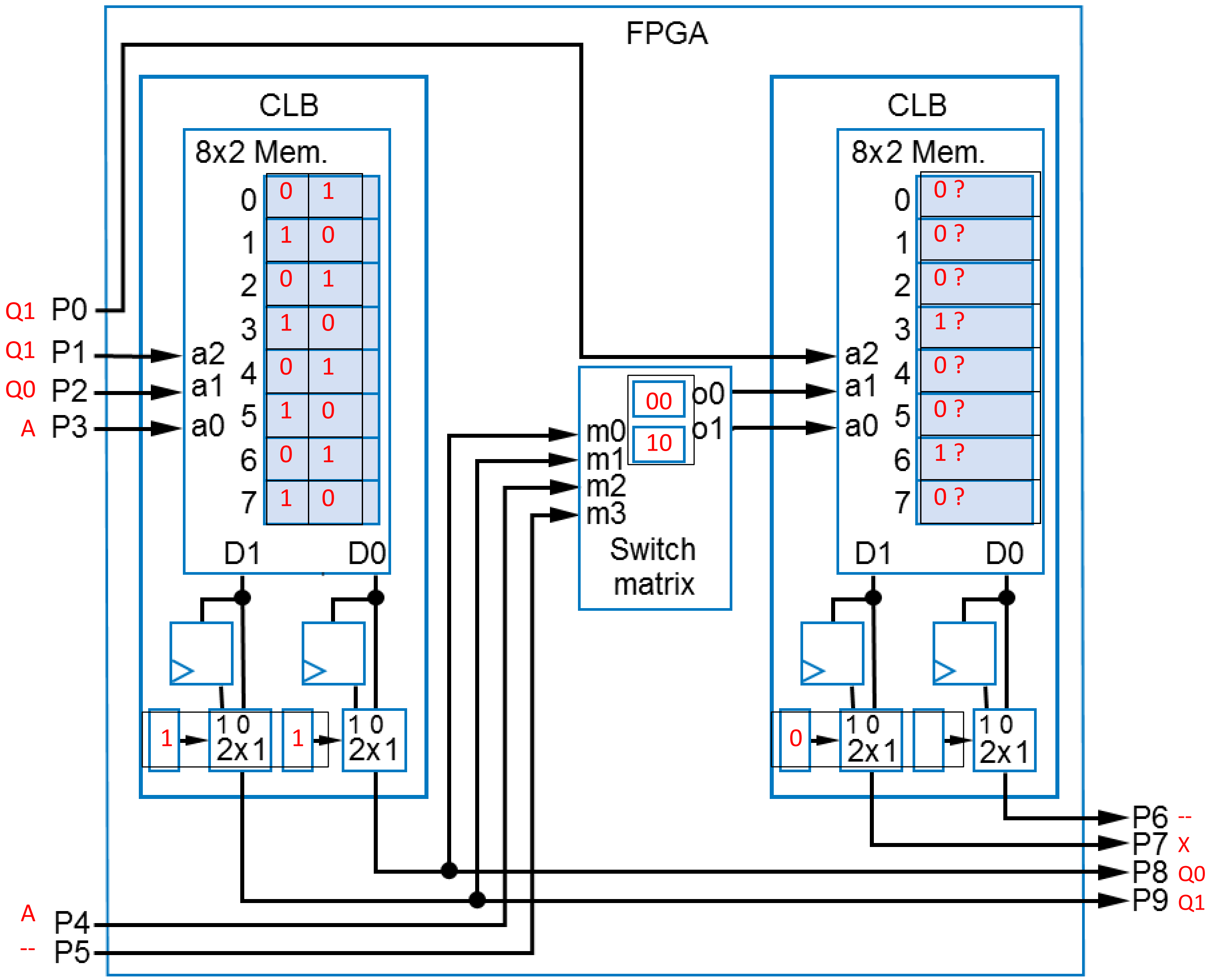
Min SoP= a!b!c!def + abcdef + !a!b!ce!f + !a!b!cd!f + !bc!d!e!f + a!b!d!e!f + !bcd!ef + !bcde!f

3



Using GN=00, “01”=01, “0”=10, “00”=11

Q1	**Q0**	**A**	**N1**	**N0**	**X**
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	0
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	0	0	0


FPGA
CLB
8x2 Mem.
CLB
8x2 Mem.
Q1 P0
Q1 P1
Q0 P2
A P3
a2
a1
a0
D1
D0
m0
m1
m2
m3
Switch matrix
00
10
o0
o1
2x1
P6 --
P7 X
P8 Q0
P9 Q1
A P4
-- P5

4. .

   a.

   - 4-bit: 12 + (2+3+4+5)=26
   - 16-bit: Need to add the logic for P and G to each 4-bit adder. Notice that we already have most of G (all the AND gates) done, so we just need the OR gate for G. So two more gates total. With those two additional that's 28 gates for each of the 4-bit adders. Plus the 14 gates for the CL logic at the next level, that's 28*4+14=126 gates.

     ```
     P = P3P2P1P0
     G = G3 + P3G2 + P3P2G1 + P3P2P1G0
     ```

   - 64-bit: That's 126 gates for each 16-bit adder. Plus the two to find P and G. Plus the 14 for the CL logic gives us 128*4+14 = 526.

   b. .

   - 4-bit: top is only 2-input gates, so 12 gates*2=24 gate-inputs. The CL logic is 4 inputs in stage 1, 8 in 2, 13 in 3, and 19 in stage 4 = 44 gate inputs. So 68 total inputs.
   - 16-bit: Each 4-bit has the 68 inputs plus another 4 for P and 4 for G. So 76*4. Add in the 44 for the CL logic, you've got 348 inputs.
   - 64-bits: Each 16-bit adder has the 348 bits plus another 8 for P and G. Then 44 for the CL logic, you've got 4*(348+8)+44 = 1468 gate inputs.

   c. (reading page 373 gives a fairly good explanation of the logic here).

   - 4-bit has 4 gate-delays (1 for P/G, 2 for CL logic, 1 for the XOR to generate s1).
   - 16-bit: Adding P and G to the 4-bit adder doesn't add any delay. So just an extra CL logic, which gets us to 6 gate-delays
   - 64-bit: Same logic gets us to 8 gate-delays.

   d. This problem was defined a bit too poorly to be graded (log base actually matters for what path is critical). Everyone was given full points if an effort was made.

5. .

There was no specific format given for data values, so anything reasonable was taken. The key is the need for self-modifying code.

```
blt   8     1000  2001  // skip increment if mem<101.
add   2000  2000  2002  // increment total
add   2     2     2002  // modify the address to check!
blt   0     2     2003  // keep going if mem[2]<2000.
halt                    // if branch not ta



2000: 0         //total
2001: 101       //value to compare to
2002: 1         //one
2003: 2000      //if mem[3] isn't this big, keep going.
```