z5209365

Design Approach

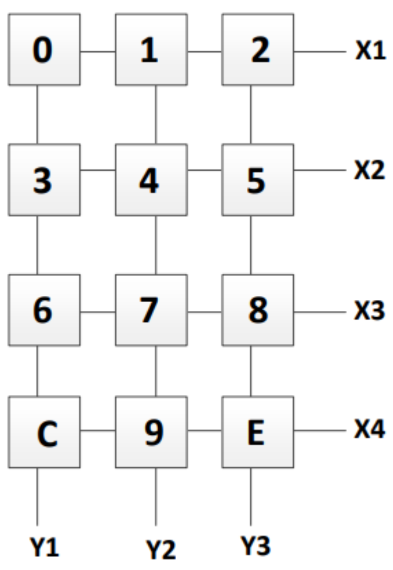
Using a 9-variable k-map to simplify the decoding to seven-segment display. Using just encoders and decoders would have increased the GIC and would not be as optimised. The 2 inputs of X and Y directly correlated with a button pressed on a keypad which means one AND gate is used for each button. After pressed, the signal would immediately light up the seven-segment display.

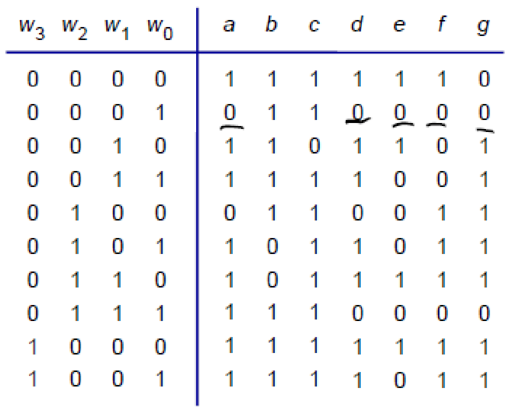
Assumptions:

1. Two buttons may not be pressed at the same time
2. When no button is pressed, nothing is displayed on the seven-segment display
3. LEDs light up when voltage is HI

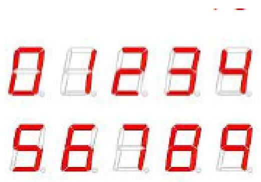
Re-specification:

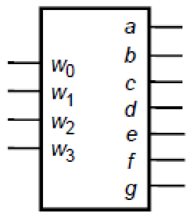
Design a seven-input combinational circuit, (inputs of rows, X1, X2, X3, X4, and columns, Y1, Y2, Y3), which outputs 12 numbers & letters (0 to 9, C & E) into a seven-segment display. Only one button is pressed at a time. When no buttons are pressed, nothing is displayed. HI turns on an LED.

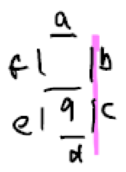




w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1







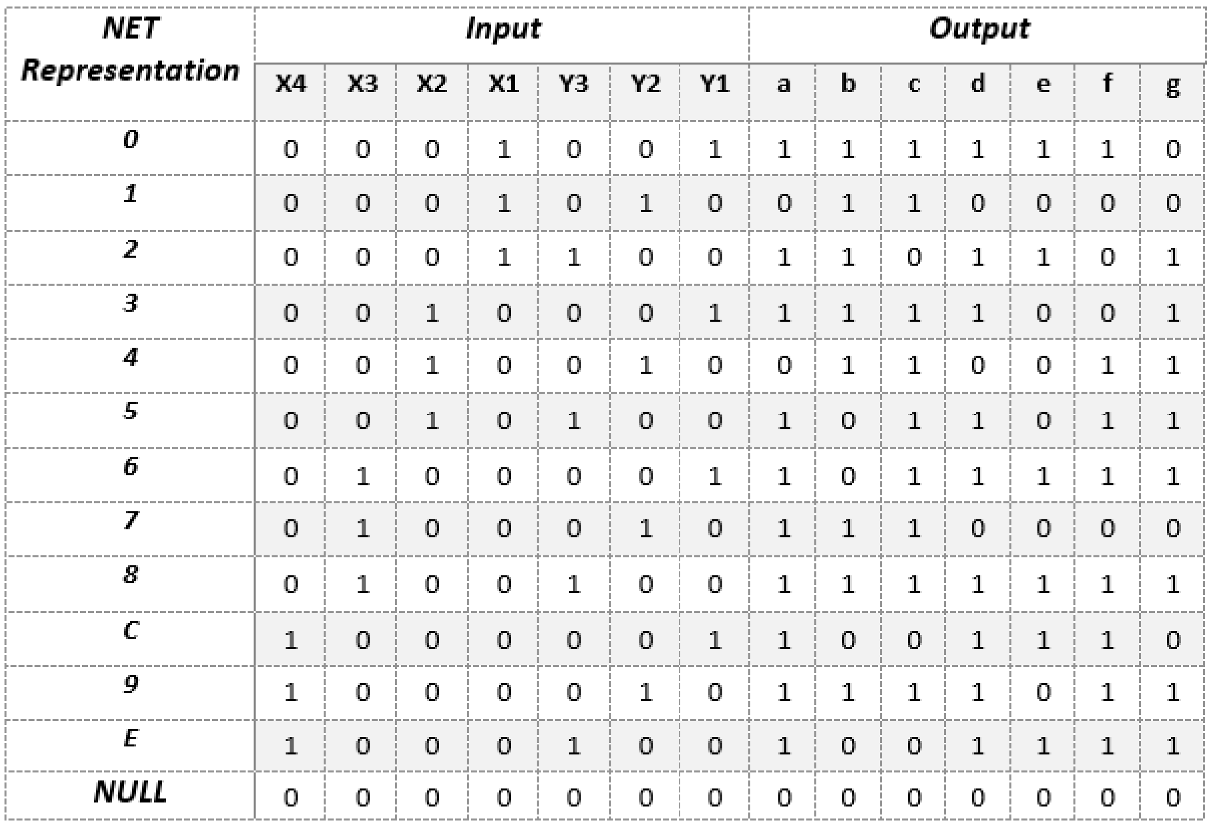
Formulation

Truth Table

This table was formulated on the assumption that:

1. Two buttons may not be pressed at the same time
2. LEDs light up when voltage is HI and LO results in no light

All other minterms with more than one button pressed were don't care conditions and therefore left out to simplify the truth table. They are left in when used in the k-mapping.



NET Representation	***Input***							***Output***						
	X4	**X3**	**X2**	**X1**	**Y3**	**Y2**	**Y1**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0	1	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	0	1	0	0	1	1	0	0	0	0
2	0	0	0	1	1	0	0	1	1	0	1	1	0	1
3	0	0	1	0	0	0	1	1	1	1	1	0	0	1
4	0	0	1	0	0	1	0	0	1	1	0	0	1	1
5	0	0	1	0	1	0	0	1	0	1	1	0	1	1
6	0	1	0	0	0	0	1	1	0	1	1	1	1	1
7	0	1	0	0	0	1	0	1	1	1	0	0	0	0
8	0	1	0	0	1	0	0	1	1	1	1	1	1	1
C	1	0	0	0	0	0	1	1	0	0	1	1	1	0
9	1	0	0	0	0	1	0	1	1	1	1	0	1	1
E	1	0	0	0	1	0	0	1	0	0	1	1	1	1
NULL	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Optimisation

An online 7-variable k-map generator was used to find optimal solutions. (src: https://charlie-coleman.com/experiments/kmap/). Below are screenshots of the solution and working out and a summary of the essential prime indicates that were selected and substitution equations made for my optimised Boolean expression:

Essential Prime Indicates:

A: Y1, Y2X2’X1’, Y3
B: Y1X4’X3’, Y2, Y3X4’X2’
C: Y1X4’, Y2, Y3X4’X1’
D: Y1, Y2X3’X2’X1’, Y3
E: Y1X2’, Y3X2’
F: Y1X2’, Y2X3’X1’, Y3X1’
G: Y1X4’X1’, Y2X3’X1’, Y3

Boolean expressions:

Let…

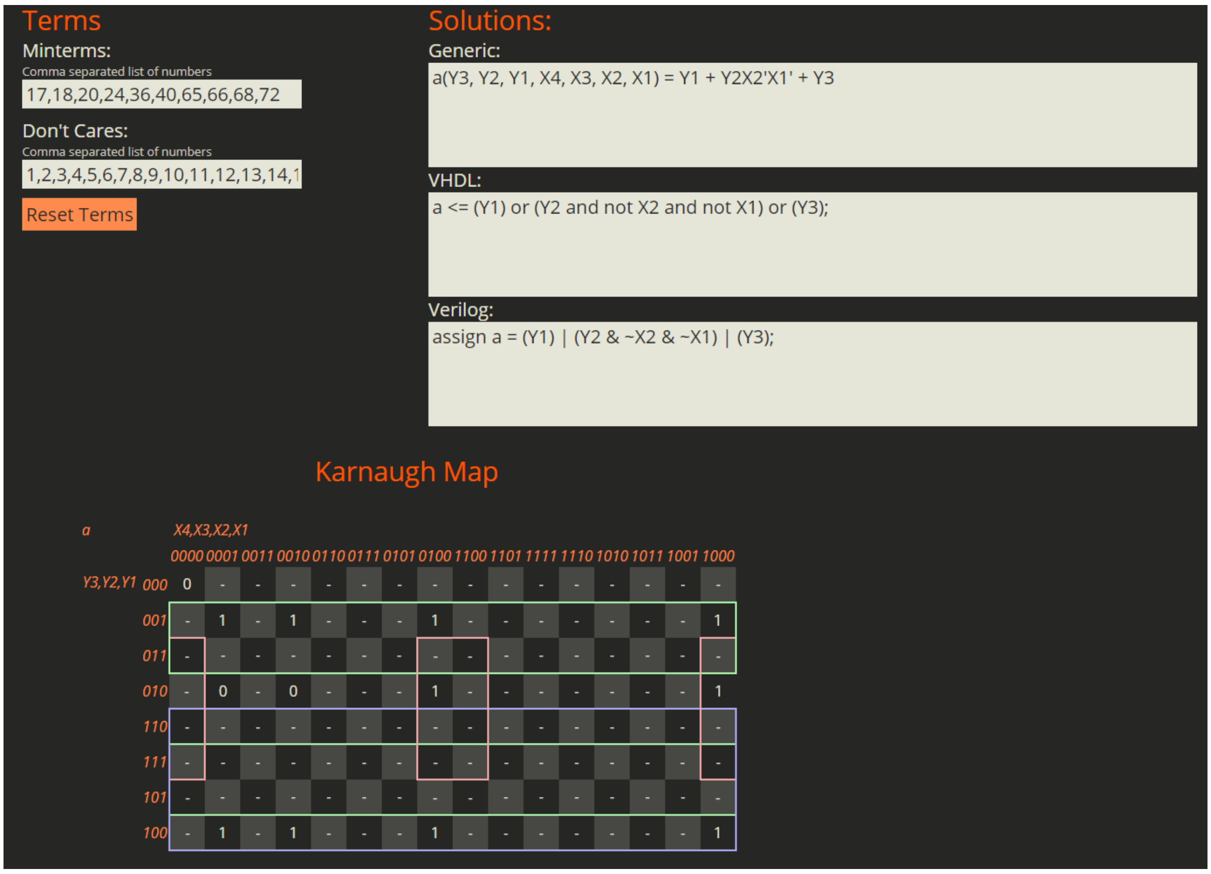
X2’X1’ = m
Y1X4’ = n
Y3X4’ = o
Y1X2’ = p
Y2X3’X1’ = q

A = Y1 + Y2m + Y3
B = nX3’ + Y2 + oX2’
C = n + Y2 + oX1’
D = Y1 + Y2X3’m + Y3
E = p + Y3X2’
F = p’ + q + Y3X1’
G = Y1X4’X1’ + q + Y3

Using substitution and the algebraic method, the total GIC was 53.

z5209365

A



Terms

Minterms:

Comma separated list of numbers

17,18,20,24,36,40,65,66,68,72

Don't Cares:

Comma separated list of numbers

1,2,3,4,5,6,7,8,9,10,11,12,13,14,1

Reset Terms

Solutions:

Generic:

a(Y3, Y2, Y1, X4, X3, X2, X1) = Y1 + Y2X2'X1' + Y3

VHDL:

```
a <= (Y1) or (Y2 and not X2 and not X1) or (Y3);
```

Verilog:

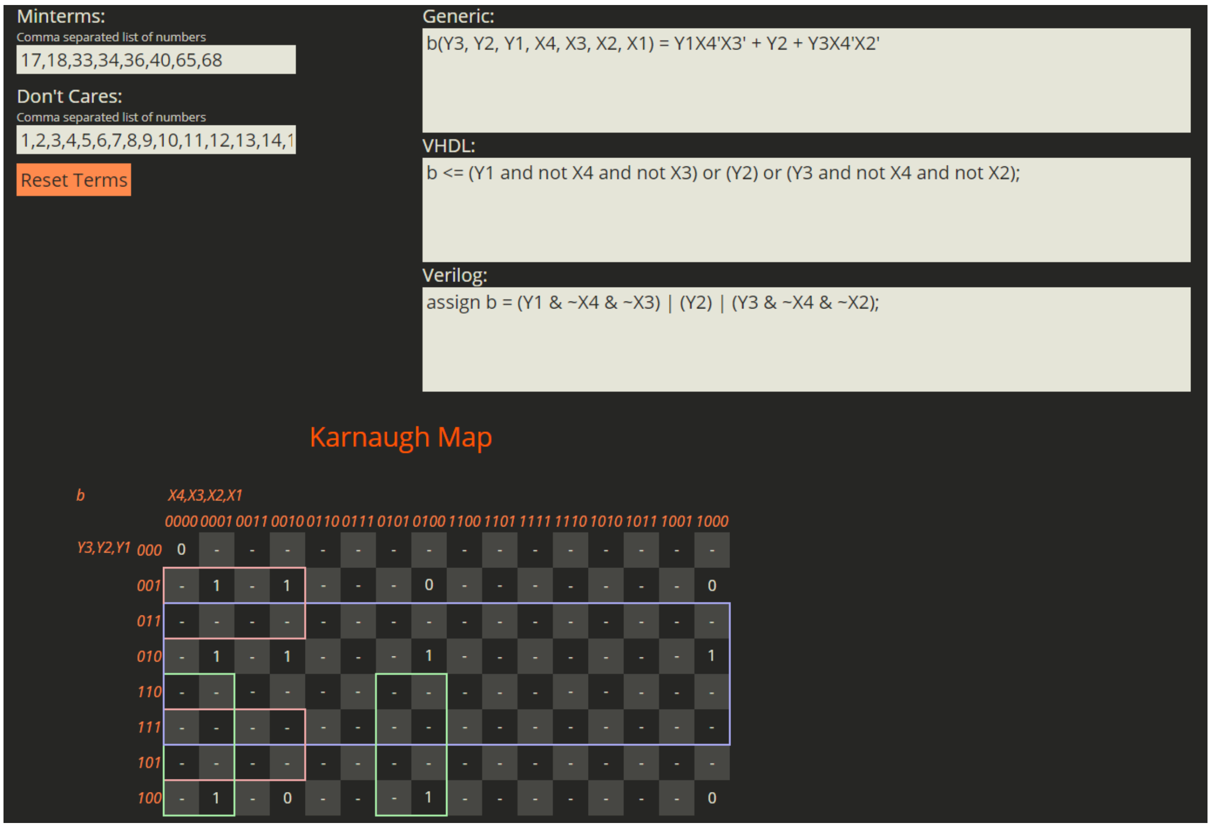
```
assign a = (Y1) | (Y2 & ~X2 & ~X1) | (Y3);
```

Karnaugh Map

a X4,X3,X2,X1

Y3,Y2,Y1	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
001	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-	1
011	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
010	-	0	-	0	-	-	-	1	-	-	-	-	-	-	-	1
110	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
111	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
101	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
100	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-	1

B



Minterms:

Comma separated list of numbers

17,18,33,34,36,40,65,68

Don't Cares:

Comma separated list of numbers

1,2,3,4,5,6,7,8,9,10,11,12,13,14,1

Reset Terms

Generic:

b(Y3, Y2, Y1, X4, X3, X2, X1) = Y1X4'X3' + Y2 + Y3X4'X2'

VHDL:

```
b <= (Y1 and not X4 and not X3) or (Y2) or (Y3 and not X4 and not X2);
```

Verilog:

```
assign b = (Y1 & ~X4 & ~X3) | (Y2) | (Y3 & ~X4 & ~X2);
```

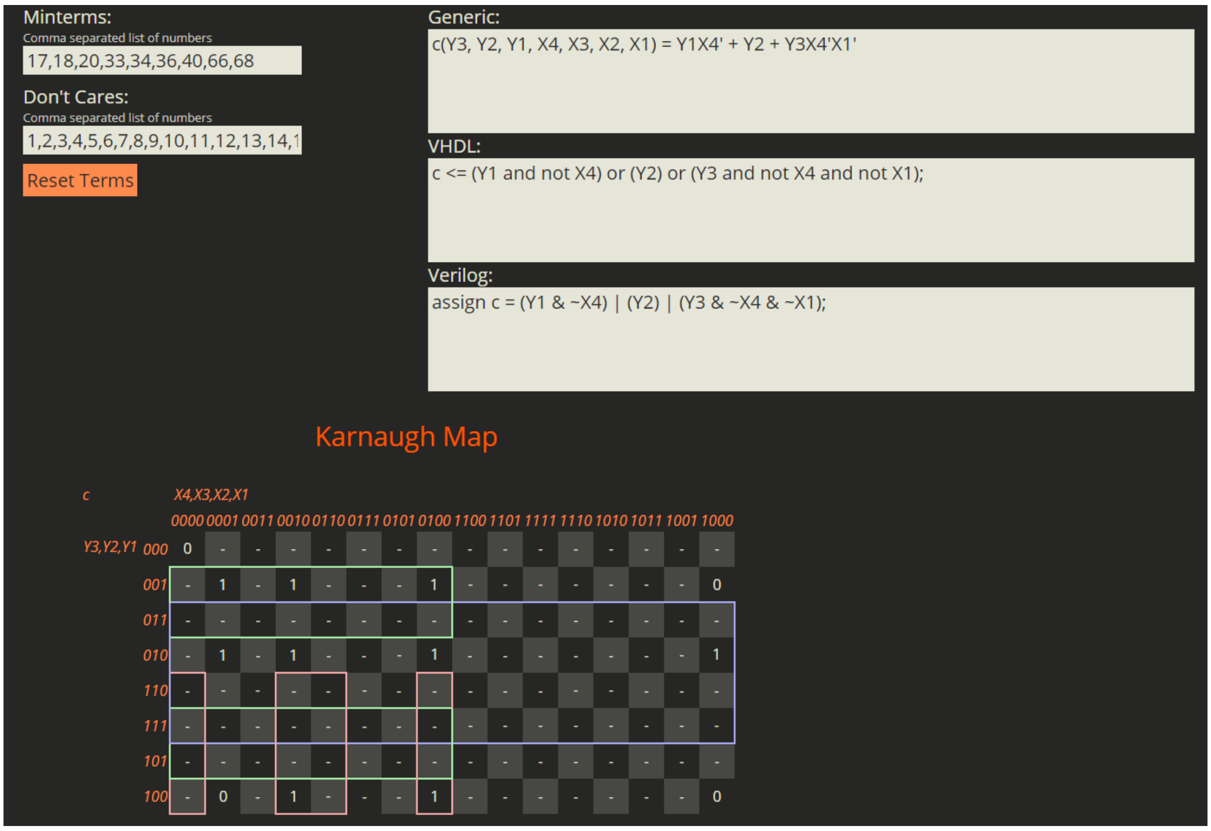
Karnaugh Map

b X4,X3,X2,X1

Y3,Y2,Y1	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
001	-	1	-	1	-	-	-	0	-	-	-	-	-	-	-	0
011	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
010	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-	1
110	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
111	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
101	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
100	-	1	-	0	-	-	-	1	-	-	-	-	-	-	-	0

z5209365

C



Minterms:
Comma separated list of numbers
17,18,20,33,34,36,40,66,68

Don't Cares:
Comma separated list of numbers
1,2,3,4,5,6,7,8,9,10,11,12,13,14,1

Reset Terms

Generic:
```
c(Y3, Y2, Y1, X4, X3, X2, X1) = Y1X4' + Y2 + Y3X4'X1'
```

VHDL:
```
c <= (Y1 and not X4) or (Y2) or (Y3 and not X4 and not X1);
```

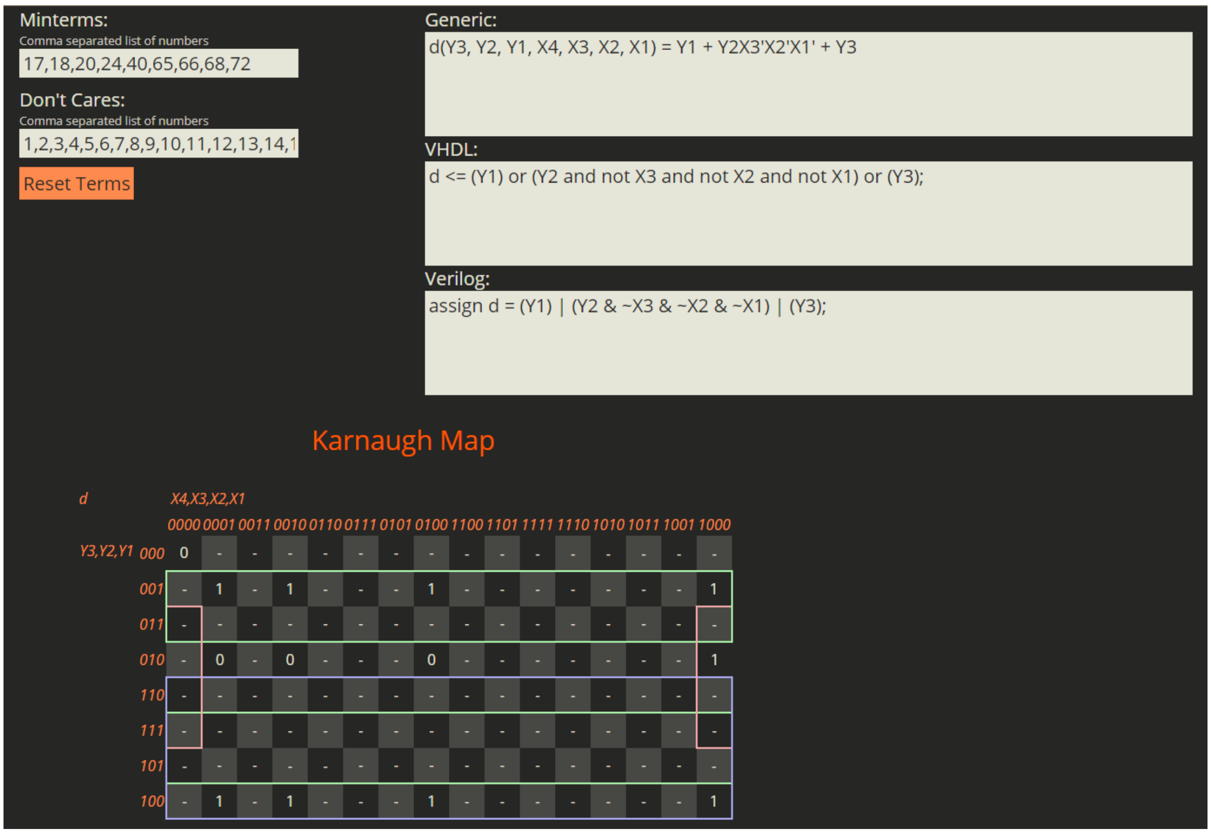
Verilog:
```
assign c = (Y1 & ~X4) | (Y2) | (Y3 & ~X4 & ~X1);
```

Karnaugh Map

c X4,X3,X2,X1

Y3,Y2,Y1	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
001	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-	0
011	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
010	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-	1
110	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
111	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
101	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
100	-	0	-	1	-	-	-	1	-	-	-	-	-	-	-	0

D



Minterms:
Comma separated list of numbers
17,18,20,24,40,65,66,68,72

Don't Cares:
Comma separated list of numbers
1,2,3,4,5,6,7,8,9,10,11,12,13,14,1

Reset Terms

Generic:
```
d(Y3, Y2, Y1, X4, X3, X2, X1) = Y1 + Y2X3'X2'X1' + Y3
```

VHDL:
```
d <= (Y1) or (Y2 and not X3 and not X2 and not X1) or (Y3);
```

Verilog:
```
assign d = (Y1) | (Y2 & ~X3 & ~X2 & ~X1) | (Y3);
```

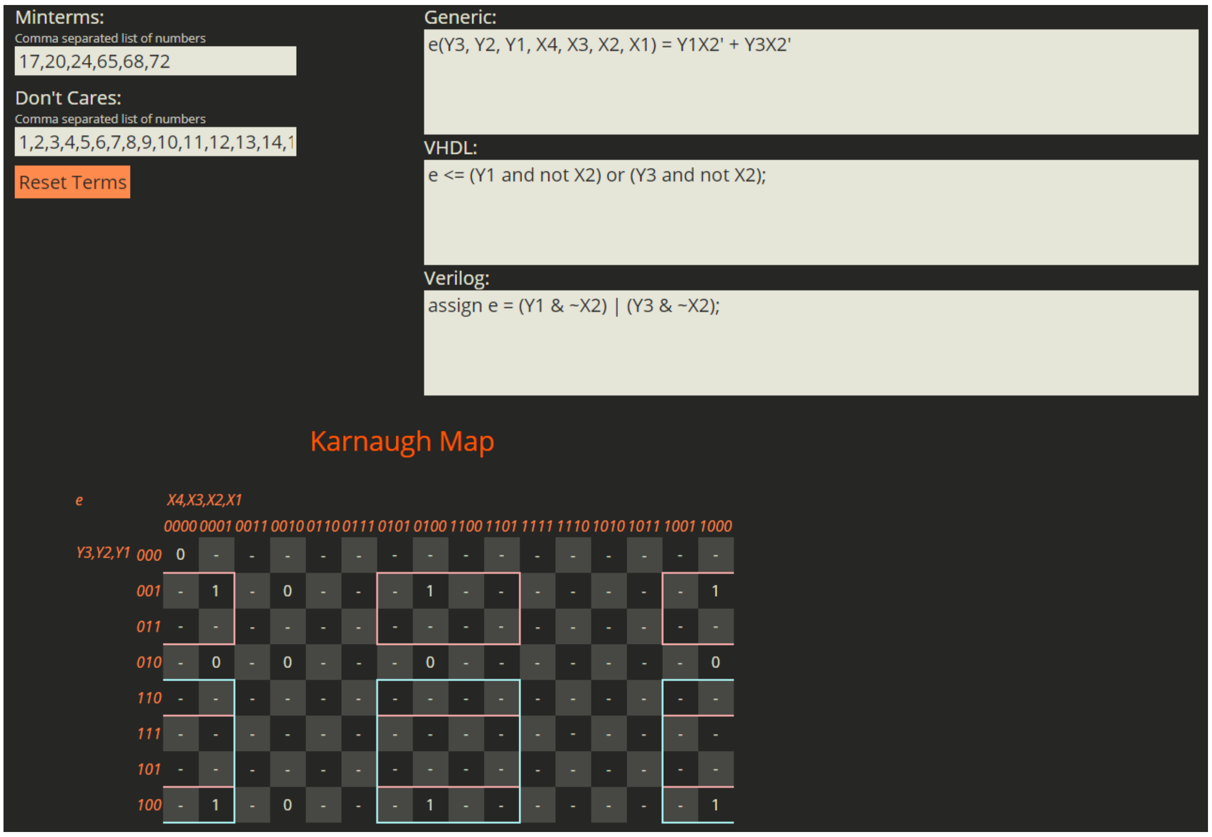
Karnaugh Map

d X4,X3,X2,X1

Y3,Y2,Y1	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
001	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-	1
011	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
010	-	0	-	0	-	-	-	0	-	-	-	-	-	-	-	1
110	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
111	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
101	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
100	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-	1

E

z5209365



Minterms:
Comma separated list of numbers
17,20,24,65,68,72

Don't Cares:
Comma separated list of numbers
1,2,3,4,5,6,7,8,9,10,11,12,13,14,1

Reset Terms

Generic:
e(Y3, Y2, Y1, X4, X3, X2, X1) = Y1X2' + Y3X2'

VHDL:
```
e <= (Y1 and not X2) or (Y3 and not X2);
```

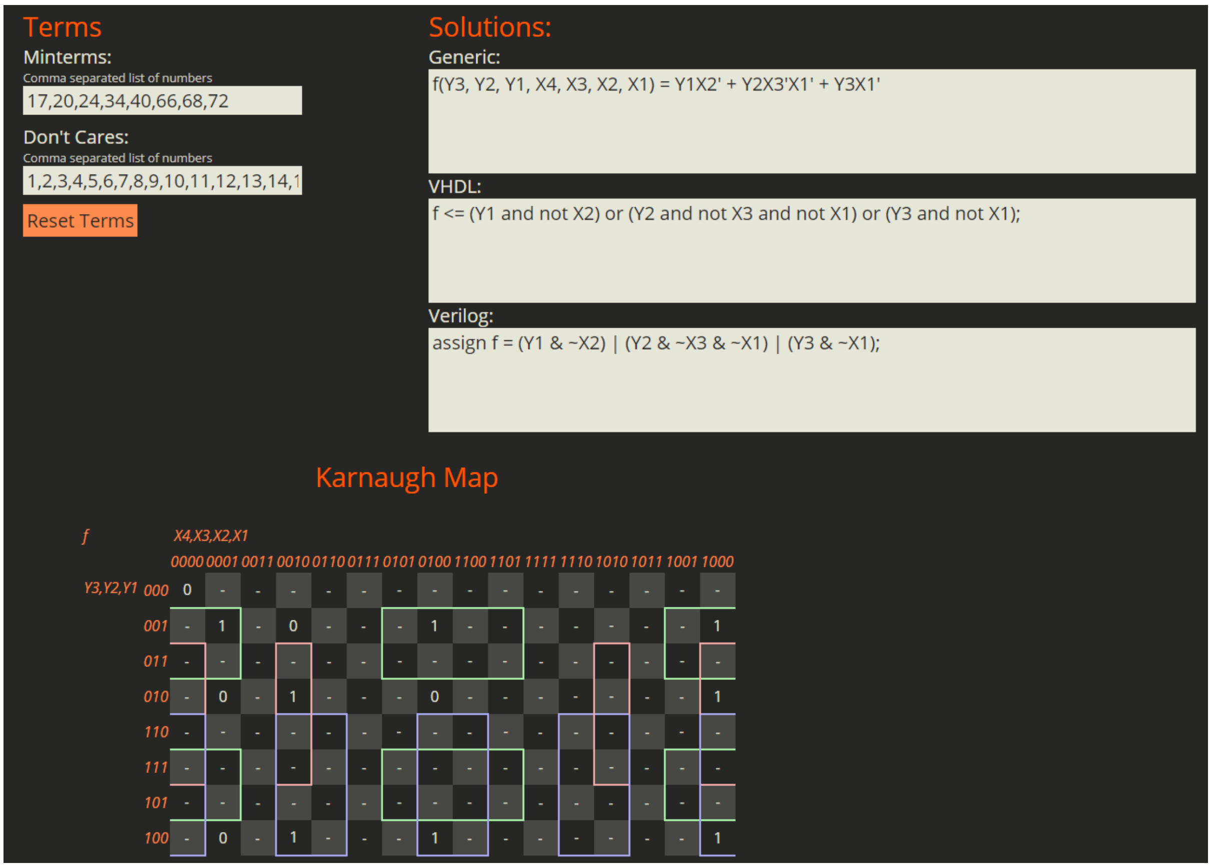
Verilog:
```
assign e = (Y1 & ~X2) | (Y3 & ~X2);
```

Karnaugh Map

e X4,X3,X2,X1

Y3,Y2,Y1	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
001	-	1	-	0	-	-	-	1	-	-	-	-	-	-	-	1
011	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
010	-	0	-	0	-	-	-	0	-	-	-	-	-	-	-	0
110	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
111	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
101	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
100	-	1	-	0	-	-	-	1	-	-	-	-	-	-	-	1

F



Terms

Minterms:
Comma separated list of numbers
17,20,24,34,40,66,68,72

Don't Cares:
Comma separated list of numbers
1,2,3,4,5,6,7,8,9,10,11,12,13,14,1

Reset Terms

Solutions:

Generic:
f(Y3, Y2, Y1, X4, X3, X2, X1) = Y1X2' + Y2X3'X1' + Y3X1'

VHDL:
```
f <= (Y1 and not X2) or (Y2 and not X3 and not X1) or (Y3 and not X1);
```

Verilog:
```
assign f = (Y1 & ~X2) | (Y2 & ~X3 & ~X1) | (Y3 & ~X1);
```

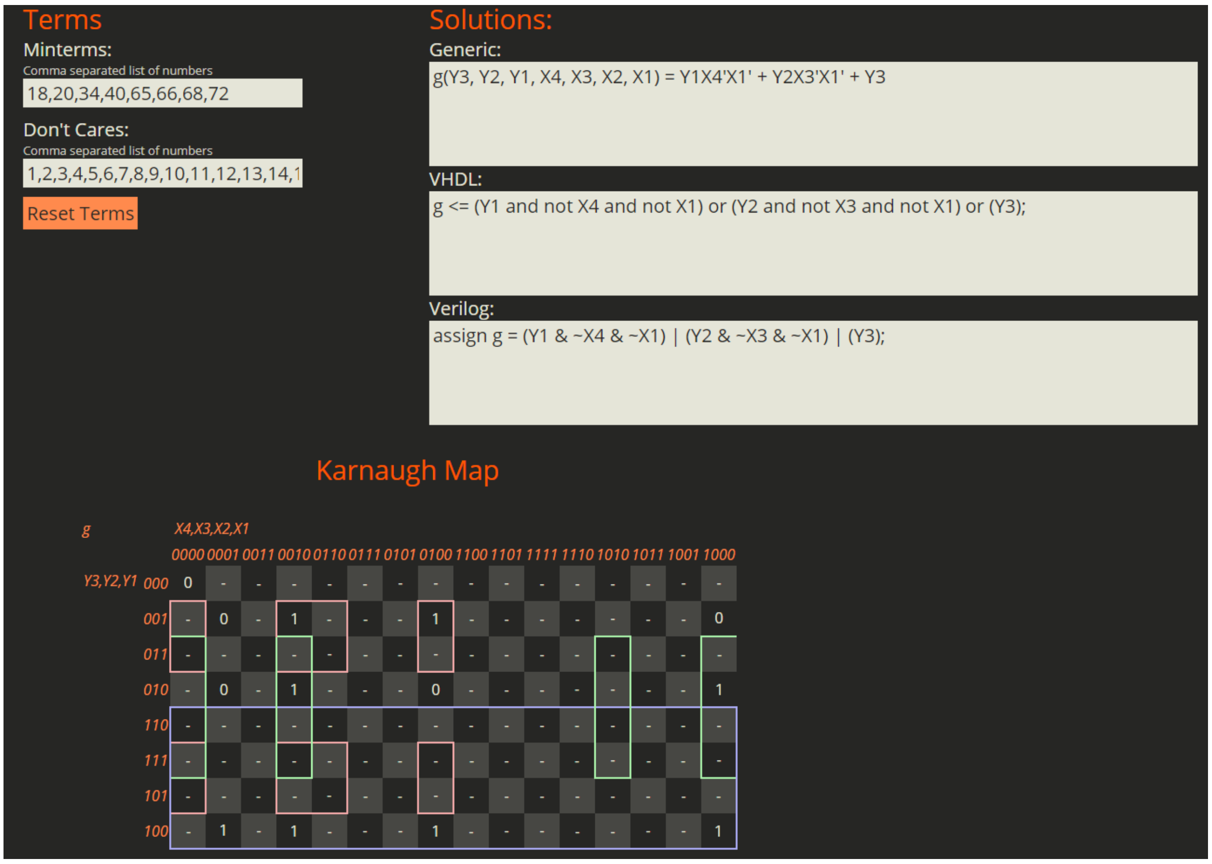
Karnaugh Map

f X4,X3,X2,X1

Y3,Y2,Y1	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
001	-	1	-	0	-	-	-	1	-	-	-	-	-	-	-	1
011	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
010	-	0	-	1	-	-	-	0	-	-	-	-	-	-	-	1
110	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
111	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
101	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
100	-	0	-	1	-	-	-	1	-	-	-	-	-	-	-	1

G

z5209365



Terms

Minterms:

Comma separated list of numbers

18,20,34,40,65,66,68,72

Don't Cares:

Comma separated list of numbers

1,2,3,4,5,6,7,8,9,10,11,12,13,14,1

Reset Terms

Solutions:

Generic:

g(Y3, Y2, Y1, X4, X3, X2, X1) = Y1X4'X1' + Y2X3'X1' + Y3

VHDL:

```
g <= (Y1 and not X4 and not X1) or (Y2 and not X3 and not X1) or (Y3);
```

Verilog:

```
assign g = (Y1 & ~X4 & ~X1) | (Y2 & ~X3 & ~X1) | (Y3);
```

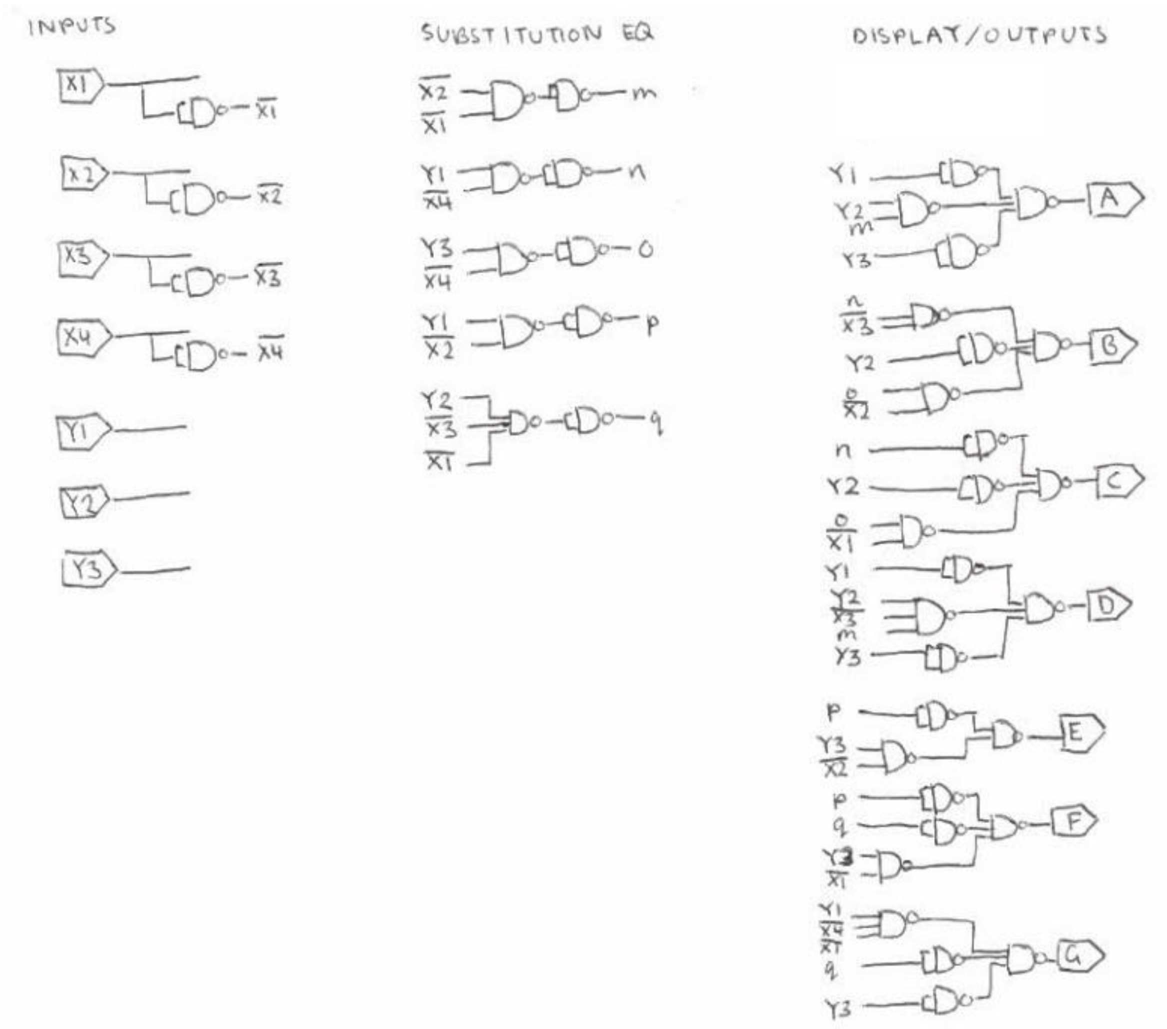
Karnaugh Map

g

Y3,Y2,Y1 \ X4,X3,X2,X1	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
000	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
001	-	0	-	1	-	-	-	1	-	-	-	-	-	-	-	0
011	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
010	-	0	-	1	-	-	-	0	-	-	-	-	-	-	-	1
110	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
111	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
101	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
100	-	1	-	1	-	-	-	1	-	-	-	-	-	-	-	1

Circuit Implementation

NAND only because there are more AND gates which makes it cheaper than OR since AND = 2 NAND and OR = 3 NANDS. NANDs and NORs are economically cheaper and easier to resource. They are also used for all other gate types because they are universal.

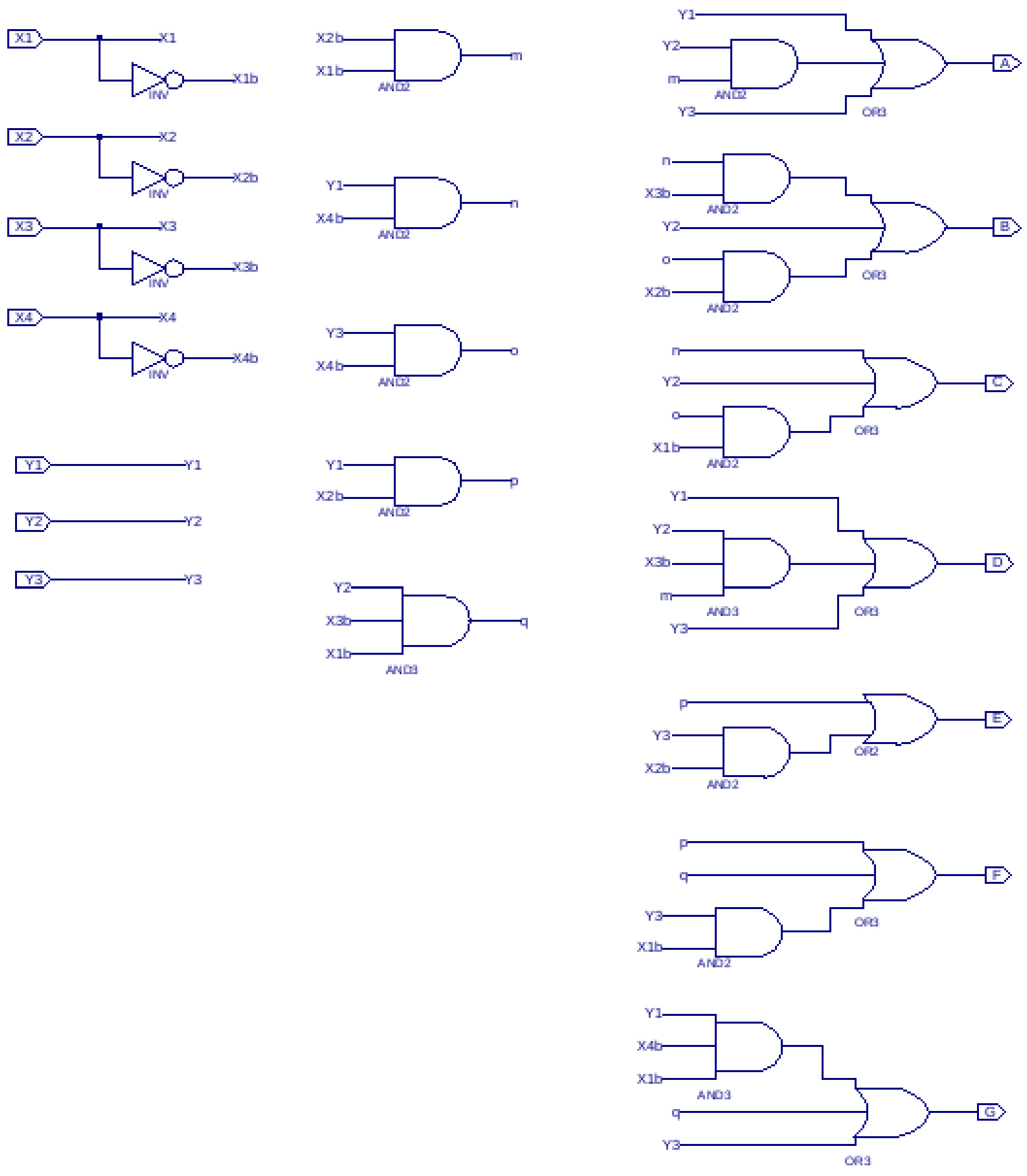


z5209365

Verification

See attached files in ZIP format for Verilog test files and ISim files, which can also be previewed very briefly below.

Schematics

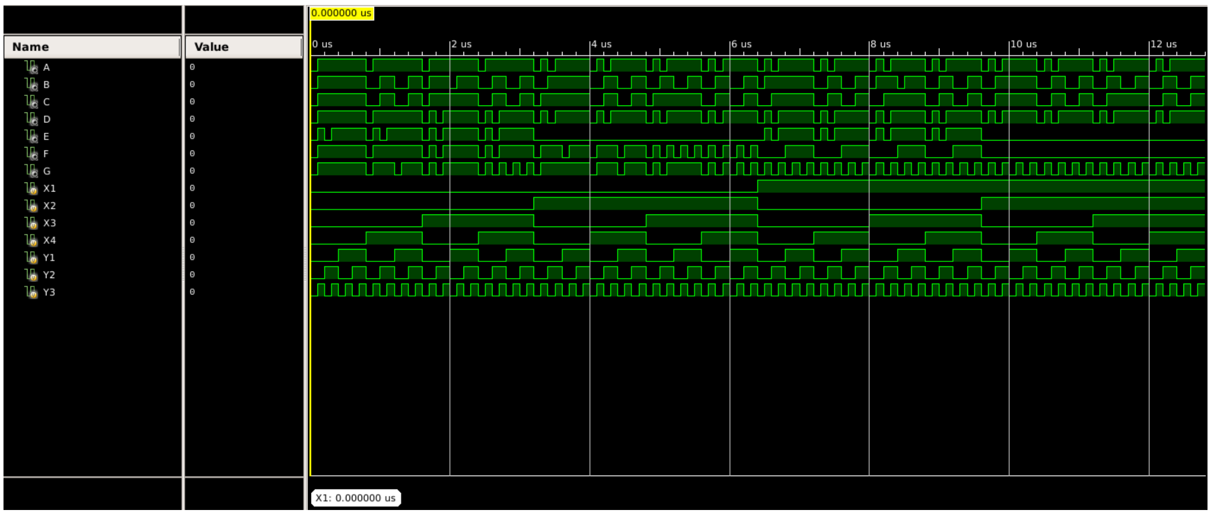


Verilog Test File

```verilog
// Initialize Inputs
//   `ifdef auto_init
       initial begin
         X1 = 1'b0;
         X2 = 1'b0;
         X3 = 1'b0;
         X4 = 1'b0;
         Y1 = 1'b0;
         Y2 = 1'b0;
         Y3 = 1'b0;
      forever begin
         #100;
         {X1,X2,X3,X4,Y1,Y2,Y3} = {X1,X2,X3,X4,Y1,Y2,Y3} + 1;
         end
      end

//   `endif
endmodule
```

Simulation File



Part 2

Design Approach

The design approach is by receiving PUMP Boolean function from specification, then using 4 variable k-maps to receive FLOW and then 3 variable k-maps to receive DISPLAY2. DISPLAY1 is automatically received from PUMP. Hierarchical design with functional blocks are also used to make the design simpler. This is because each functional block of PUMP, FLOW, DISPLAY1 and DISPLAY2 each have a few amount of inputs and outputs which can be linked to other outputs and circuitry. Moreover their functions are each unique to their own block.

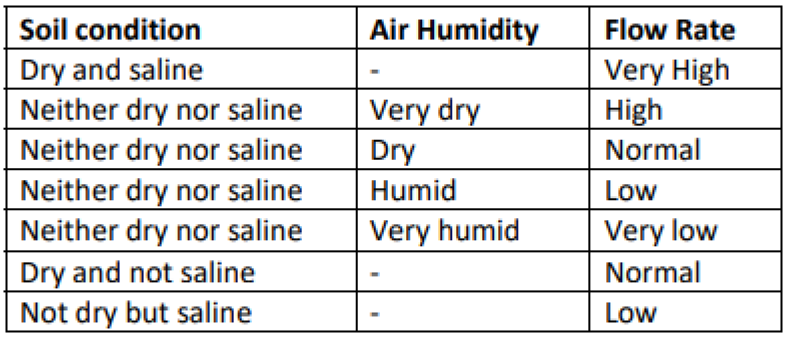
Assumptions:

- HI turns on the LED and LO shows no light for a single LED on the display
- When it is raining, it is on the OFF condition

Re-specification:

7 inputs of 1-bit CLOCK, TSWITCH, SALINE, DRY, RAIN and 2-bit HUMIDITY are used to create a pumping system that controls flowrate and display its status. From the specifications, HUMIDITY's bits represent the following: 00: Very dry 01: Dry 10: Humid 11: Very humid. Other inputs are self-explanatory in that when they are on, it is the set condition e.g. dry, raining.

There are 4 outputs, 1-bit PUMP, 3-bit FLOW, and 7-bit DISPLAY1 and DISPLAY2. When (CLK + TSWITCH + SAL.DRY).(RAIN'), PUMP is on. FLOW is 3-bits and represents the following conditions from given specifications:



Soil condition	Air Humidity	Flow Rate
Dry and saline	-	Very High
Neither dry nor saline	Very dry	High
Neither dry nor saline	Dry	Normal
Neither dry nor saline	Humid	Low
Neither dry nor saline	Very humid	Very low
Dry and not saline	-	Normal
Not dry but saline	-	Low

DISPLAY1 is a seven-segment display, along with DISPLAY 2. DISPLAY1 shows S when PUMP is off and P when PUMP is on. DISPLAY2 represents flow with numbers according to the following: 0: No flow 3: Normal 1: Very low 4: High 2: Low 5: Very high.

Formulation

There are 7 bits of input, 1-bit CLOCK, TSWITCH, SALINE, DRY, RAIN, and 2-bit HUMIDITY. There are four outputs of 1-bit PUMP, 3-bit FLOW, 7-bit DISPLAY1 and DISPLAY2. DISPLAY1 relies on the output, PUMP, and DISPLAY2 relies on FLOW. FLOW relies on PUMP to act as an enabler which means PUMP must be formulated first.

1-bit PUMP

PUMP is generated directly from the specification. It says ON should occur when CLOCK ***or*** TSWITCH are true, and ***also*** activated whenever both salinity is too high ***and*** soil is too dry. It says the ON is true ***unless*** it is raining.

PUMP = (CLK + TSWITCH + SAL.DRY).(RAIN')

3-bit FLOW

Truth Table

This truth table has 5 inputs including PUMP which acts as an enabler, therefore when k-mapped, only 4 variables are used but PUMP becomes EN in the end.

INPUT					*OUTPUT*			*FLOW*
P/EN	**D**	**S**	**H1**	**H0**	**F2**	**F1**	**F0**	***STATE***
0	X	X	X	X	0	0	0	**NF**
1	0	0	0	0	1	X	0	**H**
1	0	0	0	1	0	1	1	**N**
1	0	0	1	0	0	1	0	**L**
1	0	0	1	1	0	0	1	**VL**
1	0	1	X	X	0	1	0	**L**
1	0	1	X	X	0	1	0	**L**
1	0	1	X	X	0	1	0	**L**
1	0	1	X	X	0	1	0	**L**
1	1	0	X	X	0	1	1	**N**
1	1	0	X	X	0	1	1	**N**
1	1	0	X	X	0	1	1	**N**
1	1	0	X	X	0	1	1	**N**
1	1	1	X	X	1	X	1	**VH**
1	1	1	X	X	1	X	1	**VH**
1	1	1	X	X	1	X	1	**VH**
1	1	1	X	X	1	X	1	**VH**

Which is derived from two tables

INPUT					*OUTPUT*					
P/EN	**D**	**S**	**H1**	**H0**	**NF**	**VL**	**L**	**N**	**H**	**VH**
0	X	X	X	X	1	0	0	0	0	0

1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0	0
1	0	0	1	1	0	1	0	0	0	0
1	0	1	X	X	0	0	1	0	0	0
1	0	1	X	X	0	0	1	0	0	0
1	0	1	X	X	0	0	1	0	0	0
1	0	1	X	X	0	0	1	0	0	0
1	1	0	X	X	0	0	0	1	0	0
1	1	0	X	X	0	0	0	1	0	0
1	1	0	X	X	0	0	0	1	0	0
1	1	0	X	X	0	0	0	1	0	0
1	1	1	X	X	0	0	0	0	0	1
1	1	1	X	X	0	0	0	0	0	1
1	1	1	X	X	0	0	0	0	0	1
1	1	1	X	X	0	0	0	0	0	1

INPUT						***OUTPUT***		
NF	**VL**	**L**	**N**	**H**	**VH**	**F2**	**F1**	**F0**
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	0	1	0	0	0	1	1
0	0	0	0	1	0	1	X	0
0	0	0	0	0	1	1	X	1

DISPLAY1

NET Representation	***INPUT***	***OUTPUT***						
	P	**a**	**b**	**c**	**d**	**e**	**f**	**g**
S	0	1	0	1	1	0	1	1
P	1	1	1	0	0	1	1	1

DISPLAY2

NET Representation	***INPUT***			***OUTPUT***						
	F2	**F1**	**F0**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0	1	1	1	1	1	1	0
1	0	0	1	0	1	1	0	0	0	0
2	0	1	0	1	1	0	1	1	0	1
3	0	1	1	1	1	1	1	0	0	1
4	1	X	0	0	1	1	0	0	1	1
5	1	X	1	1	0	1	1	0	1	1

z5209365

Optimisation

4 variable k-maps were used for flow. 3 variable k-maps for DISPLAY2. DISPLAY1 had one input so no k-maps were used, and PUMP was derived from specification and could not be algebraically simplified. The remainder of this section will show a summary of Boolean expressions derived from k-maps, then k-maps with used EPI underneath.

Boolean Expressions:

F2 = DS + D’S’H1’H0

The total GIC for the entire circuit is 46.

K-Maps

The k-maps below show all EPI and PI. Only the EPI was used for the final optimised expression. Templates were used for all k-maps and therefore default variables were used and must be substituted.

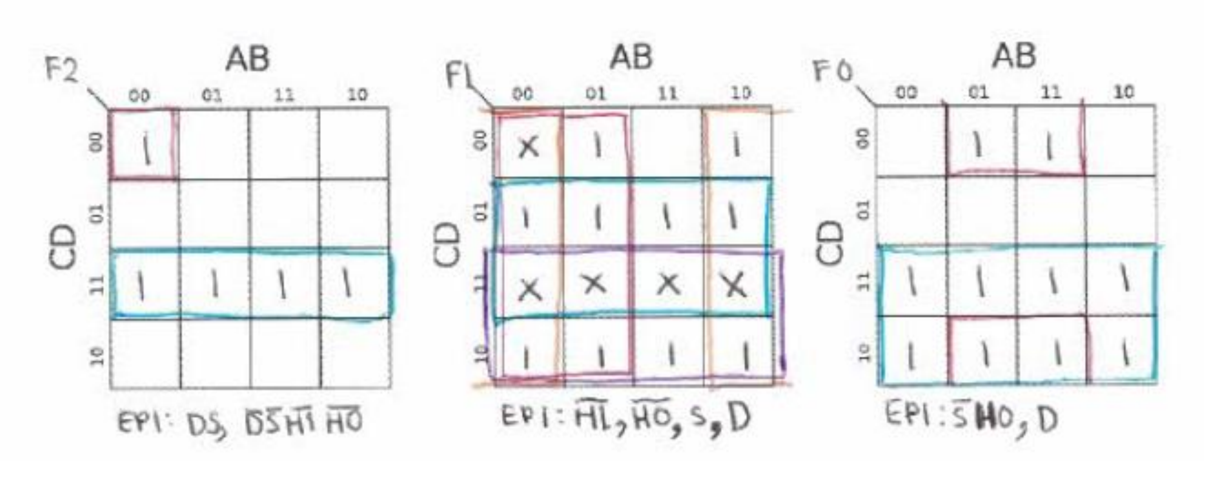
For FLOW, A = H1 (humidity 1), B = H0 (humidity 0), C = D (dry) and D = S (salinity).

For DISPLAY2, A = F2, B = F1, C = F0 which are the flow outputs.

DA2 and DD2 have the same inputs, as seen in the truth table above. Therefore, they use the same k-map.

FLOW

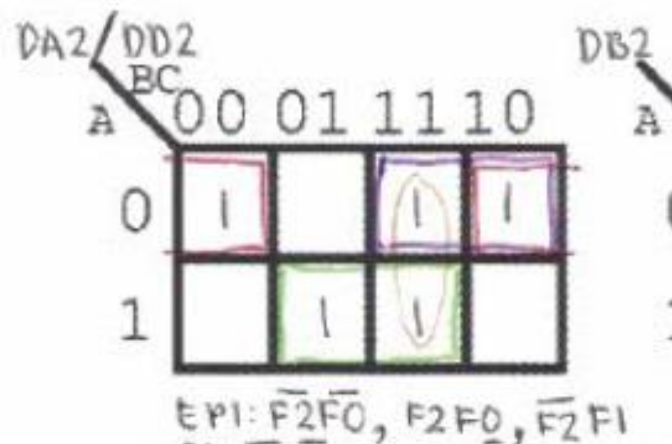
For FLOW, A = H1 (humidity 1), B = H0 (humidity 0), C = D (dry) and D = S (salinity).


F2
AB
CD
00 01 11 10
EPI: DS, $\overline{D}\overline{S}\overline{H1}\ \overline{H0}$
F1
EPI: $\overline{H1}$, $\overline{H0}$, S, D
F0
EPI: $\overline{S}$H0, D

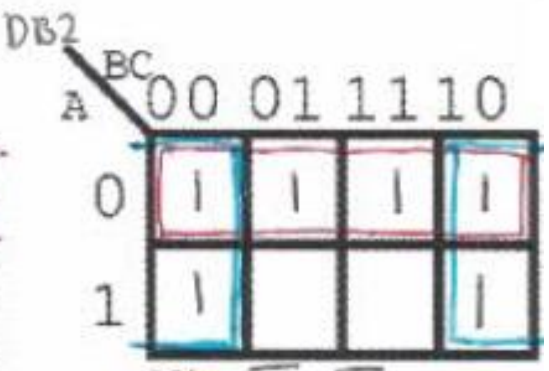
z5209365

DISPLAY2

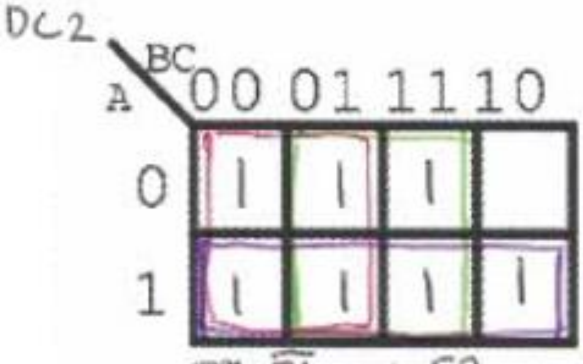
For DISPLAY2, A = F2, B = F1, C = F0 which are the flow outputs.



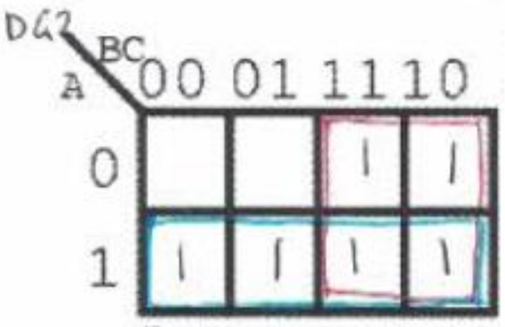
EPI: $\overline{F2}\,\overline{F0}$, $F2F0$, $\overline{F2}F1$
PI: $\overline{F2}\,\overline{F0}$, $F2F0$, $\overline{F2}F1$, $F1F0$



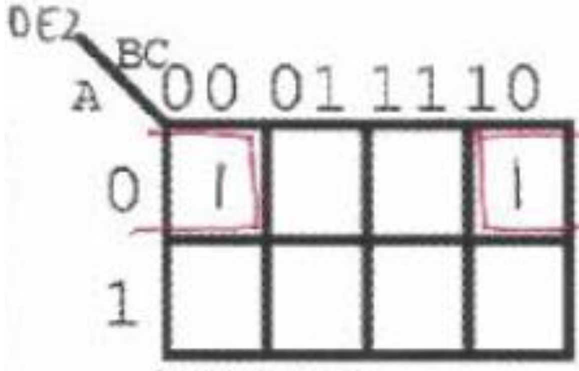
EPI: $\overline{F2}$, $\overline{F0}$



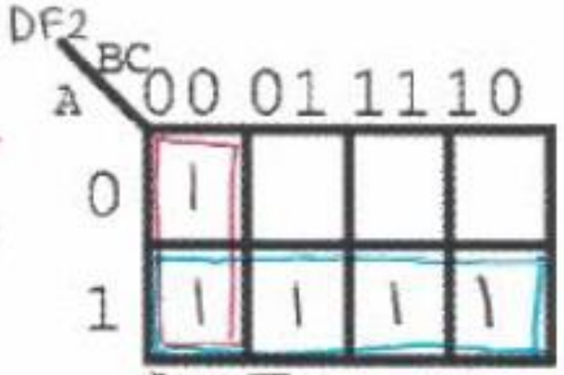
EPI: $\overline{F1}$, $F0$, $F2$



EPI: $F1$, $F2$



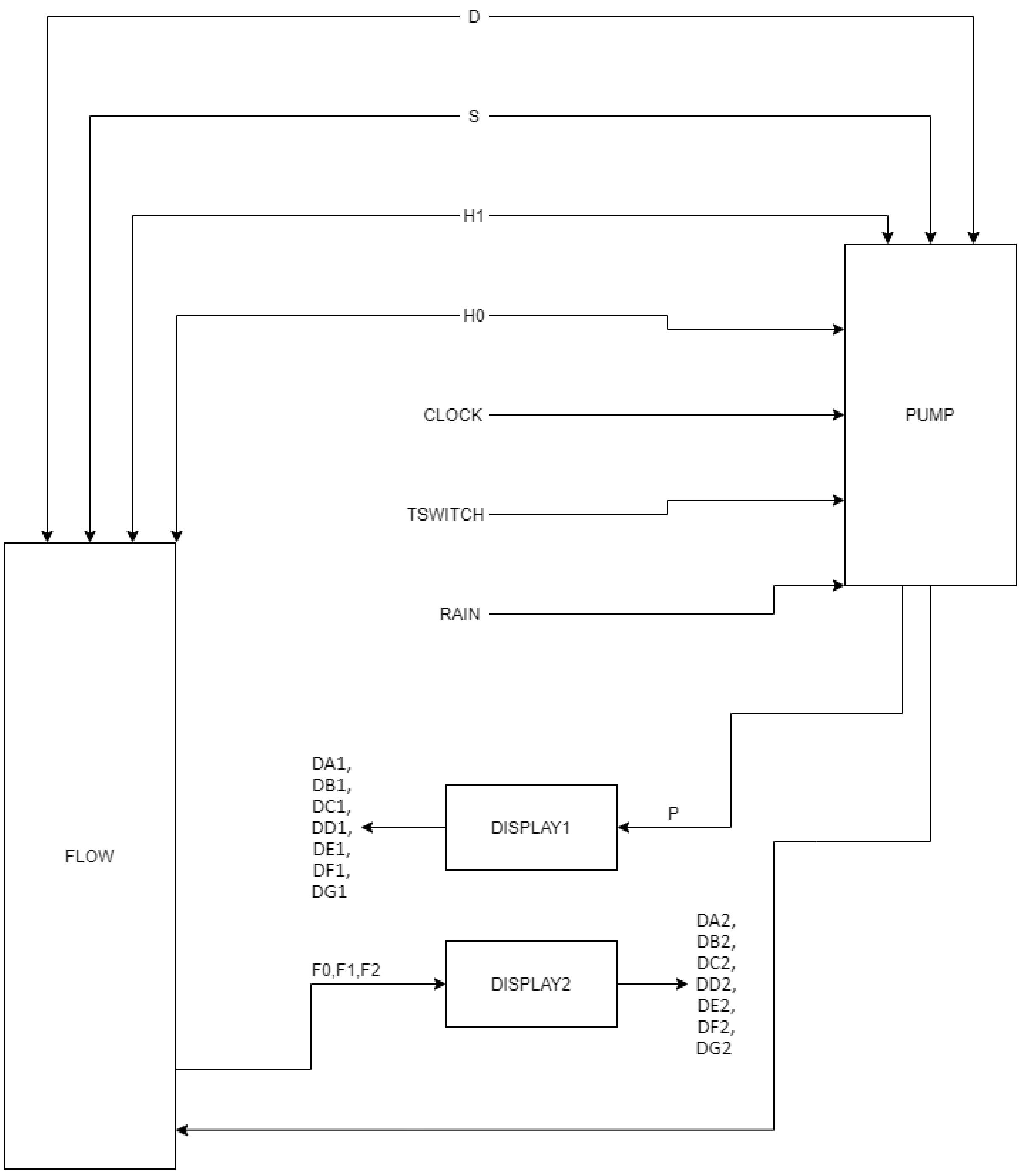
EPI: $\overline{F2}\,\overline{F0}$



EPI: $\overline{F1}\,\overline{F0} + F2$

z5209365

Hierarchical/Functional Blocks


D
S
H1
H0
CLOCK
TSWITCH
RAIN
PUMP
FLOW
P
DISPLAY1
DA1, DB1, DC1, DD1, DE1, DF1, DG1
F0,F1,F2
DISPLAY2
DA2, DB2, DC2, DD2, DE2, DF2, DG2

Circuit Implementation

NAND only because there are more AND gates which makes it cheaper than OR since AND = 2 NAND and OR = 3 NANDS. XOR is also cheaper in NAND by 1 gate. NANDs and NORs are economically cheaper, easier to resource and are universal.

Each hierarchical block of PUMP, FLOW, and DISPLAY1 & 2 are underneath titles.

The inputs are: D, S, H1, H0, CLOCK, TSWITCH, RAIN.

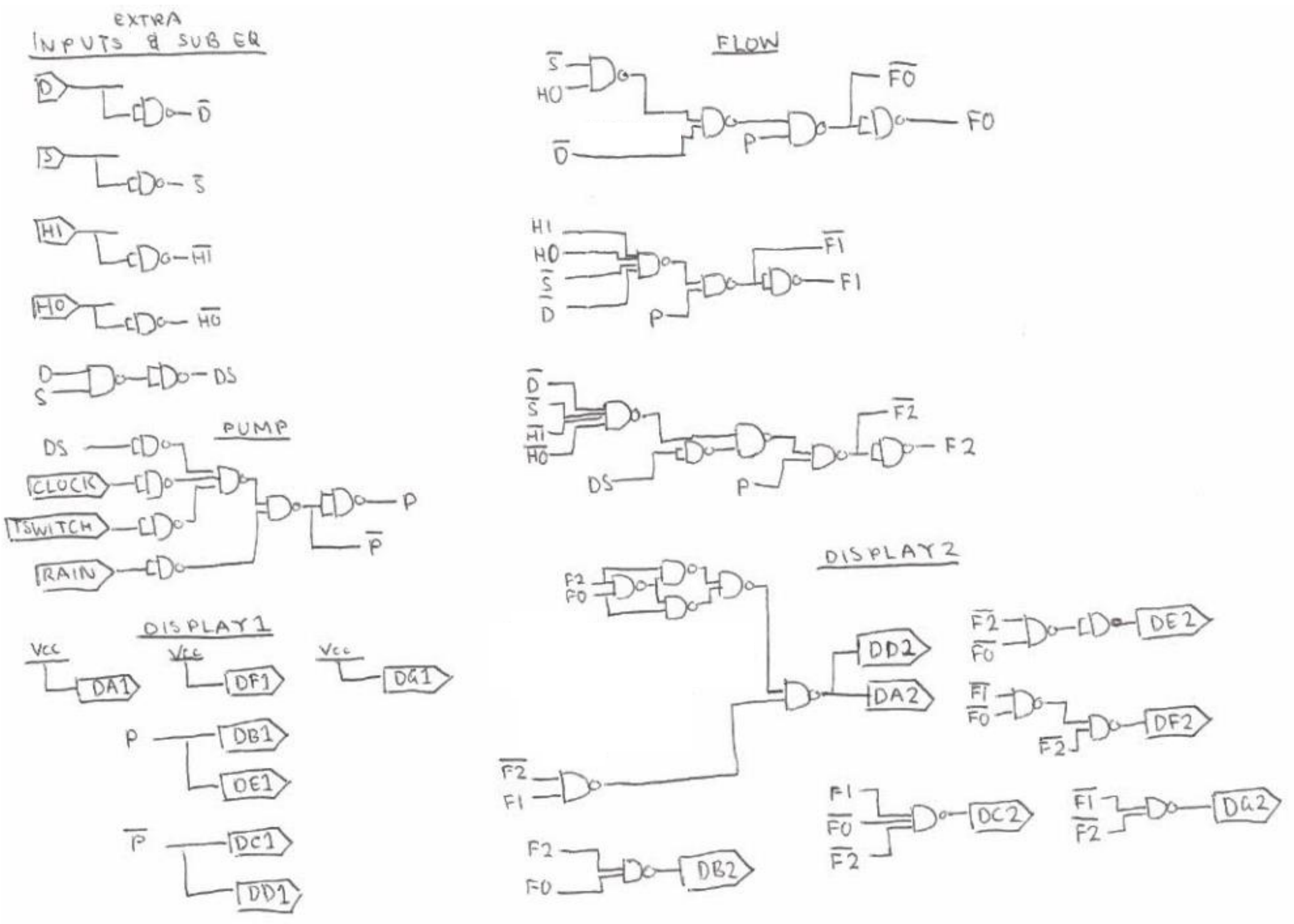
The outputs are:

P for PUMP

F0, F1, F2 for FLOW

DA1, DB1, DC1, DD1, DE1, DF1, DG1 for DISPLAY1

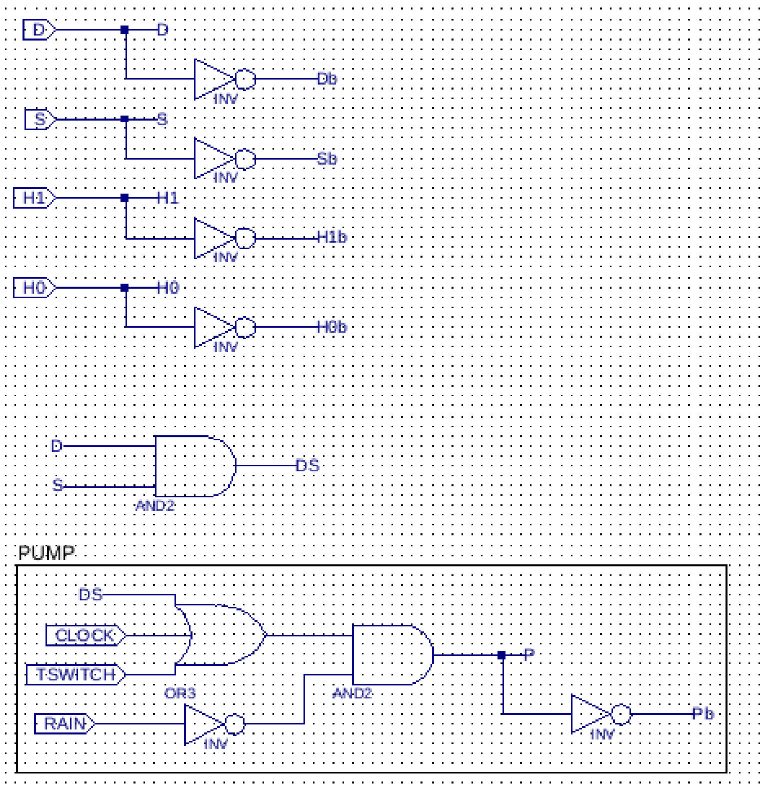
DA2, DB2, DC2, DD2, DE2, DF2, DG2 for DISPLAY2

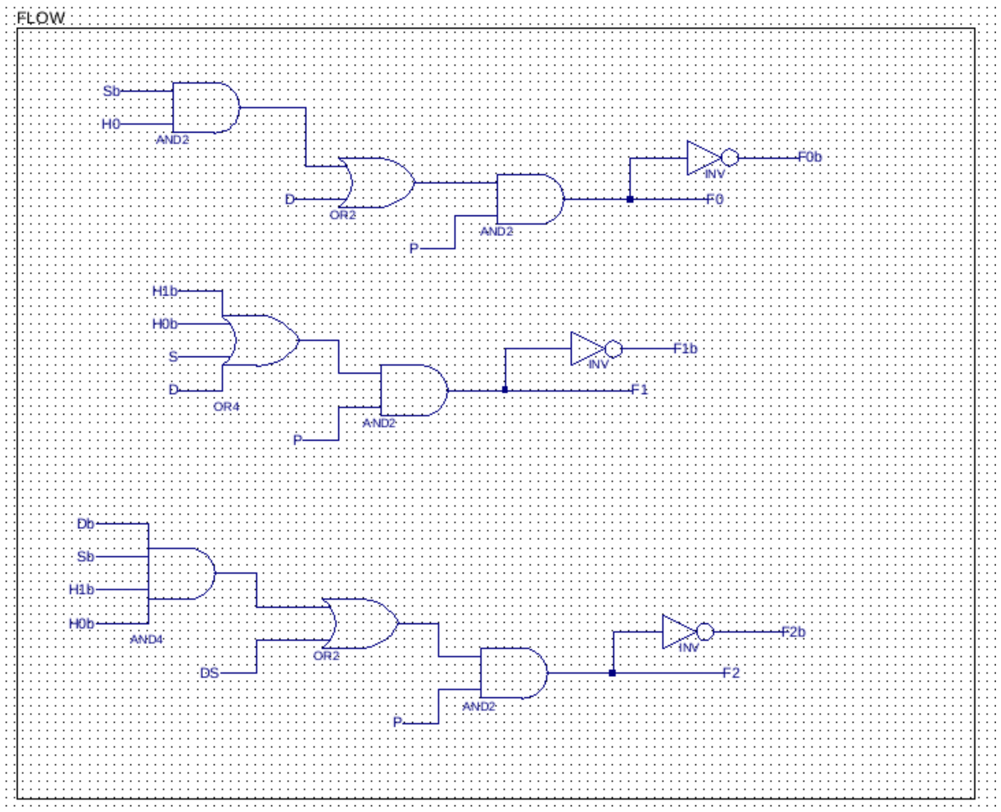


Verification

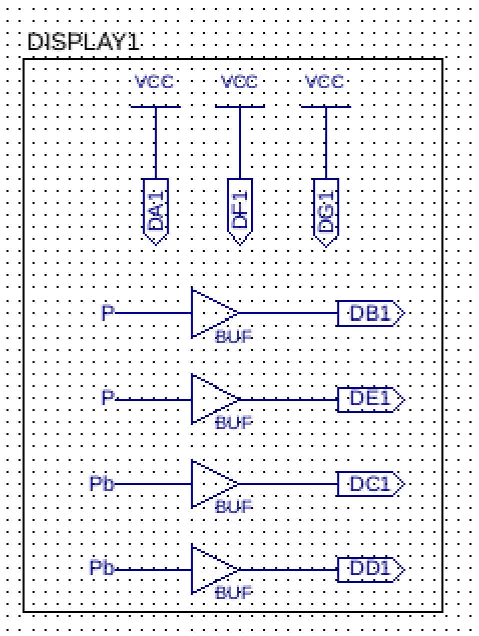
See attached files for schematics, Verilog test file and simulation result and files. Below are a few screenshots for preview.

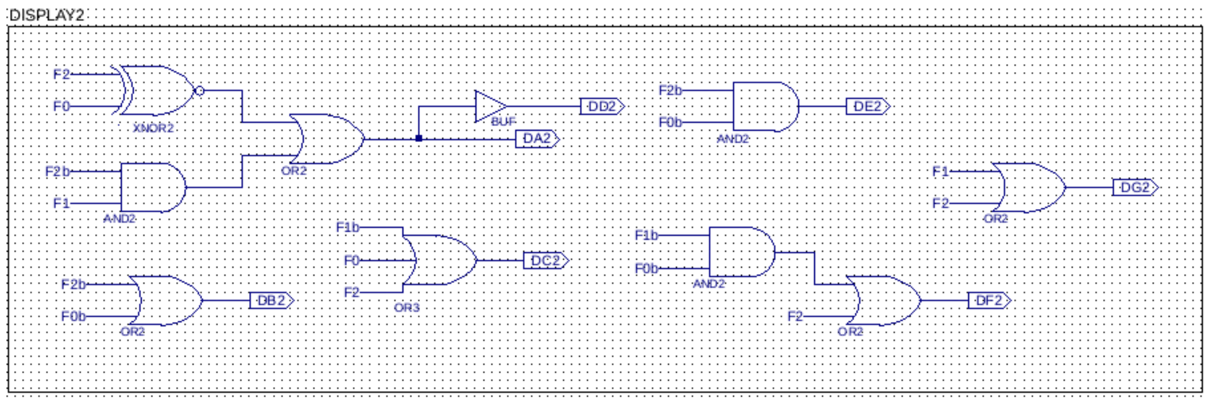
Schematics





z5209365



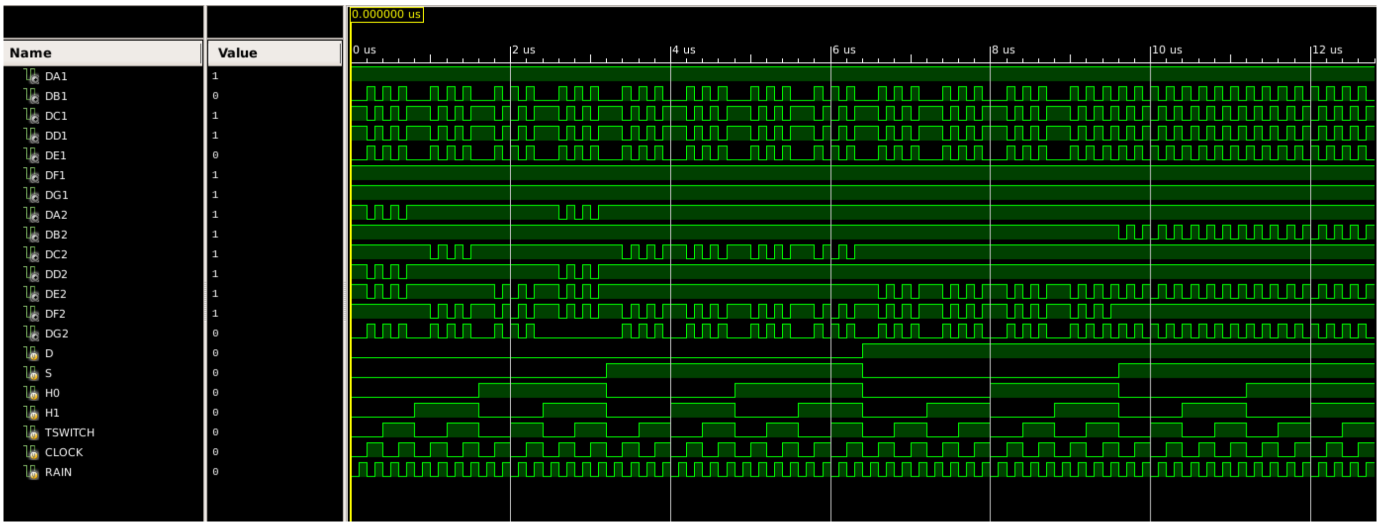


Verilog Test File

```
// Initialize Inputs
//   `ifdef auto_init
       initial begin
         D  = 1'b0;
         S  = 1'b0;
         H0  = 1'b0;
         H1  = 1'b0;
         TSWITCH = 1'b0;
         CLOCK = 1'b0;
         RAIN = 1'b0;
      forever begin
         #100;
         {D,S,H0,H1,TSWITCH,CLOCK,RAIN} = {D,S,H0,H1,TSWITCH,CLOCK,RAIN} + 1;
         end
      end
```

z5209365

Simulation



Name	Value
DA1	1
DB1	0
DC1	1
DD1	1
DE1	0
DF1	1
DG1	1
DA2	1
DB2	1
DC2	1
DD2	1
DE2	1
DF2	1
DG2	0
D	0
S	0
H0	0
H1	0
TSWITCH	0
CLOCK	0
RAIN	0