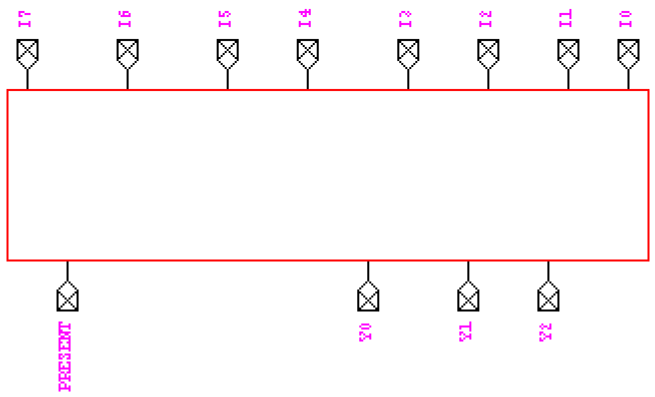
The Problem

Design a *combinatorial circuit* with 8-bit input **I** ($I_7 \ldots I_0$) which detects the patter **101** and outputs :

- A single bit output **P** which tells whether the pattern is present at all or not.
- A *3-bit* output **Y**($Y_2Y_1Y_0$) which tells the location at which the pattern is detected and in case of multiple instance gives the smallest number amongst the locations .



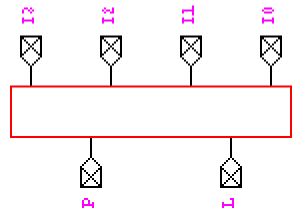
For example
If **I=** **00001010** then **Y=001** because it is located in location **1**
5 4 3 2 **1** 0
and if **I=00010101** then **Y=100** because the pattern is located at locations 0,2, and 4 and of them **0** is the minimum.

- An additional constraint that s there is to divide problem into blocks so that NO BLOCK has *more than four* input **except** one which can have five.

A point to note is the output **Y** is don’t care for when pattern is NOT detected, that is when PRESENT is zero.

The Solution

First of all lets see a simpler problem which is just to solve the problem for four inputs **I(**$I_3I_2I_1I_0$) and two single bit outputs ,P for present and L for location (note that the 3-bit pattern can only be in two locations 0 or 1 so the only 1 bit required for location)



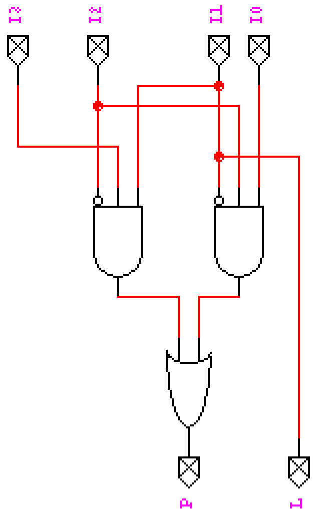
So more P to be one there are two cases
I = -101 or I = 101-

Which gives

$$\mathbf{P = I_3I_2'I_1 + I_2I_1'I_0}$$

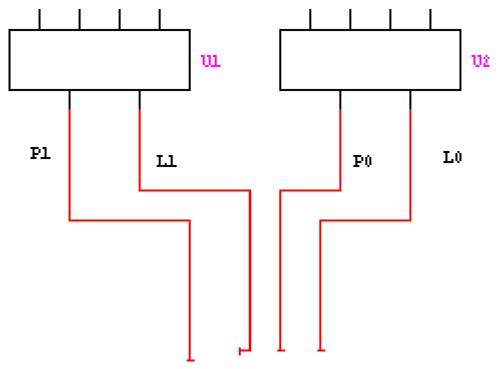
Now to find out the expression for L. Obviously the expression for L can be found out easily by making up an K-map and minimizing it. However a lot simpler expression can be obtained if we note the fact that **L** needs to have a valid output ONLY when pattern IS present. i.e. ONLY when **P** is 1.

I3	I2	I1	I0	**L**
-	1	0	1	**0**
1	0	1	-	**1**
Rest of combinations				**X**



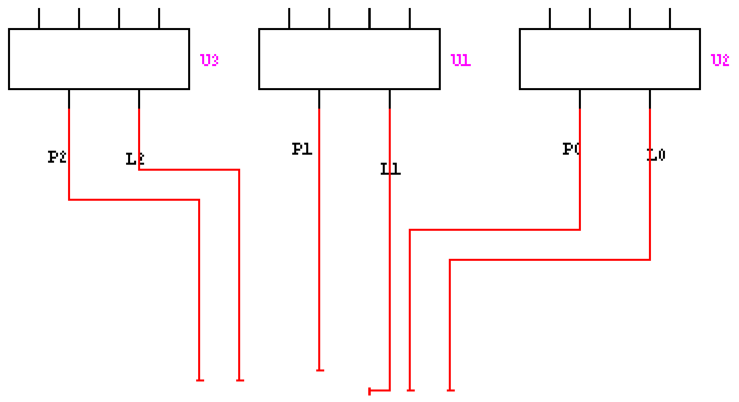
Looking at the truth table it is easy to see that the output is nothing but I_1. An important thing to realize is that the output L can be 0 and 1 even when pattern is not present (when **I = 0000**, L is 0 and **I=1111,** L is 1, but according to the specification of the problem that is allowed. Any user of this block is expected to realize that L is NOT valid when P=0).
Note: other simple implementation of L like I_2' are EQUALLY valid by the same argument).

Now that we have the blocks it is about time we started using them. So the naïve way would be



And feed these four outputs to some logic which using them solves for the **P** and **Y**. However in the cases in which the pattern in between the two four inputs would not be detected. (for e.g. **I=0001 0100 or I= 1110 1000**) . Therefore three such blocks are required each with Inputs $I_7I_6I_5I_4$, $I_5I_4I_3I_2$ and $I_3I_2I_1I_0$ respectively).

Now there is another problem here. Three blocks have SIX outputs , but the restriction is to have gate with MAX 5 inputs. So we need to reduce the number of the inputs somehow.



Now note we do not really need 8 cases (all possible combinations of P2P1P0) because of the relative priority involved. For example

If P0=1 we don't really need to see P2 or P1 because even if the pattern is detected by these blocks as the PRSENT output will anyway remain one and the locations ouput **Y** will be either **000 or 001** depending on where the pattern is. (Because even if the middle block **also** detects the pattern it means it is at location **010 or 011** which will not be output as location as the pattern is also present at *smaller* location !!)

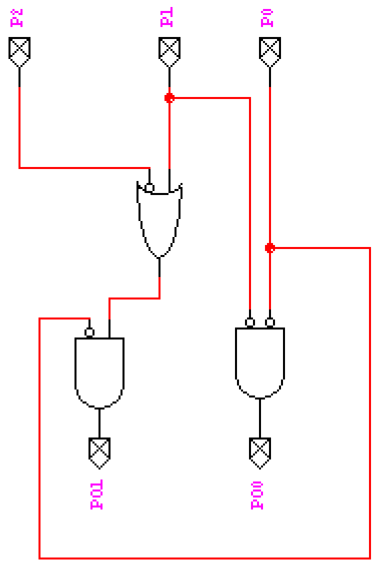
We will use the notation first block is *active* if it is the one which finally decides what the output of Location would be . So first blcok (rightmost) is *active* when P0=1. Middle one is active when P1=1 AND P0=0.

So actually there are just four cases conceptually.

- Pattern not present **(11)** *(none active)*
- Pattern presence detected by rightmost (first) block **(00)** *(first block active)*
- Pattern presence detected by middle block AND NOT by first block.**(01)** *(second block active)*
- Pattern presence detected ONLY by the third (leftmost) block.**(02)** *(third block active).*

So we just need **two** bits to encode these fours cases. Which gives us the following truth table corresponding to the encoding shown in the brackets above:

P2	P1	P0	**PO_1**	**PO_0**
0	0	0	**1**	**1**
-	-	1	**0**	**0**
-	1	0	**0**	**1**
1	0	0	**1**	**0**



This gives the following minimized Boolean expressions

PO_1=P0'P1'
PO_0=P0'(P2'+P1)

Now that we have the basic elements we can go ahead and see the full design at the block level. (see LAST page).

Now the only thing left is to design the last block which takes in PO_0,PO_1, L_2,L_1,L_0 as inputs and outputs **PRESENT** and **$Y_2Y_1Y_0$.**

There are two concepts involved here which will help us minimize the design complexity and the time required to think about it ☺

The first of these is our old friend “Location can be garbage when the pattern is not present”. Which means **Y** is like “don’t-care” in cases when P=0.

The second is to realize that inputs L0,L1 and L2 affect the output **Y** only when their respective block is the *active* (If you have forgotten what *active* is by now see above, and I also suggest you start paying attention ☺)

For example if middle block is active then the Location is 010 OR 011 depending on L1.
So The **PRESENT** output is very simple.
It is **0** if and only if PO_0PO_1=**11**
Which gives

PRESENT = (PO_0PO_1)'

Regarding $Y_2Y_1Y_0$ note the following

$P0_1$	PO_1	$\mathbf{Y_2}$	$\mathbf{Y_1}$	$\mathbf{Y_0}$
1	1	**X**	**X**	**X**
0	0	**0**	**0**	**L0**
0	1	**0**	**1**	**L1**
1	0	**1**	**0**	**L2**

Note that is nice compact way of writing another way
That

If P_0P_1 are 11 then all the Y's are don't care
00 then **Y**=$00L_0$

and so on.

And the way to **solve** these *if-then* is also pretty easy as read from the K-map.

Note an very interesting feature that Y_2 and Y_1 **do not** depend on L0, L1 or L2 (the location output of the 4-input blocks) !!. This because these inputs just will indicate *which block* was finally *active*. And that is expected if you think a little more because locations possible in the three blocks respectively are 000&001 , 010&011, and 100&101 respectively which means that only the last bit is different . Rest is common for both the locations corresponding to the block!

So we get the following expressions for the outputs

Y_2 = P1P0'
Y_1=P1'P0
Y_0=P1'P0'L0+P1'P0L1+P1P0'L2

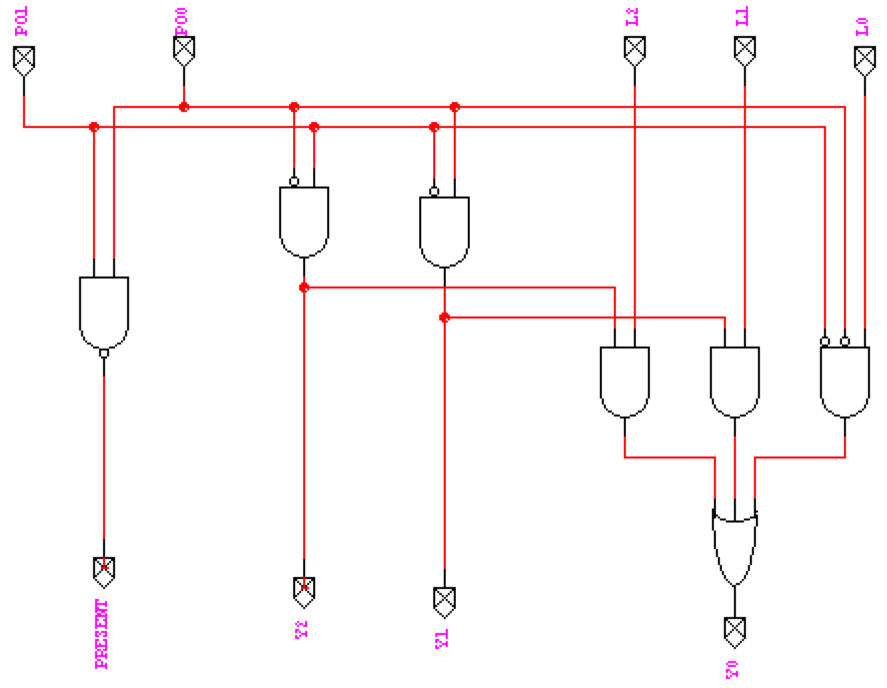
So as a general trick (*optional*)

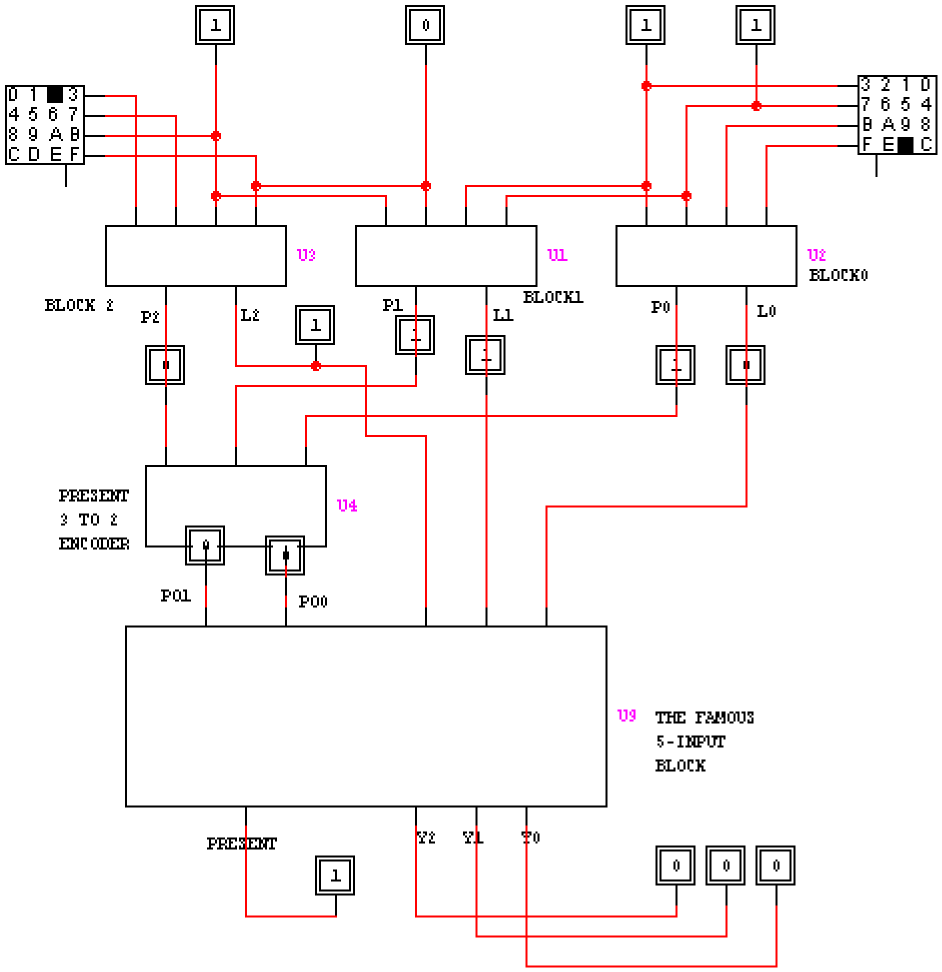
Inputs	Output=F
Case1	Out1
Case2	Out2
Case3	Out3
And so on	

Then

F = (Case1*Out1)+(Case2*Out2)+(Case3*Out3) ….

Note that the minterm expansion from Truth Table or Sum-of-Products is just a special case of this in which Out's can only be **0** or **1!**!. Which means the minterms (Cases) for which the Out is **1** , are included and rest are not !!




1
0
1
1
0 1 ■ 3
4 5 6 7
8 9 A B
C D E F
3 2 1 0
7 6 5 4
B A 9 8
F E ■ C
U3
U1
U2
BLOCK0
BLOCK 2
P2
L2
1
P1
L1
BLOCK1
P0
L0
PRESENT
3 TO 2
ENCODER
U4
P01
P00
U9
THE FAMOUS
5-INPUT
BLOCK
PRESENT
Y2
Y1
Y0
1
0
0
0