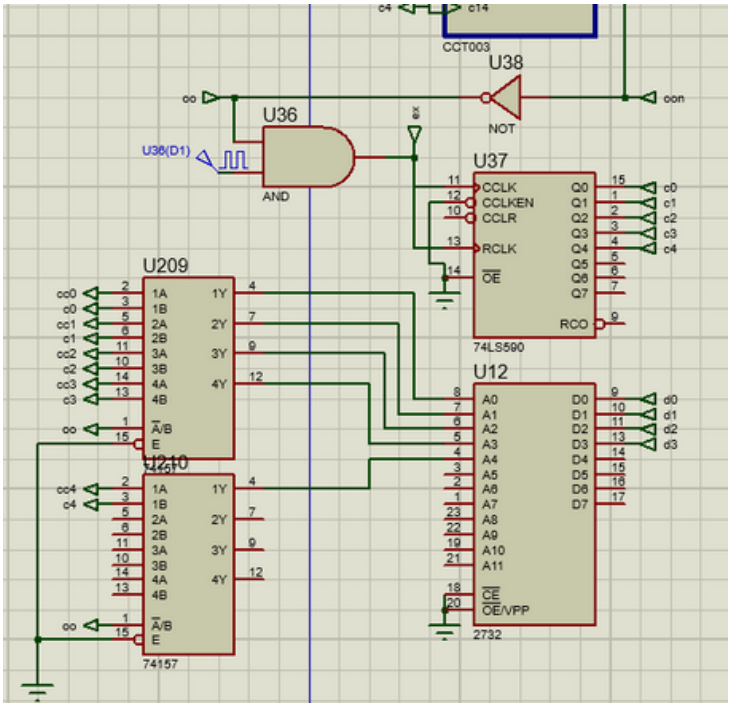
PROBLEM STATEMENT
CONVEX HULL CALCULATION

In geometry, a convex hull is the smallest convex shape that contains a set of points. It's also known as the convex envelope or convex closure.

"Starting from a leftmost point of the data set, we keep the points in the convex hull by anti-clockwise rotation. "



We used only one **ROM**. For being able to use only one ROM we needed two counters, one for the calculation of left-most point and the other one for calculation of orientation and updation.

We used two MUXs to create a two **5-bit data selector** which selects which indices should be given to the ROM. the select condition for the MUXs were whether we were performing left-most point calculation or orientation calculation.

Firstly we tried to figure out how to find the leftmost point and we observed that it can be done by only comparing the x-coordinates of points unless the x-coordinates are equal, in that case , we compared the y-coordinates.

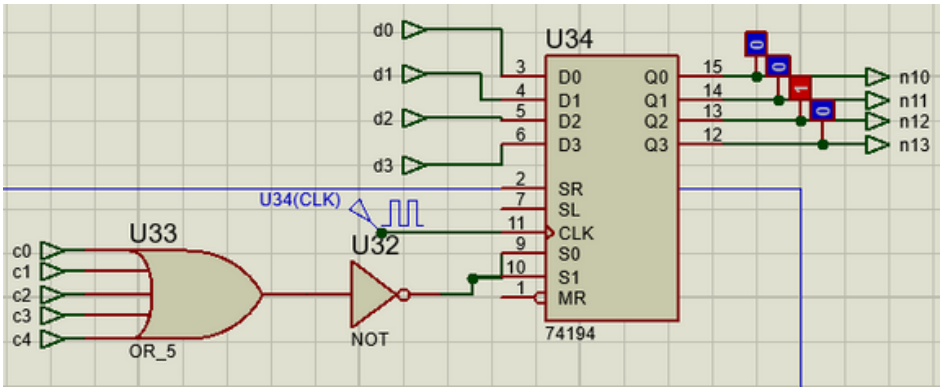
The next step was to construct a circuit that would calculate the orientation of the points, for that matter we choose 9 registers and assigned (p2,x",y"), (po,x',y') and (i,x0,y0) to them with appropriate loading conditions.

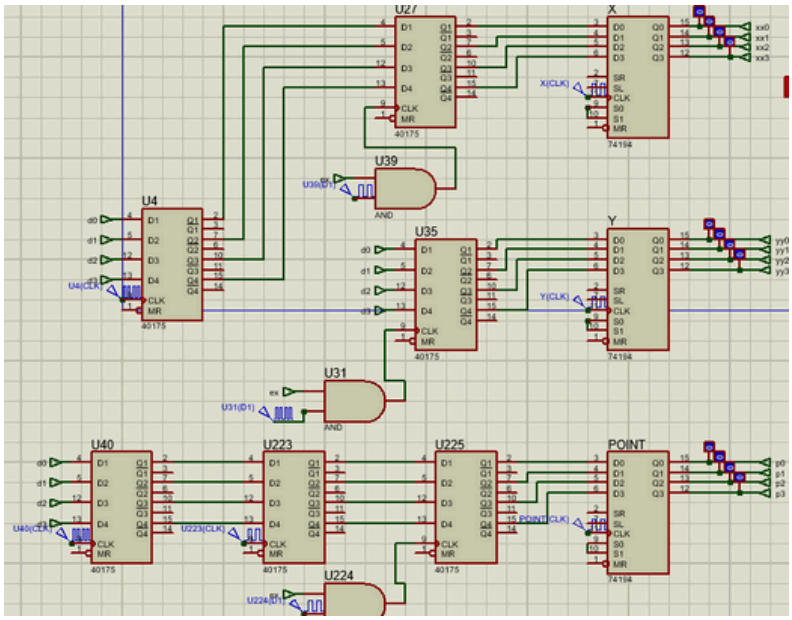
Our circuit consists of a subcircuit which deals with the collinear case as well, as in a strict convex hull, only the vertices are considered and not every point which lies on the edge of the hull. For creating this subcircuit, we again considered comparison of coordinates of points for deciding which one is farther.

In the end, for the circuit to stop, we considered the condition when the point number of the point on the convex hull is again equal to the point number of the leftmost point, the process is complete and the circuit stops.

SUBCIRCUIT MIN

We need the **number of coordinates** for any further action hence we created a circuit which stores the number of coordinates. The circuit consisted of a register which loads values only when the **count** is 0.
Here the number of coordinates were 4 hence the register stores value 4 (4'b0100).

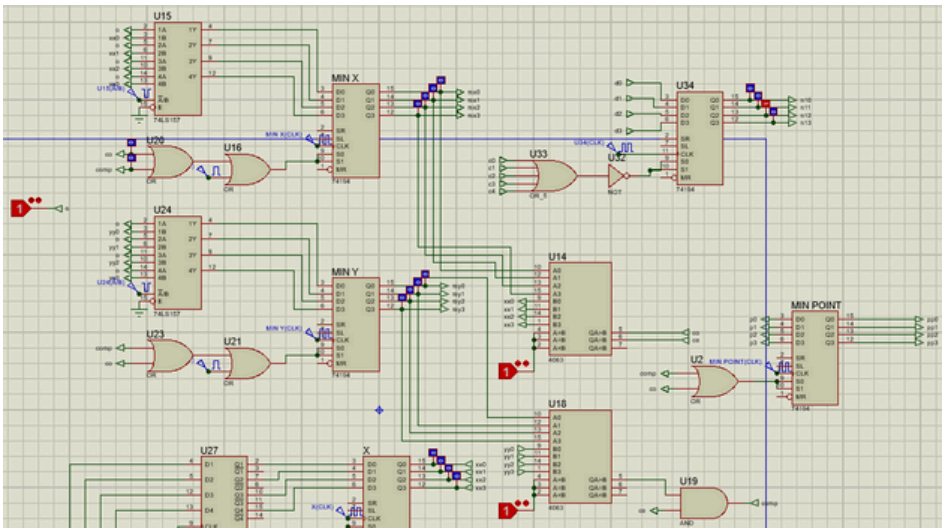




For a particular coordinate, first we obtain the point number, then the x-coordinate and then the y-coordinate, but we need all of them at once for any calculation hence we introduced a **delay of Two clock cycles** in point number and a **delay of one clock cycle** in x-coordinate.
For the registers to store only the entity they are supposed to store, we gave a pattern as clock to them which only loads the required values to a particular register. (point number, x-coordinate and y-coordinate to X,Y and POINT registers respectively.

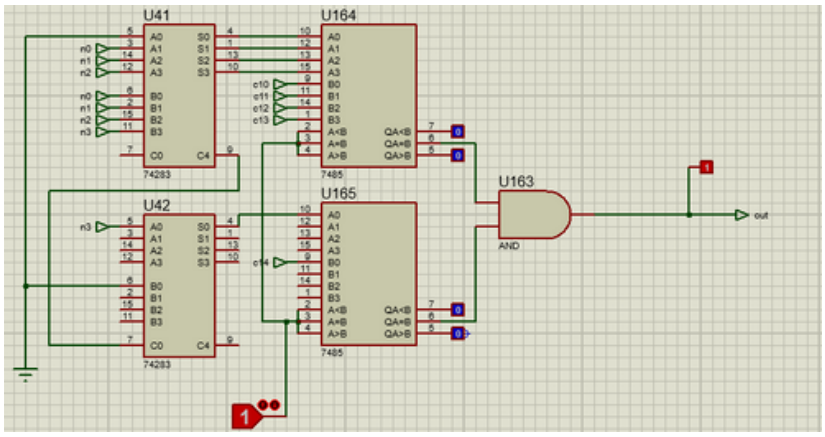
For calculation of the left-most point firstly we stored value 15 (4'b1111, maximum possible value) in the MIN X, MIN Y registers so as when the comparison starts, the registers don't store any arbitrary trash value and the initial value in the registers are greater than any possible value the coordinates can take.

We compared the x-coordinates, if current x-coordinate is less than the x-coordinate stored in the MIN X register then MIN X, MIN Y and MIN POINT registers are updated with current values of (point number, x, y).
If the x-coordinates are equal then y-coordinates are compared and if current y is less than Min y then MIN X, MIN Y and MIN POINT are updated.



The process stops once the counter reaches the maximum index.
For the process to stop we need **maximum index.** Maximum possible is calculated in yet another sub-circuit **COUNT** .

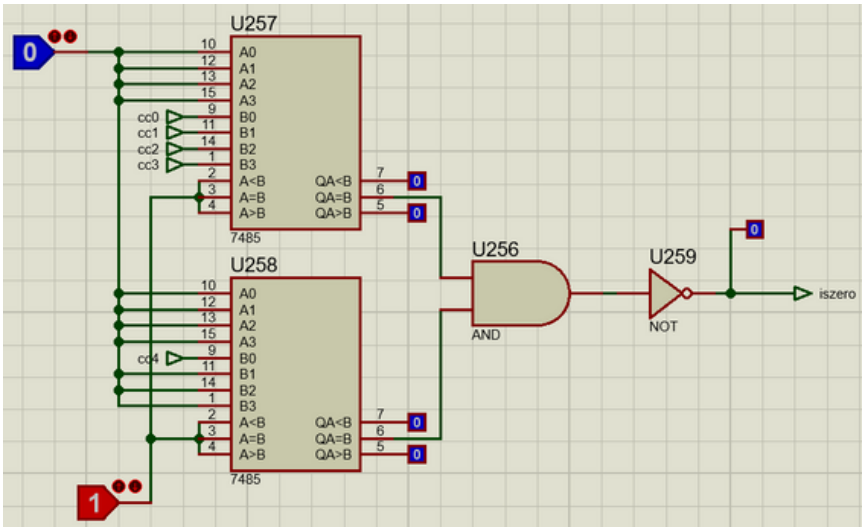
SUBCIRCUIT COUNT AND COUNT2



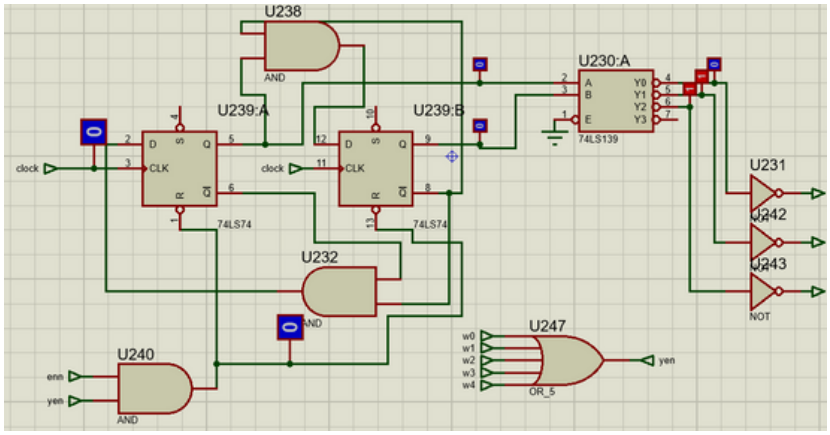
We know that the maximum possible index will be three times the number of points hence we created a circuit to calculate **3*N**.
(**N=number of points**)
Firstly we left shifted N (obtained from min sub-circuit) so as to get 2*N and then added N to the result to get 3*N.
Then we compare the value of counter to

3*N, if the values are equal this means all the coordinates have been traversed and we got our bottom left point. now the counter for minimum point (left-most and bottom-most point) stops.

This circuit is present only in sub-circuit **COUNT2,** it compares the value of counter to **zero** and tells if the counter is zero or not.
We need to know if the counter is zero or not as we need to initialize the counter again and again (looping).
We didn't initialize the counter to 1 as it introduced unnecessary delay in the first traversal.

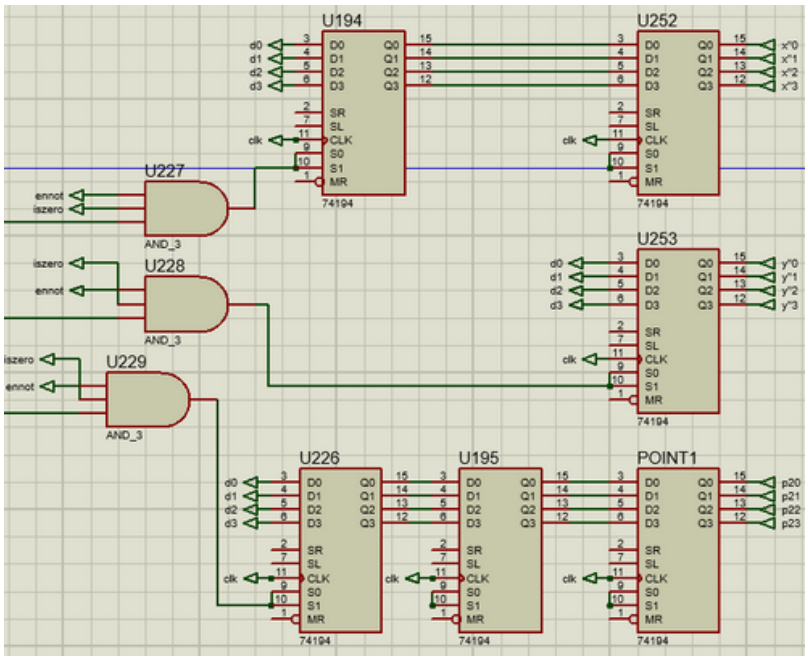


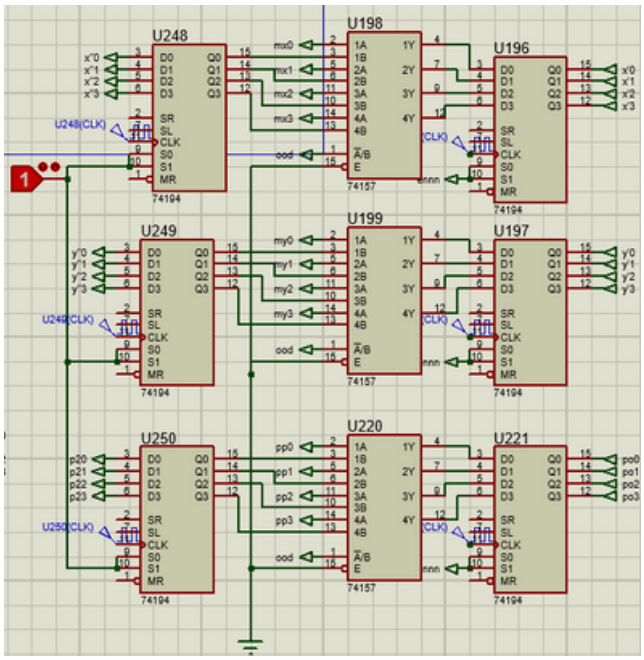
SUBCIRCUIT DECODE

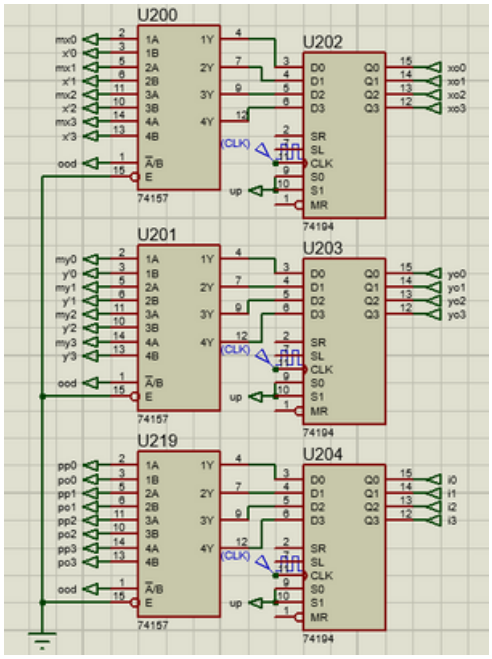


This is the FSM of modulo 3.
This circuit is synchronized with the counter and outputs **mod0**, **mod1** or **mod2** in a cyclic manner. The circuit activates when the counter is active and the value is greater than zero, that is, it starts when the value of counter is 1 or more. These will be used to activate x,y and point registers

TERMS & NOMENCLATURE







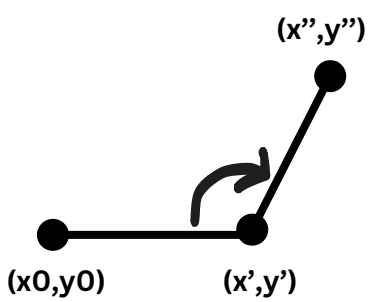
(x0,y0) :- Reference point (Already on Hull).
(x',y') :- Point being checked to be on the vertex of convex hull.
(x",y") :- Every other point.

We included delays in the updation of **x"** (one clock cycle) and point number **p2** (two clock cycles) so that (p2,x",y") all are obtained at the same time.

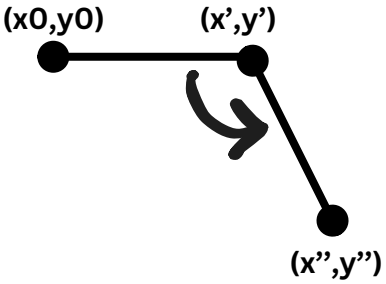
There is no need to include delays in loading of (po,x',y') and (i,x0,y0) as their updation depends on the orientation and various calculations being performed on (x",y").

Orientation = (y'-y0)(x"-x')-(y"-y')(x'-x0)

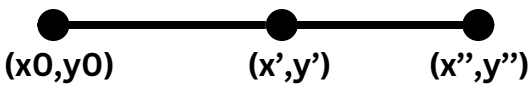
Orientation>0 Clockwise
Orientation<0 Counter clockwise
Orientation=0 Collinear



Clockwise
Orientation>0

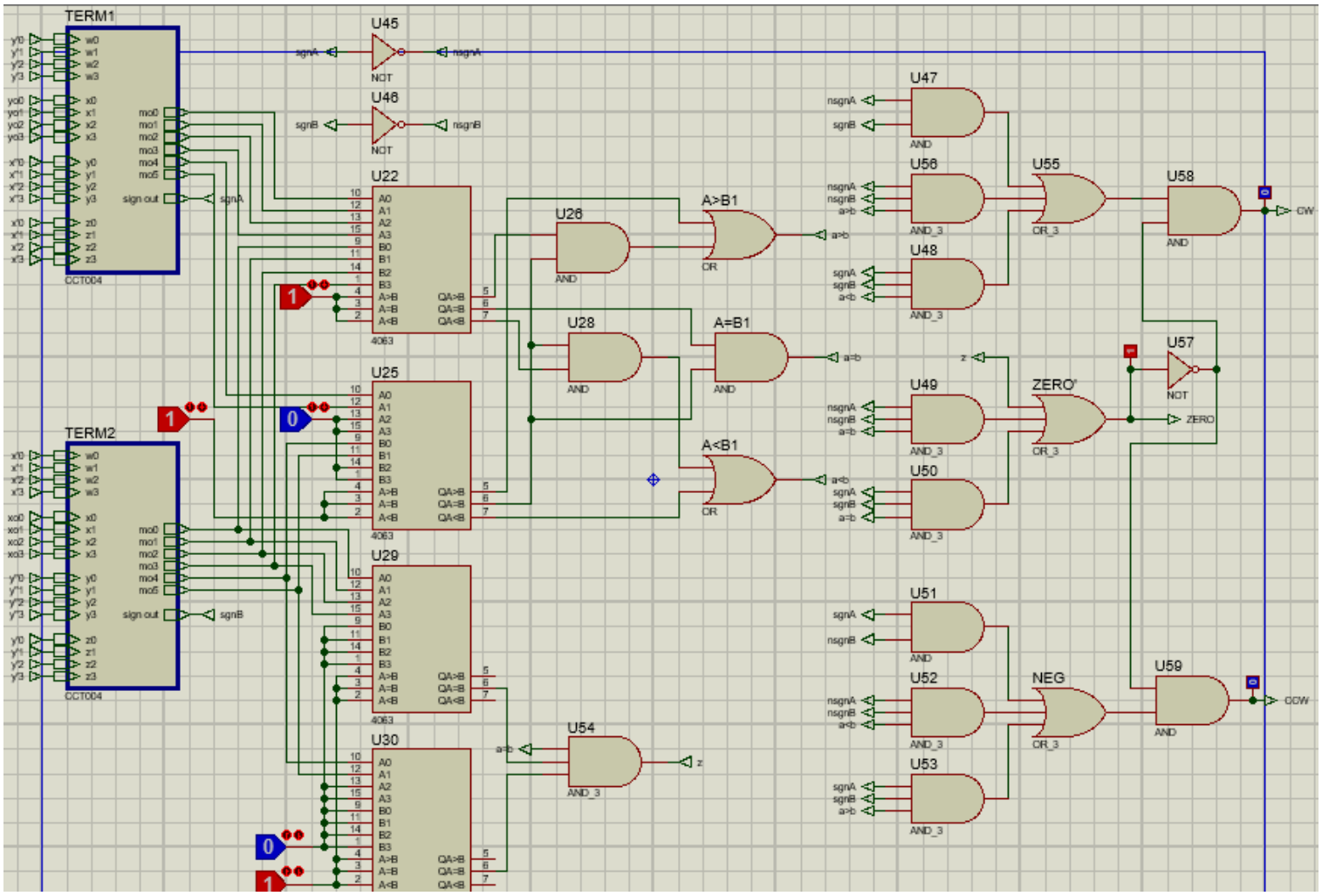


Counter Clockwise
Orientation<0



Collinear
Orientation=0

SUBCIRCUIT ORIENTATION



TERM1 = (y'-y0)*(x''-x')

TERM2 = (y''-y')(x'-x0)

Let:-

a = TERM1

b =T ERM2

Orientation = (y'-y0)*(x''-x') - (y''-y')(x'-x0) = a - b

Now, if we consider the signs of **a** and **b** :-

a	b	a==b	a>b	a<b	Orientation
Positive	Negative	1	0	0	Positive
Positive	Negative	0	1	0	Positive
Positive	Negative	0	0	1	Positive
Negative	Positive	1	0	0	Negative
Negative	Positive	0	1	0	Negative
Negative	Positive	0	0	1	Negative
Positive	Positive	1	0	0	Zero (Positive)
Positive	Positive	0	1	0	Positive
Positive	Positive	0	0	1	Negative
Negative	Negative	1	0	0	Zero (Positive)
Negative	Negative	0	1	0	Negative
Negative	Negative	0	0	1	Positive

From the table, we observed that is **a** and **b** are of **opposite signs** then the orientation depends on the sign of **a** only, doesn't matter if **a** is greater than **b** or vice-versa, unless both are equal to zero, in that case we see whether **a** and **b** are equal or not and if **b** is equal to zero or not, here **Orientation** will become **ZERO.**

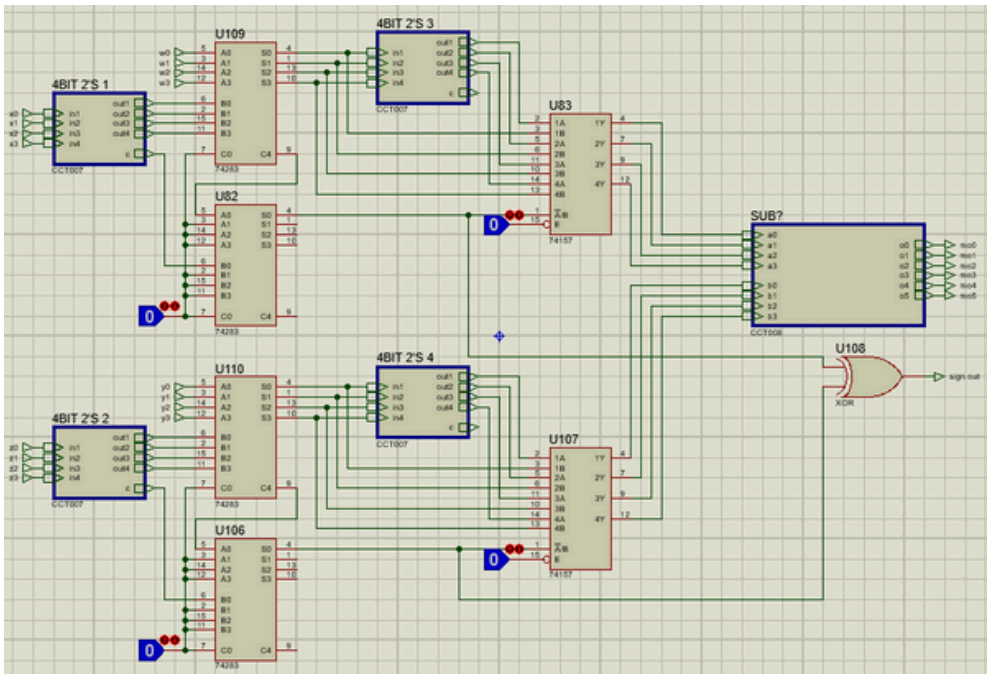
If **a** and **b** are of same signs then value of Orientation depends on whether **a** is greater than **b** or vice-versa. Orientation will be **ZERO** if both **a** and **b** are of same sign and are equal.
If both are positive then orientation will be **Positive** if **(a>b)**, and **Negative** if **(a<b)**.
If both are negative then orientation will be **Positive** if **(a<b)**, and **Negative** if **(a>b)**.

Hence, our circuit compares the signs of **a (TERM1)** and **b (TERM2)** and their magnitudes and computes whether **Orientation** will be **positive (Clockwise) , negative (Counterclockwise)** or **Zero**.

SUB-SUBCIRCUIT TERM1 & TERM2

TERM1 = (y2-y1)(x3-x2) = c*d
If we consider calculation of **c (y2-y1):-**
First we calculate the 2's compliment of y1 (in the subcircuit **4BIT 2'S**) and then add it to y2.
Carry bit will tell us whether the subtraction resulted in positive value (in 2's compliment form) or negative value.

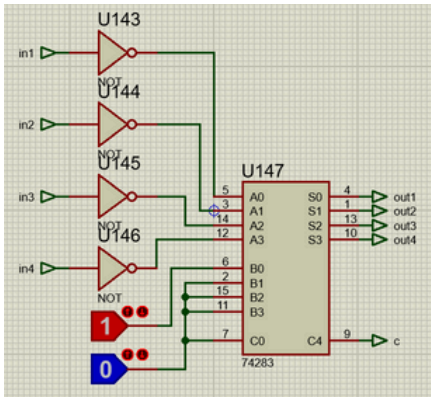
We multiply the magnitude of **c** and **d** in the subcircuit **SUB?** and the sign of the product depends on the signs of **c** and **d** , if both are of same sign then the product will be positive and is they are of opposite signs then the product will be negative.
We used XOR gate to obtain the sign of the product.



SIGN:-
Positive = 0
Negative = 1

To calculate the magnitude of subtraction, when result is positive, we pass the result to **4BIT 2'S** subcircuit to again take 2's compliment. We pass the 2's complement of the result and the actual result to a MUX whose select line is the carry of the adder. The MUX outputs the magnitude of the result of subtraction to the SUB? subcircuit for multiplication.

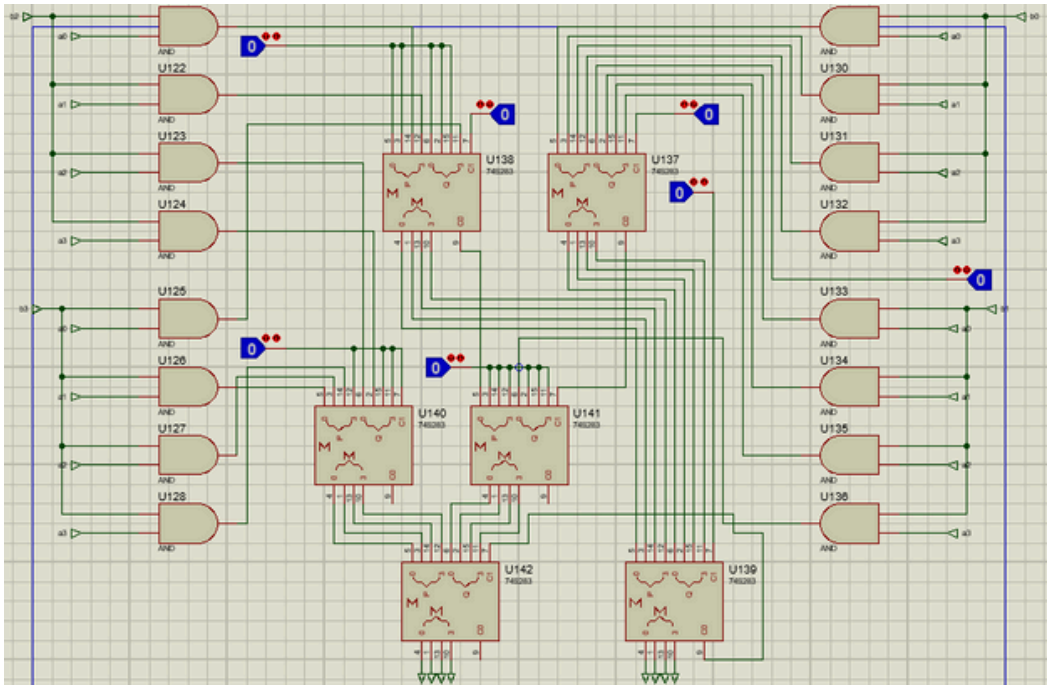
SUB-SUB-SUBCIRCUIT 4BIT 2'S



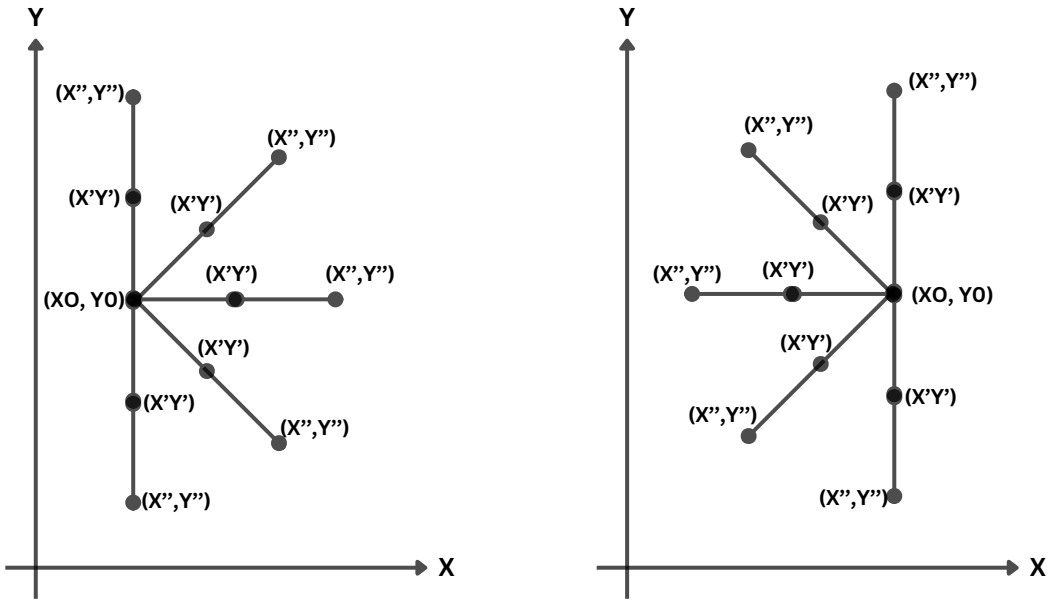
4BIT 2'S calculates 2's compliment of a number by inverting it's bits and then adding 1 to it.

SUB? multiplies two 4 bit numbers.

SUB-SUB-SUBCIRCUIT SUB? (MULTIPLICATION)



SUBCIRCUIT COLLINEAR CASE

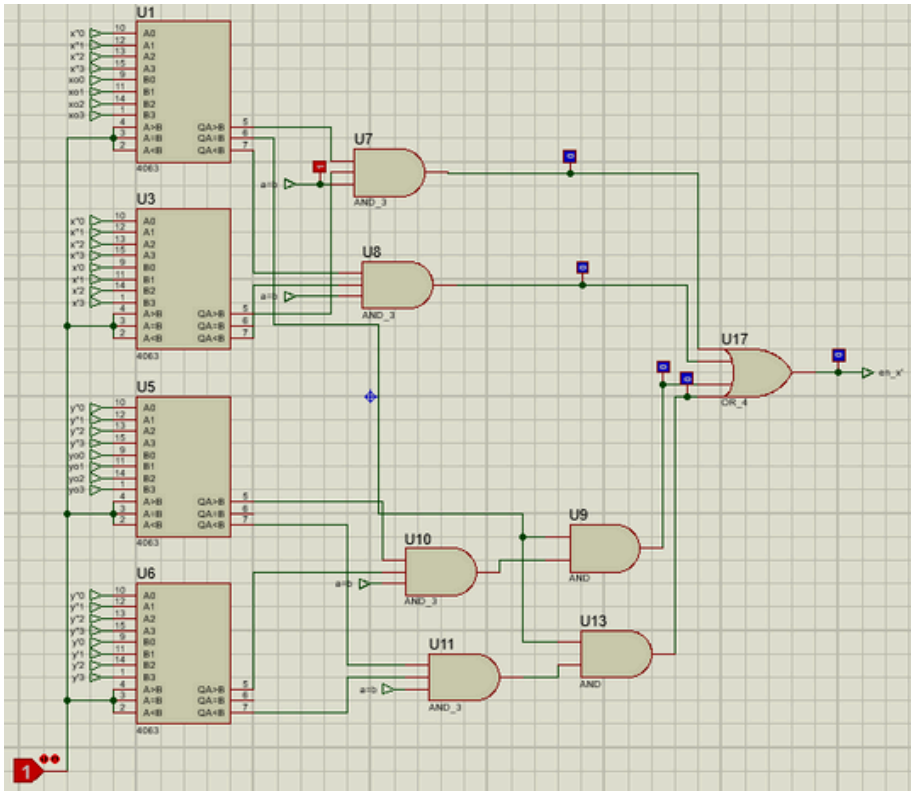


When **Orientation** is **ZERO**, that is, when points are collinear, then **(x',y')** should be updated only when **(x",y")** is farther from **(x0,y0)** than **(x',y')**, this is possible for cases shown above.

```
if (x">x0)
    if(x">x')
      update (x',y')

if (x"<x0)
    if(x"<x')
      update (x',y')
```

```
if(x"=x0)
    if(y"<y0)
      if(y"<y')
        update (x',y')
    if(y">y0)
      if(y">y')
        update (x',y')
```



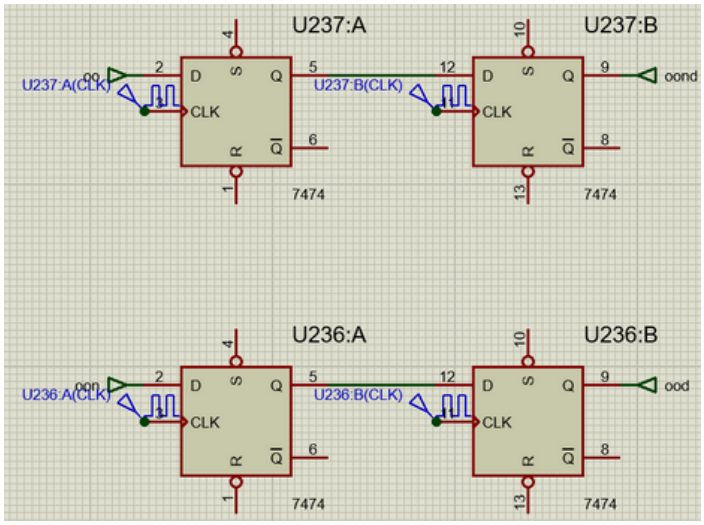
The subcircuit COLLINEAR CASE configures whether **(x",y")** is farther from **(x0,y0)** than **(x',y')** or not.

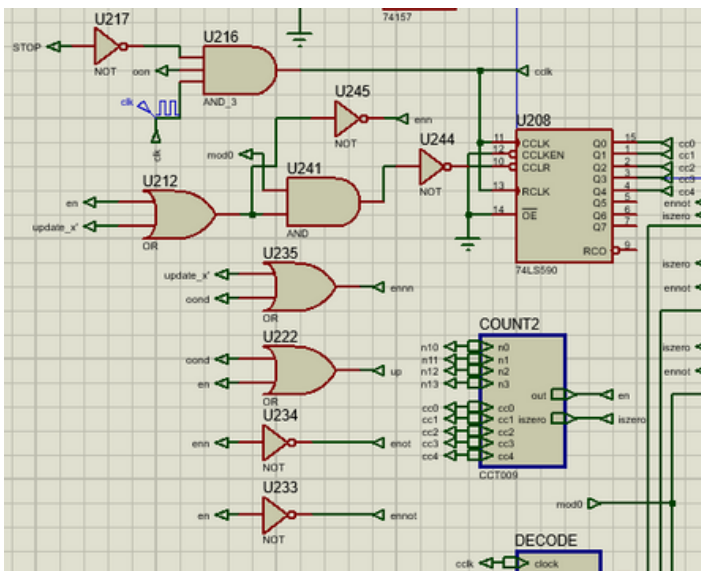
The subcircuit operates/is active only when **Orientation** is **ZERO.**

The circuit consists of four magnitude comparators and a combination of their outputs.

(x',y') will be updated if **Orientation** is either **clockwise** or if **(x",y")** is farther from **(x0,y0)** when **Orientation** is **ZERO**.

UPDATE CONDITIONS





oon = count (for min / leftmost) calculation is equal to **3*N**

oo = NOT(**oon**)

ood = **oon** delayed by one clock cycle

oond = **oo** delayed by one clock cycle

en = count2 (for Orientation calculation) is equal to **3*N**

iszero = is count2 equal to zero or not

update_x' = condition for updating (x',y')

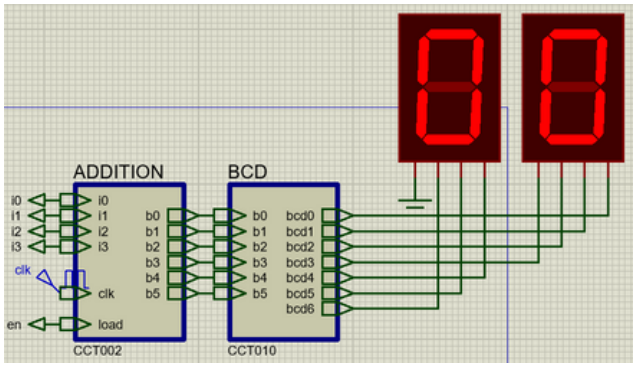
ennn = if min calculation is complete or not and should (x',y') be updated or not

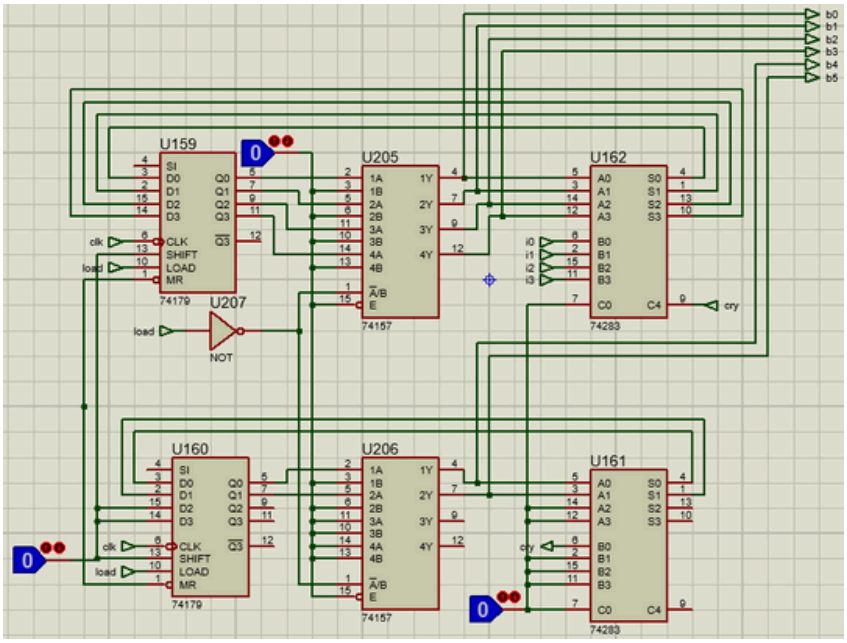
up = condition for updating (x0,y0), if count2 is equal to **3*N** or **oond** (to update initial value of (x0,y0)

enot = NOT(**enn**)

ennot = NOT(**en**)

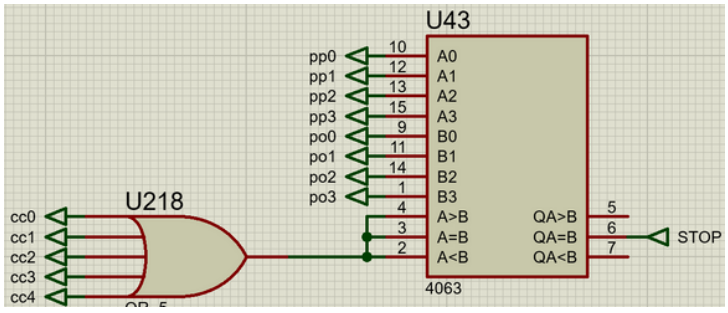
SUBCIRCUIT ADDITION





The subcircuit ADDITION consists of adders and registers. the circuit is active whenever (x0,y0) is updated. The circuit add the point number corresponding to the new (x0,y0) and the result is stored in a register. There is a feedback which adds the new point number to the sum of point numbers of previous points/coordinates which were on convex hull.

STOP



This part of the circuit decides when the whole circuit should stop.
This is a comparator which compares when the updated point number (which updates whenever (x0,y0) updates) is equal to the point number of the leftmost-bottommost point.