Convex Hull Circuit Design - Gift Wrapping Algorithm

Introduction

This report explains our approach to designing a circuit that implements the Gift Wrapping Algorithm (Jarvis March) to compute the convex hull of a set of points. The circuit takes input from a ROM containing point coordinates and outputs the sum of point indices in the convex hull on seven-segment displays.

Problem Specifications

- Find the convex hull of 3-10 points with coordinates between 0-7
- Handle collinear points correctly
- Input is provided via ROM in a specified format
- Output is the sum of point indices displayed on 7-segment displays

Overall Approach

Our implementation uses a looping mechanism in the following order:

1) We access the ROM via a counter which keeps incrementing on every clock pulse and resets once all points have been accessed.

2) In our first traversal through the ROM we find the point which has the min y value while having a tiebreaker condition for points with same y value that is to find the one among them with the least x value.

3) Once our first point of the convex hull i.e. that is our min point , the nature of our traversal through the ROM changes. From now on through each traversal we will find the next point of the convex hull by computing the cross product of 3 values (i , j , k) :
   i: the latest point of our convex hull
   j: the current most suitable point to be part of convex hull
   k: the next point with which j will be compared.

4) In the second traversal of loop i is our min point while j is given a placeholder value till our traversal begins and then we compute our new values. At the end of each traversal the value stored in the j becomes our new i and our sum counter is incremented. Then these traversals continue till the new value to be stored in I becomes equal to our original point. Now that we have reached the original point has been reached we have made our convex hull and process is stopped.

We have several sub circuit and will proceed through them step by step-

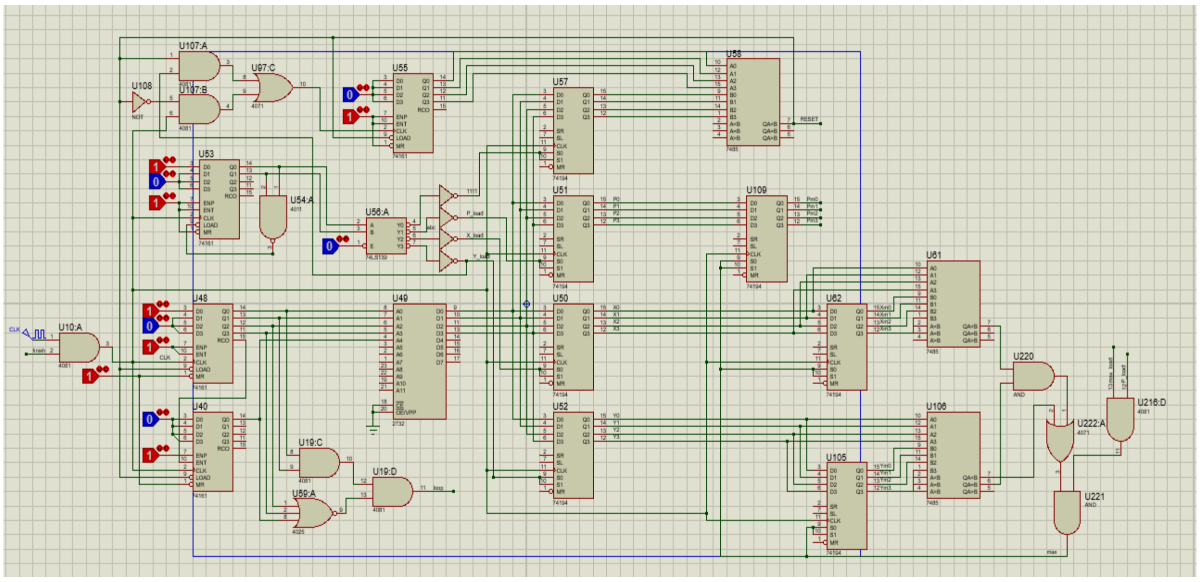
1) ROM interface and min point calculation (U-Number indicates label of part in circuit)

Inputs: Address lines (controlled by counter)

Outputs: Point data (point number, x-coordinate, y-coordinate)

Function: Reads the input points stored in ROM

`



This part of the circuit is used to access ROM and for finding our starting point (min point).

1) We traverse through the ROM using a cascaded counter (U48 AND U40).
   The counter increments at each clock pulse.
   When we reach the end of the array our reset condition activates and we start from the beginning.

2) In our first traversal through the ROM , the counter starts from 0. This gives us the number of points given (stored at address 0). We have used a counter (U55) which increments every time we read the value of a y point. Once this value reaches the value of the number of points (U57) the reset condition activates and all counters are reset to the given preset values.

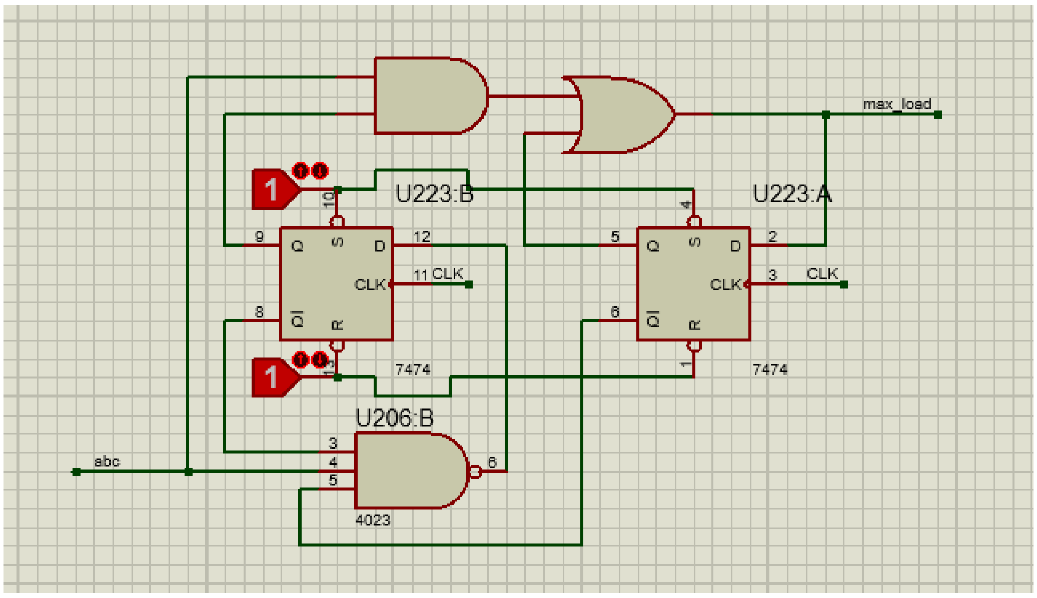
3) For proper storing of values we have used a counter (U53) that first counts from 0 to 3 but then counts from 1 to 3 for the rest of its lifetime. The output of this counter is connected to a demultiplexer which enables the loading of one of four registers which store the number of points (U57), point number (U51) , x value (U50) and y value (U52).

4) In our first traversal through the ROM we find the point which has the min y value while having a tiebreaker condition for points with same y value that is to find the one among them with the least x value. The comparison occurs everytime we load the y value of a point.

5) From the second traversal this part of circuit is simply used for loading values into our k point which we will discuss in our next sub circuit.

6) The Traversal of ROM ends when mentioned above. However we have a separate condition called loop which turns high everytime our main counter connected to ROM input reaches 3. This is when loading into i from j occurs. We have done this to ensure that during cross product comparison comparison with all points was done before next hull poit was finalised.

`



Another FSM used by us to prevent loading into min register before point comparison actually begins.

2) Cross Product

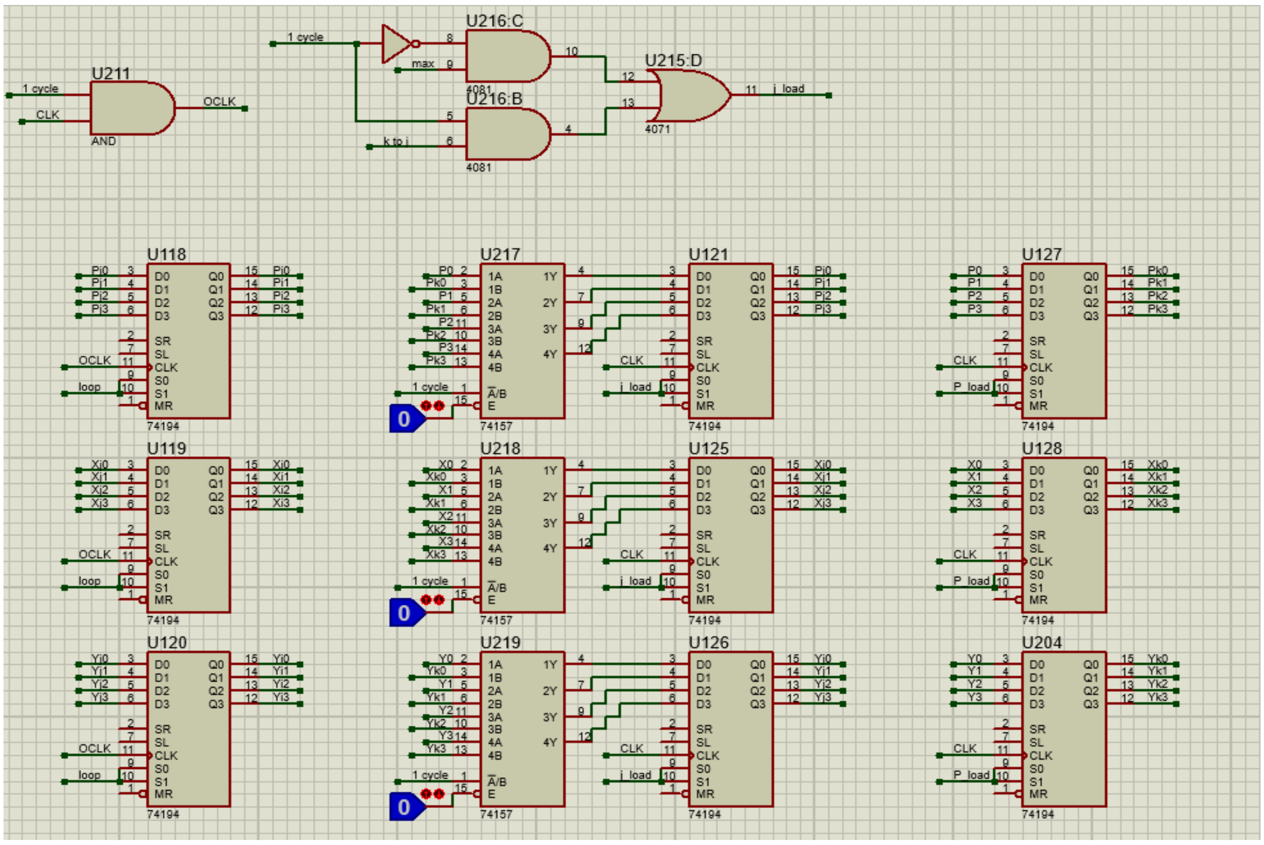


FIG 2

- Once we have found our min point we have got the first point of our convex hull. This point will now help us find our next hull point by using the jarvis march algorithm. To find the point we use the cross product [(Xj-Xi) * (Yk-Yi) - (Xk - Xi) * (Yj-Yi)]
  [Here Xk means x value of k point and similarly for other values]

- The above cross product expression helps determine that for a given pair of points we have to rotate in which direction to reach the third point. The result of expression equate to:
  Cross product is positive: Point k is counterclockwise
  Cross product is negative: Point k is clockwise

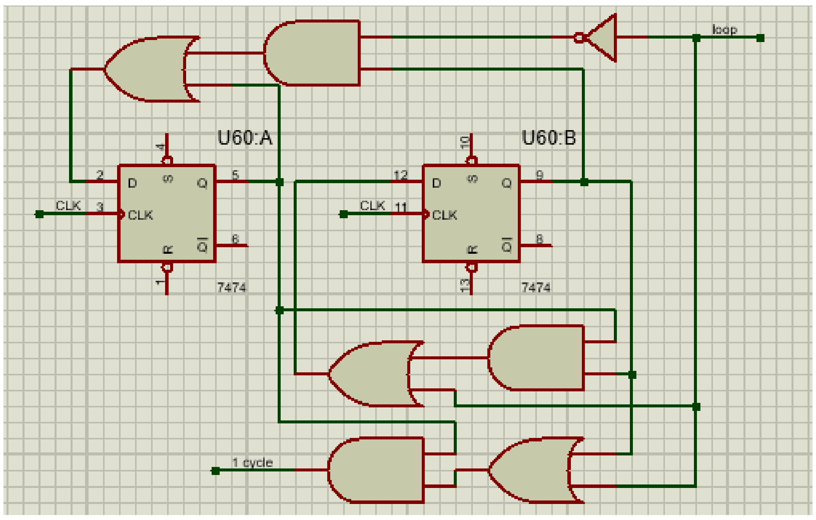
`

Cross product is zero: Point k is collinear

- Our algorithm works by:

1. Finding the bottom point (smallest y-coordinate) as the starting point. (For tie breaking we take point with smallest x among points with smallest y)
2. Repeatedly finding the point with the most counterclockwise orientation
3. Continuing until we reach the starting point again

- So from second traversal everytime we find a point (k) more counterclockwise relative to i (out latest hull point) in comparison to j we store that k point in j . Fig 2 shows how for each point i,j,k we are maintaining 3 separate registers storing point number, x value and y value.
- For each point the load conditions are different:
    - For i: We load the point in j into i everytime we complete a traversal of the ROM [3 pulses after completion of traversal using condition loop]
    - For j: We load in j everytime we get new min in 1st traversal and everytime we get a better candidate for next hull point afterwards. [Condition is shown in fig 2 at top right called j_load]
    - For k: We load in k everytime we get a new point. [Loading occurs]



This FSM is used to generate an output which remains low for 1st traversal of ROM and the turns high for remaining duration. It takes our condition loop as input and the second time loop goes high 1 cycle turns high permanently.

`

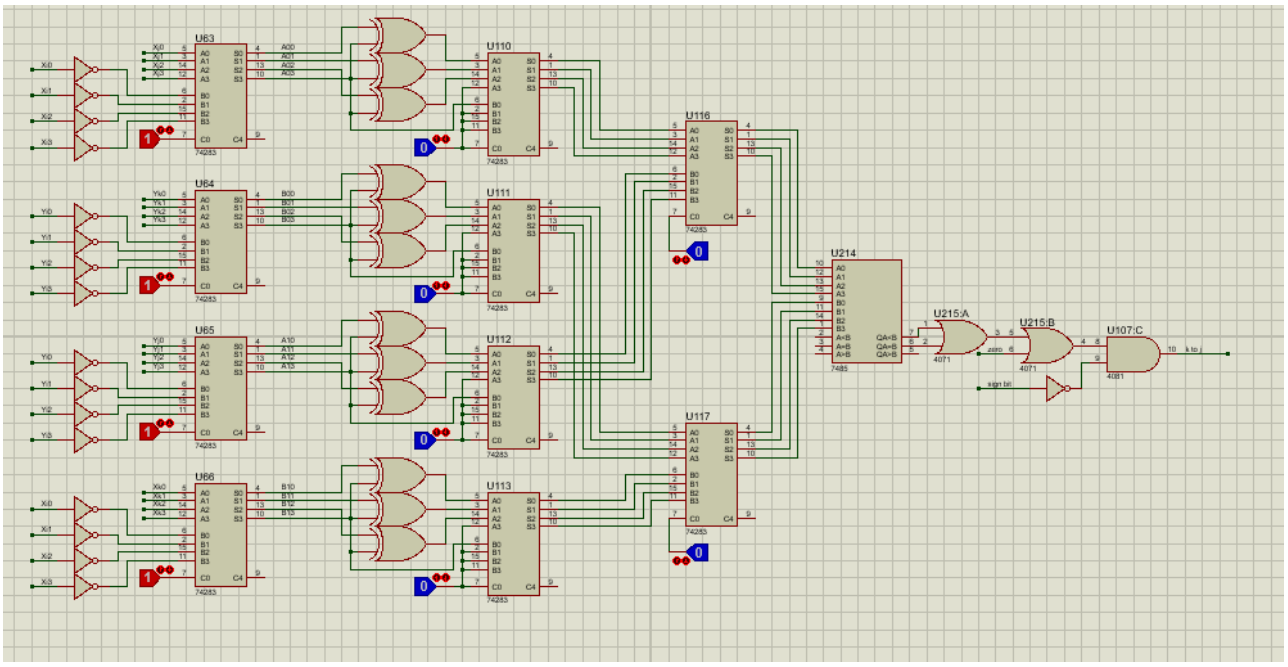


Fig 3

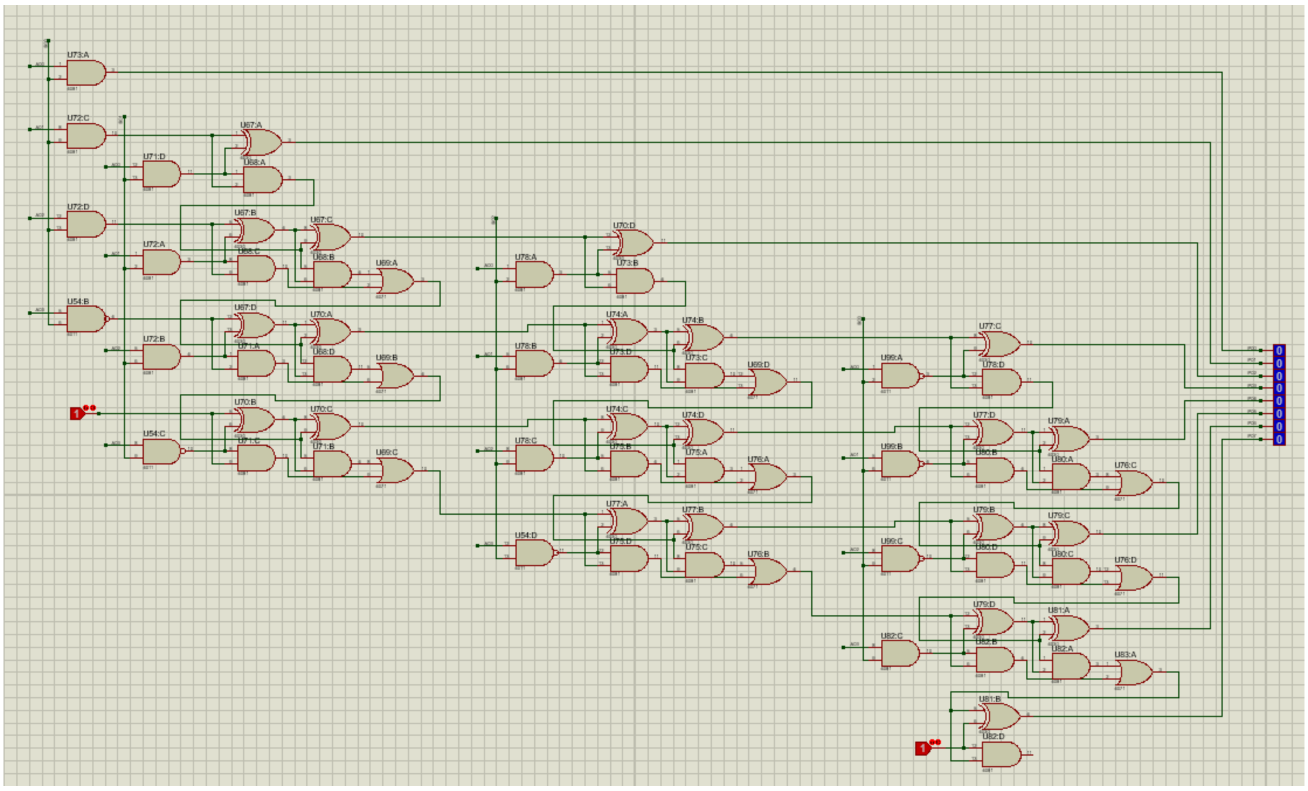


Fig 4

`

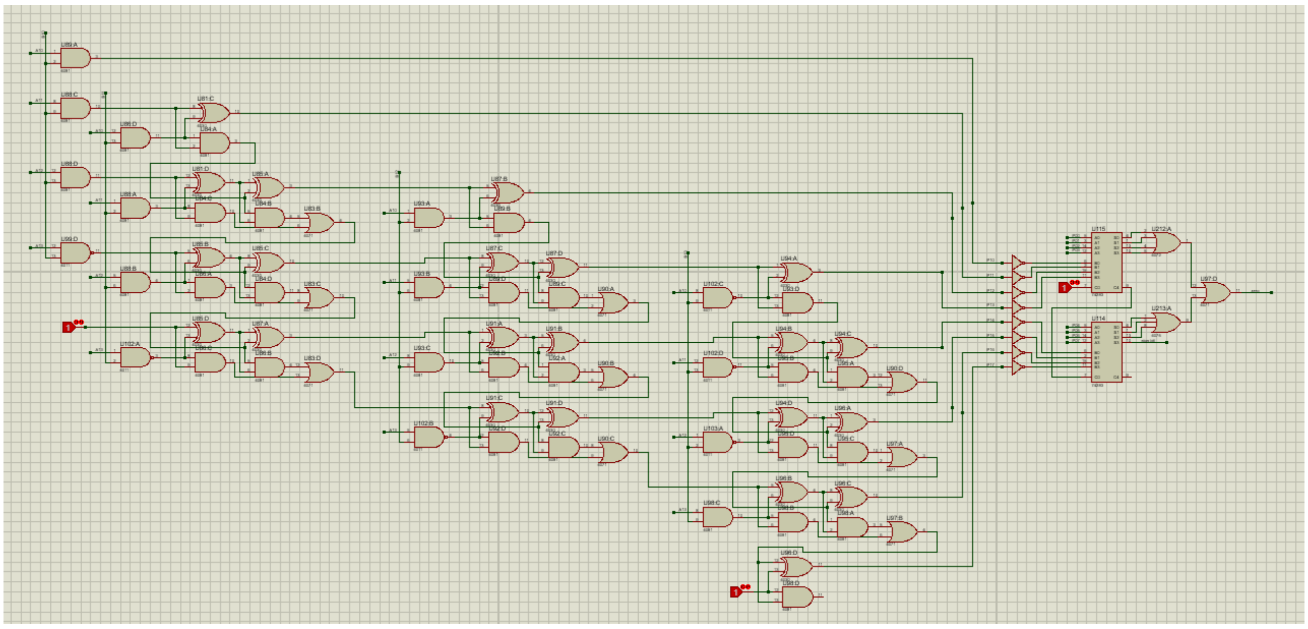


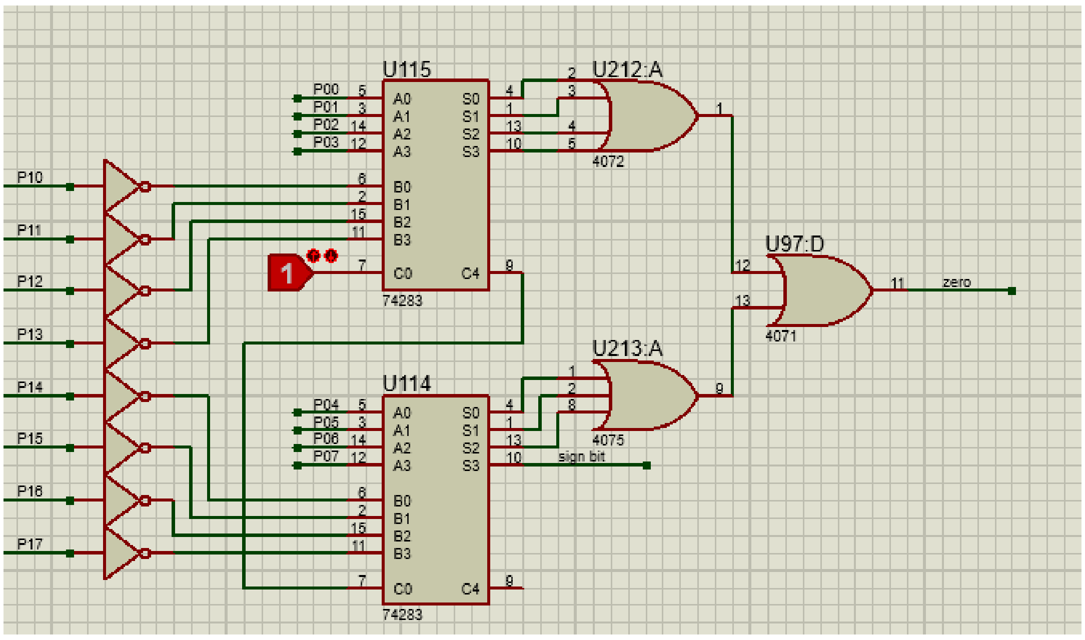
Fig 5

To compute : [(Xj-Xi) * (Yk-Yi) - (Xk - Xi) * (Yj-Yi)   ]

Inputs: Values of i,j,k stored in registers in fig 2

Outputs: Cross Product of input , distance for tiebreaker (cross product is 0)

- The first half of the circuit takes in the coordinates of i,j and k and gives us the differences required in above expression.
- The figure 4 and 5 show the circuits in which these to differences are fed to get the product. The circuits shown in figure 4 and 5 are signed 4 bit number multipliers and give an eight bit output.



This part shown in figure 5 computes the final difference of the 2 products.

- We take the eighth bit of the result after the difference (The sign bit) and determine using it whether cross product is negative or positive.

`

- Afterwards we declare a zero condition which computes the or of all bits except the sign bit to determine whether cross product is zero this occurs when i,j and k are collinear. In that case:

Collinear Point Handler (Second half of circuit in Fig 3)

Inputs: Current hull point i, two collinear points j and k

Outputs: The point farther from i

Function: When points are collinear, selects the farther point

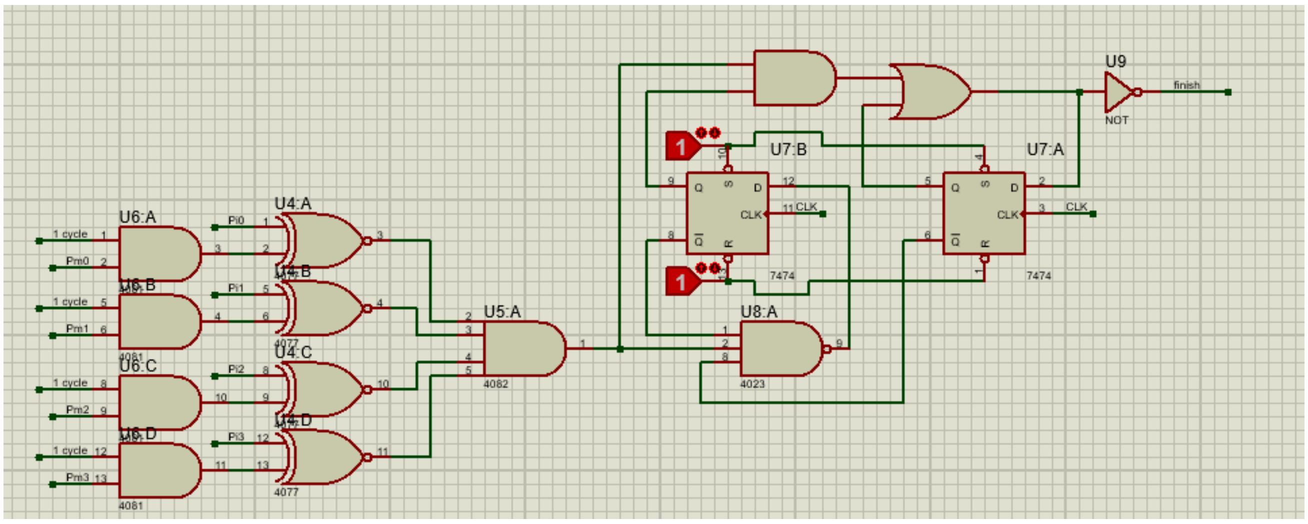
Implementation:

- Calculates Manhattan distance (|Xj - Xi| + |Yj-Yi|) for given points.
- Uses 2's complement for absolute value (using XOR gates)
- Adders to sum the absolute differences
- Comparator to determine which point is farther

Finally there is transfer of point from j to k in 2 cases:

1) Cross product is positive
2) Cross product is 0 but distance of k is greater than j

Termination



Inputs: Next hull point i , min point

Outputs: Finish signal

Finally from second traversal onwards after every time we obtain our next hull point we use this circuit to compare it to our first hull point that is the min point. The moment our next finalised hull point becomes equal to our min point the circuit gives a finish signal which is used to stop the whole convex hull circuit.

`

Summation and Display Units

Inputs: Hull point indices

Outputs: Sum displayed on 7-segment displays

Implementation:

- Adder circuit (74283) to calculate running sum
- BCD converter for display
- 7-segment display drivers (7447)
- 7-segment displays

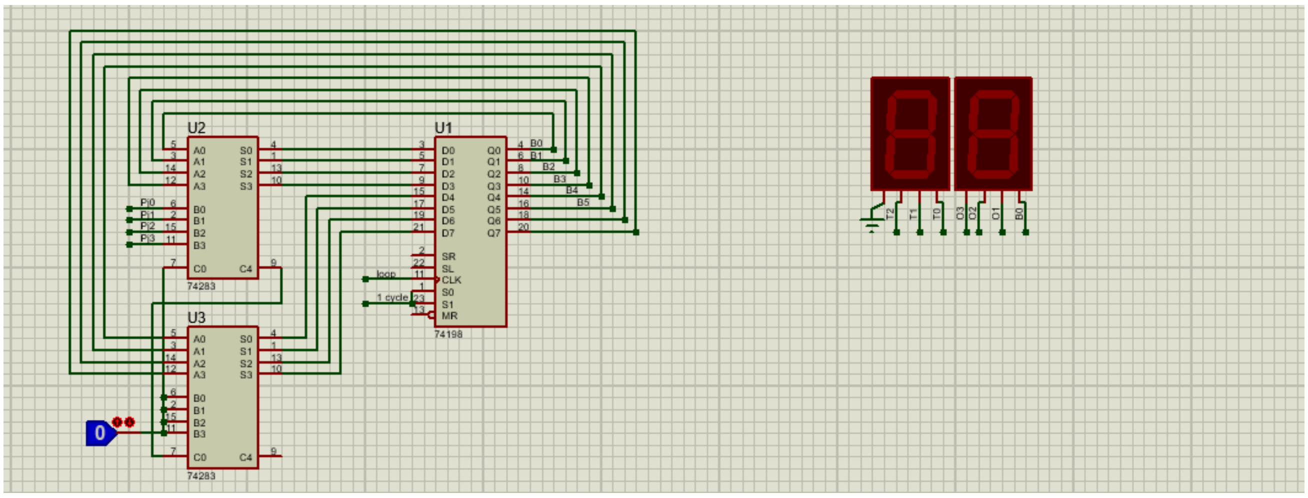
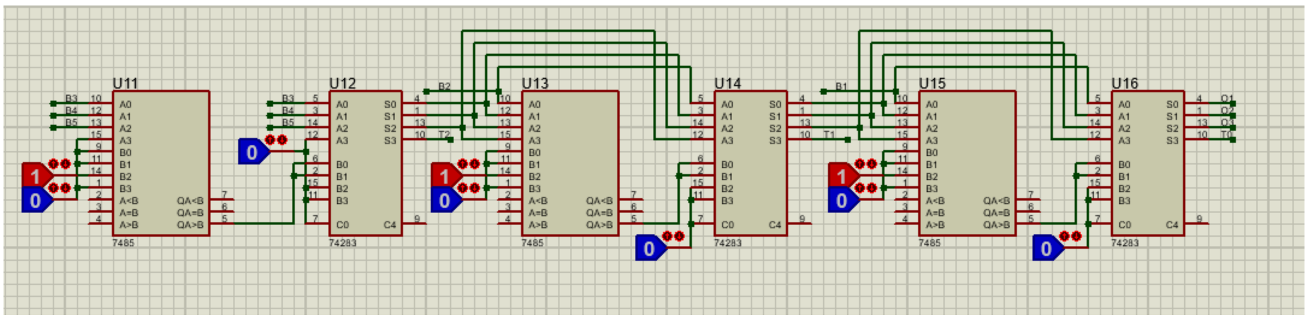


Fig: Shows our adder circuit it gives output in binary so circuit below converts binary to BCD for display.



.

Conclusion

Our circuit design successfully implements the Gift Wrapping Algorithm to find the convex hull of a set of points. By using a state machine approach with specialized sub-circuits for orientation calculation and collinear point handling, we ensure correct results even in edge cases. The implementation efficiently uses the available components while minimizing cost.

Final Cost : 225.3

`