**Introduction to Convex Hull and Jarvis March Algorithm**

The **Convex Hull** is the set of points that form the smallest convex polygon enclosing a given set of points. We use the **Jarvis** **March Algorithm (Gift Wrapping)** to compute this.

### **Algorithm Steps:**

1. Select the leftmost point as the starting point.
2. Choose the next point so that all other points are to its left.
3. Repeat until returning to the starting point.
4. The collected points form the convex hull.

## **Implementation Details**

### **Design Specifications**

* The convex hull is implemented using **digital circuits**.
* The **ROM (Read-Only Memory)** stores the input points.
* The number of points n is stored at **index 0**.
* The points are stored in the format: {n, (point\_no, x, y)}.

### **Circuit Components and Their Functions:**

#### **U21 (Counter):**

* Determines whether the extracted ROM value represents the number of points, a point number, or its coordinates.
* Loaded to 01.

#### **Shift Registers (POINT, NUMBER, X COR, Y COR):**

* Use U21 counter’s lines to load and assign values.

| **I1** | **I0** | **Load** |
| --- | --- | --- |
| 0 | 0 | Number |
| 0 | 1 | Point |
| 1 | 0 | X Coordinate |
| 1 | 1 | Y Coordinate |

#### **U4, U5 (Counters):**

* Provide address lines ranging from 0 to 3N.
* Loaded to 00001.

#### **3N (Computation for Total Points Storage):**

* Computes 3 \* number of points.
* Compared with address lines; when equal, U4 resets to 01 and U5 resets to 00.

#### **U2 (Nested Loop Counter):**

* Resets after every **4 loops**.
* Assigns x = 1 (load) before resetting.

#### **U7, U28, U29 (Shift Registers):**

* Store the leftmost point.
* All initialized with value 8.

#### **U6, U115 (Comparators):**

* **U6**: Checks if min y > y in ROM, outputs 1.
* If min y == y in ROM, **U115** checks if min x > x in ROM.
* If min x > x, store the new minimum values in U7, U28, U29.
* **Load condition for U7, U28, U29:**Load = (min y > y) + (min y == y) . (min x > x)

#### **Finite State Machine (FSM) for Selecting Points:**

* Implemented using D flip-flops **S0:B, S1:B, S0:A**.
* Flip-flop update rule: D(t+1) = input, initialized to 000.
* State transitions:

| Previous | Next | Purpose |
| --- | --- | --- |
| s1=0, s0=0, y=0 | s1=0, s0=1, y=0 | First leftmost point found and stored |
| s1=0, s0=1, y=0/1 | s1=1, s0=0, y=1/1 | Assigns minimum value to P1, X1, Y1 / store value from P2 to P1 |
| s1=1, s0=0, y=1 | s1=1, s0=1, y=1 | Checks the next probable value and stores in P2, X2, Y2 |
| s1=1, s0=1, y=1 | s1=0, s0=1, y=1 | Stores values from P2 to P1 |

* Clock cycle X: Updates every **4 ROM iterations**.

#### **MUX Components (U79, U83, U85, U84, U87, U88):**

* Determine which values go to P1 and P2 registers based on FSM states.
* Initially set with leftmost point values stored in U7, U28, U29.

#### **Registers for Storing Convex Hull Points:**

* **U34-32 (P1, X1, Y1)**: Stores selected best point when s0=1, meaning comparator is inactive.
* **U35-37 (P2, X2, Y2)**: Stores next probable point in 101 FSM state, finalizes best point in 111 state.
* **U35-37 (P3, X3, Y3)**: Gives iteration of points on a clk when we get all the set of POINT X, COR,Y COR which stores in it when I0 and I1 gets 11.
* **Load condition for U35-37:**Load = y' . s0 . s1' + s1 . s0' . LOAD  
  *(Circuit elements for Load: G1, U90, K:A, U89)*

### **Handling Collinear and Non-Collinear Points:**

For this we will open the statement and put positive terms on one side and - ve terms on other since subtraction might be difficult. We have a predesigned 4\*4 multiplier circuit to multiply numbers.

#### **Non-Collinear Case:**

* Using cross-product formula:  
  (x3 - x1) \* (y2 - y1) - (x2 - x1) \* (y3 - y1) < 0
  + If **negative** or P3 == P2 (where P1 ≠ P3), then LOAD for U35-37 is set to 1.

#### **Collinear Case:**

* Uses distance formula to choose the farther point:  
  If (x3 - x1) \* (y2 - y1) - (x2 - x1) \* (y3 - y1) == 0 Then compare: x3² + y3² + 2(x1 \* x2 + y1 \* y2) > x2² + y2² + 2(x1 \* x3 + y1 \* y3) If true, set LOAD = 1.

### **Summation of Convex Hull Points:**

* **Adder Circuit (U94, U95)**:
  1. 8-bit adder, connected to register inputs **(U97, U96)**.
  2. Register outputs connected to adder input.
  3. The sum is stored and later converted using **Binary to BCD converter** for display.
  4. Clock condition: S0' . S1 . (Output of U99:A)'.
  5. (Output of U99:A)'- becomes 0 when FSM is in state of 101 and P3 = the leftmost point MP0

## **Limitations**

Number of points = [3,10]

X coordinate = [0,7]

Y coordinate = [0,7]  
  
**COST: 297.2**