1. Follow the design procedure to design a circuit to add two 2-bit binary numbers. Show truth table, K-maps, sum-of-products or product-of-sums, and draw a circuit.

2. For the following equations draw a K-map, list the prime implicants, and list the essential prime implicants.

$$X(A,B,C,D) = \Sigma m(2,3,4,5,7,8,9,10,12,15)$$
$$Y(A,B,C,D) = \Pi M(1,2,5,6,8,10,11)$$

3. Follow the design procedure to design a 2-bit by 2-bit multiplier. Show truth table, K-maps, sum-of-products, or product-of-sums, and draw a circuit.

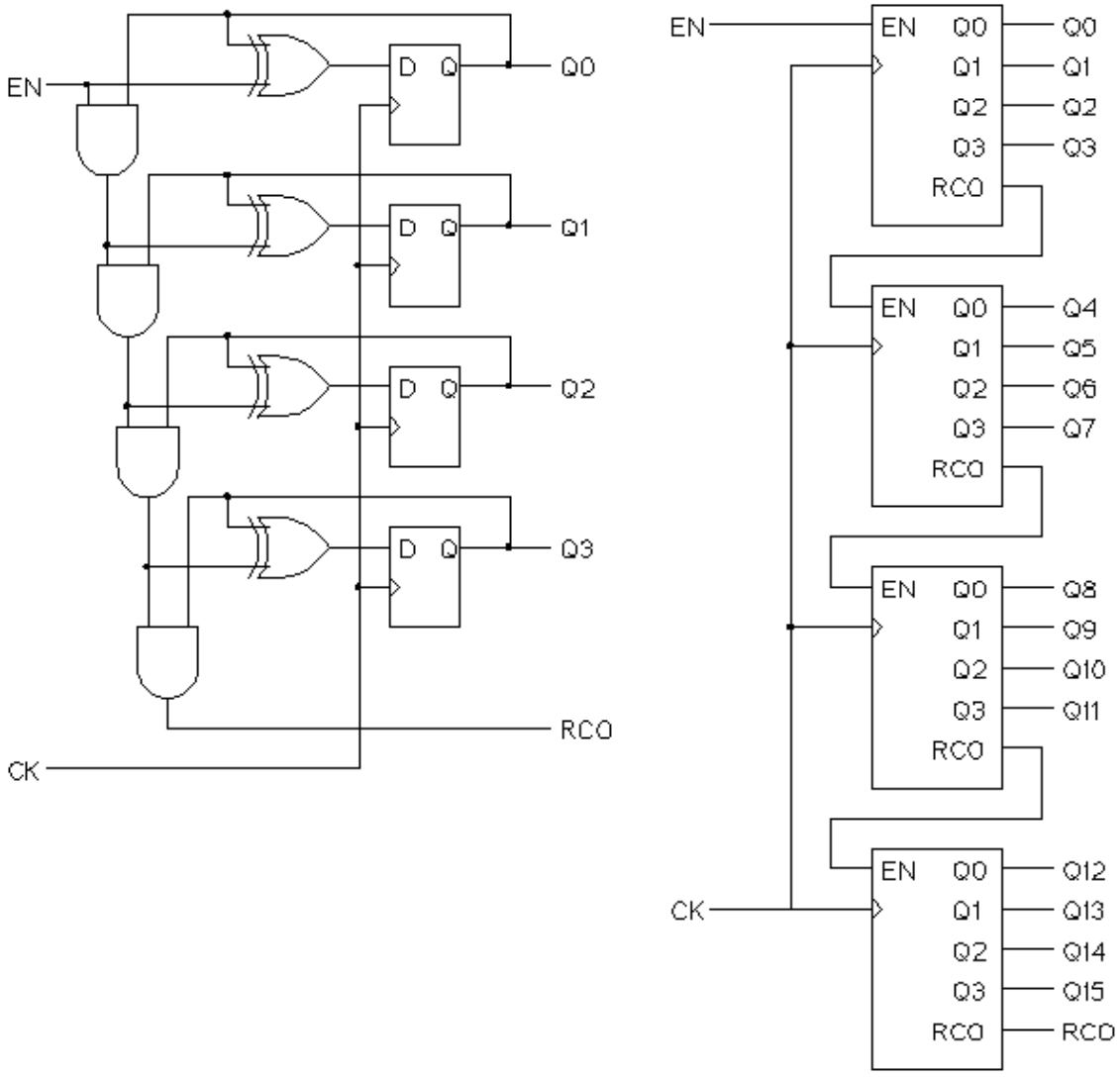
4. Using the design procedure design a 4-bit input priority encoder. The priority encoder has the following requirements:

- Four input bits
- The binary output value is the bit position of the most significant bit that is a one.
- The binary output has enough bits to represent the bit position of the most significant input bit.
- There is an additional output bit to indicate an error.
- If all of the input bits are zeros the error output signal is one.

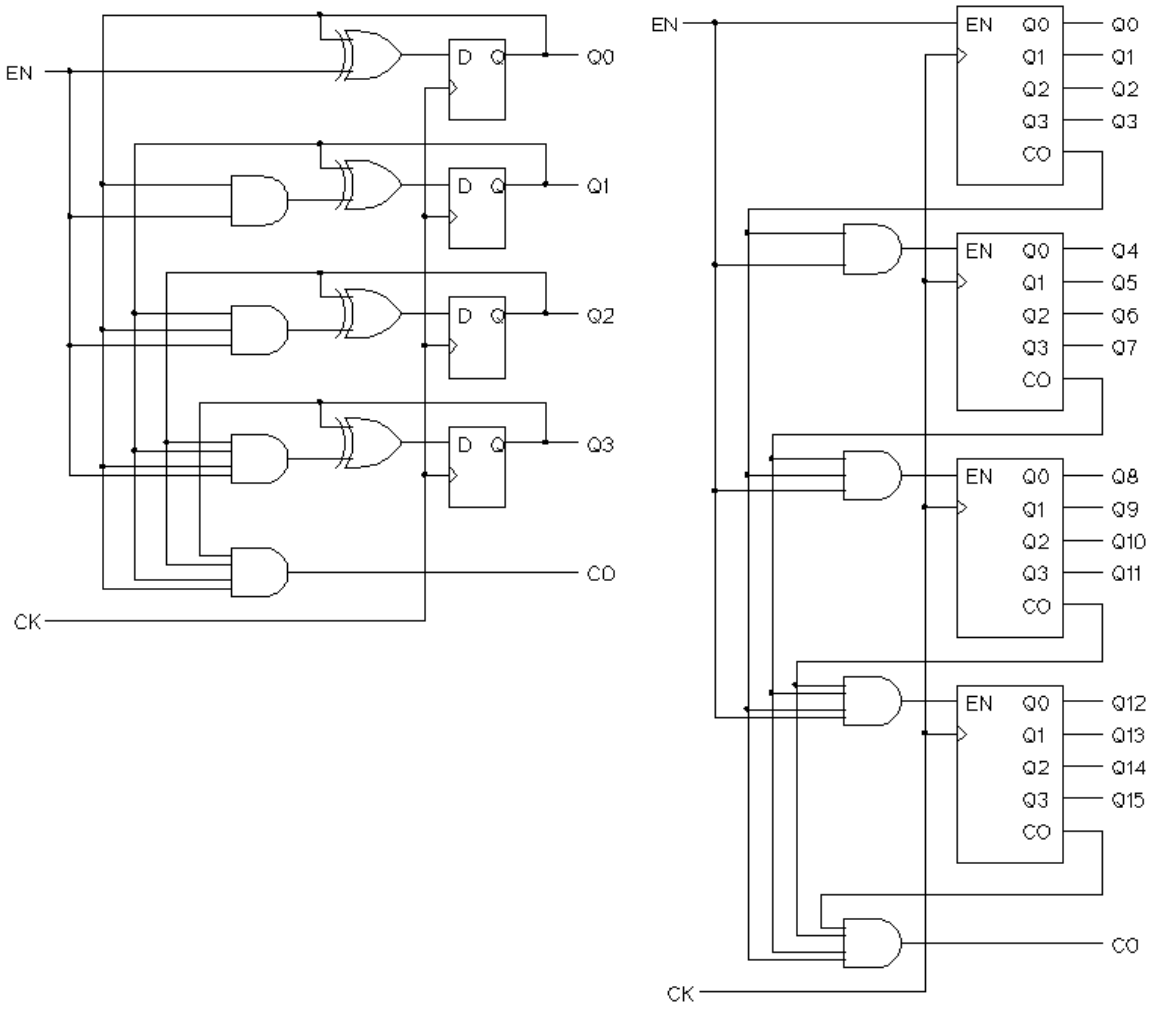
Part A: Create a truth table.
Part B: Write optimized sum-of-products or product-of-sums equations for your circuit.
Part C: Draw a circuit schematic of your design.

5. Design a Moore model circuit (where the output can change only on a positive clock edge) that detects the bit pattern 1100110 (least significant bit first). This pattern can be detected even if it overlaps. Use D-type flip-flops. The output detection bit goes 1 when the pattern is detected and 0 otherwise. Show a state table, state diagram, how you simplified the logic to sum-of-products or product-of-sums. Do NOT draw a circuit.

6. Analyze the timing for the following 16 bit counters. What is the maximum operating frequency of each 16 bit counter? An AND gate has a 2 nS propagation delay, an XOR gate has 3 nS, and a flip-flop has 4 nS. The setup time for a flip-flop is 1 nS and the hold time is 0.5 nS.



First Counter using serial gating



Second Counter using parallel gating

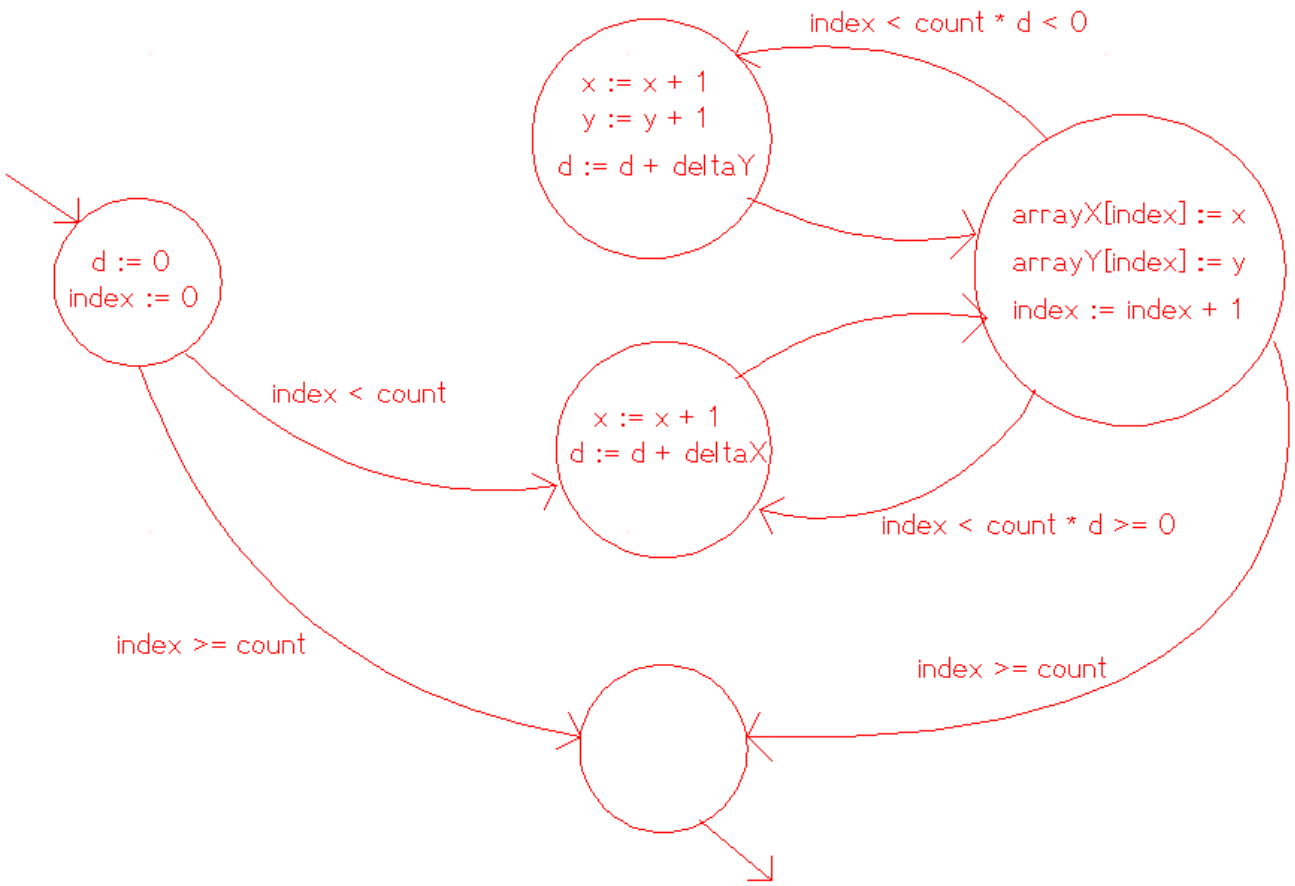
7. Part A. Design a high level state machine that generates a set of points of a line in the first quadrant using Bresenham's line drawing algorithm. Your circuit can only perform one arithmetic operation per clock cycle. The index increment, x increment, and y increment do not count as arithmetic operations since they are implemented as counters. A C function to generate a set of points can be written as the following:

```
/*
   deltaX is always negative.
   deltaY is always positive
   x and y are the starting point
*/
void lineDraw(int count, int deltaX, int deltaY,
              int x, int y, int* arrayX,
              int* arrayY)
{
   int d = 0;
   for( int index = 0; index < count; ++index)
   {
      x = x + 1;  // always increment x by 1
      if (d < 0)
      {
         y = y + 1;  // sometimes increment y
         d = d + deltaY;   // makes d positive again
      }
      else
      {
         d = d + deltaX;   // reduce d by some small amount
      }
      arrayX[index] = x;
      arrayY[index] = y;
   }
}
```

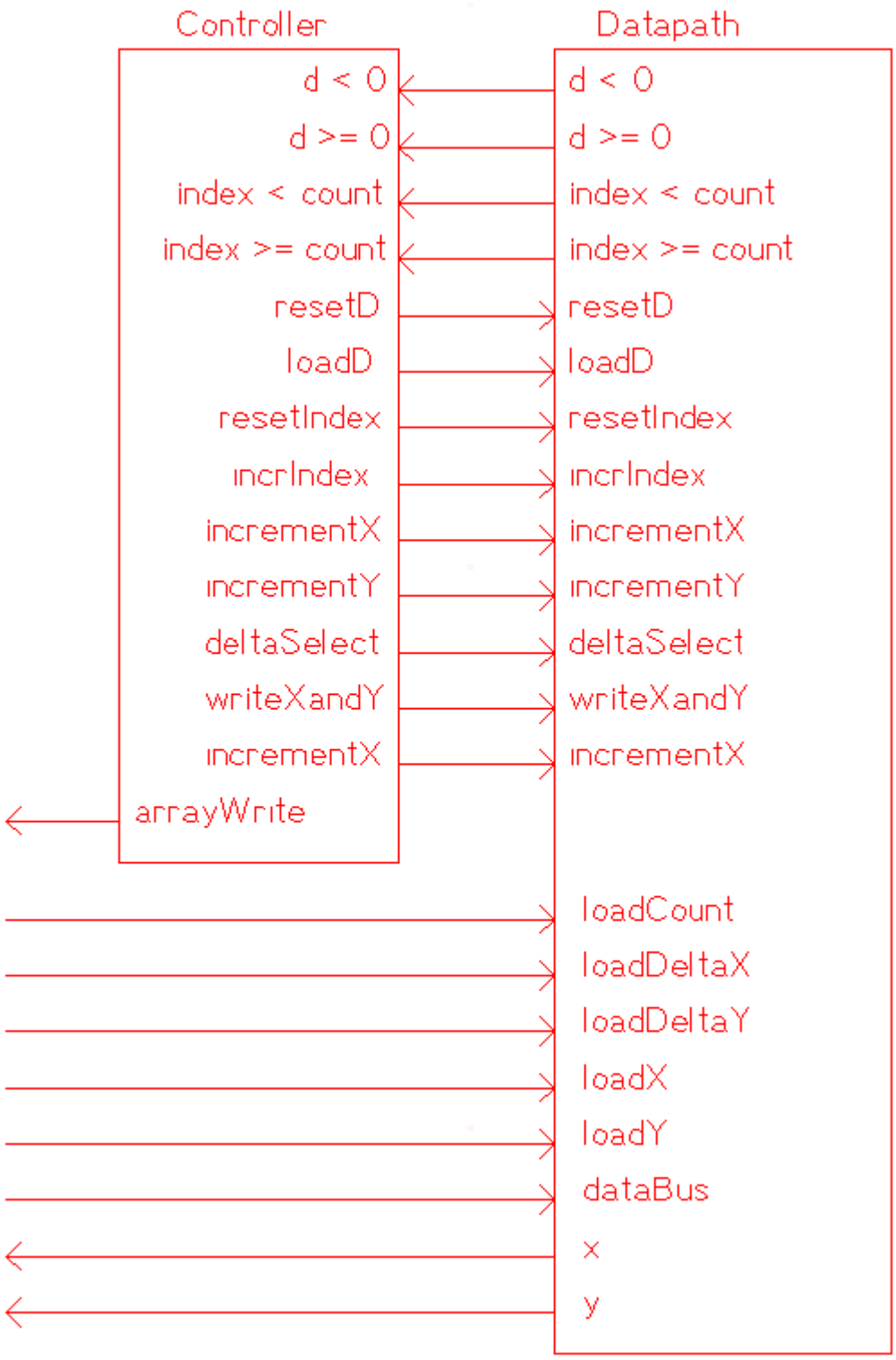
Inputs: x(int), y(int), deltaX(int), deltaY(int), count(int)

Outputs: arrayX[](int), arrayY[](int)

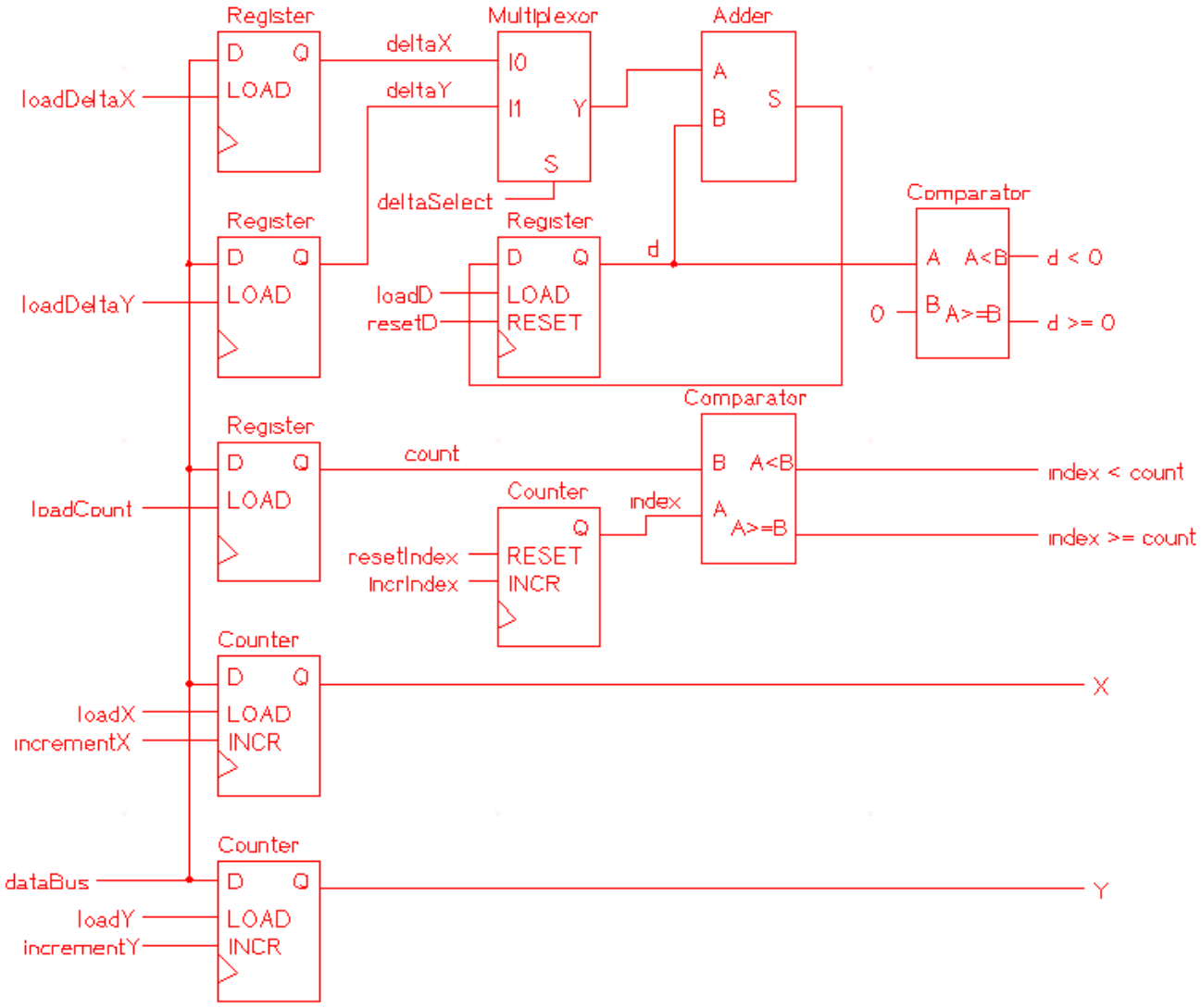
Local Storage: d(int), index(int)



7 Part B. Draw a block diagram of the controller and datapath for the Bresenham circuit.



7 Part C. Draw a block diagram of the Bresenham datapath. Show registers, counters, comparators, adders, and the control and status signals.



8. Design a high level state machine that generates a set of points on a circle using the sine and cosine addition formulas. Your circuit can only perform two arithmetic operations per clock cycle. The index increment does not count as an arithmetic operation. The formulas are as follows:

$$SIN(\Theta + \Delta\Theta) = COS(\Theta) * SIN(\Delta\Theta) + SIN(\Theta) * COS(\Delta\Theta)$$
$$COS(\Theta + \Delta\Theta) = COS(\Theta) * COS(\Delta\Theta) - SIN(\Theta) * SIN(\Delta\Theta)$$

Where $\Delta\Theta$ is the angle between points on the circle and is a constant. A C function to generate a set of points using the above formulas can be written as the following:

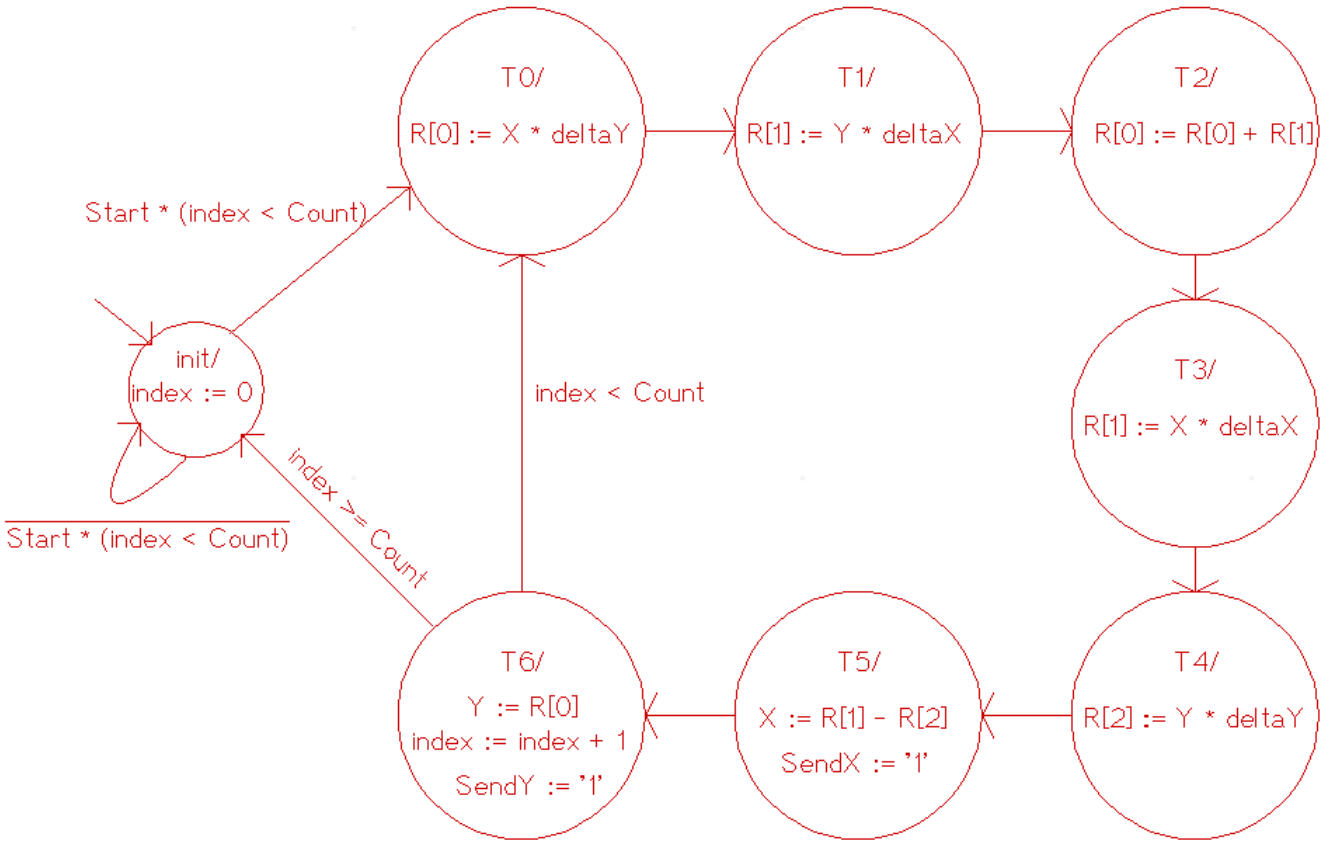
```
/*
   deltaX is cos(deltaTheta)
   deltaY is sin(deltaTheta)
   x and y are the starting point
*/
void circle(int count, float deltaX, float deltaY, float x,
float y, float* arrayX, float* arrayY)
{
   for( int index = 0; index < count; ++index)
   {
      float tempX = x * deltaX – y * deltaY;
      y = x * deltaY + y * deltaX;
      x = tempX;
      arrayX[index] = x;
      arrayY[index] = y;
   }
}
```

One way of solving this problem is to write down the register transfers for each step. In the JEE 3620 class, we would make a table of time steps and register transfers. The idea for the transfers in this design is based on all registers sharing a single input bus. This means only one transfer per clock cycle. There are other designs based on multiple transfers per clock cycle.

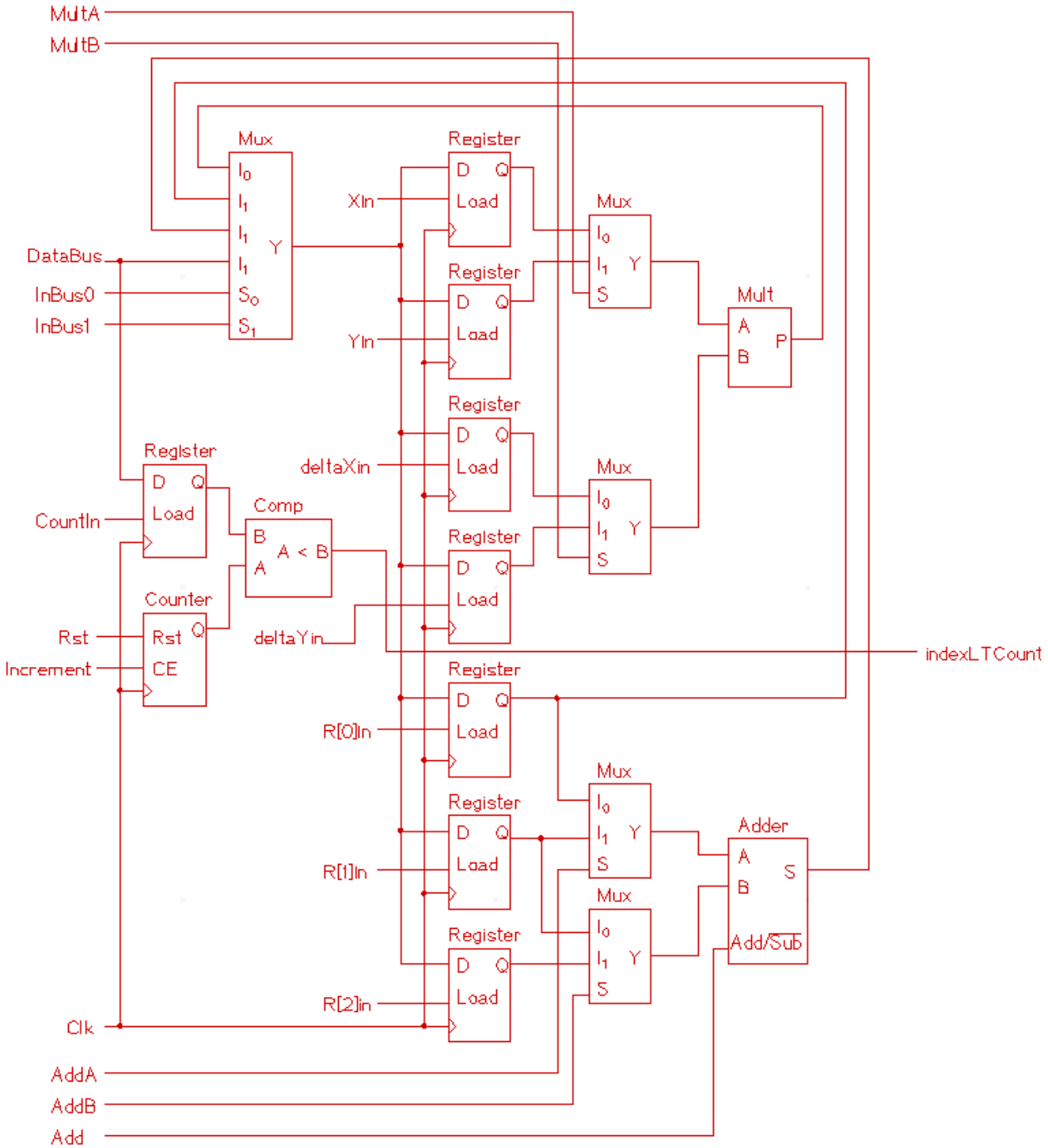
Time Step	Register Transfer
T0	R[0] ← X * deltaY
T1	R[1] ← Y * deltaX
T2	R[0] ← R[0] + R[1]
T3	R[1] ← X * deltaX
T4	R[2] ← Y * deltaY
T5	X ← R[1] - R[2]
T6	Y ← R[0] : index ← index + 1

There are 3 general purpose registers used to hold temporary variables, R[0], R[1], and R[2]. There is nothing special about these three register names. Below is the HLSM for this problem using the Time Step from the table as the state. One thing to note about the HLSM is that it doesn't show how X, Y, deltaX, deltaY, and Count are loaded. How these are loaded is a detail in how the interface is set up between the external circuit and this circuit. This design assumes that the above 5 registers are loaded by the external circuit then a Start signal is applied to start the circuit.

Note: float is a 32-bit floating point, int is a 32-bit signed.
Inputs: X (float), Y (float), deltaX (float),
deltaY (float), Count (int), Start (bit)
Outputs: X (float), Y (float), Stop (bit), SendX (bit),
SendY (bit)
Local Storage: R[2..0] (float), index (int)



If the question asked for a circuit or block diagram of the circuit, it would look as shown below. This is the datapath part of the circuit. To design the controller you still need to create a FSM. In the JEE 3620 class we would add a third column to the previous table to show the control sequences.



9. Design a mixed SRAM and ROM 2Kx8 memory using 1Kx4 SRAMS and 1Kx4 ROMs and the minimum amount of additional combinational logic necessary. The least significant 1K words of memory must be read only while the most significant 1K words must be read and write.

10. Complete the memory circuit below. The ROM address range is 0 to 8191. The RAM address range is 8192 to 12287. The word bit depth is 8 bits. All ROMs are the same size. All RAMs are the same size. Show the memory sizes you have selected, address bit ranges, data bit ranges, and any missing signal lines. You may add a minimal amount of combinational logic.