**Lab 2**

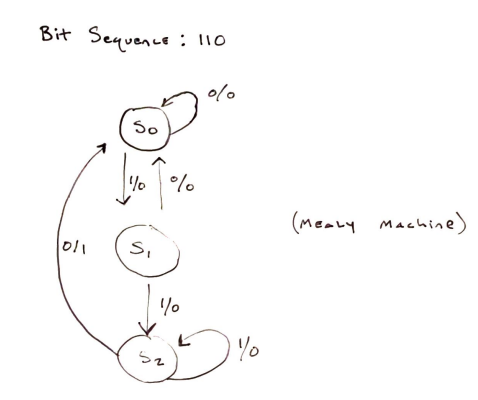
**Robert Everhart**

**Jacob Navarro**

**Joseph Hyatt**

**9 January 2021**

**CSE 4010 Winter 2021**

**State Diagram:**

**It appears to be a Mealy Machine since the output relies on the current state and the current input values.**

**State Transition Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Present State** | **Next State** | **Output** |  | |
|  | **x = 0** | **x = 1** | **x = 0** | **x = 1** |
| **A** | **A** | **B** | **0** | **0** |
| **B** | **A** | **C** | **0** | **0** |
| **C** | **A** | **C** | **1** | **0** |

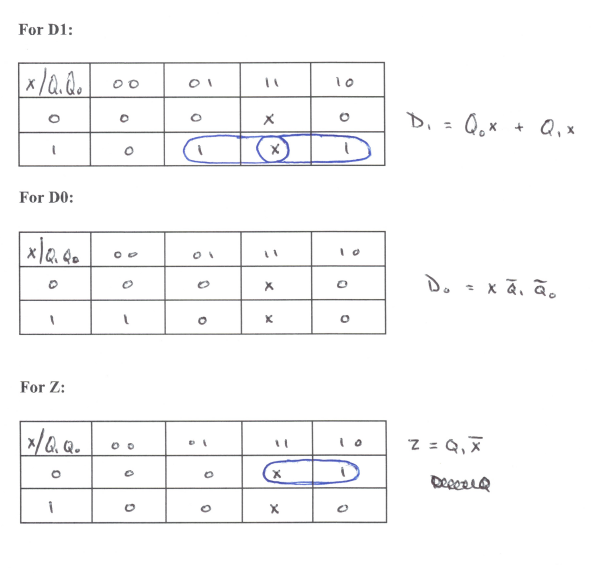
**Binary Encoding:**

|  |  |  |
| --- | --- | --- |
| **States** | **Q1** | **Q0** |
| **A** | **0** | **0** |
| **B** | **0** | **1** |
| **C** | **1** | **0** |

**Excitation Table:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Present State** | **Input** | **Next State** | **Excitation Input** | **Output** |  | | |
| **Q1** | **Q0** | **X** | **Q1(n+1)** | **Q0(n+1)** | **D1** | **D0** | **Z** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |

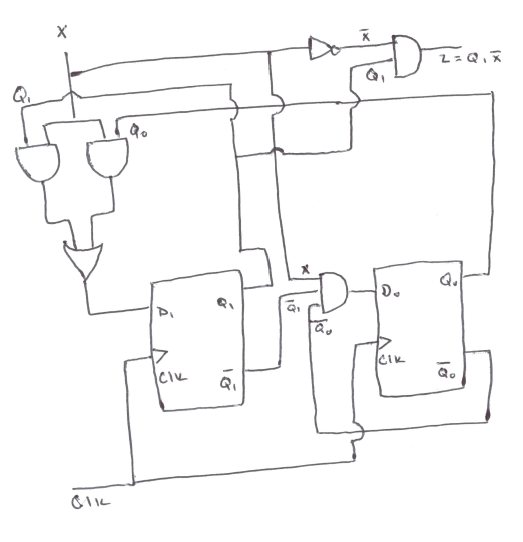
**3-Variable K-Maps:**

****

**Logic Circuit:**

For simplified functions:

D1 = Q0x + Q1x, D2 = xQ1’Q0’, z = Q1x’

****

Mealy finite state machine for locating a sequence of “110.”

**seqrec.v**

**// Mealy finite state machine for locating a sequence of "110."**

**module seqrec(Y, X, RST, CLK);**

**input X, RST, CLK;**

**output Y;**

**reg Y;**

**reg [1:0] Prstate, Nxtstate;**

**// Reset, clock**

**always @ (posedge CLK or negedge RST)**

**if(~RST) Prstate = 2'b00;**

**else Prstate = Nxtstate;**

**// State Transitions**

**always @ (Prstate or X)**

**case (Prstate)**

**2'b00: if(X) Nxtstate = 2'b01; // Initial state**

**else Nxtstate = 2'b00;**

**2'b01: if(X) Nxtstate = 2'b10;**

**else Nxtstate = 2'b00;**

**2'b10: if(X) Nxtstate = 2'b10; // Loop if we encounter many 1 bits.**

**else Nxtstate = 2'b00;**

**endcase**

**// Handles output**

**always @ (Prstate or X)**

**case (Prstate)**

**2'b00: Y = 0;**

**2'b01: Y = 0;**

**2'b10: if(X) Y = 0; else Y = 1;**

**endcase**

**endmodule**

The corresponding test bench:

**seqrec\_tb.v**

**// Test bench for seqrec.v**

**module seqrec\_tb();**

**reg x, CLK, RST;**

**reg [14:0] sequence;**

**wire y;**

**integer i = 0;**

**seqrec DUT (y, x, RST, CLK);**

**initial**

**begin**

**// 010110101101100**

**x = 0;**

**CLK = 0;**

**RST = 0;**

**sequence = 15'b010110101101100;**

**// Reset needs to be on**

**#10 RST = 1;**

**for( i = 0; i < 15; i = i + 1)**

**begin**

**// Simulate the clock, check each bit of the sequence.**

**x = sequence[14-i];**

**#5 CLK = 1;**

**#5 CLK = 0;**

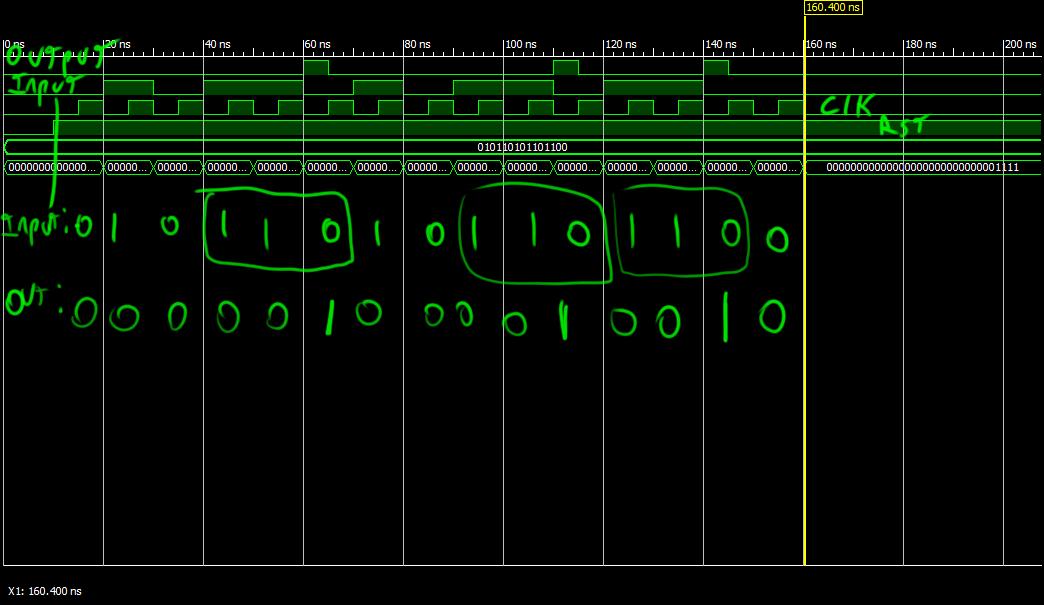
**end**

**end**

**endmodule**

The simulation of the seqrec using the test bench:

The output value turns 1 when it finds the ‘110’ input sequence as displayed in the simulation.

****