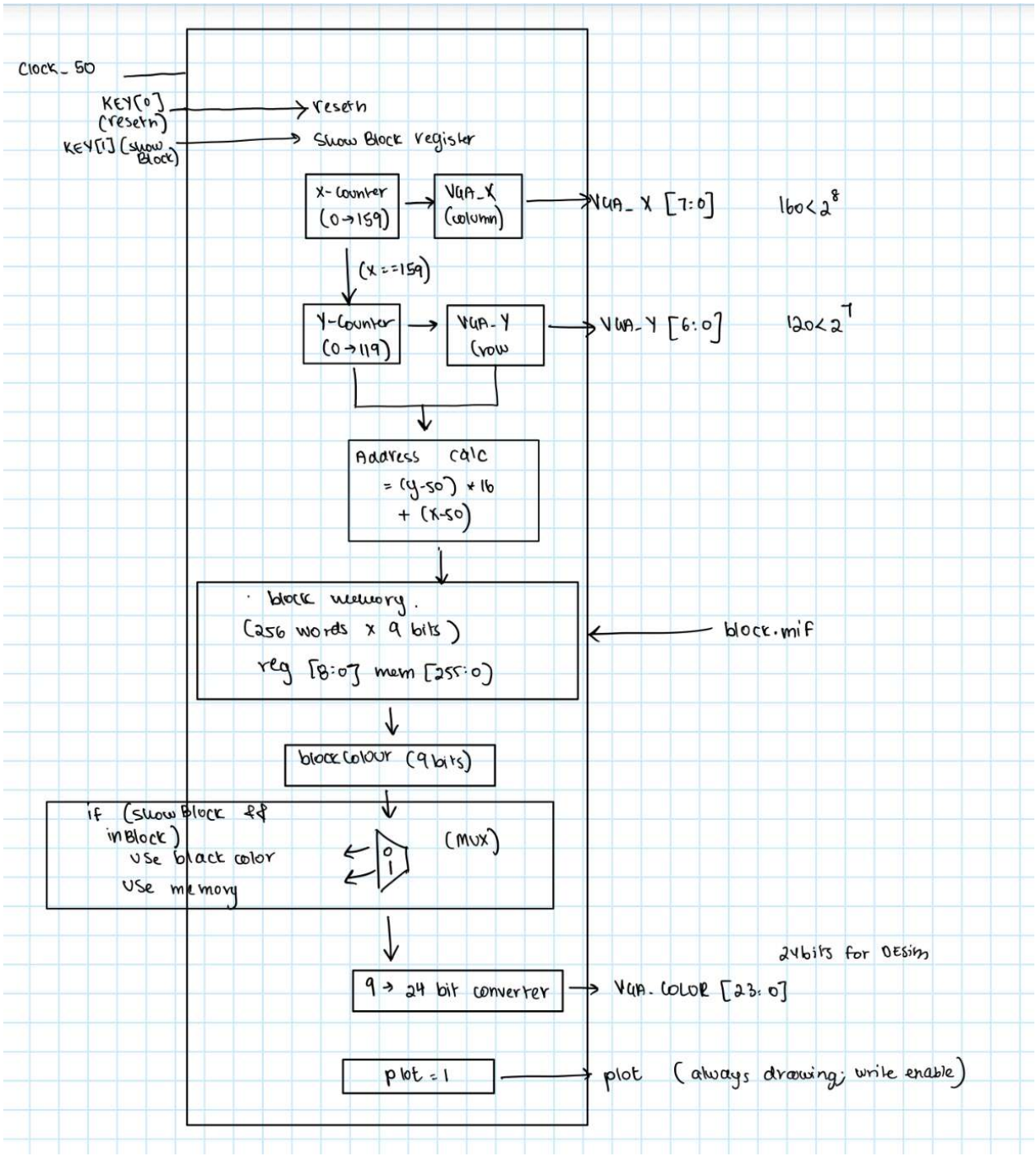
2) Memory (ROM)

Block Diagram:



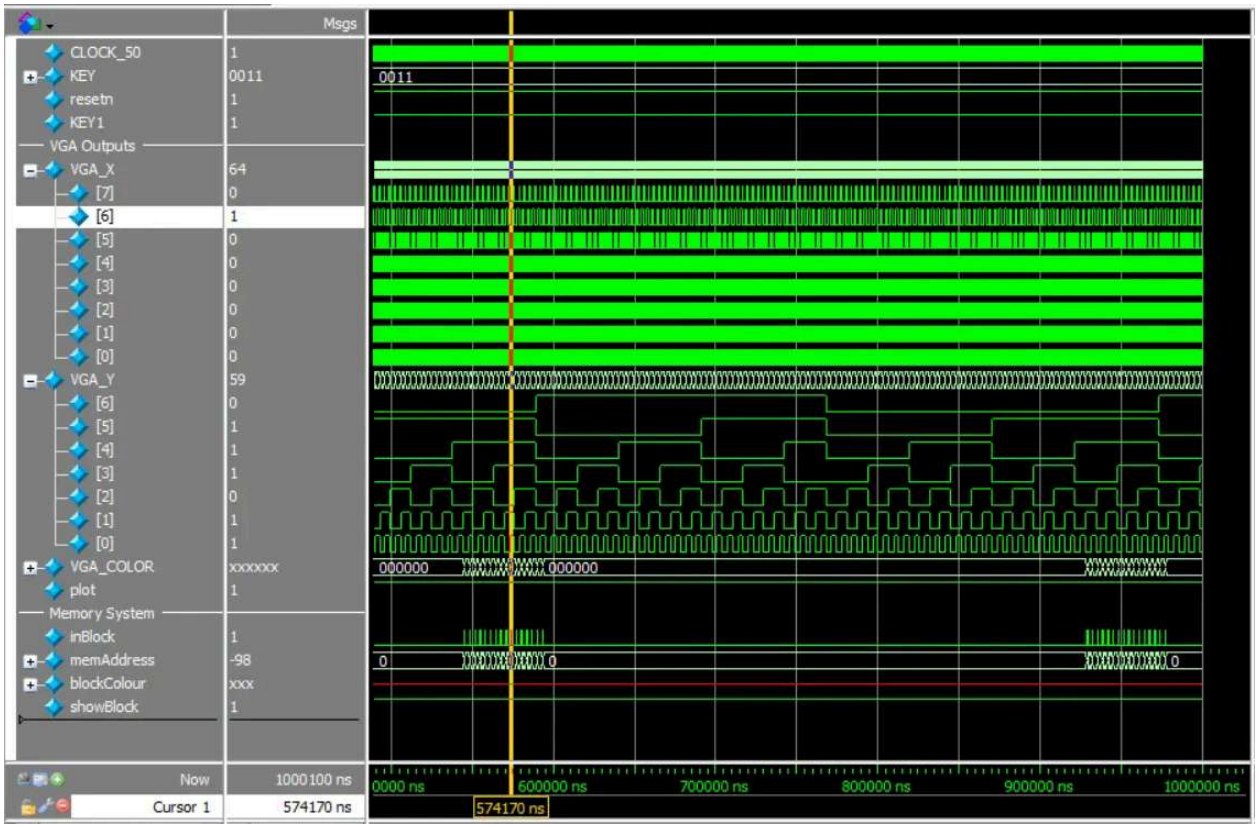
Code:

```
//reading block from memory

//In reset state, where KEY[0]=0, black screen and counters at at origin.
//During normal operation, when KEY[0] = 1, counters scan the screen but it is still black
//When KEY[1] is pressed, the square is shown on the screen at (50,50)



module memory(CLOCK_50, KEY, VGA_X, VGA_Y, VGA_COLOR, plot);
   input wire CLOCK_50;

   input wire [3:0] KEY; // KEY[0] for resetn, KEY[1] to show block

   //screen dimension from 160*120, so 8 bits for 2^8 and 7 bits for 2^7 respectively
   output reg [7:0] VGA_X;
   output reg [6:0] VGA_Y;
   output wire [23:0] VGA_COLOR; //for 24 bit colour
   output reg plot;

   wire resetn = KEY[0]; //active low reset (when low (!resetn), reset is triggered)
   wire KEY1 = KEY[1]; //press key1 to show block

   //X counter (columns)
   always @(posedge CLOCK_50)
   begin
      if(!resetn) //when reset is 0, it is triggered
         VGA_X <=0;
      else if (VGA_X==159)
         VGA_X <=0;
      else VGA_X<=VGA_X+1;
   end

   //X counter essentially works like a cursor across the screen in the x direction
   //when Clockedge is positive, and reset is triggered, then, x-coordinate is 0
                               // or if x coordinate is at the right end of the screen,
                               // it goes back to 0
                               // otherwise, x coordinate keeps incrementing by 1
```

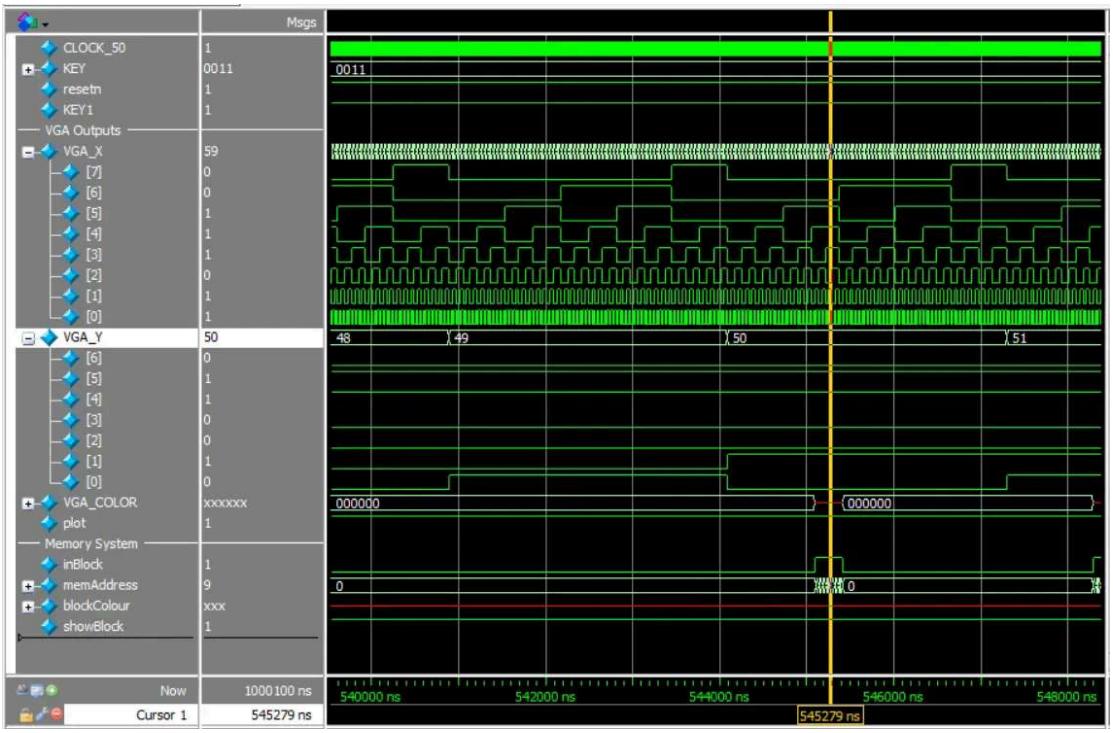
```
41
42       // Y counter (rows)
43       always @(posedge CLOCK_50)
44       begin
45          if(!resetn)
46             VGA_Y <= 0;
47          else if(VGA_X==159)
48          begin
49             if(VGA_Y==119) VGA_Y<=0; //end of the frame, where y coordinate is reset to 0
50             else VGA_Y <=VGA_Y+1; //otherwise, y coordinate keeps incrementing by 1
51          end
52       end
53
54       //plot, write enable signal
55       always@(posedge CLOCK_50)
56       begin
57          plot<=1'b1;
58       end
59
60
61       //Block memory - 16x16
62       reg [8:0] block [0:255];
63       //16*16=256 pixels total, where each needs 9 bits for colour
64
65       //initializing memory with the block mif
66       initial
67       begin
68          $readmemh("C:/Users/niran/Downloads/Stack5/memory/block.mif", block);
69       end
70
71       //calculate address into block memory
72       reg [7:0] memAddress;
73       wire inBlock;
74       assign inBlock=(VGA_X>=50&&VGA_X<66)&&(VGA_Y>=50&&VGA_Y<66);
75
76       //calculating address of pixels; are they within the 16x16 block?
77       always@(*)
78       begin
79          if(inBlock) memAddress = (VGA_Y-50)*16+(VGA_X-50);
80          //converting the 2D coordinates into 1D memory addresses
81          //multiply the row by 16 (since each row has 16 pixels), then add column to yield linear address
82          else memAddress = 0;
83       end
84
```

```
65      //initializing memory with the block mif
66      initial
67      begin |
68          $readmemh("C:/Users/niran/Downloads/Stack5/memory/block.mif", block);
69      end
70
71      //calculate address into block memory
72      reg [7:0] memAddress;
73      wire inBlock;
74      assign inBlock=(VGA_X>=50&&VGA_X<66)&&(VGA_Y>=50&&VGA_Y<66);
75
76      //calculating address of pixels; are they within the 16x16 block?
77      always@(*)
78      begin
79          if(inBlock) memAddress = (VGA_Y-50)*16+(VGA_X-50);
80          //converting the 2D coordinates into 1D memory addresses
81          //multiply the row by 16 (since each row has 16 pixels), then add column to yield linear address
82          else memAddress = 0;
83      end
84
85      //reading from memory
86      reg [8:0] blockColour;
87      always @(posedge CLOCK_50)
88      begin
89          blockColour<=block[memAddress];
90      end
91
92      //show block if KEY1 is pressed and within block area, else show black background
93      reg showBlock;
94      always@(posedge CLOCK_50)
95      begin
96          if(!resetn) showBlock<=0;
97          else if (~KEY1) //active low; KEY 1 is presssed
98              showBlock<=1;
99      end
100
101
102
103         // Convert 9-bit to 24-bit color
104     wire [23:0] color_24bit = {
105         {8{blockColour[8]}}, // Red
106         {8{blockColour[5]}}, // Green
107         {8{blockColour[2]}}  // Blue
108     };
109
110     assign VGA_COLOR = (showBlock && inBlock) ? color_24bit : 24'h000000;
111
112 endmodule
```

ModelSims:



Currently, inside the block at VGA_X = 64, VGA_Y = 59 (inBlock =1, showBlock =1 because KEY1 =1). Memory values are output at VGA_COLOR. However, blockColour is uninitialized/ not being read.



Currently, near the start of the block (59, 50), where memory address is 9 (9th pixel in first row of 16x16 block, because (50-50)*16+(59-50)=9), so inBlock = 1. memAddress is thus incrementing as VGA controller is scanning through the 16x16 block area. blockColour signal is however not being read/ memory not initialized. VGA_COLOR transitions between black and other values.
As VGA_X increments from 50 to 65 bits (which is one row of the 16x16 block), the memAddress increments from 0 to 15 bits. Every clock cycle should read the next pixel colour from the block ROM (ROM address scans/reads through the block sprite).