MINIATURIZED GRAPHIC ENGINE

DIGITAL SYSTEMS DESIGN WITH FPGA (PROF. DEBAYAN DAS)

Hemanth Raj (23120)

Kishan Baranwal (23483)

Kushal Gowda (22445)

Rahul Kumar (23773)

Department of Electronic Systems Engineering

INDIAN INSTITUTE OF SCIENCE

भारतीय विज्ञान संस्थान

Contents

1. Aim
2. VGA controller
3. Animation
4. Rotation
5. Bresenham's algorithm
6. Background formation
7. Full screen text editor

Aim

1. Background accelerator

All rooms will have same background with only extra objects and wall colors.

2. Animation

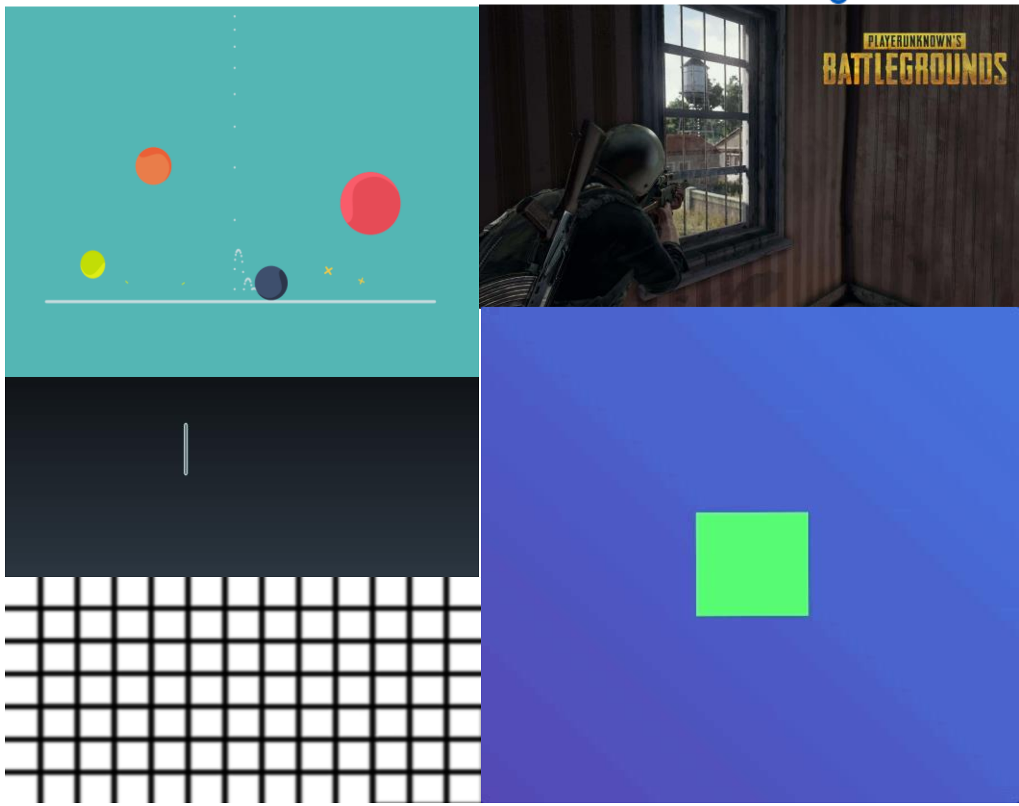
Animation shows basic law of reflection and translated object.

3. Rotation and line formation

Using sin cos matrix approximation rotation

4. Full Screen Text display

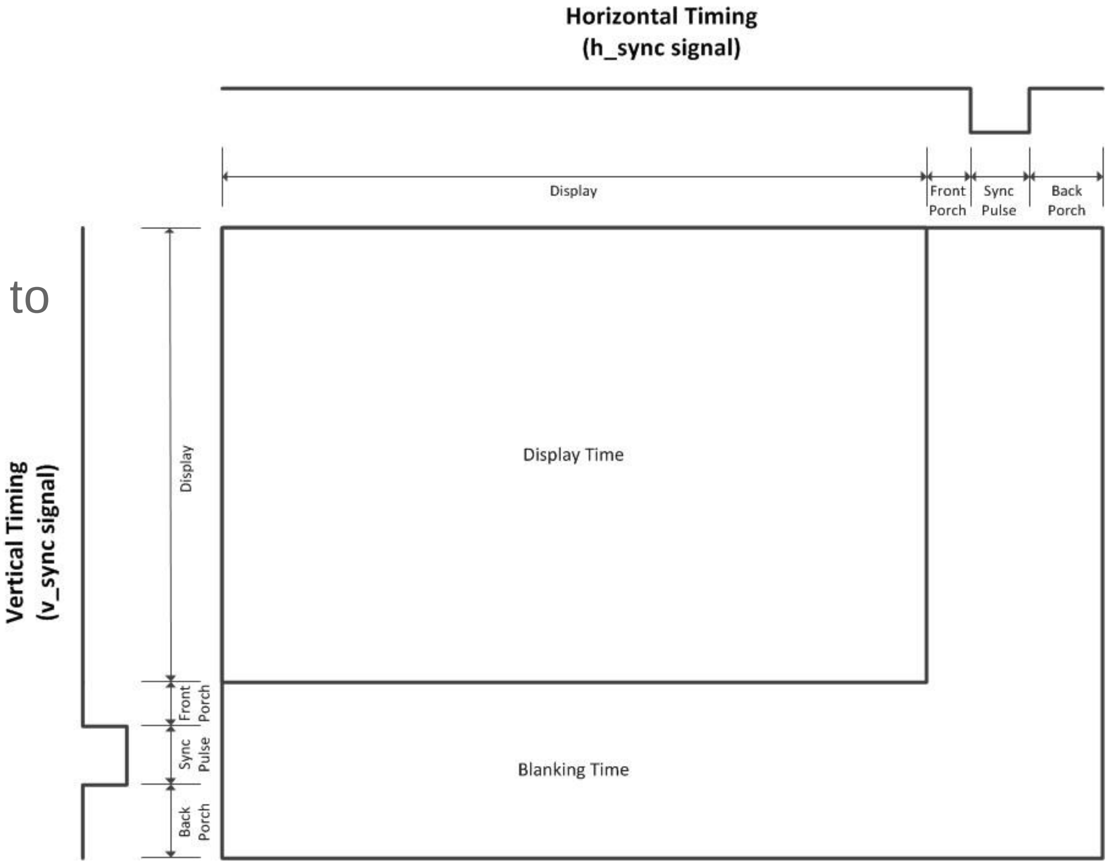
Displays text on Full Screen

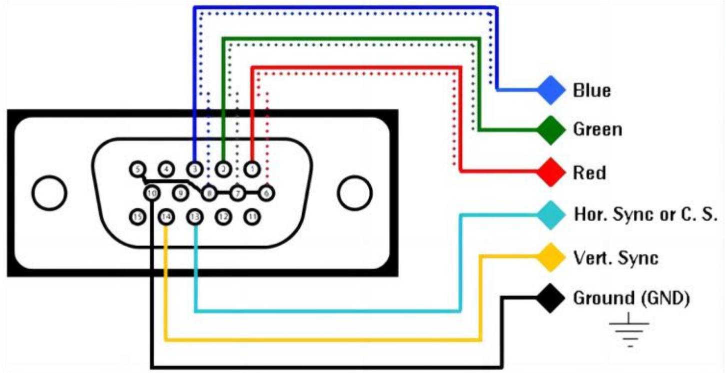


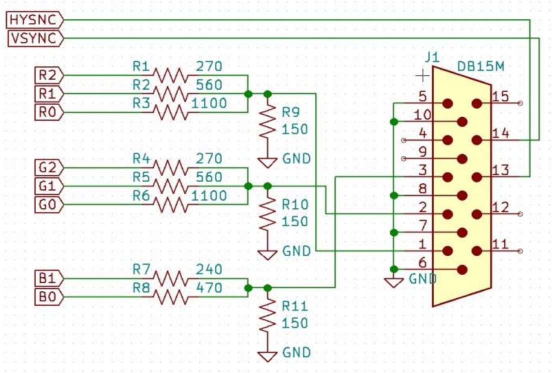
Source: Google image

VGA Controller

VGA RED : red color channel
VGA GREEN : green color channel
VGA BLUE : blue color channel
VGA HSYNC : Tells when horizontal screen has been completed
VGA VSYNC : Tells when all horizontal lines are completed
FRONT PORCH and BACK PORCH are safety timing for circuitry to have time to move beam. In between image can not be generated.







Reference Book: Chu, Pong P. Wiley, 2008 "FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version"

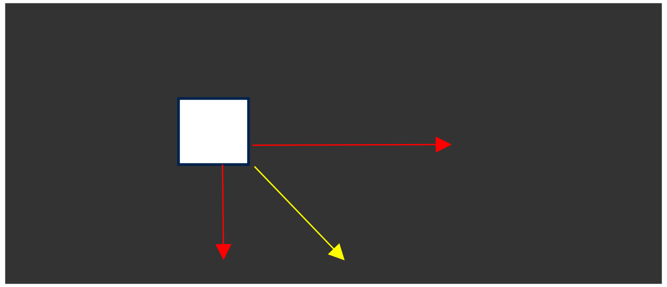
640 * 480 pixels

25 MHz each pixel update

Updating frame without flickering at 60 Hz

Object Animation

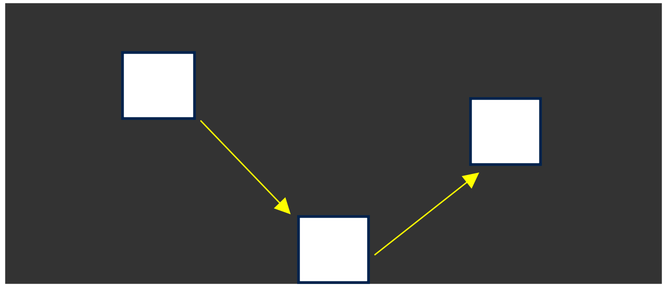
- Changing objects location from frame to frame creates an illusion of motion
- The screen is redrawn 60 times per second as the monitor supports 60Hz refresh rate.
- Each time, the object is moved by a small amount (move by less number of pixels) making it feel like the object is moving.



- Registers are used to track X and Y coordinates of the object.
- Four registers are used to track right, left, top and bottom edges of the square
- For each frame, registers are incremented or decremented based on the direction of motion.

Object Animation

- If the object is incremented in both right and down directions, it moves diagonally downwards.
- Upon collision with the bottom of the screen, the Y value flips and starts decrementing while X keeps incrementing until it hits the right most edge of the screen



- Upon collision with the right most edge of the screen, the X value also flips and starts decrementing and the motion will be as shown.

Matrix Rotation using approximation

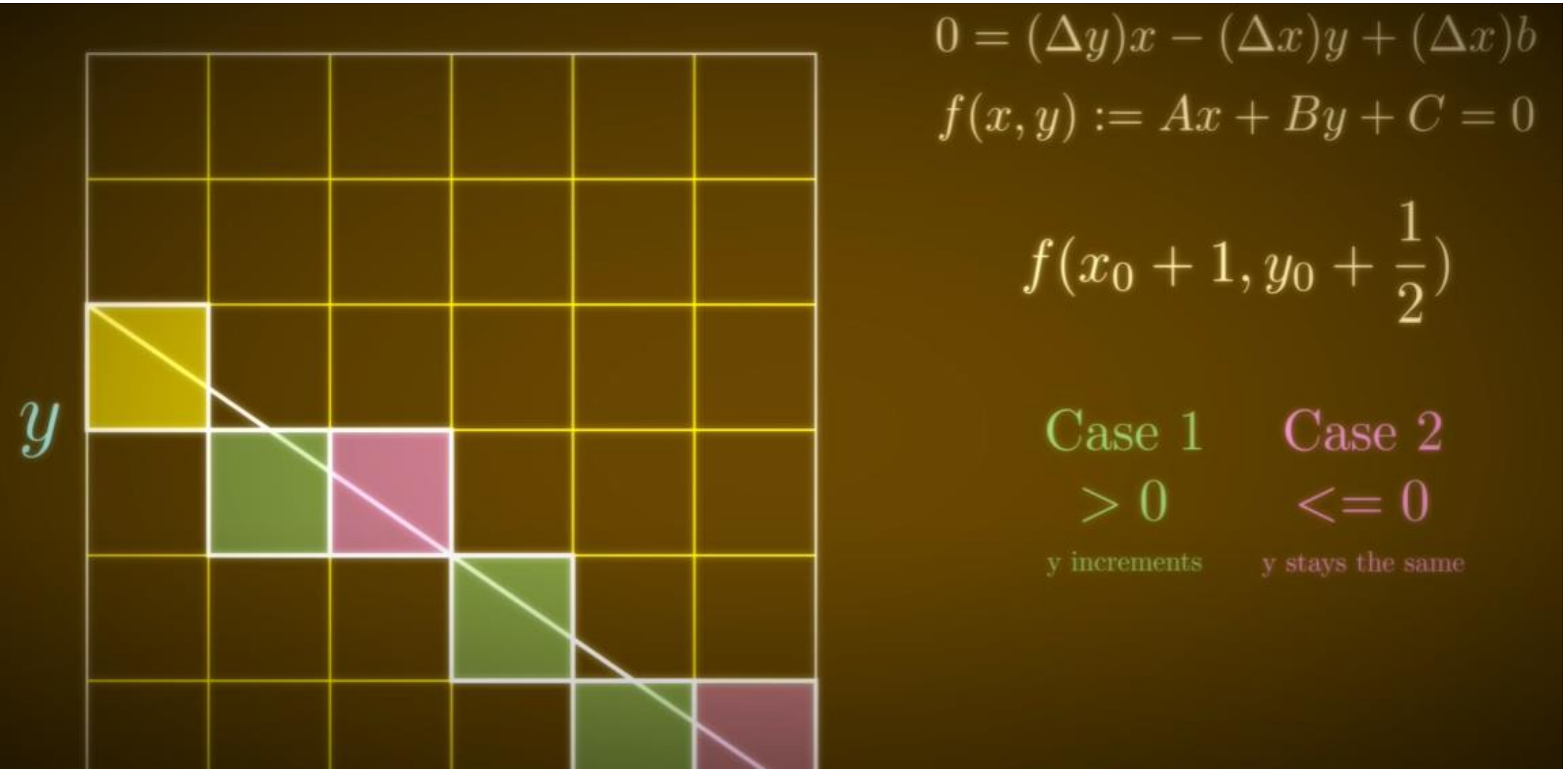
- Rotation matrix is transformation matrix in linear algebra that is used to perform rotation in Euclidean space.
- Rotation matrix: $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix}$$
- Verilog code for rotating the point after every 15 degree :

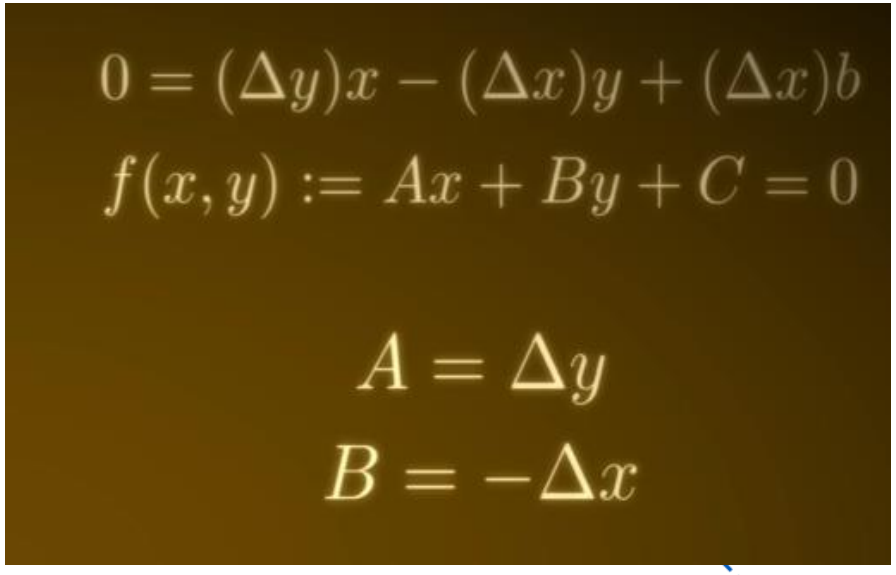
```
rotated_x <= x_pt - (x_pt >>> 5)- (y_pt >>> 2) - (y_pt >>> 7);
rotated_y <= (x_pt>>>2) + (x_pt >>> 7) + y_pt - (y_pt >>> 5);
```

- x_pt – (x_pt>>>5) = x(1 - 1/32) = x(0.968) , which is approximately equal to cos(15)= 0.967.
- (y_pt>>>2) + (y_pt>>>7) = y(1/4 + 1/128) = y(0.2578) , which is approximately equal to sin(15)= 0.2588.
- The changed cordinated are fed again to x_pt and y_pt to rotate continuously for 15 degree .

Bresenham's Line Algorithm:-

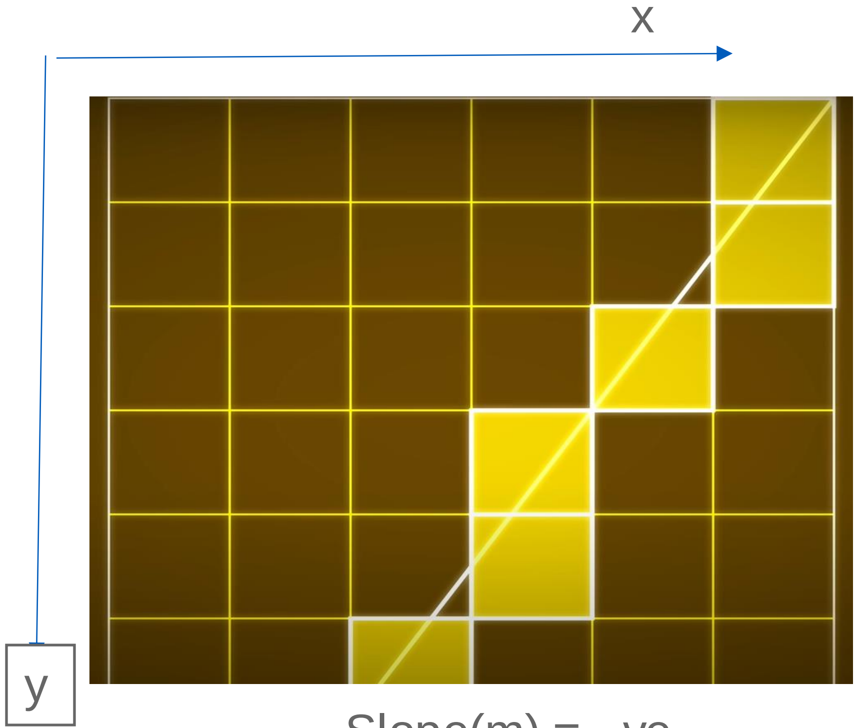
- Bresenham's line algorithm is a line drawing algorithm that determined all the points in a plane that should be selected in order to form a close appoximation to a straight line between two points.
- Consider line Y= mX + C , slope (m) is ΔY/ΔX, the equation will become : (for m <1)



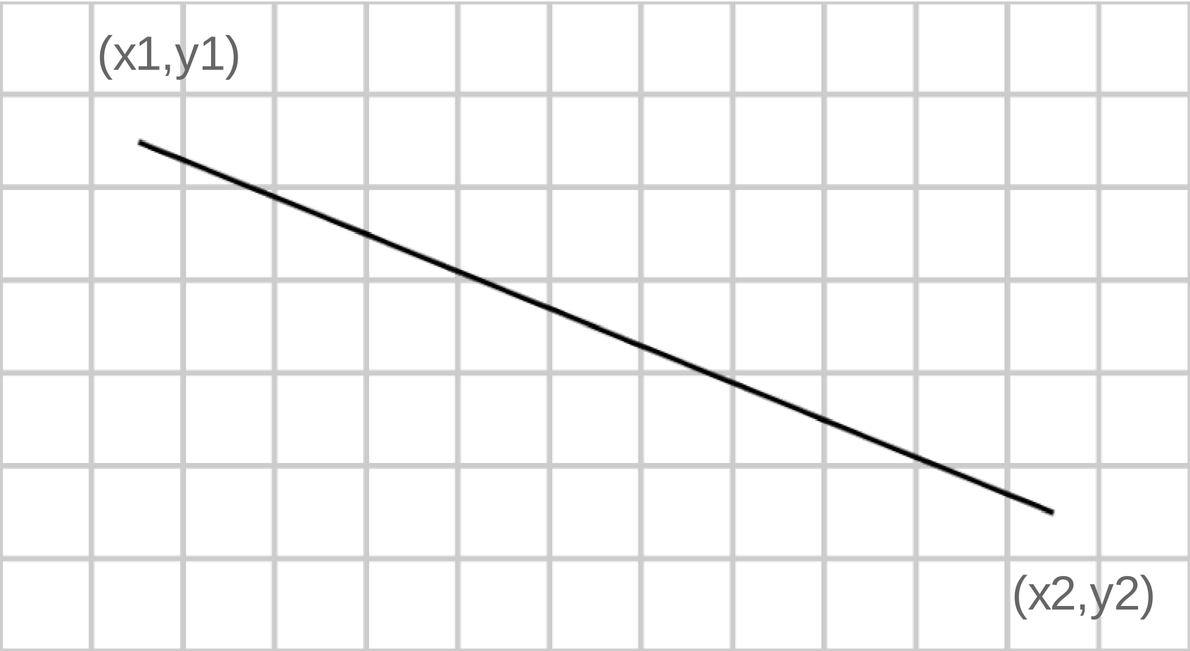


Bresenham's Line Algorithm:-

- Now the incremented y and x is chacked again in line equation F(x0 , yo+½), to find whether the new point is above the line or below the line.
- Similarly for m >1, Y =Y+1; and X need to check for line positon,
- For m<0, complete algorithm follows with slope=(- Δy /Δ x).



Slope(m) = -ve



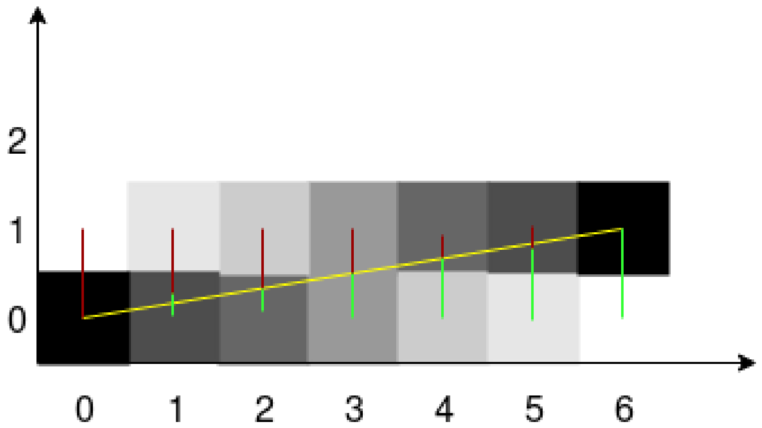
Tracing of line b/w two point using Bresenham's algo.

Limitations

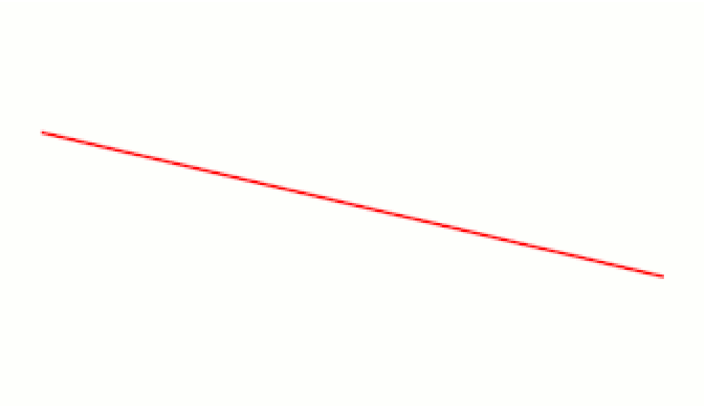
Bresenham's algorithm do not plot static line.
Bresenham's algorithm is not capable of plotting steep line.

Future Modifications

Bresenham's line algorithm should be replaced with Xiaolin Wu's line algorithm to get static and steep line for all slope.
So static cube formation with rotation can easily be formed.
Xiaolin emphasized the significance of pixel intensity when illuminating pixels while plotting lines.



Intensity of pixel



Xiaolin Wu's line algorithm

Formation of dots

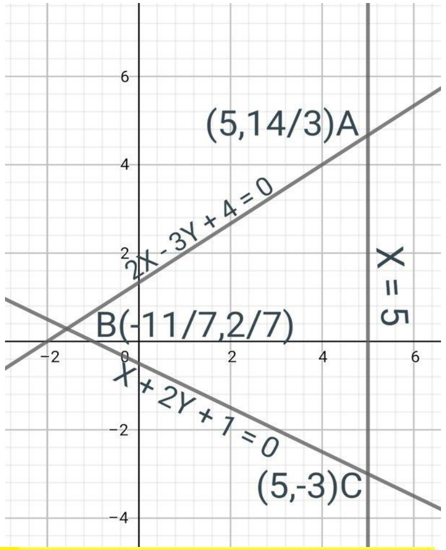
```
assign dot0 = ((x_pt >= (x_out_pt0_coordinate - 1)) && (x_pt < (x_out_pt0_coordinate + 1)) && (y_pt >= (y_out_pt0_coordinate - 1)) && (y_pt < (y_out_pt0_coordinate + 1)));
```

Formation of rectangle

```
assign dot0 = ((x_pt >= (x_out_pt0_coordinate - W)) && (x_pt < (x_out_pt0_coordinate + W)) && (y_pt >= (y_out_pt0_coordinate - L)) && (y_pt < (y_out_pt0_coordinate + L)));
```

Formation of line of fixed slope

```
if((y0_new < y1) && video_on && (x == x0_new) && (y == y0_new))
y0_new <= y0_new + dx;
if((x0_new < x1) && video_on &&(x == x0_new) && (y == y0_new))
x0_new <= x0_new + dy;
```

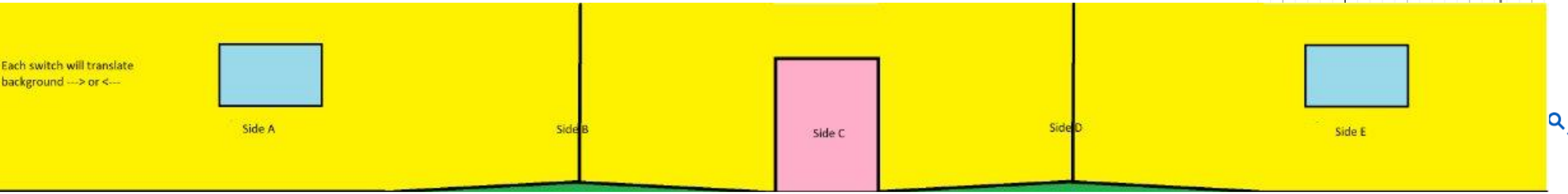


Formation of triangle

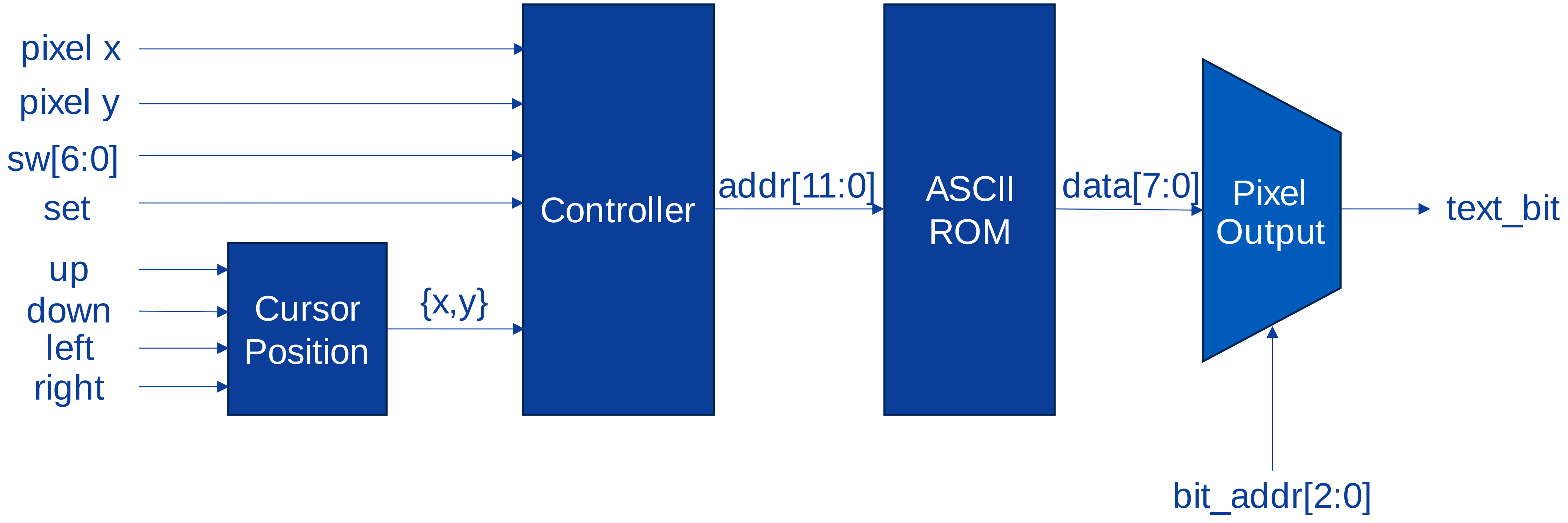
```
assign dots = (x_pt < line1_x) && (y_pt <line1_y) && (x_pt < line2_x) && (y_pt <line2_y) && (x_pt > line3_x) && (y_pt >line3_y)
```

Lateral shift of each point

```
x_out_pt0_coordinate <= x_out_pt0_coordinate – dx;
y_out_pt0_coordinate <= y_out_pt0_coordinate – dy;
```



Full Screen Text Editor



ASCII ROM

- Each ASCII character takes 8*16 space on screen, so 8*16 bits of ROM has to be set for each character
- Hence, for all 128 ASCII characters we need 128*8*16 = 16384 bits of ROM
- Thus, a total of 2048 bytes are required, that requires 11 bits of address.
- The 7-bit ASCII code for each character is used as the MSB of the address.
- The 4-bit LSB is the row value.

```
code x21 (!)
11'h210: data = 8'b00000000;
11'h211: data = 8'b00000000;
11'h212: data = 8'b00000000;
11'h213: data = 8'b00011000;
11'h214: data = 8'b00011000;
11'h215: data = 8'b00011000;
11'h216: data = 8'b00011000;
11'h217: data = 8'b00011000;
11'h218: data = 8'b00011000;
11'h219: data = 8'b00000000;
11'h21a: data = 8'b00011000;
11'h21b: data = 8'b00011000;
11'h21c: data = 8'b00000000;
11'h21d: data = 8'b00000000;
11'h21e: data = 8'b00000000;
11'h21f: data = 8'b00000000;
```

```
// code x35 (5)
11'h350: data = 8'b00000000;
11'h351: data = 8'b00000000;
11'h352: data = 8'b11111110;
11'h353: data = 8'b11111110;
11'h354: data = 8'b11000000;
11'h355: data = 8'b11000000;
11'h356: data = 8'b11111110;
11'h357: data = 8'b11111110;
11'h358: data = 8'b00000110;
11'h359: data = 8'b00000110;
11'h35a: data = 8'b11111110;
11'h35b: data = 8'b11111110;
11'h35c: data = 8'b00000000;
11'h35d: data = 8'b00000000;
11'h35e: data = 8'b00000000;
11'h35f: data = 8'b00000000;
```

```
// code x41 (A)
11'h410: data = 8'b00000000;
11'h411: data = 8'b00000000;
11'h412: data = 8'b00010000;
11'h413: data = 8'b00111000;
11'h414: data = 8'b01101100;
11'h415: data = 8'b11000110;
11'h416: data = 8'b11000110;
11'h417: data = 8'b11111110;
11'h418: data = 8'b11111110;
11'h419: data = 8'b11000110;
11'h41a: data = 8'b11000110;
11'h41b: data = 8'b11000110;
11'h41c: data = 8'b00000000;
11'h41d: data = 8'b00000000;
11'h41e: data = 8'b00000000;
11'h41f: data = 8'b00000000;
```

Thank You