

Drawing Lines with SystemVerilog

Prof. Stephen A. Edwards

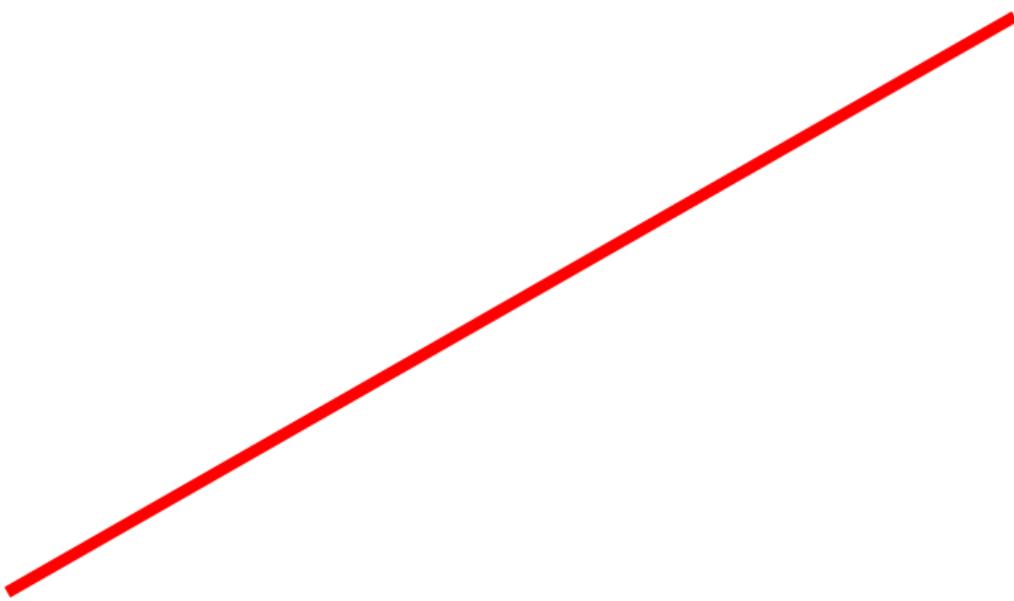
Columbia University

Spring 2020

Bresenham's Line Algorithm

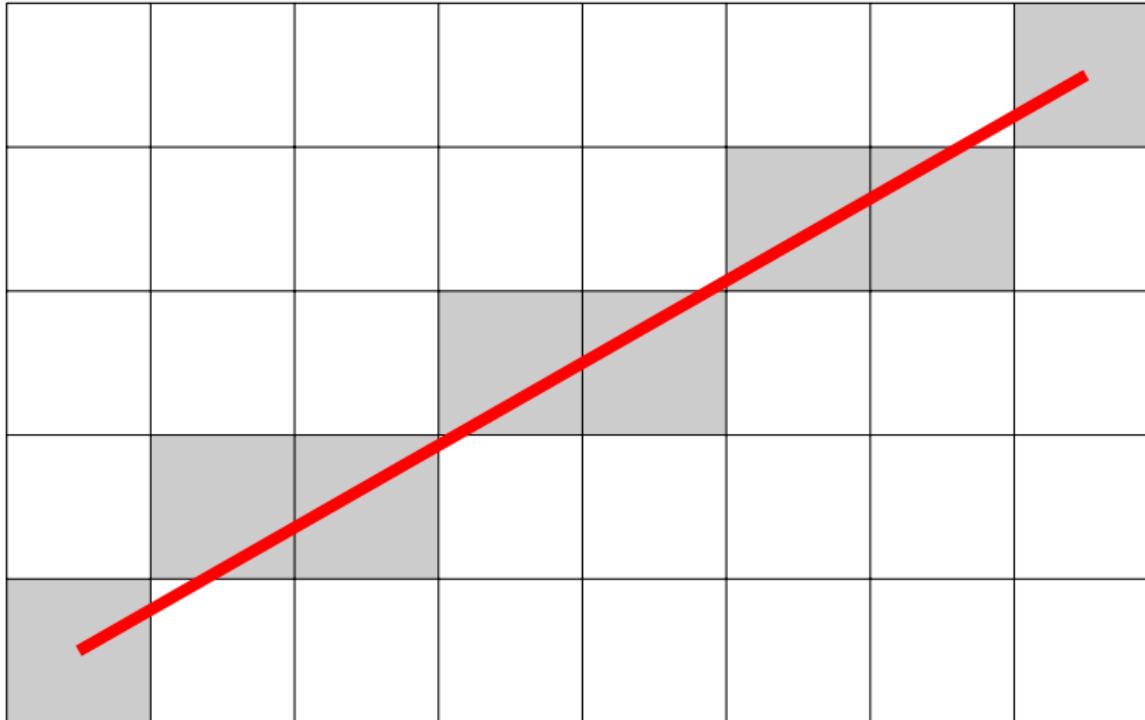
Bresenham's Line Algorithm

Objective: Draw a line...



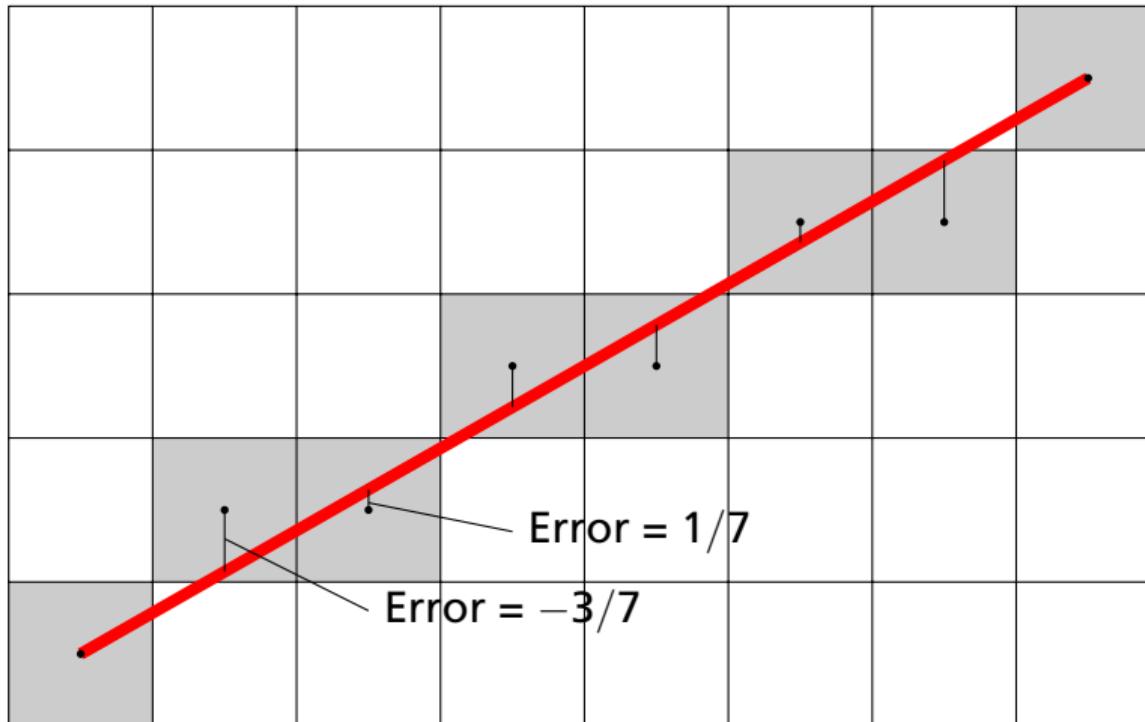
Bresenham's Line Algorithm

...with well-approximating pixels...



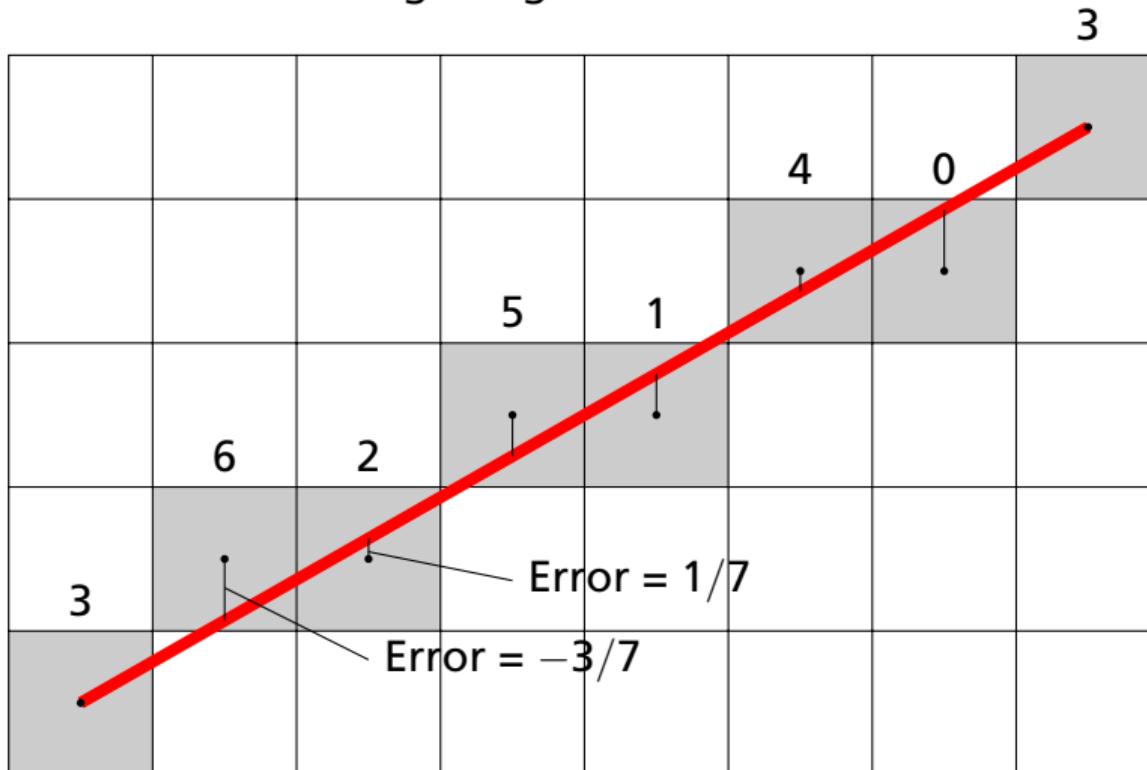
Bresenham's Line Algorithm

...by maintaining error information..



Bresenham's Line Algorithm

...encoded using integers



Approach

1. Understand the algorithm

I went to Wikipedia; doesn't everybody?

2. Code and test the algorithm in software

I used C and the SDL library for graphics

3. Define the interface for the hardware module

A communication protocol: consider the whole system

4. Schedule the operations

Draw a timing diagram!

In hardware, you must know in which cycle each thing happens.

5. Code in RTL

Always envision the hardware you are asking for

6. Test in simulation

Create a testbench: code that mimicks the environment (e.g., generates clocks, inputs).

7. Test on the FPGA

Simulating correctly is necessary but not sufficient.

The Pseudocode from Wikipedia

```
function line(x0, y0, x1, y1)
    dx := abs(x1-x0)
    dy := abs(y1-y0)
    if x0 < x1 then sx := 1 else sx := -1
    if y0 < y1 then sy := 1 else sy := -1
    err := dx-dy

loop
    setPixel(x0,y0)
    if x0 = x1 and y0 = y1 exit loop
    e2 := 2*err
    if e2 > -dy then
        err := err - dy
        x0 := x0 + sx
    end if
    if e2 < dx then
        err := err + dx
        y0 := y0 + sy
    end if
end loop
```

My C Code

```
void line(Uint16 x0, Uint16 y0, Uint16 x1, Uint16 y1)
{
    Sint16 dx, dy; // Width and height of bounding box
    Uint16 x, y; // Current point
    Sint16 err; // Loop-carried value
    Sint16 e2; // Temporary variable
    int right, down; // Boolean

    dx = x1 - x0; right = dx > 0; if (!right) dx = -dx;
    dy = y1 - y0; down = dy > 0; if (down) dy = -dy;
    err = dx + dy; x = x0; y = y0;
    for (;;) {
        plot(x, y);
        if (x == x1 && y == y1) break; // Reached the end
        e2 = err << 1; // err * 2
        if (e2 > dy) { err += dy; if (right) x++; else x--;}
        if (e2 < dx) { err += dx; if (down) y++; else y--;}
    }
}
```

Module Interface

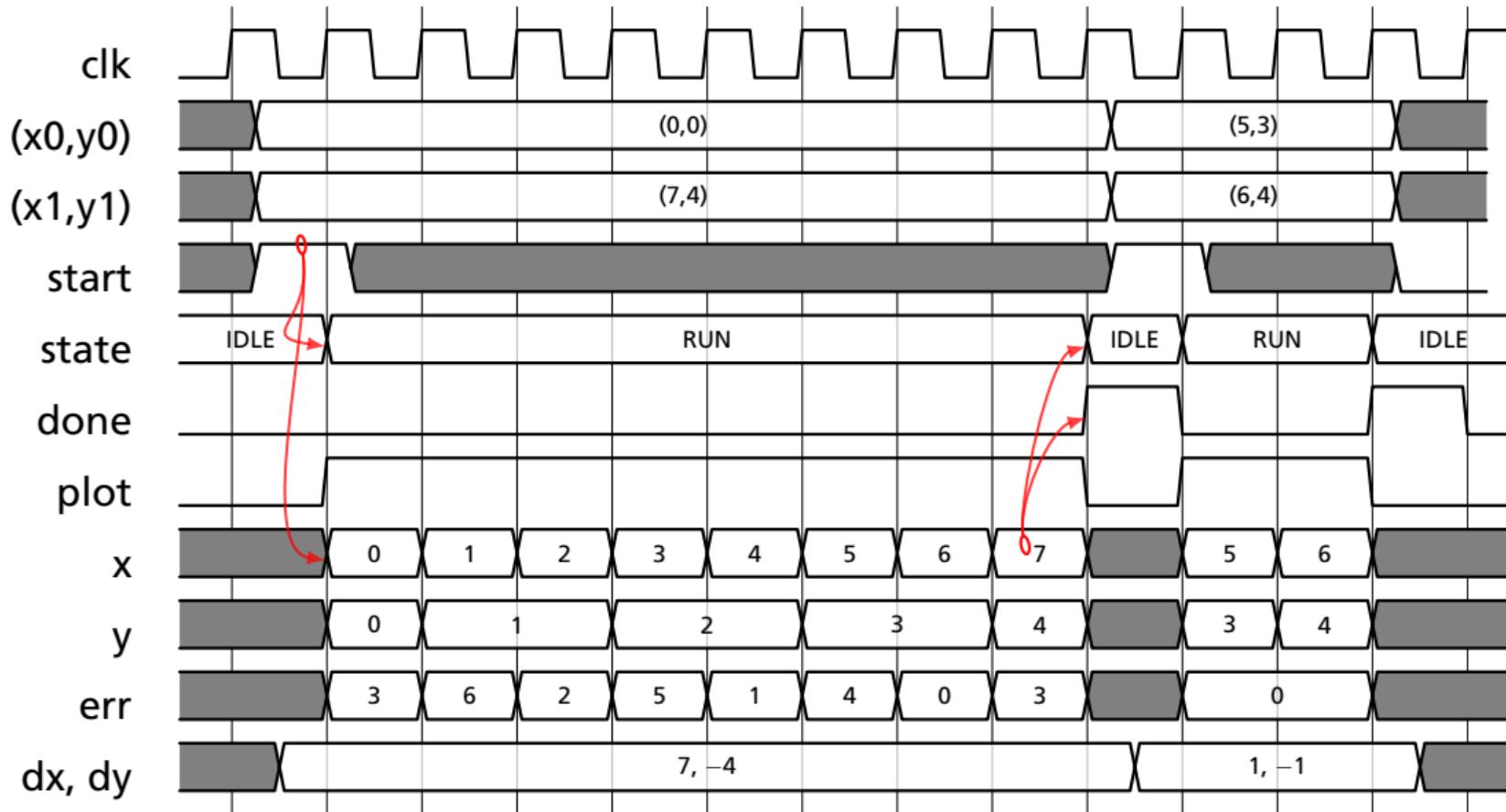
```
module bresenham(input logic          clk, reset,
                  input logic          start,
                  input logic [10:0]    x0, y0, x1, y1,
                  output logic         plot,
                  output logic [10:0]   x, y,
                  output logic         done);
```

start indicates (x_0, y_0) and (x_1, y_1) are valid

plot indicates (x, y) is a point to plot

done indicates we are ready for the next *start*

Scheduling: Timing Diagram



RTL: The IDLE state

/* C code */

```
Sint16 dx;
Sint16 dy;
Uint16 x, y;
Sint16 err;
Sint16 e2;
int right;
int down;

dx = x1 - x0;
right = dx > 0;
if (!right) dx = -dx;
dy = y1 - y0;
down = dy > 0;
if (down) dy = -dy;

err = dx + dy;
x = x0;
y = y0;

for (;;) {
    plot(x, y);
```

```
logic signed [11:0] dx, dy, err, e2;
logic right, down;

typedef enum logic {IDLE, RUN} state_t;
state_t state;

always_ff @(posedge clk) begin
    done <= 0;
    plot <= 0;
    if (reset) state <= IDLE;
    else case (state)
        IDLE:
            if (start) begin
                dx = x1 - x0; // Blocking!
                right = dx >= 0;
                if (~right) dx = -dx;
                dy = y1 - y0;
                down = dy >= 0;
                if (down) dy = -dy;
                err = dx + dy;
                x <= x0;
                y <= y0;
                plot <= 1;
                state <= RUN;
            end
    end
end
```

RTL: The RUN state

```
/* C Code */

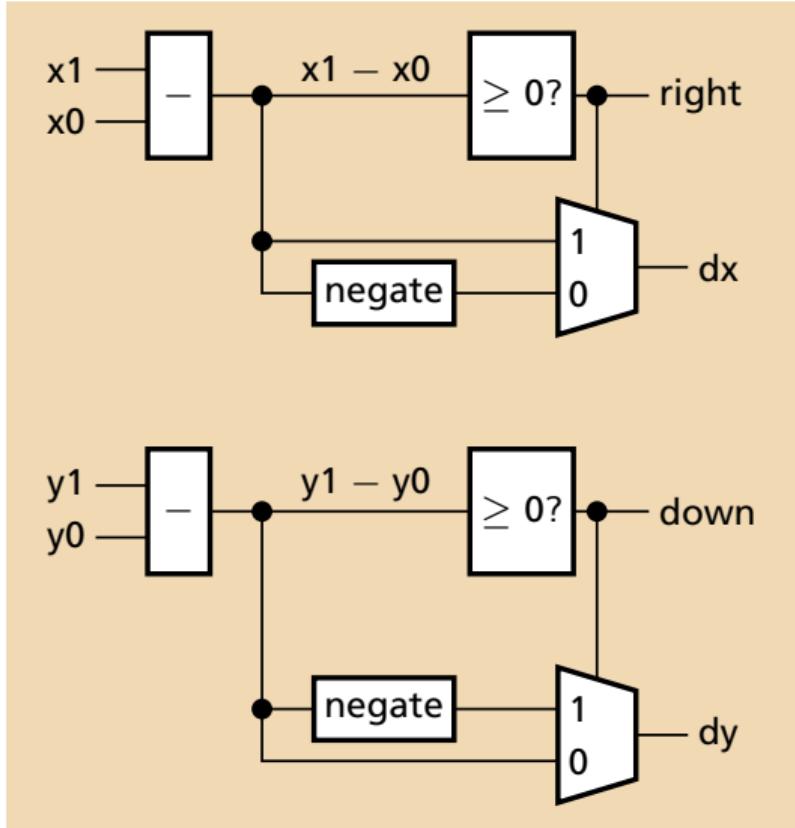
for (;;) {
    plot(x, y);
    if (x == x1 &&
        y == y1)
        break;
    e2 = err << 1;
    if (e2 > dy) {
        err += dy;
        if (right) x++;
        else x--;
    }
    if (e2 < dx) {
        err += dx;
        if (down) y++;
        else y--;
    }
}
```

```
RUN:
    if (x == x1 && y == y1) begin
        done <= 1;
        state <= IDLE;
    end else begin
        plot <= 1;
        e2 = err << 1;
        if (e2 > dy) begin
            err += dy;
            if (right) x <= x + 10'd 1;
            else         x <= x - 10'd 1;
        end
        if (e2 < dx) begin
            err += dx;
            if (down) y <= y + 10'd 1;
            else         y <= y - 10'd 1;
        end
    end
    default:
        state <= IDLE;
    endcase
end
```

Datapath for dx , dy , $right$, and $down$

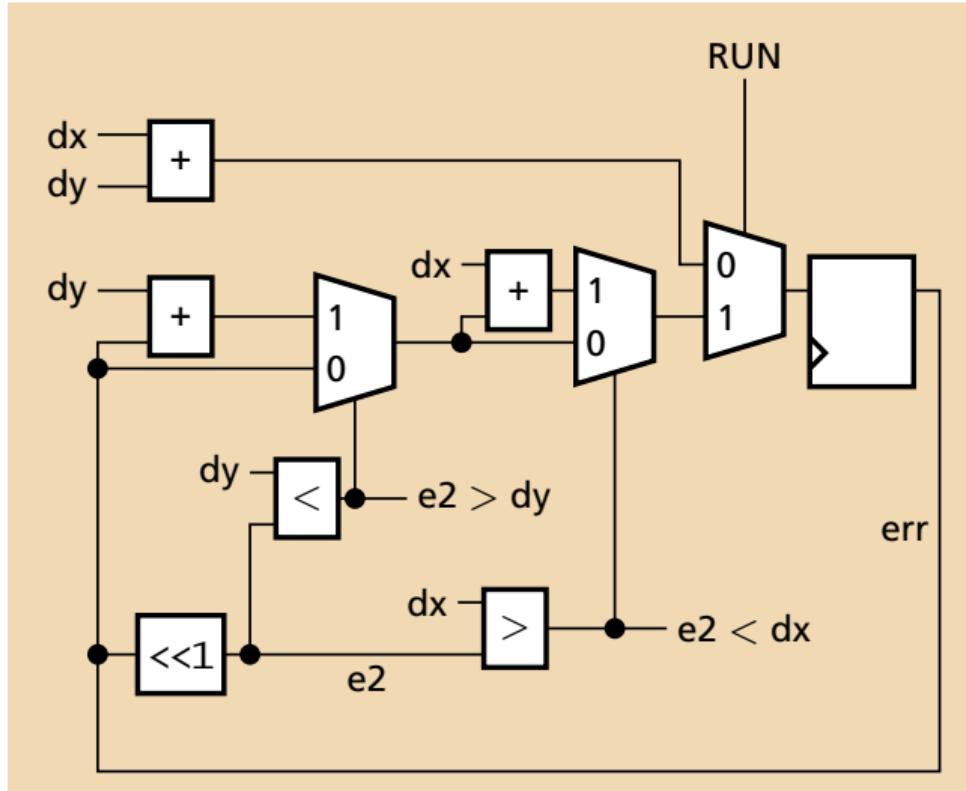
```
I: if (start)
    dx = x1 - x0;
    right = dx >= 0;
    if (~right) dx = -dx;
    dy = y1 - y0;
    down = dy >= 0;
    if (down) dy = -dy;
    err = dx + dy;
    x <= x0;
    y <= y0;
    plot <= 1;
    state <= RUN;
```

```
R: if (x == x1 && y == y1)
    done <= 1;
    state <= IDLE;
else
    plot <= 1;
    e2 = err << 1;
    if (e2 > dy)
        err += dy;
        if (right) x <= x + 10'd 1;
        else       x <= x - 10'd 1;
    if (e2 < dx)
        err += dx;
        if (down) y <= y + 10'd 1;
        else       y <= y - 10'd 1;
```



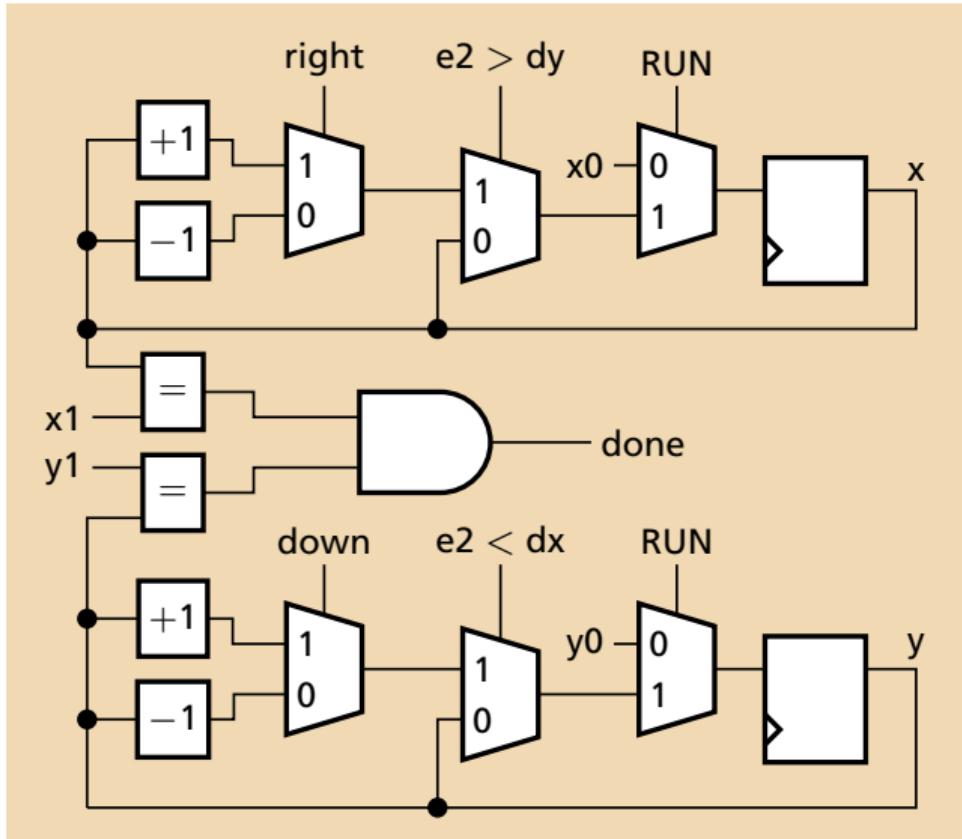
Datapath for err

```
I: if (start)
    dx = x1 - x0;
    right = dx >= 0;
    if (~right) dx = -dx;
    dy = y1 - y0;
    down = dy >= 0;
    if (down) dy = -dy;
    err = dx + dy;
    x <= x0;
    y <= y0;
    plot <= 1;
    state <= RUN;
R: if (x == x1 && y == y1)
    done <= 1;
    state <= IDLE;
else
    plot <= 1;
    e2 = err << 1;
    if (e2 > dy)
        err += dy;
        if (right) x <= x + 10'd 1;
        else      x <= x - 10'd 1;
    if (e2 < dx)
        err += dx;
        if (down) y <= y + 10'd 1;
        else      y <= y - 10'd 1;
```



Datapath for x and y

```
I: if (start)
    dx = x1 - x0;
    right = dx >= 0;
    if (~right) dx = -dx;
    dy = y1 - y0;
    down = dy >= 0;
    if (down) dy = -dy;
    err = dx + dy;
    x <= x0;
    y <= y0;
    plot <= 1;
    state <= RUN;
R: if (x == x1 && y == y1)
    done <= 1;
    state <= IDLE;
else
    plot <= 1;
    e2 = err << 1;
    if (e2 > dy)
        err += dy;
        if (right) x <= x + 10'd 1;
        else       x <= x - 10'd 1;
    if (e2 < dx)
        err += dx;
        if (down) y <= y + 10'd 1;
        else       y <= y - 10'd 1;
```



The Framebuffer: Interface and Constants

```
module VGA_framebuffer(
    input logic      clk50, reset,
    input logic [10:0] x, y, // Pixel coordinates
    input logic      pixel_color, pixel_write,
    output logic [7:0] VGA_R, VGA_G, VGA_B,
    output logic      VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

    parameter HACTIVE      = 11'd 1280,
              HFRONT_PORCH = 11'd 32,
              HSYNC         = 11'd 192,
              HBACK_PORCH   = 11'd 96,
              HTOTAL        =
                            HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH; //1600

    parameter VACTIVE      = 10'd 480,
              VFRONT_PORCH = 10'd 10,
              VSYNC         = 10'd 2,
              VBACK_PORCH   = 10'd 33,
              VTOTAL        =
                            VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; //525
```

The Framebuffer: Counters and Sync

```
// Horizontal counter
logic [10:0] hcount;
logic endOfLine;

always_ff @(posedge clk50 or posedge reset)
  if (reset) hcount <= 0;
  else if (endOfLine) hcount <= 0;
  else hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic [9:0] vcount;
logic endOfField;

always_ff @(posedge clk50 or posedge reset)
  if (reset) vcount <= 0;
  else if (endOfLine)
    if (endOfField) vcount <= 0;
    else vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

assign VGA_HS = !( (hcount[10:7] == 4'b1010) &
                    (hcount[6] | hcount[5]) );
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
```

The Framebuffer: Blanking, Memory, and RGB

```
assign VGA_SYNC_n = 1; // Sync on R, G, and B. Unused for VGA.  
  
logic      blank;  
assign blank = ( hcount[10] & (hcount[9] | hcount[8]) ) | // 1280  
              ( vcount[9] | (vcount[8:5] == 4'b1111) ); // 480  
  
logic      framebuffer [307199:0]; // 640 * 480  
logic [18:0] read_address, write_address;  
  
assign write_address = x + (y << 9) + (y << 7) ; // x + y * 640  
assign read_address =  
           (hcount >> 1) + (vcount << 9) + (vcount << 7);  
  
logic      pixel_read;  
always_ff @(posedge clk50) begin  
    if (pixel_write) framebuffer[write_address] <= pixel_color;  
    if (hcount[0]) begin  
        pixel_read <= framebuffer[read_address];  
        VGA_BLANK_n <= ~blank; // Sync blank with read pixel data  
    end  
end  
  
assign VGA_CLK = hcount[0]; // 25 MHz clock  
assign {VGA_R, VGA_G, VGA_B} = pixel_read ? 24'hFF_FF_FF : 24'h0;  
endmodule
```

The “Hallway” Line Generator

```
module hallway(input logic      clk, reset,
               input logic      VGA_VS,
               input logic      done,
               output logic [10:0] x0, y0, x1, y1,
               output logic      start, pixel_color);

// ...

// Typical state:

S_TOP:
  if (done) begin
    start <= 1;
    if (x0 < 620)
      x0 <= x0 + 10'd 10;
    else begin
      state <= S_RIGHT;
      x0 <= 639;
      y0 <= 0;
    end
  end
end
```

Connecting the Pieces

// SoCKit_Top.sv

```
logic [10:0]      x, y, x0,y0,x1,y1;
logic             pixel_color;
logic             pixel_write;
logic             done, start;

VGA_framebuffer fb(.clk50(OSC_50_B3B),
                    .reset(~RESET_n),
                    .*);

bresenham liner(.clk(OSC_50_B3B),
                 .reset(~RESET_n),
                 .plot(pixel_write),
                 .*);

hallway hall(.clk(OSC_50_B3B),
              .reset(~RESET_n),
              .* );
```

Connect the bresenham
reset port to
an inverted *RESET_n*

Connect the other
bresenham ports to wires
with the same name
e.g., .x(x), .y(y), ...