Bresenham's Line Algorithm in Hardware

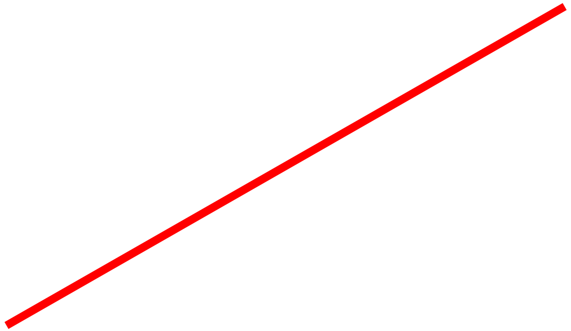
Stephen A. Edwards

Columbia University

Spring 2012

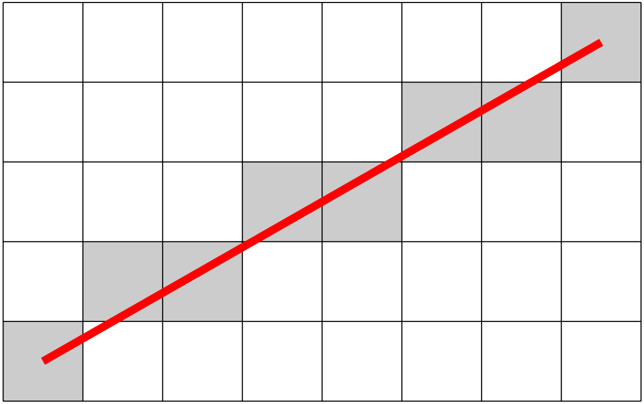
Bresenham’s Line Algorithm

Objective:Draw a line...



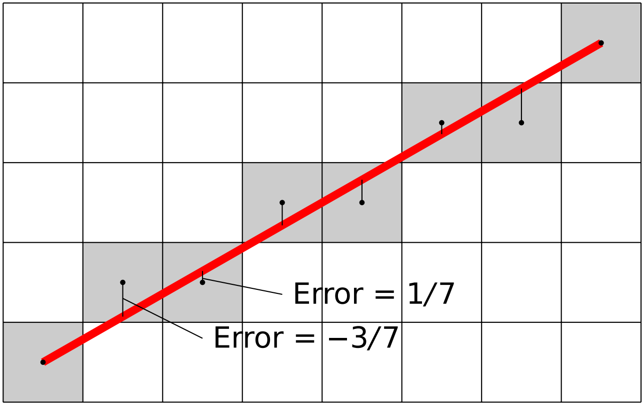
Bresenham’s Line Algorithm

...with well-approximating pixels...



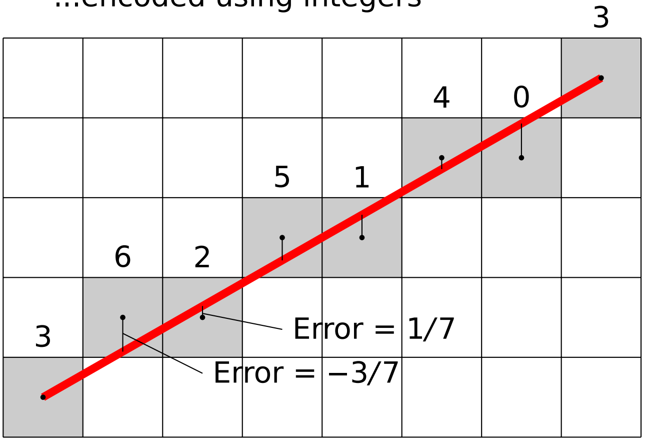
Bresenham’s Line Algorithm

...by maintaining error information..



Bresenham's Line Algorithm

...encoded using integers



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

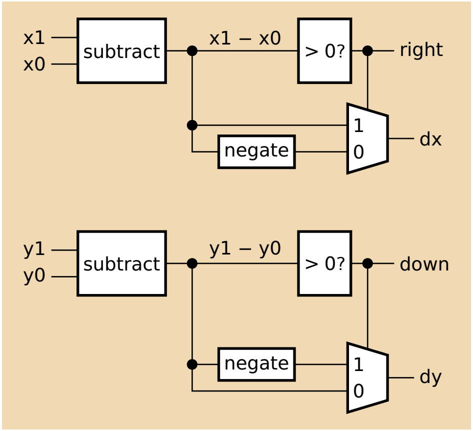
```c
void line(Uint16 x0, Uint16 y0, Uint16 x1, Uint16 y1)
{
  Sint16 dx, dy;  // Width and height of bounding box
  Uint16 x, y;    // Current point
  Sint8 sx, sy;   // -1 or 1
  Sint16 err;     // Loop-carried value
  Sint16 e2;      // Temporary variable
  int right, down;// Boolean

  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--;}
  }
}
```

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

```
void line(Uint16 x0, Uint16 y0,
          Uint16 x1, Uint16 y1)
{
  Sint16 dx;   // Width of bounding box
  Sint16 dy;   // Height of BB (neg)
  Uint16 x, y; // Current point
  Sint8 sx, sy;// -1 or 1
  Sint16 err;  // Loop-carried value
  Sint16 e2;   // Temporary variable
  int right;   // Boolean
  int down;    // Boolean
```

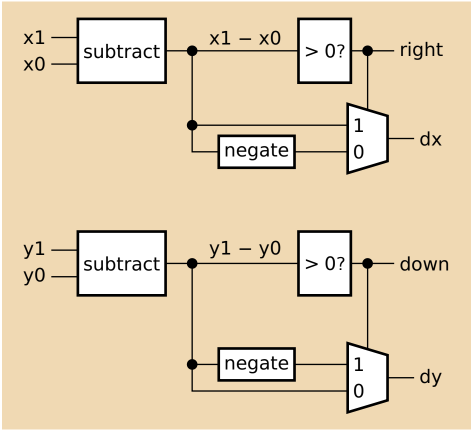
```
dx = x1 - x0;
right = dx > 0;
if (!right) dx = -dx;
dy = y1 - y0;
down = dy > 0;
if (down) dy = -dy;

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

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

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



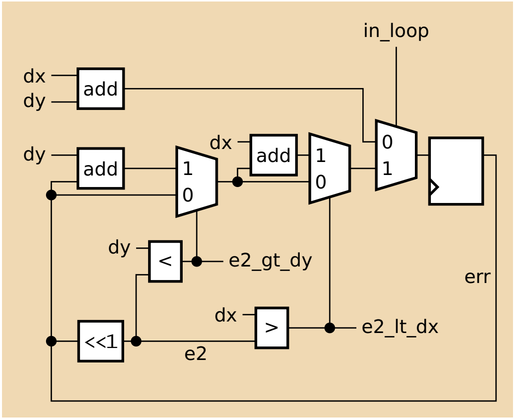
```
dx = x1 - x0;
right = dx > 0;
if (!right) dx = -dx;
dy = y1 - y0;
down = dy > 0;
if (down) dy = -dy;

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

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

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



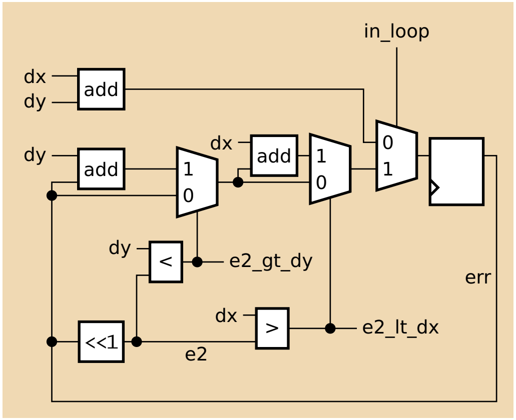
```
x1x0 <= x1 - x0;
y1y0 <= y1 - y0;

right <= not x1x0(10);
down <= not y1y0(10);

dx <= x1x0 when right = '1'
           else -x1x0;
dy <= -y1y0 when down = '1'
            else y1y0;
```

Datapath for *err*



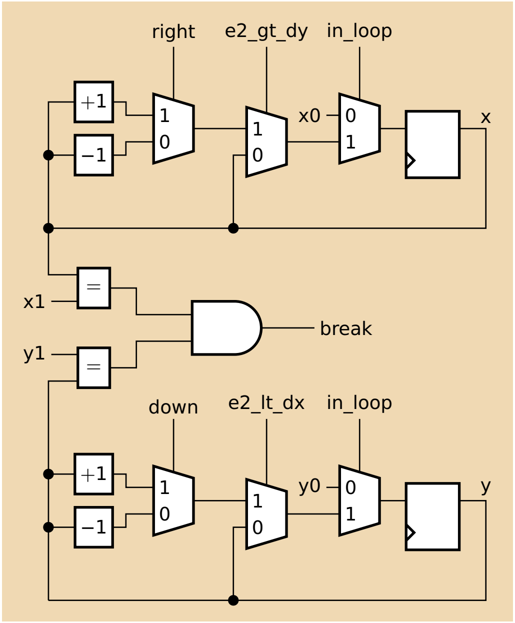
```
dx = x1 - x0;
right = dx > 0;
if (!right) dx = -dx;
dy = y1 - y0;
down = dy > 0;
if (down) dy = -dy;

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

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

Datapath for *err*



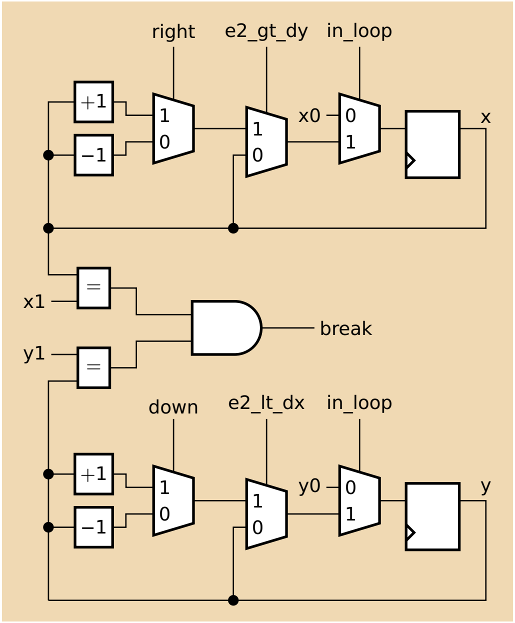
```
err_next <=
    ("0" & dx) + ("0" & dy)
    when in_loop = '0'
    else err2;

err2 <= err1 + dx
    when e2_lt_dx = '1'
    else err1;

err1 <= err + dy
    when e2_gt_dy = '1'
    else err;

e2 <=
  err(10 downto 0) & "0";

e2_gt_dy <=
    '1' when e2 > dy
        else '0';
e2_lt_dx <=
    '1' when e2 < dx
        else '0';
```

Datapath for *x* and *y*



```
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;
  e2 = err << 1;
  if (e2 > dy) {
    err += dy;
    if (right) x++;
    else x--;
  }
  if (e2 < dx) {
    err += dx;
    if (down)  y++;
    else y--;
  }
}
```

Datapath for *x* and *y*

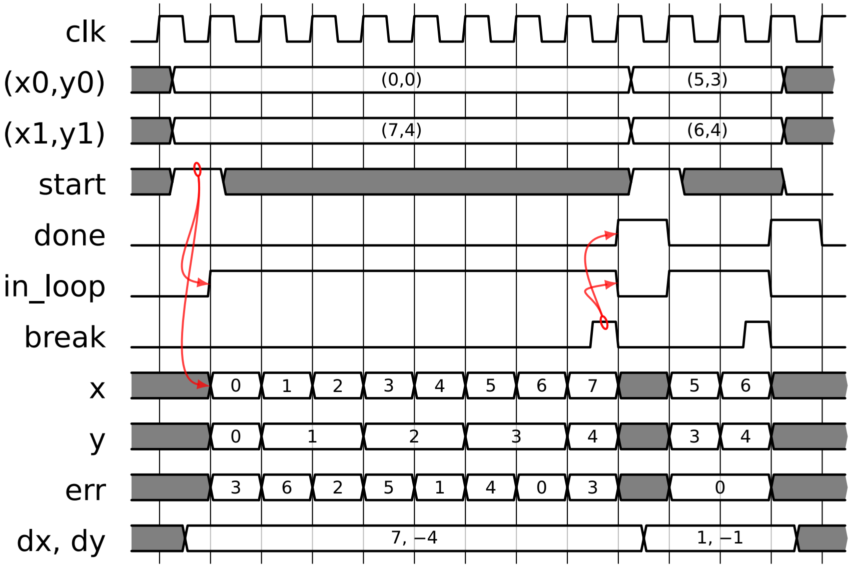


```
x_next <=
   x0 when in_loop = '0'
      else xb;
xb <= xa when e2_gt_dy = '1'
         else x;
xa <= x + 1 when right = '1'
            else x - 1;
y_next <=
   y0 when in_loop = '0'
      else yb;
yb <= ya when e2_lt_dx = '1'
         else y;
ya <= y + 1 when down = '1'
            else y - 1;
break <=
 '1' when x = x1 and y = y1
     else '0';

process (clk)
begin
  if rising_edge(clk) then
    err <= err_next;
    x <= x_next;
    y <= y_next;
  end if;
end process;
```

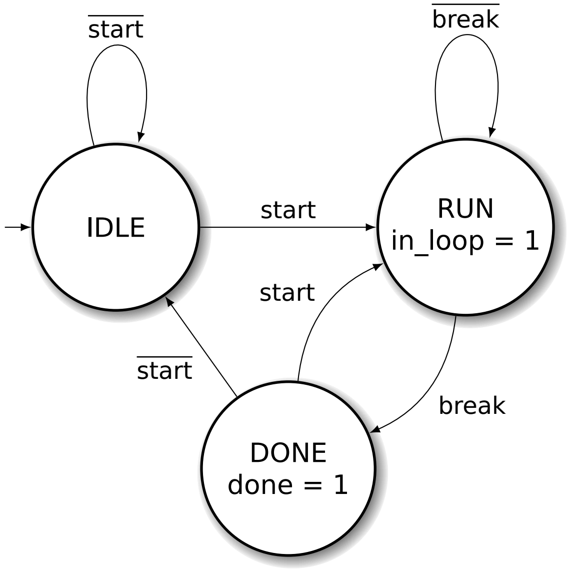
Interface and Types

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity lines is
  port (
    clk            : in  std_logic;
    rst            : in  std_logic;
    x0, y0, x1, y1 : in  signed(10 downto 0);
    x_p, y_p       : out signed(10 downto 0);
    start          : in  std_logic;
    done, plot     : out std_logic);
end lines;

architecture datapath of lines is
  signal x1x0, y1y0, dx, dy                : signed(10 downto 0);
  signal down, right, e2_lt_dx, e2_gt_dy : std_logic;
  signal in_loop, break                    : std_logic;

  signal err, err1, err2, e2, err_next   : signed(11 downto 0);
  signal x, y, x_next, y_next,
         xa, xb, ya, yb                    : signed(10 downto 0);
```

Timing


clk
(x0,y0)
(0,0)
(5,3)
(x1,y1)
(7,4)
(6,4)
start
done
in_loop
break
x
0
1
2
3
4
5
6
7
5
6
y
0
1
2
3
4
3
4
err
3
6
2
5
1
4
0
3
0
dx, dy
7, −4
1, −1

Control FSM



$\overline{\text{start}}$
$\overline{\text{break}}$
IDLE
start
RUN
in_loop = 1
start
$\overline{\text{start}}$
break
DONE
done = 1

Control FSM in VHDL

```vhdl
type states is (IDLE_STATE, RUNNING_STATE, DONE_STATE);
signal state : states;

fsm : process (clk)
begin
  if rising_edge(clk) then
    if rst = '1' then
      state <= IDLE_STATE;
    else
      case state is
        when IDLE_STATE =>
          if start = '1' then state <= RUNNING_STATE; end if;
        when RUNNING_STATE =>
          if break = '1' then state <= DONE_STATE;    end if;
        when DONE_STATE =>
          if start = '1' then state <= RUNNING_STATE;
                         else state <= IDLE_STATE;    end if;
end case; end if; end if; end process;

in_loop <= '1' when state = RUNNING_STATE else '0';
done    <= '1' when state = DONE_STATE    else '0';
plot    <= in_loop;
```

The Framebuffer

```vhdl
component fbmem port (
    data        : in std_logic_vector(0 downto 0);
    rdaddress   : in std_logic_vector(18 downto 0);
    rdclock     : in std_logic;
    wraddress   : in std_logic_vector(18 downto 0);
    wrclock     : in std_logic;
    wren        : in std_logic;
    q           : out std_logic_vector(0 downto 0));
end component;

frame_buffer_memory : fbmem port map (
  rdaddress => std_logic_vector(rdaddress(18 downto 0)),
  rdclock   => VGA_CLK,
  q         => q,
  data      => data,
  wraddress => std_logic_vector(wraddress(18 downto 0)),
  wrclock   => clk,
  wren      => wren);

read_data <= '1' when q(0) = '1'         else '0';
data      <= "1" when write_data = '1' else "0";

rdaddress <= ("0000000000" & VGA_X) +
             (VGA_Y & "0000000") + (VGA_Y & "000000000");
wraddress <= ("0000000000" & VGA_XW) +
             (VGA_YW & "0000000") + (VGA_YW & "000000000");
```