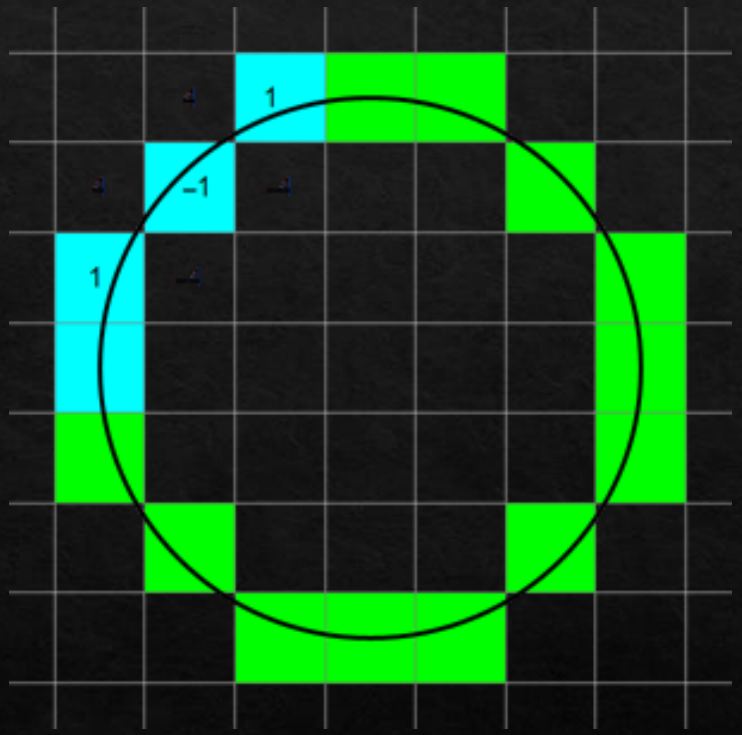
Algoritmo de Bresenham

Rasterizando um círculo

Desenhar uma primitiva 2D utilizando “line scan”

O círculo possui propriedades simétricas em qualquer ponto

Se podemos pintar um pixel (x,y) então podemos fazer o mesmo processo nos outros 7 pontos


1
−1
1

Implementação

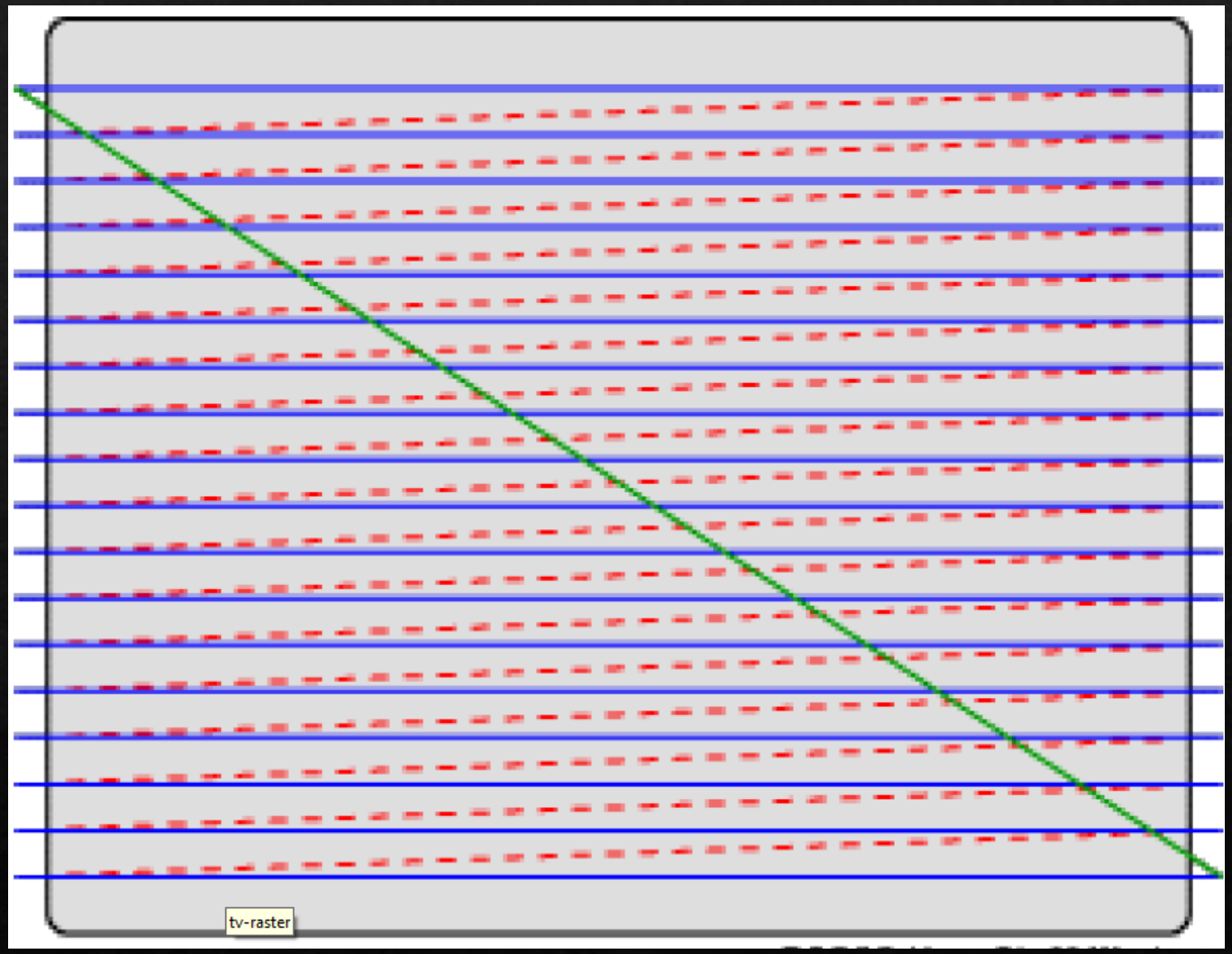
Implementamos um driver VGA com um buffer de 800x600@60Hz.

Pixel Clock de 40Mhz (1000ns)

Memória indexada para guardar os pontos a serem feitos o scan.

Dispositivos de imagens atuais precisam de 50 ~ 60 Hz para poder exibir imagens sem distorções (flickering)

Os feixes de elétrons são escaneados linha por linha e apagados em intervalos específicos, da esquerda para direita e cima para baixo.


tv-raster

SVGA Signal 800 x 600 @ 60 Hz timing

General timing

Screen refresh rate	60 Hz
Vertical refresh	37.878787878788 kHz
Pixel freq.	40.0 MHz

Horizontal timing (line)

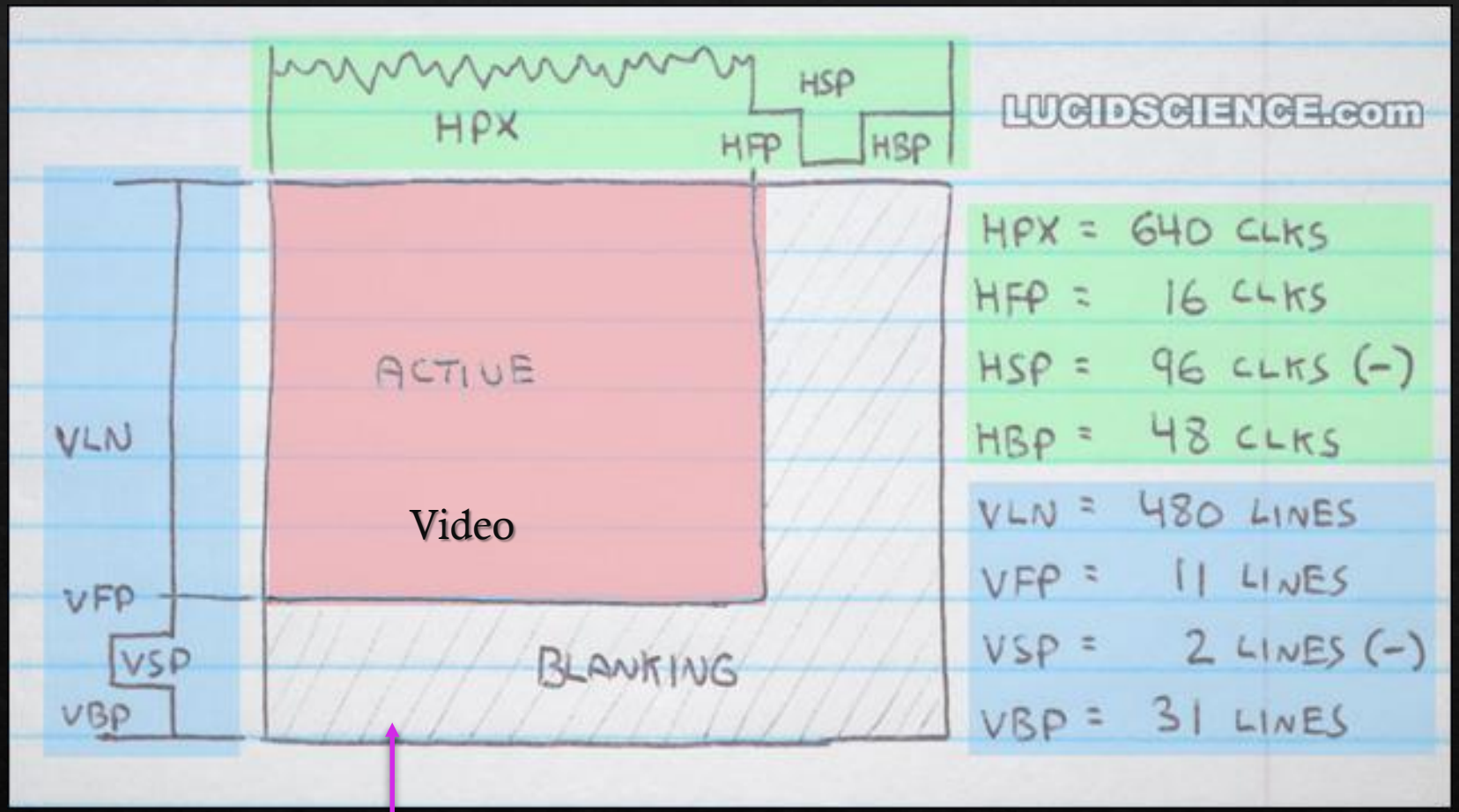
Polarity of horizontal sync pulse is positive.

Scanline part	Pixels	Time [µs]
Visible area	800	20
Front porch	40	1
Sync pulse	128	3.2
Back porch	88	2.2
Whole line	1056	26.4

Vertical timing (frame)

Polarity of vertical sync pulse is positive.

Frame part	Lines	Time [ms]
Visible area	600	15.84
Front porch	1	0.0264
Sync pulse	4	0.1056
Back porch	23	0.6072
Whole frame	628	16.5792



Não há imagem nessa região

É preciso de uma fração do clock da basys 3 para percorrer toda a tela.

Clock divider!
100/2.5 = 40

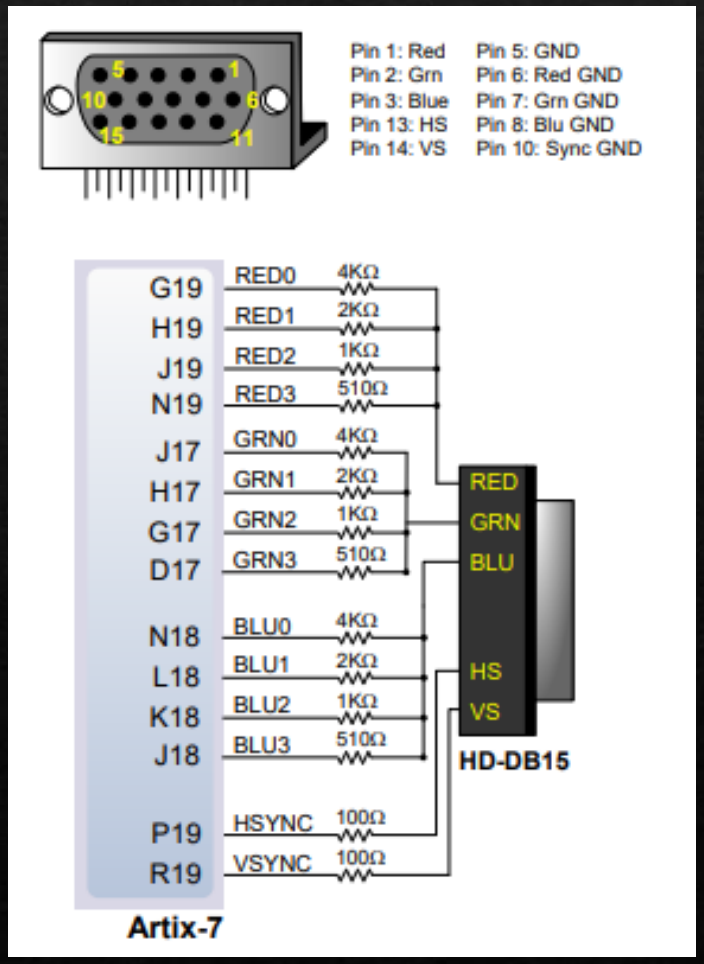
São necessários dois sinais de sincronização:

H: 128 (3.2 µs) - por linha
V: 4 (0.1056 µs) - por frame

Basys 3 tem um port VGA de 12 bits

Podemos controlar a intensidade da cor

Ex: VGA_R = 1 e VGA_B = 1 (magenta)



Alocando clock (divisor fracional)

Consiste em contar até “esperar” o sinal “virar”.

```
reg [15:0] cnt = 0;
reg pix_stb = 0;
always @(posedge clk)
    {pix_stb, cnt} <= cnt + 16'h6666;
```

A cada 1/(2.5) ciclos de clock, o sinal “pix_stb” deve estar em alto por um período de de 40Mhz e (2^16)/2.5 = 16’h666.

Algoritmo

```
initial
begin
    y0 = r;
    x0 = 0;
    d = 3 - 2 * r;

    points[y_center + y0] = {x_center + x0,x_center - x0};
    points[y_center - y0] = {x_center + x0,x_center - x0};
    points[y_center + x0] = {x_center + y0,x_center - y0};
    points[y_center - x0] = {x_center + y0,x_center - y0};

    while(y0 >= x0)
    begin
        x0 = x0 + 1;
        if(d > 0)
        begin
            y0 = y0 - 1;
            d = d + 4 * (x0 - y0) + 10;
        end
        else
        begin
            d = d + 4 * x0 + 6;
        end
        points[y_center + y0] = {x_center + x0,x_center - x0};
        points[y_center - y0] = {x_center + x0,x_center - x0};
        points[y_center + x0] = {x_center + y0,x_center - y0};
        points[y_center - x0] = {x_center + y0,x_center - y0};
    end
end
```

Componentes