VICKYGRAPH LIBRARY V0.2

Foenix C256U / c256U+ / FMX

https://github.com/econtrerasd/VickyGraph

Table of content

REQUIREMENTS

The VickyGraph library requires:

1. A Foenix U, Foenix U+ or Foenix FMX computer
   (It's meant to be portable to any Foenix computer supporting Vicky II)
2. The Calypsi Compiler for the WDC 65816 target

   https://www.calypsi.cc/

3. Calypsi's Foenix 256 board support (also available on the previous link)

INSTALLING THE LIBRARY

The VickyGraph library can be obtained from the following GitHub

https://github.com/econtrerasd/VickyGraph

Get the Zip file and uncompressed the whole directory in a subdirectory, the resulting files are based on Calypsi's "Hello World for c256U" that already include the Foenix C256 board support.

If you already have Calypsi Installed, you can go to the unpacked directory and build the project by typing

```
make grfdemo.pgz
```

to compile the demo.

DESCRIPTION

This graphics library provides a set of functions to:

1. Initialize the graphic capabilities of the Vicky II
2. Select the Bitmap mode from the available options provided by the Vicky Chip
   a. 640 x 480
   b. 800 x 600
   c. 320 x 240*

d. 400 x 300*

** Double pixel modes*

3. Render geometry (point, line, circles, etc..) in the bitmap Graphics mode selected

The Motivations to create this library were:

- Familiarize with the Calypsi Compiler / Linker
- Explore the Hardware capabilities of the Foenix computer
  - VDMA
  - FIFO
- Understand the achievable speed of bitmap graphics when working with C
- Provide the community with a native library for C programmers (Calypsi!) to easily create bitmap graphics
- Research and understand classic algorithms for graphics primitives (Bresenham algorithms)

The source code is provided on the github, so anyone can learn from it and/or improve it

GRAPHICS SETUP FUNCTIONS

int vickyVideoMode (text, txtOverlay, bitmap, tiles, sprites, gammaFix, videoOff, resolution)

Objective: Initialize graphics modules on the Vicky II graphics engine.

Parameter	Type	Description
text	boolean	Enables the text engine of VickyII
txtOverlay	boolean	Enables text to be overlayed over graphics and tiles
bitmap	boolean	Enables the bitmap engine of VickyII
tiles	boolean	Enables the bitmap engine of VickyII
sprites	boolean	Enables the sprite engine of VickyII
gammafix	boolean	Enables/disables compensation for monitor gamma (use if screen is too dark/too bright)
videoOff	boolean	Shuts down all graphic output from VickyII (improves processing speed)
resolution	Selects the screen resolution from the following options	Value: 0 / 1 / 2 / 3 — Resolution: 640 x 480 / 800 x 600 / 320 x 240 / 400 x 300

On success:

- *Identifies the total video memory according to the Foenix Computer model and updates the global variable* ***maxVickyMemory***
- *Depending on the selected resolution populates the global variables* ***vickyResX, vickyResY*** *with the maximum X and Y resolution of the selected mode.*
- *The function returns a positive number with the value of the Vicky MASTER_CTRL_REG*

In case of an error *- the function returns a negative number. The only error scenario happens when you request an invalid resolution mode outside the range of 0-3.*

Int vickyBitmap (page, enable, address, lut, collision)

Objective: Configure the bitmap mode provided by VickyII

Parameter	Type	Description
page	integer	Use to select the bitmap page. Possible values are 0 or 1
enable	boolean	Show/hide bitmap
address	long	Offset to indicate where in video graphics memory the bitmap begins (default value is 0x0)
lut	integer	Selects the palette from the available look up tables containing RGB values. Possible values are 0-7
collision	boolean	Enables the collision support for sprites vs the Bitmap

On success:

- *Populates the **vickyBitmapPage** global variable with the current Bitmap page enabled (all graphic commands will be directed to this page)*
- *Updates the global array **vBitPlane [0-1]** with true or false to keep track of which bitmap pages are enabled/disabled.*
- *The function returns 0 on a successful call*

***In case of an error** the function returns one of these possible values:*

-1	*in case that an invalid bitmap page is requested (valid range 0-1)*
-2	*When the offset requested for the bitmap would prevent the page to fit in memory*

Example: Bitmap Initialization by calling vickyVideoMode & vickyBitmap

```
Int SetupScreen ()
{
  int status;
  // 1. Ask Vicky to enable: text, textoverlay, bitmap, gammafix
  //    and video mode 2 - 320 x 240
  status=vickyVideomode(true,true,true,false,false,true,false,2);
  if (status!=-1)
  {
     // Success on setting the video mode
     // 2. Let's enable bitmap page 0, at default address
     //    use first LUT color palette, no collision
     status=vickyBitmap (0,true,0,0,false);
     if (status==0)
     // Success!
     {
         BORDER_CTRL_REG = 0;
         return 0;
     }
     else
     //Error setting bitmap
     {
         printf ("Error %d while setting bitmap",status);
         return status;
     }
  else
  //Error setting videomode
  {
     printf ("Error %d while setting VideoMode",status);
     return status;
  }
}
```

Code Discussion:

1. *Enable Vicky II features:*

```
status=vickyVideomode(true,true,true,false,false,true,false,2);
```

This enables the following modes:

text	On
txtOverlay	On
bitmap	On
tiles	Off
sprites	Off

gammafix	On
videoOff	Off
resolution	2 - (320 x 240)

2. *Setup the Bitmap mode with the following statement:*

```
status=vickyBitmap (0,true,0,0,false);
```

This enables the following bitmap characteristics:

page	0
enable	On
address	0 (offset)
lut	0 (first palette)
Collision	Off

GRAPHICS RENDERING FUNCTIONS

To paint in the bitmap area, you can use any of the graphics rendering functions and variables listed in the following 2 tables to control what is drawn onscreen.

Functions to draw basic Elements to the bitmap

Element	Function
Erase Bitmap	void clrBitmap()
Point	void plot (x, y, col)
Line	void plot_line (x0, y0, x1, y1, col)
Rectangle	void plot_rectangle (x0, y0, x1, y1, col)
Filled Rectangle	void plot_solid_rectangle (x0, y0, x1, y1, col)
Bezier Curve	void plot_bezier (x0, y0, xc0, yc0, xc1, yc1, x1, y1, col)
Circle	void plot_circle (x, y, r, col)
Filled Circle	void plot_solid_circle (x, y, r, col)
Ellipse	void plot_ellipse (x0, y0, a, b, col)
Filled Ellipse	void plot_solid_ellipse (x0, y0, a, b, col)
Fill Area	void floodFill (x, y, col)

Variables with information about bitmap

Variable	Type	Content Description
vickyResX	long	Max X resolution in current bitmap mode
vickyResY	long	Max Y resolution in current bitmap mode
vickyBitmapPage	int	Current bitmap page (0 or 1)
maxVickyMemory	long	Max VRAM memory detected

Description of graphic rendering functions

void clrBitmap()

Objective: clears the current bitmap page

Takes no parameters

void plot (x, y, col)

Objective: sets pixel at (x,y) with color (col)

Parameter	Type	Description
x	integer	X coordinate of pixel
y	integer	Y coordinate of pixel
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT)

void plot_line (x0, y0, x1, y1, col)

Objective: Draw a line from (x0,y0) to (x1,y1) using color (col)

Notes: This function has optimizations to draw horizontal lines faster

Parameter	Type	Description
x0	integer	X coordinate of initial line point
y0	integer	Y coordinate of initial line point
x1	integer	X coordinate of end line point
y1	integer	Y coordinate of end line point
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

void plot_rectangle (x0, y0, x1, y1, col)

Objective: Draw a hollow rectangle from (x0,y0) to (x1,y1) using color (col)

Parameter	Type	Description
x0	integer	X coordinate of top left point of rectangle
y0	integer	Y coordinate of top left point of rectangle
x1	integer	X coordinate of bottom right point of rectangle
y1	integer	Y coordinate of bottom right point of rectangle
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

void plot_solid_rectangle (x0, y0, x1, y1, col)

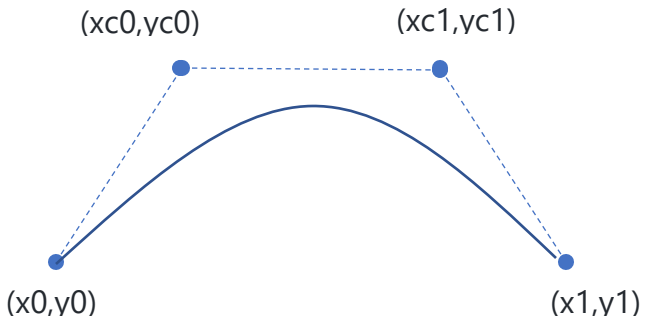
Objective: Draw a filled rectangle from (x0,y0) to (x1,y1) using color (col)

Parameter	Type	Description
x0	integer	X coordinate of top left point of rectangle
y0	integer	Y coordinate of top left point of rectangle
x1	integer	X coordinate of bottom right point of rectangle
y1	integer	Y coordinate of bottom right point of rectangle
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

void plot_bezier (x0, y0, xc0, yc0, xc1, yc1, x1, y1, col)

Objective: draws a curved line from (x0,y0) to (x1,y1) using color (col).

The curve path is controlled by 2 control points (xc0, yc0) and (xc1,yc1), the line curves toward those control points but won't touch any of those points.



For any math fanatic out there here is a summary of the algorithm:

This implementation focuses on a Bezier curve with 4 control points, and so it is calculated with the following equations for x and y

$$x=(1-u)^3 * x[0]+3u(1-u)^2 * x[1]+3u^2 * (1-u) * x[2]+u^3 * x[3]$$
$$y=(1-u)^3 * y[0]+3u(1-u)^2 * y[1]+3u^2 * (1-u) * y[2]+u^3 * y[3]$$

The equations are evaluated with values of 'u' between 0 and 1 (incremented in steps of 0.02)

x[0],y[0] = initial point of line
x[1],y[1] = first control point
x[2],y[2] = second control point
x[3],y[3] = end point of line

Parameter	Type	Description
x0	integer	X coordinate of initial line point
y0	integer	Y coordinate of initial line point
xc0	integer	X coordinate of first control point
xy0	integer	Y coordinate of first control point
xc1	integer	X coordinate of second control point

yc1	integer	Y coordinate of second control point
x1	integer	X coordinate of end line point
y1	integer	Y coordinate of end line point
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

void plot_circle (x, y, r, col)

Objective: draws a hollow circle onscreen centered on (x,y) with radius (r) using color (col)

Parameter	Type	Description
x	integer	X coordinate of center point
y	integer	Y coordinate of center point
r	integer	Radius of circle
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

void plot_solid_circle (x, y, r, col)

Objective: draws a solid circle onscreen centered on (x,y) with radius (r) using color (col)

Parameter	Type	Description
x	integer	X coordinate of center point
y	integer	Y coordinate of center point
r	integer	Radius of circle
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

void plot_ellipse (x, y, a, b, col)

Objective: draws a hollow ellipse centered on (x,y) with width (a) and height (b) using color (col)

Parameter	Type	Description
x	integer	X coordinate of center point
y	integer	Y coordinate of center point
a	integer	Ellipse width
b	Integer	Ellipse height
Col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

void plot_solid_ellipse (x, y, a, b, col)

Objective: draws a solid ellipse centered on (x,y) with width (a) and height (b) using color (col)

Parameter	Type	Description
x	integer	X coordinate of center point
y	integer	Y coordinate of center point
a	integer	Ellipse width
b	Integer	Ellipse height
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

void floodFill (x, y, col)

Objective: fills an area of pixels that share the same color as the pixel at (x,y) with a new color (col)

The algorithm will change pixel colors to the new color specified until it finds a pixel either left/right up or down that has a different color from the initial pixel

Parameter	Type	Description
x	integer	X coordinate of initial fill point
y	integer	Y coordinate of initial fill point
col	integer	Use specified color index (value between 0-255) in the current lookup table (LUT) to draw

READING VALUES FROM VIDEO MEMORY

So far, every function we have shown has 'seemingly' just put values in Video Memory, although there is one exception, the "floodFill" is a function that requires to probe pixels on the screen to decide if it should color the pixels around the initial coordinate.

VIDEO MEMORY ON VICKY

Unique to the Vicky Chip there is dedicated VRAM accessible by the chip, this allows the chip to access it fast enough to support all Bitmaps / Tiles /Sprites / Text at the same time, but the compromise is that the main CPU can't directly read video memory just as it would read regular RAM

There are two ways for reading VRAM

- The FIFO
- VDMA to DMA transfers

Each mechanism has its own pros and cons.

THE FIFO

The Vicky Chip provides a mechanic for the CPU to request values from Video Memory through a "First in First Out" (FIFO) queue mechanism. This queue can hold up to 1024 bytes of memory.

Using the FIFO is quite straightforward

1. Signal Vicky to Initialize the FIFO queue
2. Wait for the FIFO queue to Empty
3. Read all memory addresses from VRAM just as you would read memory from any other address, this will add these addresses to the FIFO queue.
4. Vicky will wait for an opportunity to read the values in the FIFO queue and will signal through a flag when the data is ready
5. Read the real content of the requested memory addresses from the 'Byte Port' of the FIFO
6. Just as a good practice signal Vicky to initialize the FIFO queue again

Example: Implementation of Read VRAM Buffer with FIFO

```
Int read_vram_buffer (uint8_t volatile __far *p, int length,
uint8_t __far *buffer)
 {
      int i;
      // 1. Send clear fifo signal before reading data
      VMEM2CPU_CTRL_REG = VMEM2CPU_Clear_FIFO;
      VMEM2CPU_CTRL_REG = 0;
      // 2. Wait for fifo to clear
      if (__fifo_count>0)
            while (__fifo_count>0)
            ;
// 3. Ask for as many bytes as the value - length
      i=0;
   while (i<length)
  {
      *p;         // dummy read
      i++;        // increment pixel request counter
      p++;        // increment VRAM pointer
  }
// 4. Wait for fifo buffer
      while (__fifo_count & 0x8000 != 0x8000)
      ;
// 5. Move all requested bytes into RAM buffer
      i=0;
      while (__fifo_count > 0)
        *(buffer+(i++)) = __byte_port;
// 6. send clear fifo signal before exiting
      VMEM2CPU_CTRL_REG = VMEM2CPU_Clear_FIFO;
      VMEM2CPU_CTRL_REG = 0;
      return i;
}
```

Functions in the library using the FIFO

int getPixel (x, y)

Objective: Returns the pixel color at coordinate (x,y)

Parameter	Type	Description
x	integer	X coordinate of initial fill point
y	integer	Y coordinate of initial fill point

On success:

- *The function returns a positive number with the value of the pixel color index corresponding to the current LUT used in the bitmap*

In case of an error *- the function returns a negative number. The only error scenario happens when you request a pixel outside the (x,y) range of the current video mode.*

int getScanLine (y, *buffer)

Objective: Returns the pixel color at coordinate (x,y)

Parameter	Type	Description
Y	integer	X coordinate of initial fill point
***buffer**	uint8_t __far	Buffer size should be -at least as long- as the max x coordinate of the current video mode. It is recommended to use a 1024 byte buffer.

On success:

- *Returns the number of bytes read*
- *The function retries until it succeeds, so there are no error cases*

Example usage:

```
// function that searches for a pixel where color = newColor
// returns x coordinate of first pixel that has this color in
// line y of the bitmap
FindNewColor(int y, int newColor)
{
    uint8_t Buffer [1024];
    // get all pixels from line y from bitmap
    st=getScanLine(y,Buffer);
    x=0;
    while(x < (vickyResX) && Buffer[x] != newColor)
      {
         x++;
       }
    If (x<vickyResX)
    // if new color found, returns position
    {
        return x-1;
    }    // if no new color found returns -1

    return -1;
}
```