NovaVGA
Reference Manual

www.micro-nova.com

OVERVIEW

NovaVGA is a low-cost shield board for the Arduino that provides an easy-to-use VGA graphics output. The adapter is controlled via a simple SPI interface from the Arduino (or any SPI-compatible microcontroller.) *NovaVGA* keeps a 160x120 pixel frame buffer in SRAM and outputs it to a monitor at an industry standard 640x480 @ 60Hz VGA signal. Meanwhile, the microcontroller is free to write to the SRAM frame buffer at any time via the SPI interface. An Arduino library is provided with several examples.

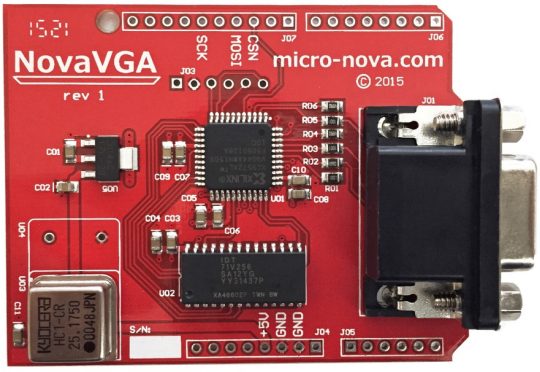

Figure 1 - NovaVGA board

SPECIFICATIONS

- Arduino Uno shield form-factor
- 160x120 pixels @ 6-bit color video buffer (2^6 = 64 possible colors)
- 640x480 @ 60Hz physical resolution (25.175MHz pixel clock)
- Controlled via standard SPI mode 1 interface (consumes only three Arduino pins!)
- Arduino library with examples: color palette, Mandelbrot, Tetris and text console
- Hardware powered by a Xilinx XC9572XL CPLD. (User modifiable via JTAG interface).

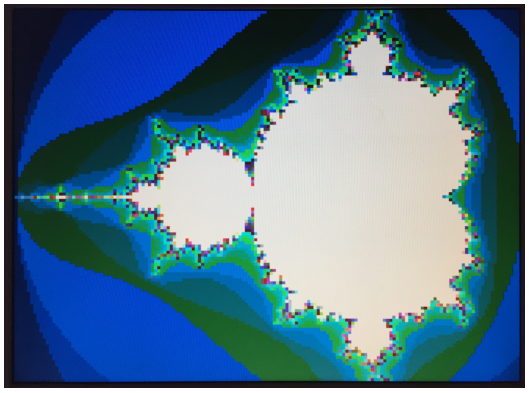

Figure 2 - Mandelbrot set on NovaVGA

GETTING STARTED

All of the required pins are labeled on the NovaVGA board. A 5V power input is required, along with SCK, CSN, and MOSI for SPI communication. If using an Arduino Uno, all of the NovaVGA pins line up with the proper Arduino pins, and NovaVGA can be plugged directly into the Arduino.

An Arduino library is provided at www.micro-nova.com.
See the next page for a listing of some of the available functions:

LIBRARY FUNCTIONS

```
void NovaVGA.init(uint8_t cspin)
```

Initialize the NovaVGA module. Chip select pin specified by cspin. (For Arduino Uno, this is pin 10.)

```
void NovaVGA.writePixel(uint8_t x, uint8_t y, uint8_t color)
```

Write a pixel to the specified coordinate (x,y). Top-left is (0,0), bottom-right is (159, 119).
*Color is byte where bits[5:4] are **RED**, bits[3:2] are **GREEN**, and bits[1:0] are **BLUE**. Bits[7:6] unused.*

```
void NovaVGA.writePixel(Point p, uint8_t color)
```

Write a pixel to the specified coordinate at point p.

```
void NovaVGA.fillScreen(uint8_t color)
```

Fill the screen with a specified color.

```
void NovaVGA.fillRect(uint8_t x, uint8_t y, uint8_t w, uint8_t h, uint8_t color)
```

Draw a filled rectangle at the specified coordinates.
Where: x = left coordinate, y = top coordinate, w = width of rectangle, h = height of rectangle

```
void NovaVGA.fillRect(Rect r, uint8_t color)
```

Draw a filled rectangle at the specified coordinates.
Where: r =rectangle coordinates.

```
void NovaVGA.drawChar(const char *bitmap, uint8_t x, uint8_t y, uint8_t color)
```

Draw a letter at the specified location (x,y).
*Pointer *bitmap points to a 8x8 array with character ROM.*

```
void NovaVGA.drawChar(char ch, Point p, uint8_t color)
```

Draw a letter at the specified location (p).
Character(ch) is an ASCII byte.

```
void NovaVGA.drawChar(char ch, uint8_t x, uint8_t y, uint8_t color)
```

Draw a letter at the specified location (x,y).
Character(ch) is an ASCII byte.

```
void NovaVGA.drawString(const String str, uint8_t x, uint8_t y, uint8_t color)
```

Draw a string at the specified location (x,y). Newline characters "\n" are supported.

```
void NovaVGA.drawString(const String str, Point p, uint8_t color)
```

Draw a string at the specified location (p). Newline characters "\n" are supported.

THEORY OF OPERATION

At the heart of NovaVGA is a CPLD that acts as a VGA graphics controller at 640x480 @ 60Hz physical resolution. Timing is provided by a 25.175MHz crystal oscillator. The CPLD is connected to an SRAM which acts as a display buffer. While the unit is powered, the contents of the SRAM are continuously displayed to the monitor at a 160x120 effective resolution. Meanwhile, the SRAM is seamlessly multiplexed with a SPI receiver, allowing the attached Arduino (or other SPI-capable microcontroller) to write into the SRAM via SPI.

In each SPI transfer, three bytes are sent: color, X coordinate and Y coordinate. The color is 6-bits, in the format **RRGGBB**, with the top two bits of the byte over SPI ignored. Valid X coordinates range from 0 to 159, while valid Y coordinates range from 0 to 119. The coordinate plane starts with (0,0) in the upper left-hand corner.

Below is a diagram illustrating the SPI timing. Three bytes are sent MSB-first in SPI mode 1 (CPOL = 0, CPHA = 1). The microcontroller (SPI master) shifts out data on the rising SCLK edge, while the NovaVGA (SPI slave) shifts in data on the falling SCLK edge.

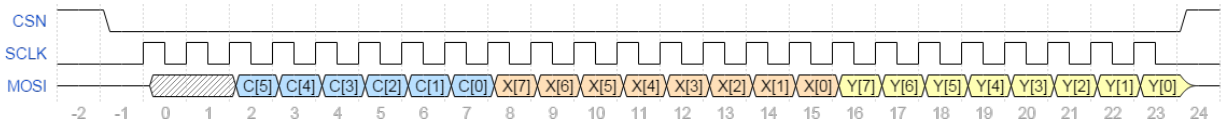

Figure 3 - SPI waveform

Below is a key snippet of source from the Arduino library that shows how a pixel is sent to the NovaVGA:

```
SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE1));
digitalWrite(cspin_, LOW);      // Assert chip select
SPI.transfer(color);            // Send in the coordinates and color via SPI
SPI.transfer(x);
SPI.transfer(y);
digitalWrite(cspin_, HIGH);     // Deassert chip select
SPI.endTransaction();           // Release the SPI bus
```

While the SPI CSN pin is asserted (logic level low), the SPI receiver implemented inside the NovaVGA CPLD is enabled and the bytes can be shifted into NovaVGA. After the bytes are shifted in, when SPI CSN is released (logic level high), the specified pixel is then written into the SRAM. The video output is registered such that writes into the SRAM do not disturb the video display.