# VGA Text Mode on the DE10-Lite FPGA

Gabe Wehrle – ENP261 Personal Project

## Description

80 column by 30 row, 16 color, VGA text mode was implemented on the DE10. This was done from scratch and without outside reference aside from the existing VGA timing module, and an existing glyph binary file. A 4.8k RAM module (implemented as 8k) contains the current character frame and a 4k ROM module contains the glyph information. A char_counter module is used to generate timing signals to step through the character frame and glyph ROM. To bring all the control signals back into sync with one another, registers are used to delay the step across each character row, and to delay the VGA vertical sync, horizontal sync, and valid signals. A palette controller is used to determine what color should be displayed for any given pixel using multiplexing. Refer to "rtl.pdf" for a system diagram.

### Character Counter

The counter steps down and across each row of each character from 0 to 15 and 0 to 7 respectively. The counter also steps down each row of characters as they are displayed on screen (i.e. out of 30 rows as opposed to 525 rows). Figure 1 shows this in more detail.

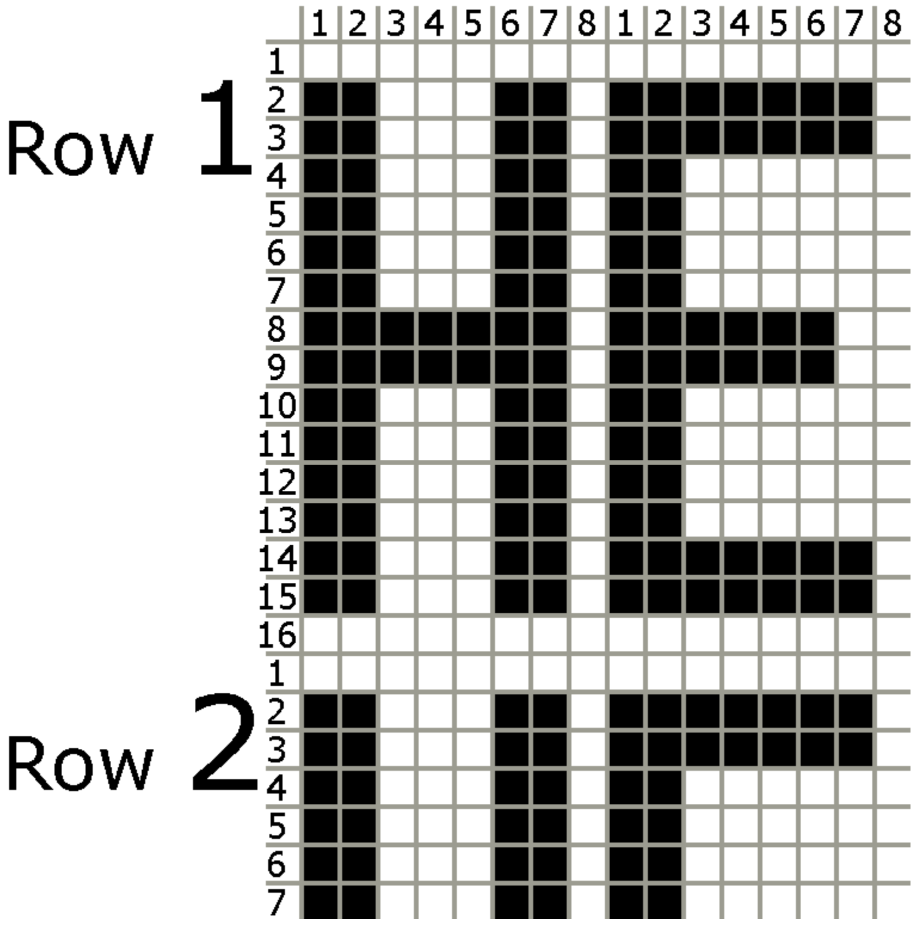


*Figure 1 – Character row and column layout*

## Character Frame RAM and Glyph ROM Layout

The character frame is made up of 2400, 16-bit (2 byte) words laid out as shown in Table 1. The foreground and background colors are each specified as 4-bit. The character is specified as 8-bit and is ASCII compatible.

| | +0 | +1 |
|---|---|---|
| 0x00 | Foreground \| Background | Character |
| 0x02 | FG \| BG | CHAR |

*Table 1 – Character frame RAM layout*

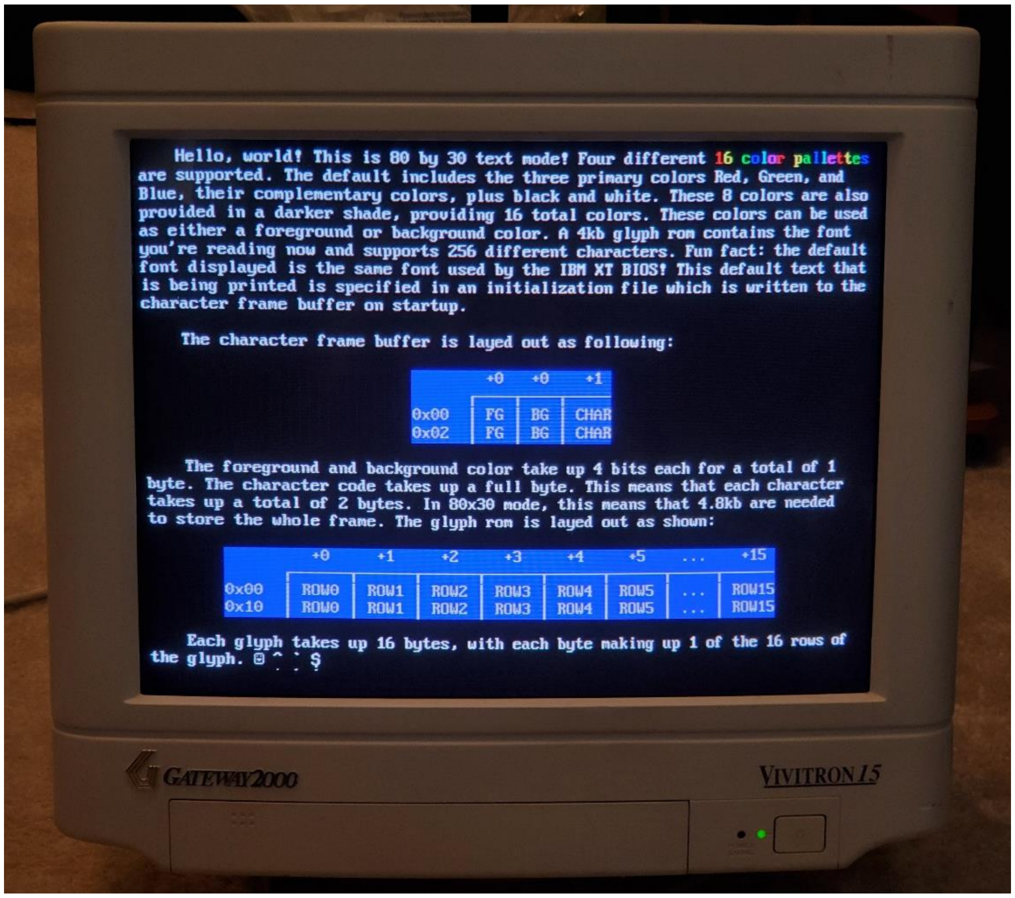
The glyph ROM is made up of 256 characters, with each row of each character specified as 1 byte as shown in Table 2.

| | +0 | +1 | +2 | … | +F |
|---|---|---|---|---|---|
| 0x00 | Row 0 | Row 1 | Row 2 | … | Row 15 |
| 0x10 | Row 0 | Row 1 | Row 2 | … | Row 15 |

*Table 2 – Glyph ROM layout*

## Results

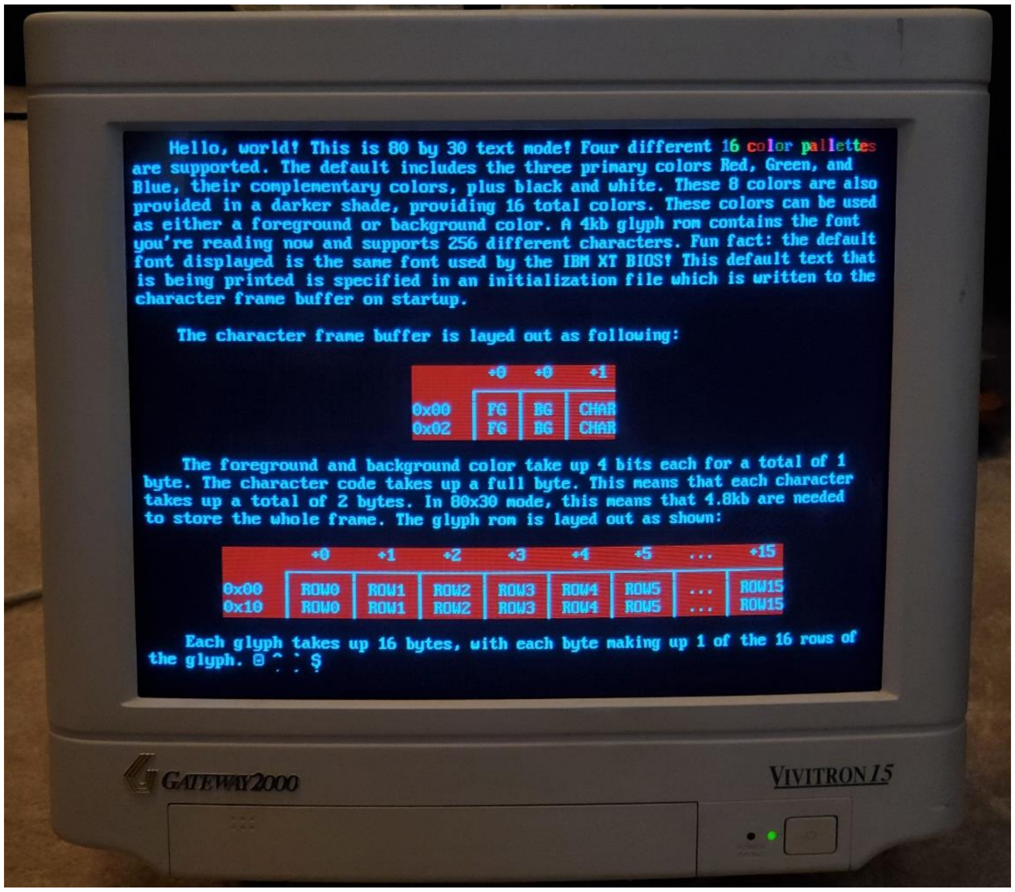
The system was tested by attaching an external monitor to the VGA port on the FPGA. As shown in Figure 2, the system was successfully implemented. However, a minor bug still exists which causes the top row of each glyph to wrap around to the bottom. This is only noticeable on glyphs which extend fully to the top of the row and can be seen clearly on the last three characters displayed in Figure 2.



```
    Hello, world! This is 80 by 30 text mode! Four different 16 color pallettes
are supported. The default includes the three primary colors Red, Green, and
Blue, their complementary colors, plus black and white. These 8 colors are also
provided in a darker shade, providing 16 total colors. These colors can be used
as either a foreground or background color. A 4kb glyph rom contains the font
you're reading now and supports 256 different characters. Fun fact: the default
font displayed is the same font used by the IBM XT BIOS! This default text that
is being printed is specified in an initialization file which is written to the
character frame buffer on startup.

     The character frame buffer is layed out as following:

                               +0   +0    +1
                             ┌────┬────┬─────
                       0x00  │ FG │ BG │ CHAR
                       0x02  │ FG │ BG │ CHAR

     The foreground and background color take up 4 bits each for a total of 1
byte. The character code takes up a full byte. This means that each character
takes up a total of 2 bytes. In 80x30 mode, this means that 4.8kb are needed
to store the whole frame. The glyph rom is layed out as shown:

                 +0     +1     +2     +3     +4     +5    ...    +15
             ┌──────┬──────┬──────┬──────┬──────┬──────┬─────┬──────
       0x00  │ ROW0 │ ROW1 │ ROW2 │ ROW3 │ ROW4 │ ROW5 │ ... │ ROW15
       0x10  │ ROW0 │ ROW1 │ ROW2 │ ROW3 │ ROW4 │ ROW5 │ ... │ ROW15

     Each glyph takes up 16 bytes, with each byte making up 1 of the 16 rows of
the glyph. ⊡ ^ ⁚ $
```

*Figure 2 – VGA output showing the system was fully implemented*

Figure 3 shows an alternate color palette being displayed, demonstrating the feature was fully implemented.


Hello, world! This is 80 by 30 text mode! Four different 16 color pallettes are supported. The default includes the three primary colors Red, Green, and Blue, their complementary colors, plus black and white. These 8 colors are also provided in a darker shade, providing 16 total colors. These colors can be used as either a foreground or background color. A 4kb glyph rom contains the font you're reading now and supports 256 different characters. Fun fact: the default font displayed is the same font used by the IBM XT BIOS! This default text that is being printed is specified in an initialization file which is written to the character frame buffer on startup.

The character frame buffer is layed out as following:

| | +0 | +0 | +1 |
|---|---|---|---|
| 0x00 | FG | BG | CHAR |
| 0x02 | FG | BG | CHAR |

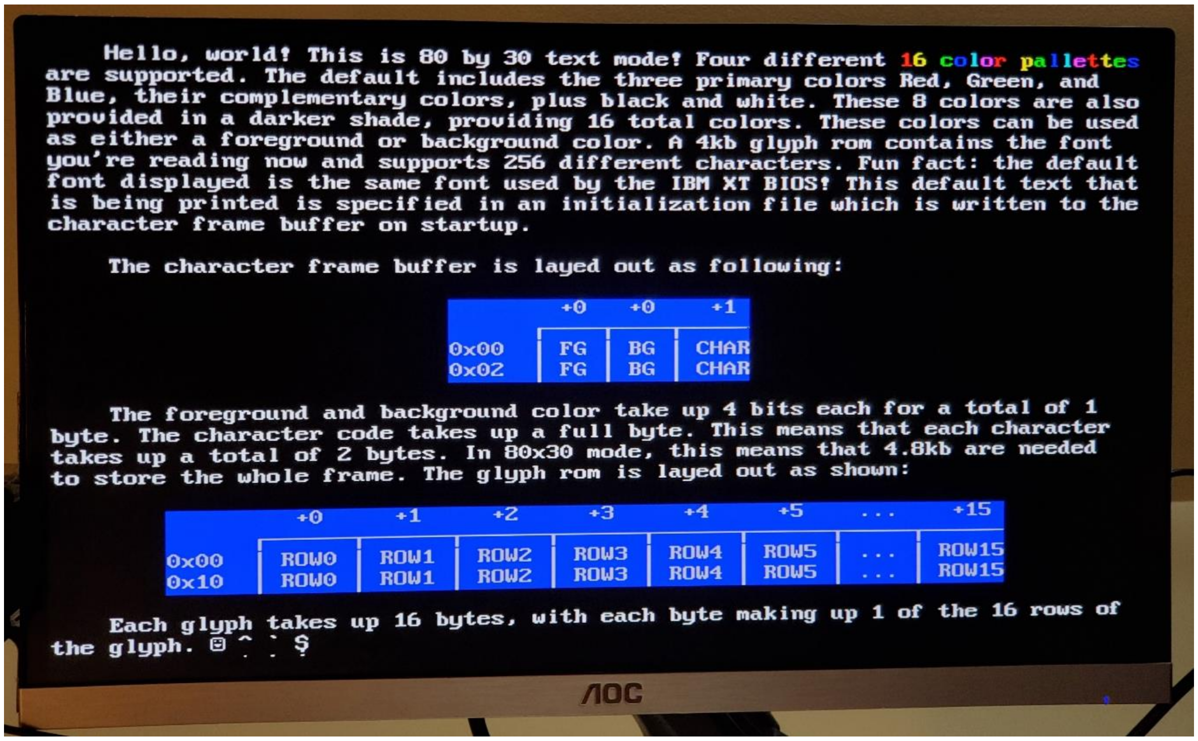
The foreground and background color take up 4 bits each for a total of 1 byte. The character code takes up a full byte. This means that each character takes up a total of 2 bytes. In 80x30 mode, this means that 4.8kb are needed to store the whole frame. The glyph rom is layed out as shown:

| | +0 | +1 | +2 | +3 | +4 | +5 | ... | +15 |
|---|---|---|---|---|---|---|---|---|
| 0x00 | ROW0 | ROW1 | ROW2 | ROW3 | ROW4 | ROW5 | ... | ROW15 |
| 0x10 | ROW0 | ROW1 | ROW2 | ROW3 | ROW4 | ROW5 | ... | ROW15 |

Each glyph takes up 16 bytes, with each byte making up 1 of the 16 rows of the glyph. ☺ ^ ⁚ $

*Figure 3 – An alternate color palette being displayed*

Figure 4 shows the same text displayed on an LCD as opposed to a CRT. The signal is displayed, however since the motor does not “know” the signal’s resolution, the picture is stretched to fill the entire width of the screen.



Hello, world! This is 80 by 30 text mode! Four different 16 color pallettes are supported. The default includes the three primary colors Red, Green, and Blue, their complementary colors, plus black and white. These 8 colors are also provided in a darker shade, providing 16 total colors. These colors can be used as either a foreground or background color. A 4kb glyph rom contains the font you're reading now and supports 256 different characters. Fun fact: the default font displayed is the same font used by the IBM XT BIOS! This default text that is being printed is specified in an initialization file which is written to the character frame buffer on startup.

The character frame buffer is layed out as following:

|  | +0 | +0 | +1 |
|---|---|---|---|
| 0x00 | FG | BG | CHAR |
| 0x02 | FG | BG | CHAR |

The foreground and background color take up 4 bits each for a total of 1 byte. The character code takes up a full byte. This means that each character takes up a total of 2 bytes. In 80x30 mode, this means that 4.8kb are needed to store the whole frame. The glyph rom is layed out as shown:

|  | +0 | +1 | +2 | +3 | +4 | +5 | ... | +15 |
|---|---|---|---|---|---|---|---|---|
| 0x00 | ROW0 | ROW1 | ROW2 | ROW3 | ROW4 | ROW5 | ... | ROW15 |
| 0x10 | ROW0 | ROW1 | ROW2 | ROW3 | ROW4 | ROW5 | ... | ROW15 |

Each glyph takes up 16 bytes, with each byte making up 1 of the 16 rows of the glyph. ☺ ^ : $

*Figure 4 – The same text as in Figures 2 and 3 being displayed on an LCD*

## Appendix – RAM Encoder

Since it is fairly involved and time consuming to encode the needed information to display a frame of text into a memory initialization file by hand, an external utility was written in C++ to allow text files to be easily encoded into a properly formatted HEX file. This is included in the "resources/ram_encoder/" folder and can be executed in a terminal.