VGA Adapter

The VGA Adapter is used to draw images on your computer monitor. An image consists of a rectangular array of picture elements, called *pixels*. Each pixel appears as a dot on the screen, and the entire screen consists of 320 columns by 240 rows of pixels. Pixels are arranged in a rectangular grid, with the coordinate (x,y) = (0, 0) at the top-left corner of the screen, and the coordinate (x,y) = (319, 239) at the bottom-right corner of the screen.

The VGA Adapter connects the Nios II processor to the DE1-SoC Video DAC chip which then outputs to your monitor. The interface to the adapter is identical to that of a memory: the address corresponds to the pixel you want to read/write, and the data you read/write from/to that address is the colour for that pixel.

Device	VGA Adapter
Configuration	320x240 pixel resolution, 80x60 character resolution, 16-bit colour
Input/Output	Input and Output
Address Base	Pixel Buffer: 0x08000000, Character Buffer: 0x09000000
Address Map	pixel base+{y[7:0],x[8:0],1'b0} corresponds to pixel (x,y) character base+{y[5:0],x[6:0]} corresponds to character (x,y)
Initialization	None needed
Interrupts	None
Hardware Setup	Connect the VGA plug of your monitor to the VGA port of the DE1-SoC
Reference	ADV7123 Video DAC datasheet

Notes

Pixel Colour - The colour of a pixel is a combination of three primary colours: red, green and blue. By varying the intensity of each primary colour, any other colour can be created. A 16-bit halfword is used to represent the colour of a pixel. The five most-significant and least-significant bits in this halfword represent the intensity of the red and blue components, respectively, while the remaining six bits represent the intensity of the green colour component, as shown in the table below.

Red 15...11	Green 10...5	Blue 4...0

For example, a red colour would be represented by a value 0xF800, a purple colour by a value 0xF81F, white by 0xFFFF, and gray by 0x8410.

Pixel Address - The colour of each pixel in an image is stored at a corresponding address in the memory. The address of a pixel is a combination of a base address and an (x, y) offset. In the DE1-SoC Computer, the base address is 0x08000000. The (x, y) offset is computed by concatenating the 9-bit x coordinate starting at the 1st bit and the 8-bit y coordinate starting at the 10th bit, as shown below.

Pixel Address

Bits	31:18	17:10	9:1	0
Function	00001000000000	y[7:0]	x[8:0]	0

To determine the location of each pixel in memory, we add the (x, y) offset to the base address. A formula offset = 2*x + 1024*y, can be used to calculate the offset. Using this scheme, the pixel at location (0, 0) has the

address 0x08000000, the pixel at (1, 0) has the address (base + 0x00000002) = 0x08000002, the pixel at (0, 1) has the address (base + 0x00000400) = 0x08000400, and the pixel at location (319, 239) has the address (base + 0x0003BE7E) = 0x0803BE7E.

Drawing Shapes - The pixel buffer can be used to draw only single pixels at a time, thus to draw lines or shapes you need to manually draw each pixel of the shape.

Characters - Besides drawing each pixel individually, there is also a character buffer allowing you to easily place text on the screen. This operates in conjunction with with pixel buffer, so you can draw shapes and images and then place text over image, or vice versa. Similar as for the pixel buffer, the entire screen is represented as a rectangular grid of 80 columns by 60 rows of characters. The character coordinate (x,y) = (0, 0) represents a character at the top-left corner of the screen, and the coordinate (x,y) = (79, 59) represents a character at the bottom-right corner of the screen. In memory, characters are represented with their ASCII codes. Each character occupies one byte of memory.

Character Address - Characters are represented in memory with their ASCII codes. Each character occupies one byte of memory. The address of a character is a combination of a base address and an (x, y) offset. In the DE1-SoC Computer, the base character address is 0x09000000. The (x, y) offset is computed by concatenating the 7-bit x coordinate and the 6-bit y coordinate, as shown below.

Character Address

Bits	31:13	12:7	6:0
Function	00001001000000000000	y[5:0]	x[6:0]

To determine the location of each character in memory, we add the (x, y) offset to the base address. A formula offset = x + 128*y, can be used to calculate the offset. Using this scheme, the character at location (0, 0) has the address 0x09000000, the character at (1, 0) has the address (base + 0x00000001) = 0x09000001, the character at (0, 1) has the address (base + 0x00000080) = 0x09000080, and the pixel at location (79, 59) has the address (base + 0x00001DCF) = 0x09001DCF.

Assembly Example: Draw white dot at pixel (4,1) and character 'A' at character (4,1)

```
.equ ADDR_VGA, 0x08000000
.equ ADDR_CHAR, 0x09000000

  movia r2,ADDR_VGA
  movia r3, ADDR_CHAR
  movui r4,0xffff  /* White pixel */
  movi  r5, 0x41   /* ASCII for 'A' */
  sthio r4,1032(r2) /* pixel (4,1) is x*2 + y*1024 so (8 + 1024 = 1032) */
  stbio r5,132(r3) /* character (4,1) is x + y*128 so (4 + 128 = 132) */
```

C Example: Program to write two lines and some text on the screen

```
/* set a single pixel on the screen at x,y
 * x in [0,319], y in [0,239], and colour in [0,65535]
 */
void write_pixel(int x, int y, short colour) {
  volatile short *vga_addr=(volatile short*)(0x08000000 + (y<<10) + (x<<1));
  *vga_addr=colour;
}

/* use write_pixel to set entire screen to black (does not clear the character buffer) */
void clear_screen() {
  int x, y;
  for (x = 0; x < 320; x++) {
    for (y = 0; y < 240; y++) {
```

```
          write_pixel(x,y,0);
        }
    }
}

/* write a single character to the character buffer at x,y
 * x in [0,79], y in [0,59]
 */
void write_char(int x, int y, char c) {
  // VGA character buffer
  volatile char * character_buffer = (char *) (0x09000000 + (y<<7) + x);
  *character_buffer = c;
}

int main () {

   clear_screen();
   int x;

   for (x=0;x<320;x++)
   {
      // Draw a straight line in red across the screen centre
      write_pixel(x, 59, 0xf800);
      // Draw a "diagonal" line in green
      if (x<240)
         write_pixel(x, x, 0x07e0);
   }

   // Write Hello, world!
   char* hw = "Hello, world!";
   x = 15;
   while (*hw) {
     write_char(x, 10, *hw);
         x++;
         hw++;
   }
   return 0;
}
```