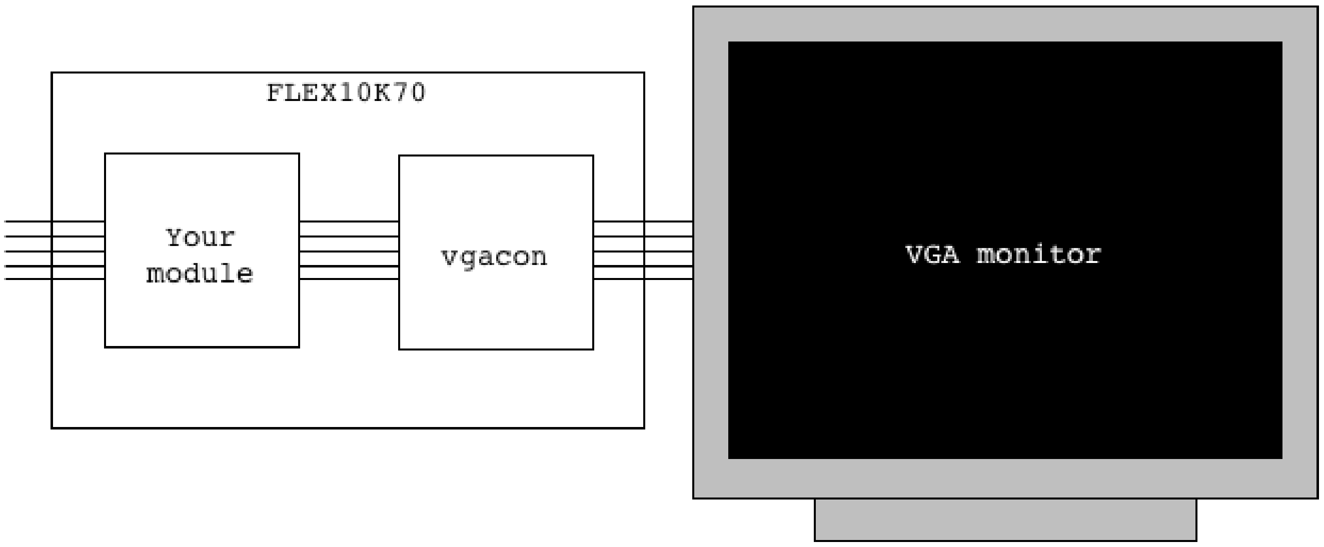
A Simple VGA Controller for the Altera UP2 Board

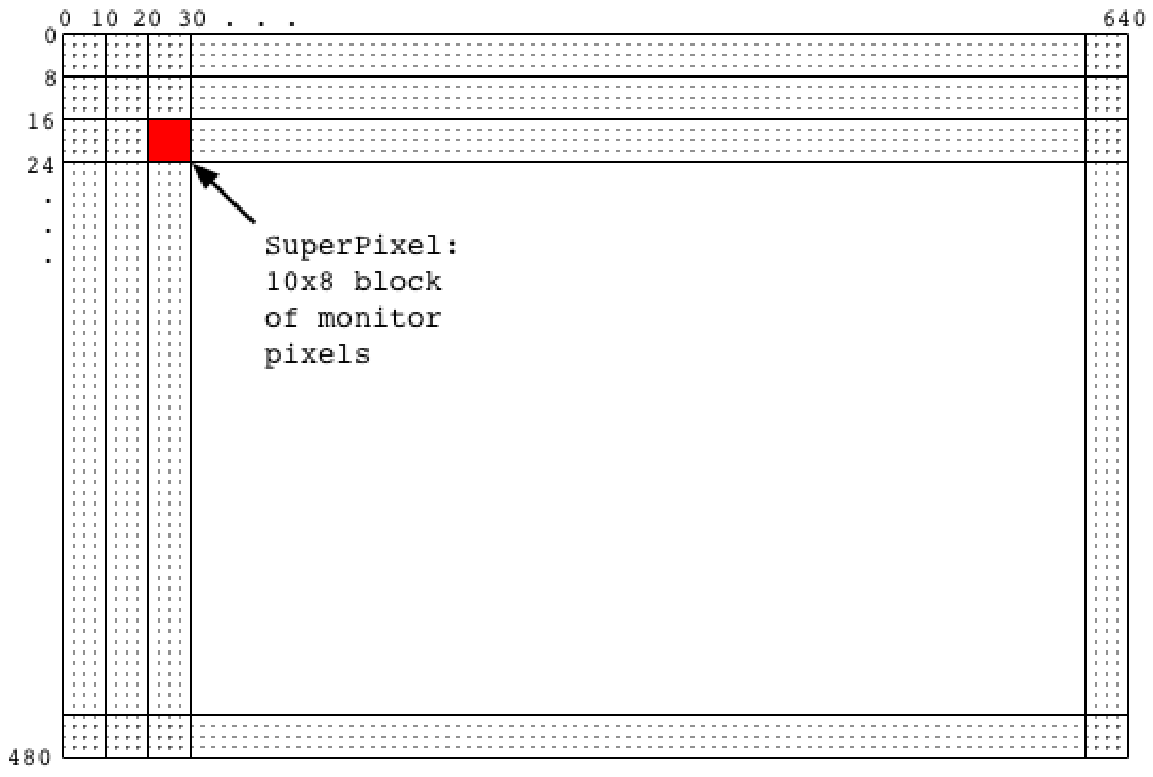
Overview

This page contains basic information describing the use of the VGA output port provided on the Altera UP2 Educational board. VHDL source code for a simple VGA controller (**VgaCon**) is provided. The purpose of VgaCon is to isolate the details of VGA signal generation from all the other modules in a design. VgaCon allows the pixel information to be written into its internal video memory using a very simple interface, while it is alone responsible for generating the required signals for displaying the pixel information on a VGA monitor. Thus, for modules interfacing with VgaCon, the process of drawing on the screen consists of a request to colour a point located at any valid pixel location.

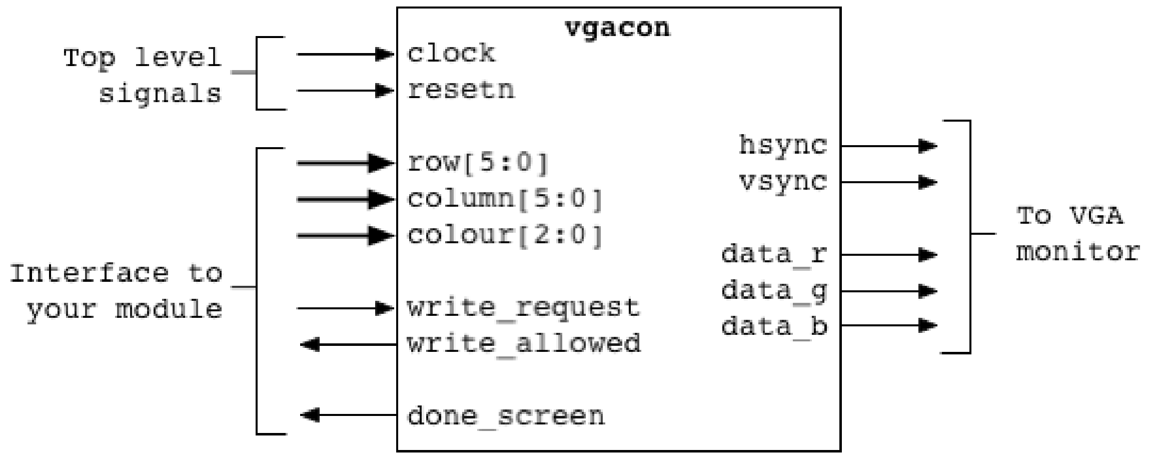


SuperPixels

VgaCon contains a memory block for storing the colour of each pixel to be drawn on the screen. However, the FLEX10K70 does not have enough on-chip memory to save a value for each monitor pixel. A simple solution to this problem is to map a large block of monitor pixels to a single memory location. In the monitor diagram below, a 10x8 pixel block (called a SuperPixel) is mapped into a single location in memory. This scheme produces a virtual resolution of **64x60** SuperPixels. VgaCon actually stores a grid of 64x64 SuperPixels in its memory. However the last four rows are not displayed.

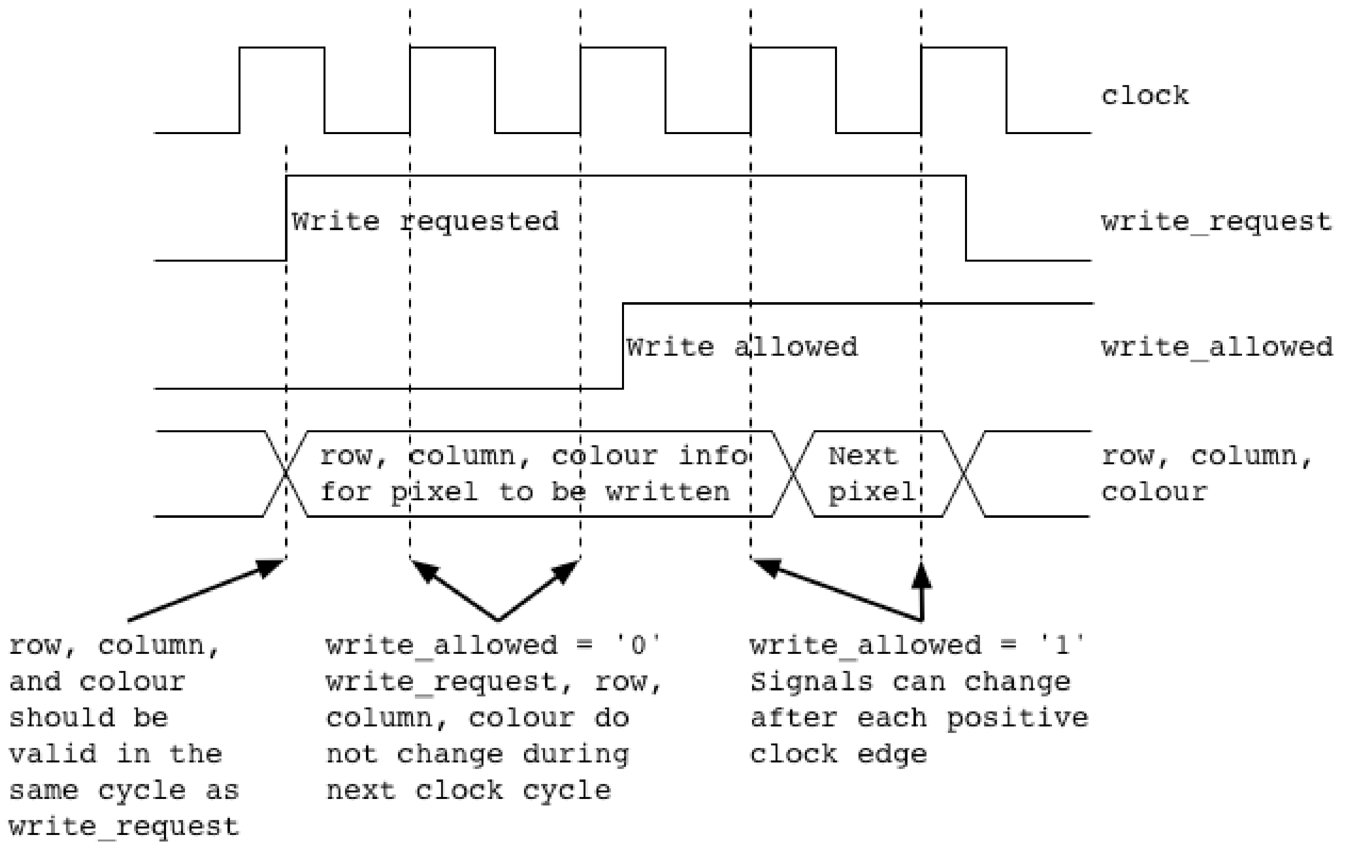


VgaCon Interface



The row, column, and colour signals define the coordinates and colour of the SuperPixel that you wish to be placed on the screen. Set the write_request signal to '1' to write the SuperPixel into the VGA controller memory. However, the controller is not always able to write the pixel immediately. It indicates this with the write_allowed signal. You must wait until the write_allowed signal is '1' for the write to complete. When write_allowed changes to '1' or if it is already at '1', the SuperPixel is written on the next positive clock edge. At that point, you can return write_request to '0'. Alternatively, you can keep write_request at '1' to write a new SuperPixel during the next clock cycle. You can write a new pixel every clock cycle until write_allowed is '0'.

Write SuperPixel Timing


clock
Write requested
write_request
Write allowed
write_allowed
row, column, colour info for pixel to be written
Next pixel
row, column, colour
row, column, and colour should be valid in the same cycle as write_request
write_allowed = '0' write_request, row, column, colour do not change during next clock cycle
write_allowed = '1' Signals can change after each positive clock edge

NOTE: With a 25 MHz clock, it will take between 25 and 30 microseconds until write_allowed is '1' for the first time. When write_allowed returns to '0' it will take a similar amount of time for it to be '1' again. So when simulating with a 25 MHz clock, be sure to set the simulation end time to at least **30 microseconds** to see write_allowed change.

Port/ Param	Description
clock	This port must be connected to the clock pin in the top-level design. The VGA controller requires that the clock pin is connected to the onboard clock on the UP2 board for correct functionality. This can be accomplished by assigning the clock pin to **pin 91** on the FLEX10K70.
resetn	The resetn port should connect to an active low signal that forces the controller back to its initial state. Typically this port should be connected to a resetn pin in the top-level design.
row[5:0]	These ports should each be connected to a 6-bit signal. The signals define the row and column coordinates of the SuperPixel that you wish to be placed on the screen. These coordinates are automatically transformed into an address for the memory in the controller. The column number should be between 0 and 63, while the row number should be between 0 and 59 for the SuperPixel to be displayed on the monitor.
column[5:0]	
	This port should be connected to a 3-bit signal which defines the colour of the SuperPixel to be placed on the screen. Eight colours can be produced by the Altera UP2 board. However, using all three bits per SuperPixel consumes 66% of the dedicated memory blocks in the FLEX10K70. If

colour[2:0]	your project uses a lot of memory then you should use the **2-colour version** so that less resources are consumed. The 2-colour version uses these colours:
	colour = "000" — **Black**
	colour = "001" — **Bright Green**

write_request	Set this signal to '1' to write the SuperPixel defined by (row[5:0], column[5:0], colour[2:0]) into the controller memory. Keep write_request set to '1' until write_allowed is '1'. At that point, the SuperPixel is written on the next positive clock edge. Then you can return write_request to '0'. Alternatively, you can keep write_request at '1' to write a new SuperPixel during the next clock cycle. You can write a new pixel every clock cycle until write_allowed is '0'.
write_allowed	This signal indicates that writing pixels is currently allowed. If write_request is set to '1', a SuperPixel will be written on the next positive clock edge. You can write a new pixel every clock cycle for as long as write_allowed is '1'. When write_allowed is '0' the controller is not accepting write requests and you must wait until write_allowed is '1' for your write to complete.
done_screen	This signal indicates that the entire screen has been drawn. The controller asserts this signal high for one clock cycle every **16.6ms**. This signal is useful for timing and delay purposes. The demo program below uses this signal to flash a diagonal line on the screen approximately every half-second.
hsync vsync	Produces the horizontal and vertical sync signals for the VGA monitor. These ports must connect to top-level output pins that must be assigned to **pins 240 and 239** respectively on the FLEX10K70.
data_r data_g data_b	Produces the red, green, and blue signals for the VGA monitor corresponding to the contents of the VGA controller RAM. Again, these ports must connect to top-level output pins which are assigned to **pins 236, 237, and 238** respectively on the FLEX10K70.
ramfile	This is a parameter which should be assigned to a MIF file that contains the initial contents of the controller RAM memory. It allows one to easily create an initial picture that is displayed on the screen. A utility is provided below which can convert standard graphic files to MIF format.

VGA Demo

Instructions:

1. Save the 2-colour demo or the 8-colour demo to your home directory.
2. Double-click and downloaded file to create a directory that contains the files listed below.

3. Open the project file (vga_demo.qpf) in Quartus.
4. Compile the design.
5. Download the design to the FLEX10K70 on the UP2 board. The programming switch on the board must be in the **up** position. The demo displays a mailbox in the background. Push the button labelled FLEX_PB2 to turn on the flashing diagonal line.

File	Description
vga_demo.qpf	The Quartus II project file for the VGA demo.
vga_demo.qsf	Contains the PIN and DEVICE assignments for the VGA demo.
vgacon.vhd	Simple VGA controller written in VHDL.
vga_demo.v	A Verilog file which demonstrates the use of VgaCon.
vga_demo.mif	Initial RAM contents for the VGA controller. This contains the picture of the mailbox that is displayed initially by the demo.
pbmtomif.c	Converts a graphics file in PBM (ASCII) format to a MIF file that can be loaded into the VGA controller RAM memory. This program was used to convert a PBM graphic of the mailbox into the demo MIF file. It is particularly useful if you wish to create an initial background for applications such as tic-tac-toe game. You can draw the background with any graphic editor, shrink it to a 64x64 resolution and save it in PBM format. **xv** is a graphical UNIX program that will allow one to convert common graphic formats such as GIF, JPEG, and BMP to PBM. Instructions for compiling and running PBMtoMIF are in this source file.

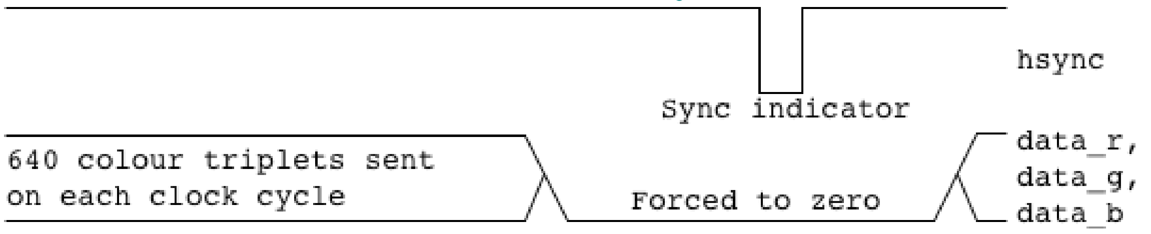
Creating Your Own Project

You need to download projects that involve VgaCon into the FLEX10K70 on the UP2 board. **Pin locations** for this device can be found in the Altera UP2 User Guide.

How VGA Monitors Work

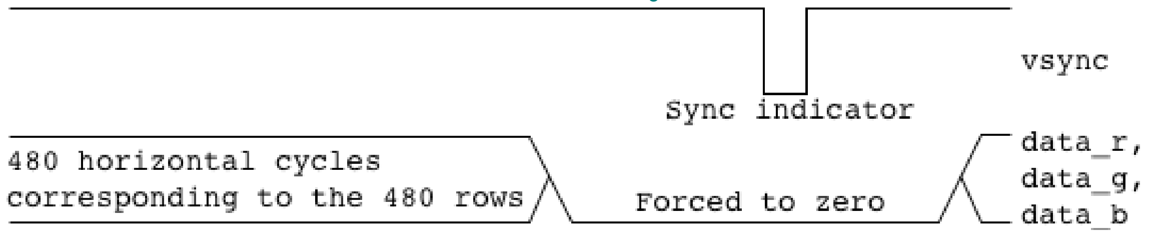
The VGA monitor can be thought of as a grid of pixels (picture elements which can be individually set to a specific colour). It contains 480 rows of 640 horizontal pixels. Most monitors, including VGA, use a serial scheme to set the colour of each pixel. This means that the VGA controller sends the colour information for each pixel one at a time, rather than being able to set all of the colours at once in a parallel scheme. This colour information for each pixel is provided by a RGB (Red, Green, Blue) triplet. Three analog signals are used to represent relative amounts of red, green, and blue that compose the colour. However, the UP2 board produces digital signals. Thus, the controller is only able to provide full intensity or turn off for each of the RGB components. The VGA monitor does not save any of the pixels written to it so the pixels must be continuously written to the monitor for the image to remain stable. The protocol for the transmission is shown below.

Horizontal Cycle



The horizontal cycle is part of the VGA standard and defines a method for setting the colours of all the pixels in a particular row. The colour values for the 640 pixels are sent out on each of the first 640 clock cyles. The colour values are then forced to zero (black), while the hsync signal is asserted low to tell the monitor to move to the next horizontal row of pixels. The exact timing between these signals is defined in the VHDL source code for the VGA controller. The VGA controller only allows your requests to write pixels while the colour signals are forced to zero. At all other times, it is reading from memory and it cannot write to the memory at the same time.

Vertical Cycle



The vertical cycle allows the monitor to synchronize which rows are being written by the horizontal cycles. 480 horizontal cycles are produced during the first stage of the vertical cycle. In the next stage, the colour values are again forced to zero (black), while the vsync signal is asserted. The vsync signal tells the monitor to go back to the (0,0) pixel. The data that follows starts with the first horizontal row of pixels.

ECE 241 HOME PAGE

Created by Deshanand Singh
Modified by Aaron Egier
Last Modified Nov 26, 2004.