VHDL serial to VGA Text Controller (SER2VGA-01)

FPGA IP Core for UART to text VGA/SVGA converter.

Revision History:

Revision	**Date**	**By**	**Comments**
1.05	**Dec. 16, 2007**	**GL**	**Format for PDF**

Table of Content:

1 Introduction

The majority of electronic products in the market today do not use a high resolution graphic display and are typically designed around a small 8 or 16-bit microcontroller. As the prices for the LCD and CRT displays are coming down a whole area of new applications are now available to take advantage of these larger and newer display technologies.

The hurdle for any company is the migration to a more powerful microcontroller with built-in VGA controller when the only purpose of the microcontroller is adding graphical interface to an existing system. This typically requires a major investment.

The design described in this document allows any microcontroller design to be updated with very little effort to allow the use of any off-the shelf low cost CRT or LCD display.
The only requirement for the microcontroller is to have an available UART interface which is required to drive the serial to VGA adapter.

This document describes the FPGA implementation for a serial to VGA/SVGA interface.
Based on the low cost Xilinx Spartan 3E family, this design provides an efficient yet powerful solution for adding graphical text capabilities to any microcontroller design.

The design supports the following display resolutions:

640x480 pixels /60 Hz refresh rate
800x600 pixels /60 Hz refresh rate
800x600 pixels /72 Hz refresh rate
1024x768 pixels/60 Hz refresh rate

Selecting between the four display resolutions is done using two FPGA select inputs or by using software control.

The design uses 5 Spartan 3E Block RAMs to implement the character maps and the video buffer.

Four character maps are available to the user:

1. The 8x16 pixels character map is the default option and it is implemented as a Character ROM using one FPGA Block RAM. A maximum number of 128 characters on every line can be displayed when the 1024x768 resolution is used. Characters corresponding to the complete ASCII table are provided as part of the VHDL design.
2. Three other character sizes are provided: 8x32, 16x16 and 16x32. Selecting the character size is done using a software command. In order to minimize the memory usage, the same character map is being used in all cases.

In order to minimize design costs (memory required inside the FPGA), only one bit per color was utilized in this design. As a result a maximum number of 8 colors can be used for background and foreground (text). These colors are user configurable. Changing the text or the background color is done by writing the color value in one of the internal FPGA registers.

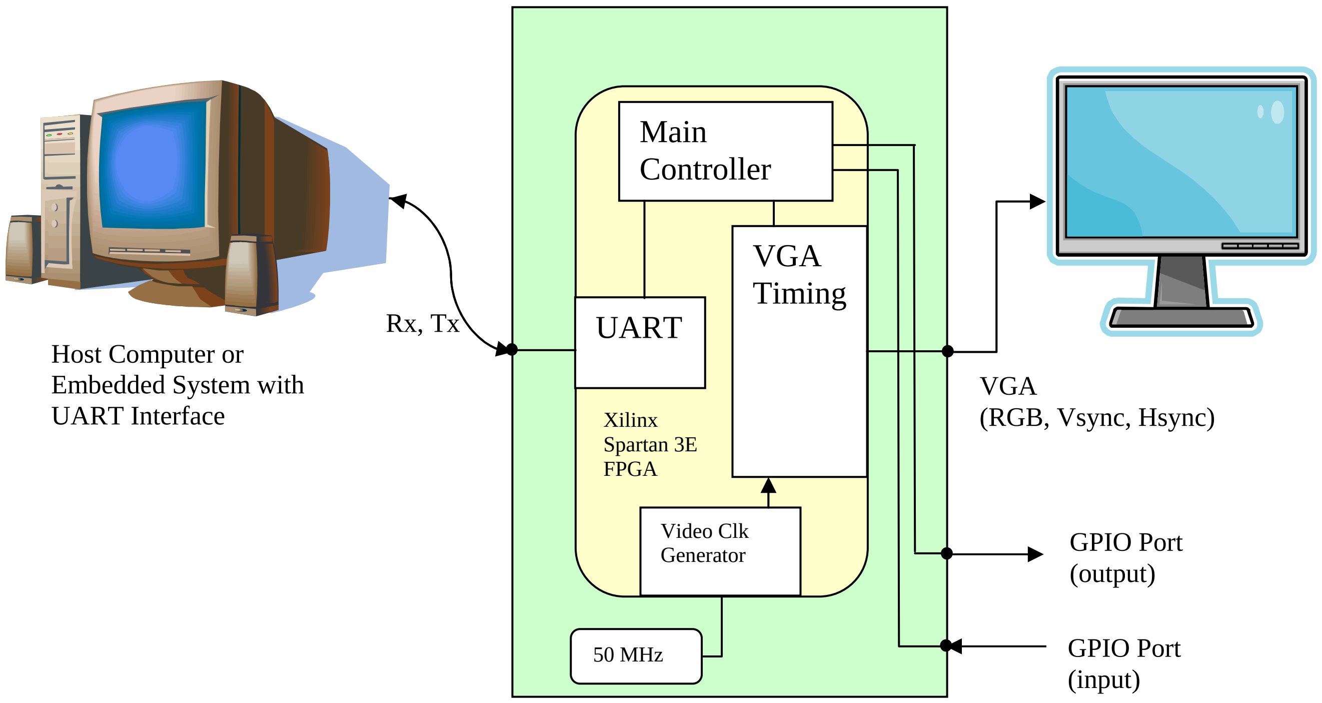


Figure 1: Ser2VGA Block Diagram

1.1. Core Derivatives

A smaller memory footprint version of this core is available (SSIP-UART2VGA-02). It only supports the 640x480/ 60 Hz displays and uses 3 Block RAMs for the character map and the video buffer. All the other functions and controls are identical to the main version of the core.

A more complex version of this core that requires the use of external memory for storing the video data is available. For details please check the website for SSIP-SER2VGA-G core.

2. Software Interface

The VGA interface is controlled using the following commands:

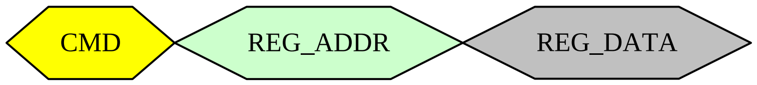
Command	Code (Hex)	Description
Write Register	0x81	Writes one byte to a register
Read Register	0x82	Reads the value in the specified register
Write Video Data	0x83	Writes an ASCII character to the screen
Write Video Data Inc	0x84	Write video data with relative buffer address pointer
Clear Screen	0x85	Clears the screen
Backspace	0x86	Clears the last written character

The structure for each command is described in details below:

2.1. Write Register Command

The Write Register command is used to write the Picoblaze registers with VGA configuration values.

The command consists of 3 successive bytes:



1. Command (0x81)
2. Register Address –as described in register memory map (see below)
3. Register value to be written

2.1.1. Register Memory Map (Write mode)

Address	Register name	Description
0x00	VGA_CTRL	Used to configure the display resolution
0x02	COLOR_REG	Changes the text and background color
0x03	GPIO_OUT	Changes the status of the output port
0x04	VIDEO_ADDRL	Lower byte of the Video Buffer Address
0x05	VIDEO_ADDRH	Higher byte of the 16-bit Video Buffer Address
0x07	T_SIZE_REG	Changes the size of the text displayed on the entire screen
0x08	V_BACK_PORCH	Allows the vertical centering of the image
0x09	H_BACK_PORCH	Allows the horizontal centering of the image

Each of the registers is described in detail below:

2.1.2. VGA_CTRL register (0x00)

Register bit	7	6	5	4	3	2	1	0
Function	C_RST	TEST	RES_S(1)	RES_S(0)	CR_DET	N/A	N/A	S_RST

The default power up value for this register is 0x00

C_RST: When this bit is set the video buffer address pointer resets. It can be used to bring the cursor to the origin (0,0).

When CR_DET is used, upon receiving a write video command the video buffer address pointer advances to the beginning of the next line when the CR (0x0D) is detected in the video string.
Note: in this version of the core, the CR_DET functionality is not implemented.

The TEST bit is used to put the controller in test mode. 8 color vertical bars are generated automatically. They can be used for display adjustment.

The RES_S pins control the display resolution as follows:

00 -640x480 pixels /60 Hz refresh rate
01 -800x600 pixels /60 Hz refresh rate
10 -800x600 pixels /72 Hz refresh rate
11 -1024x768 pixels/60 Hz refresh rate

S_RST: when set all the microcontroller registers are reset (loaded with the power up value). The video buffer is also cleared.

2.1.2 COLOR_REG (0x03)

Bit	7	6	5	4	3	2	1	0
Function	N/A	BG(3)	BG(1)	BG(0)	N/A	TXT(2)	TXT(1)	TXT(0)

TXT(2:0) –Text Color

BG(2:0) –Background Color

The default value for the register COLOR_REG is 0x0F (white text on back background)

2.1.3. GPIO_OUT register (0x02)

Register bit	7	6	5	4	3	2	1	0
Function	OUT(7)	OUT(6)	OUT(5)	OUT(4)	OUT(3)	OUT(2)	OUT(1)	OUT(0)

The default value for GPIO_OUT register is 0x00 (all outputs are 0 logic)

2.1.4 T_SIZE_REG (0x07)

Bit	7	6	5	4	3	2	1	0
Function	N/A	N/A	N/A	CURSOR_ON	N/A	N/A	CHAR_SZ(1)	CHAR_SZ(0)

CHAR_SZ –Defines the character size as follows:

CHAR_SZ	Character Size
00	8x16
01	16x16
10	8x32
11	16x32

CURSOR_ON –When 0 the cursor is active (power up setting). When this option is active, the underscore character (ASCII "_") is displayed alternately with a blank character on the screen every 0.65 seconds.

The default value for the T_SIZE_REG is 0x00 (cursor is active and character size is 8x16).

2.1.5. V_BACK_PORCH (0x08)

Bit	7	6	5	4	3	2	1	0
Function	V_BACK_PORCH(7)							V_BACK_PORCH (0)

The register is loaded at power up with a default value representing the number of inactive video lines between the rising edge of the V_SYNC and the beginning of the active video area on the screen. The default value can be modified using the Write Register command.

2.1.6. H_BACK_PORCH (0x09)

Bit	7	6	5	4	3	2	1	0
Function	H_BACK_PORCH(7)							H_BACK_PORCH (0)

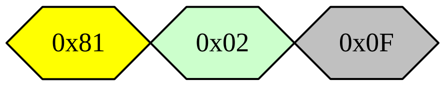
The register is loaded at power up with a default value representing the number of inactive video pixels between the rising edge of the H_SYNC and the beginning of the active video on every line. The default value can be modified using the Write Register command.

2.1.7. Using the Write Register Command

When the controller receives the Write Register command, after writing the register with the appropriate value an ACK byte (0x06) is sent back to the host. If the register address is out of range, or the command is not recognized a NAK character (0x15) is sent back to the host.

The Write Register command can be utilized to change the VGA Controller settings: Text Color, Screen Size, Character size, Cursor On/Off, etc. Please consult the register memory map for all the control signals.

Example:



Write 0x0F to the output register (address 0x02)

2.2. Read Register Command

The read command is used to read the content of one of VGA controller registers.

2.2.1. Register Memory Map (Read Mode)

Address	Register name	Description
0x00	UART_STATUS	Used to read the status of the UART
0x02	COLOR_REG	Reads the current settings for the background color
0x03	GPIO_IN	Reads the external input pins
0x04	VIDEO_ADDRL	Lower byte of the Video Buffer Address
0x05	VIDEO_ADDRH	Higher byte of the 16-bit Video Buffer Address
0x07	T_SIZE_REG	Changes the size of the text displayed on the entire screen
0x08	V_BACK_PORCH	Allows the vertical centering of the image
0x09	H_BACK_PORCH	Allows horizontal centering of the image

2.2.2 VGA_CTRL_REG (0x00)

Bit	7	6	5	4	3	2	1	0
Function	C_RST	TEST	RES_S(1)	RES_S(0)	CR_DET	RX_FF	TX_FF	UART_D

Bits 3 to 7 have the same function as defined in 2.1.2

RX_FF: RX_FIFO_FULL – 0 when less than 16 bytes are stored into the Rx FIFO

TX_FF: TX__FIFO_FULL -0 when less than 16 bytes are stored into the Tx FIFO

UART_D: UART Data Present -0 when no data is in the UART Rx FIFO buffer

Please note that for simplicity the memory mapped registers have different meaning in read and write mode.

2.2.3 COLOR_REG

Bit	7	6	5	4	3	2	1	0
Function		BG(3)	BG(1)	BG(0)		TXT(2)	TXT(1)	TXT(0)

TXT(2:0) –Text Color

BG(2:0) –Background Color

2.2.4 T_SIZE_REG (0x07)

Bit	7	6	5	4	3	2	1	0
Function	N/A	N/A	N/A	CURSOR_ON	N/A	N/A	CHAR_SZ(1)	CHAR_SZ(0)

All the bits in this register are defined in 2.1.4.

2.2.5. V_BACK_PORCH (0x08)

Bit	7	6	5	4	3	2	1	0
Function	V_BACK_PORCH(7)							V_BACK_PORCH (0)

The register is loaded at power up with a default value representing the number of inactive video lines between the rising edge of the V_SYNC and the beginning of the active video area on the screen. The default value can be modified using the Write Register command.

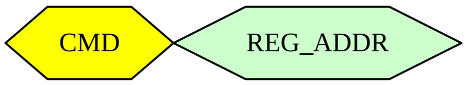
2.2.6. H_BACK_PORCH (0x09)

Bit	7	6	5	4	3	2	1	0
Function	H_BACK_PORCH(7)							H_BACK_PORCH (0)

The register is loaded at power up with a default value representing the number of inactive video pixels between the rising edge of the H_SYNC and the beginning of the active video on every line. The default value can be modified using the Write Register command.

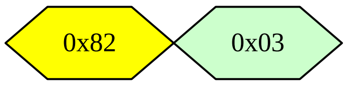
2.2.7. Command Usage:

The Read Register Command is typically used in conjunction with Write Register Command to allow the masking of control bits that are not to be affected.
For example, if the application has to modify the Cursor mode the following sequence has to be used:
Read Register T_SIZE_REG (0x07)
Mask bits CHAR_SZ
Write Register T_SIZE_REG(0x07) with the required CURSOR_ON setting
The command consists of two successive bytes:



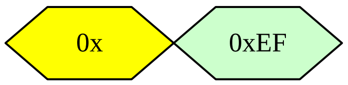
1. Command (0x82)
2. Register address (hex value)

Example:
Read the status of the GPIO input pins:



Upon receiving the command an ACK followed by the data byte will be sent back to the host. If the register address is out of range or the command is not recognized a NAK character is sent back to the host.

Example of the VGA_CTRL response to the example command described above:



2.3. Write Video Data Command

The Write Video Data command is used to place ASCII characters to a fixed position on the screen.
The address or location of each character can be calculated using the diagram below:

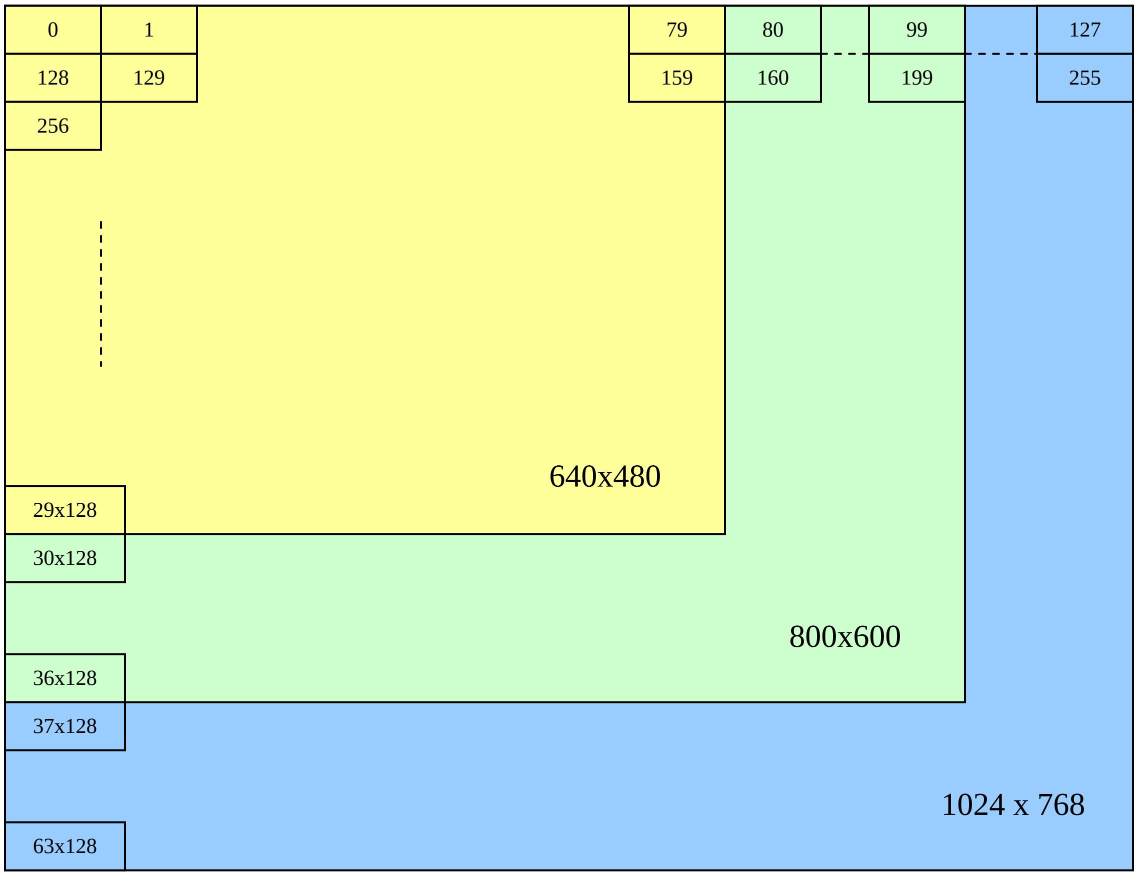


Figure 2: Screen Layout

The screen is partitioned into a maximum of 128x 64 characters (128 columns and 64 rows), when the 8x16 character size is used along with the 1024x768 resolution.
Three other text sizes are available in this design: 8x32, 16x16, 16x 32.

When using increased text size, fewer characters will fit in the active area of the screen. The position of the data in the video buffer will not change as a result of changing the character size.
Example: If the character size changes to 16x16 only 40 characters can be displayed on every line. The first character in the second row will still have address 128 (0x0080).

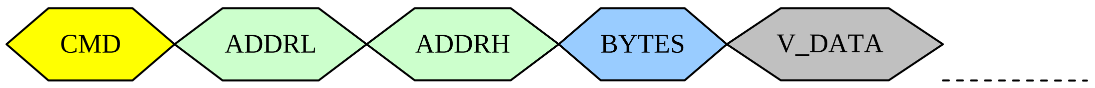
The following formula describes the address that has to be written in the video buffer when a string of characters has to be displayed starting at (Row#, Column#):

Video Buffer Address = Row# x128 + Column#
Where Row# is the row where the first character is situated on the screen and Column# is the column of the character.

A video buffer is used to store all the video characters that have to be displayed on the screen during a frame. The video data is updated on the screen during the horizontal blanking (a maximum of one character per video line).

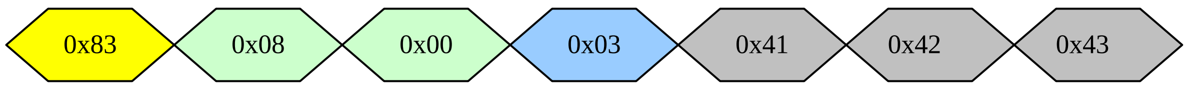
By using the Write Video Data command, the specified locations in the video buffer will be updated with the appropriate values. The ADDRL and ADDRH define the location in the video buffer where data will be written. Each Video Data byte received during one command will be written to the subsequent location (next address), starting at the address defined as part of the command.

The command has the following structure:



1. Command (0x83)
2. Lower byte Video Buffer Address (hex value)
3. Upper byte Video Buffer Address (hex value)
4. Number of video characters to be written (hex value)
5. Data bytes –the number of bytes transmitted has to match the number of characters in the command (4th field)

Example:

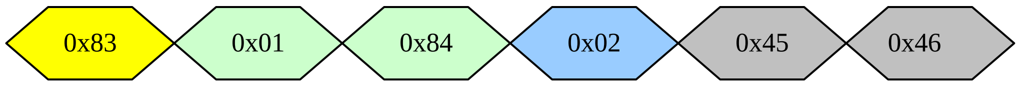


This command writes at address 0x08 (first row, 8th column) three characters: A, B and C

Example 2:

Write 2 characters, E and F on the third row starting with 4th column of the screen.
The address in the video buffer is calculated according to the following formula:

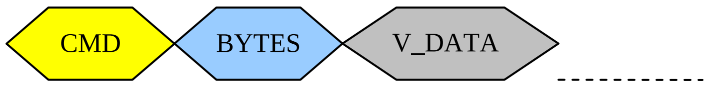
Row# x128 + Column# = 3x128 + 4 = 388 = 0x184 (hex)



2.4. Write Video Data Inc Command

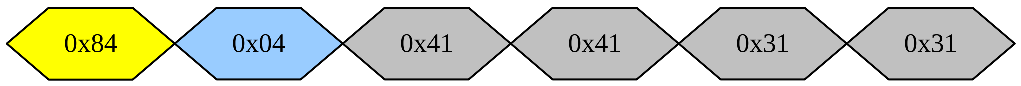
When using the Write Video Data Inc command, the start of the address pointer for the video buffer is not modified. The pointer will increment after writing each of the bytes received during the command.

The command has the following structure:



1. Command (0x84)
2. Number of video characters to be written (hex value)
3. Data bytes –the number of bytes transmitted has to match the number of characters in the command (hex value)

Example:
Write the following string of four characters "AA11" from the current cursor position (video buffer address pointer):



After the command is executed the character ACK is returned to the host.

2.6. Clear Screen Command

When receiving this command the VGA_CTRL core deletes all the video data in the buffer and the video buffer address pointer is reset to the origin (row 0, column 0).
To achieve this, upon receiving the command the clear_screen flag is set. When a new frame starts the data at every address in the video buffer is reset (0x00). It takes exactly a frame for the entire buffer to clear (15 ms).

The command has the following structure:

1. Command (0x85)

After executing the command the VGA module sends an ACK character to the host.

2.7 Backspace Command

When receiving this command the video buffer address pointer is decremented by one position and the last character in the video buffer is deleted.

The command has the following structure:

1. Command (0x86)

After executing the command the VGA module sends an ACK character to the host.

3. FPGA Pinout

The following FPGA pins are used in this design:

Pin Name	Function	Comments
clk_in	50 MHz clock input	Used to clock the DCMs and the serial interface
rx	UART Rx	Input for the serial interface
tx	UART Tx	Output for the serial interface
h_sync_mon	Horizontal Sync	VGA output to the LCD/CRT monitor
v_sync_mon	Vertical Sync	VGA output to the LCD/CRT monitor
r_out	Red Output	VGA output to the LCD/CRT monitor
g_out	Green Output	VGA output to the LCD/CRT monitor
b_out	Blue Output	VGA output to the LCD/CRT monitor
rst	Hardware reset	Resets all the FPGA blocks
gpio_out	GPIO outputs	General Purpose Outputs (optional)
gpio_in	GPIO inputs	General Purpose Inputs (optional)

3. Implementation Results

The implementation results are provided below for a few versions of the core:

3.1. FPGA Resources Used

UART2VGA-01

Features: Multiple resolutions supported, 8KB video buffer, single color text and background.

For the implementation using Xilinx XC3S500E-FG320 as a target device uses the following FPGA resources:

Total Number Slice Registers:	331 out of	9,312	3%
Number of occupied Slices:	377 out of	4,656	8%
Total Number of 4 input LUTs:	624 out of	9,312	6%
Number of bonded IOBs:	27 out of	232	11%
Number of Block RAMs:	6 out of	20	30%
Number of GCLKs:	6 out of	24	25%
Number of DCMs:	2 out of	4	50%

UART2VGA-02

Features: Single resolution (640x480), smaller core footprint

For the implementation using Xilinx XC3S500E-FG320 as a target device uses the following FPGA resources:

TBD

3.2. Design Performance

The maximum clock frequency is over 100 MHz, providing enough design margin for most of the possible applications.

If the user application requires one DCM (Digital Clock Manager) for other purposes, the video clock generator block can be simplified at the expense of removing one or two possible VGA resolutions.

5. Demo Version

The demo version was adapted to run on the Xilinx Spartan 3E Starter Kit. The core has full functionality.
A hardware counter was added to the design. The counter has a timeout period of around 10- 15 minutes. When the counter expires a part of the design will stop operating.

Re-cycling the power or re-programming the FPGA will re-initialize the counter and the design will run for the same period of time.

For testing the demo version, the FPGA or the Xilinx Prom have to be programmed using any Xilinx download cables.
Use the .mcs file to program the prom or the .bit file to program the FPGA.

Using the Windows software utility comtest.exe provided, the user can send UART commands to test all the functions of the UART2VGA controller.

After starting the comtest.exe utility please select the COM port connected to the Spartan 3E Board and select 38400 baud from the drop down list.
Before sending the commands please unselect the “Automatic Framing” box.

If other software is used for the purpose of testing the core, the UART on the host or embedded system has to use the following settings:
-Baud Rate: 38400
-8-bit, no parity
-no flow control

6. Deliverables

Upon the full licensing of the core the customer will receive the following files:

- Xilinx ISE 9.1 project with all the VHDL files required for complete implementation
- Detailed documentation containing a block diagram and description of each of the blocks that is part of the FPGA design.
- Microcontroller assembly source file
- 60 days web support.

Disclaimers: