VT100 – Project Report

EDA385 – Computer Science – LTH

Mattias Jernberg, D06 (dt06mj4@student.lth.se)
Andreas Lord, D08 (dt08al1@student.lth.se)
Björn Uusitalo, D06 (tn06bu0@student.lth.se)
Supervisor: Flavius Gruian (Flavius.Gruian@cs.lth.se)

November 8, 2010

Abstract

This projects implements a VT100-like serial terminal using a Nexys-2 FPGA board, VGA display, and a PS/2 keyboard. It features the most common subset of the ANSI escape codes defined in the ECMA-48 standard, interrupt based input, serial port IO and a character-based video controller. The final prototype is capable of running most common terminal applications supporting the ECMA-48 standard, such as – but not limited to – top, less and vim.

Contents

1 Introduction

This project aims to produce a serial terminal supporting ANSI escape codes for formatting output and supporting a keyboard for output back to the host device. Support is available for colourizing output, cursor placement and inserting text at any point on the screen.

The VT100 was a video terminal that was made by Digital Equipment Corporation (DEC) in 1978. It connected to a host computer (read: mainframe) over a serial connection and allowed display of formatted text as well as user input. The original standard for controlling the output was specified in ANSI X3.64, which later became ECMA-48.

The ECMA-48 is a standardization of the escape codes (usually known as ANSI escape codes) which are interpreted by the terminal. The wide array of control codes and display modes, as well as its extensibility, has allowed it to stay relevant to this day and is enough to configure most embedded equipment with a text-based serial connection.

Today most terminals are served using terminal emulation software, freeing you from the hardware terminal and enabling use of any computer to interface to a serially connected device. VT100 emulation can be seen as a common factor in pretty much every such software.

2 System design

The general system design of our system is very similar to what is found in a description of a normal VT100:

- VGA output
- Keyboard input
- Serial connection
- Microcontroller

In our case however we use standard connections to all I/O instead of, like the VT100, provide them as part of the hardware. A PS/2 port provides an interface to the hardware and a HD15 VGA connector allows any VGA screen to be connected. The microcontroller used is the Xilinx MicroBlaze which ties together communication between the peripheral devices. Our final design is shown in figure 1. This is nearly identical with our proposal. The shared video memory has been defined as being outside the VGA, whereas the original integrated the memory inside the VGA controller. Secondly, the PmodAMP for audio beeps has been abandoned as the hardware was not available.

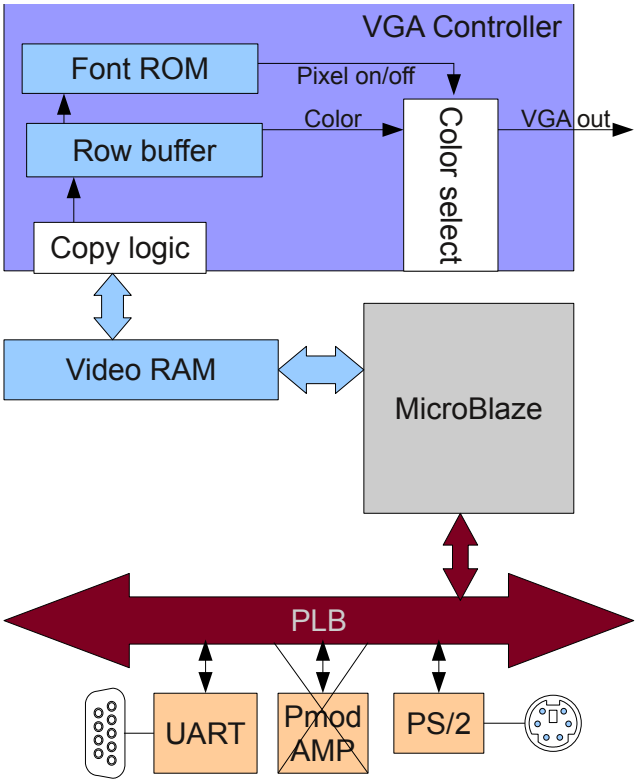


Figure 1: Architecture for the implementation

3 Protocols

As this is a system which is designed to interact with standardised peripherals it is necessary to follow a set of protocols detailing the interaction specifics of these peripherals.

3.1 ECMA-48

The ECMA-48 standard defines the specifics of terminal emulation for 7 and 8 bit long characters. In addition to normal characters it defines control sequences, functions, and strings that can be used to control the output to the terminal. In this document we'll be describing the 7 bit protocol since it's the one we're using.

The normal way to control the output is through Control Sequences. A control sequence has the following form:

CSI P...P I...I F

Where the Control Sequence Inducer (CSI) is the byte 0x5B, Parameter

bytes (P) are bytes between 0x30-0x3F, Intermediate bytes (I) are between 0x20 and 0x2F and Final Bytes (F) are between 0x40-0x7E.

The parameter bytes encode the parameters in string of parameters in string representation. 0x30-0x39 represents the numbers 0-9, 0x3A acts as a separator between integer and decimal parts, and 0x3B separates parameters. This maps well to the ASCII table where 0x30-0x39 maps to '0'-'9' and 0x3A, 0x3B maps to ':' and ';' respectively.

Control sequence provides a call to a control function which is specified by the intermediate byte and the final byte. The parameter bytes specify the arguments to the function.

It's also possible to control the formatting through Independent control functions and control strings. The independent control functions are mostly aimed at controlling the device written to or the connection to the host. Control strings are predefined commands passed through the terminal to the receiver without a predefined meaning in ECMA-48. Since neither is implemented in our project we will not describe them in detail.

3.2 PS/2

A keyboard connecting via the PS/2 connector communicates using the protocol devised by IBM for their PC AT computer. This protocol consists of 8 bit commands and so called scan codes which are transferred over a synchronous serial connection.

The connector to the keyboard consists of 4 terminals, *Vcc*, *Ground*, *Clock* and *Data*. The host connector contains pull up resistors on Clock and Data, keeping them normally high. The keyboard then pulls the signals to low to send data. The keyboard clocks bits by pulling the clock low and then releasing it making it return to a high state. First a start bit with value 0 is sent, followed by 8 data bits, 1 parity bit and a stop bit (a 1).

The host can also send data to the keyboard by forcing the clock to low for at least $60\mu s$. This tells the keyboard to start clocking, expect a start bit and then the start of data. After the sequence of data the keyboard acknowledges by pulling the Data line low for one clock cycle.

The host to keyboard sending has priority. Thus whenever the host forces the clock to low to keyboard must abort or inhibit its current transmissions to accept data from the host. To perform this safely the keyboard has a small buffer internally for storing key presses. This can be used to stop data from coming if the host cannot process any more at this time. By forcing the clock to low and when releasing it keeping Data high (thus not sending any start bit) the keyboard will abort the receiving of data and proceed to send any data in its buffer.

For an AT compatible keyboard each key has a defined scancode of at least 8 bits. The scan code is sent when the key is pressed as well as on regular intervals as long as the key is held down. This key repeat functionality,

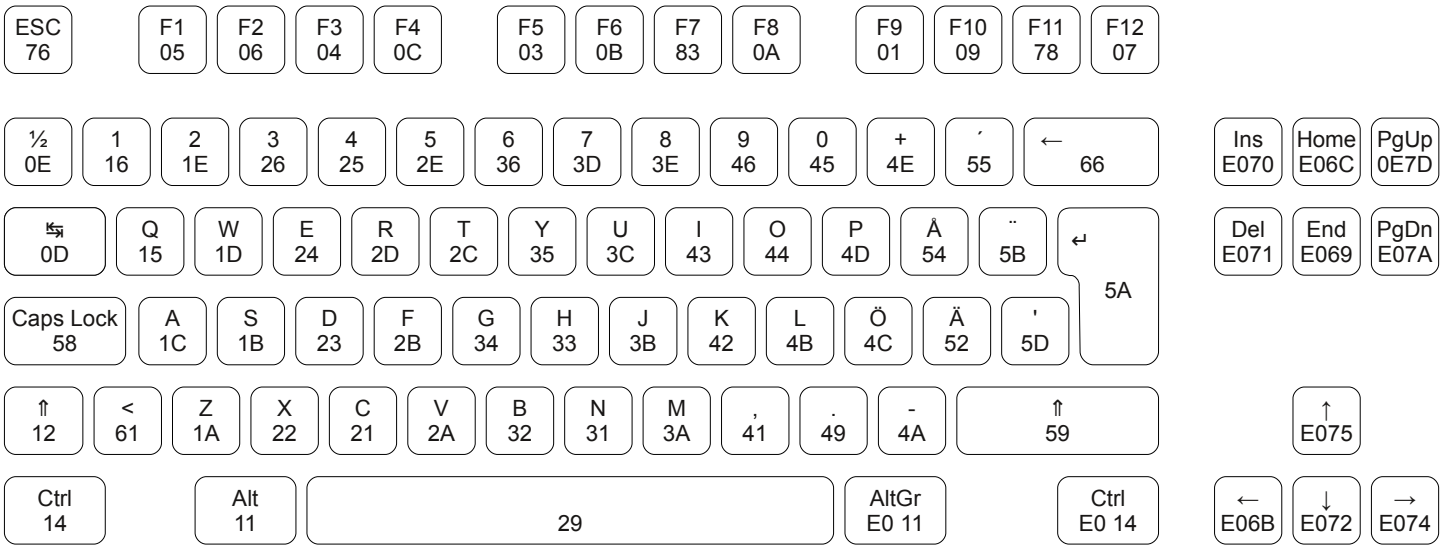


Figure 2: Scancodes on a swedish keyboard layout

called typematic, is on by default in keyboards but can be configured by sending commands to the keyboard. When a key is released the keyboard sends the byte F0 followed by the scan code. The scancodes and the corresponding keys on a keyboard with Swedish layout can be seen in figure 2.

3.3 VGA

The VGA output signal is very simple; it consists of three analogue channels, one for each colour red, green and blue. In addition to these there are two signals named vertical sync (*Vsync*) and horizontal sync (*Hsync*). These are used to tell the monitor when a new line or page is to be started.

The signal is based on how CRT monitors work. A cathode ray is guided using an electrical field which makes it sweep across the screen. As the horizontal voltage increases in this field the ray sweeps across the line outputting the colour which is currently put on the inputs. Hsync serves to discharge to component supplying voltage to the electrical field. This is the amount of time that Hsync is active. On a CRT there is also a part of the screen that is hidden by the casing. To keep image stability there are two areas, one at the beginning and one at the end named front and back porch respectively which do not contain data.

All this also applies to vertical order as well, with Vsync instead of Hsync and a different timing. Modern LCDs instead incorporate a chip

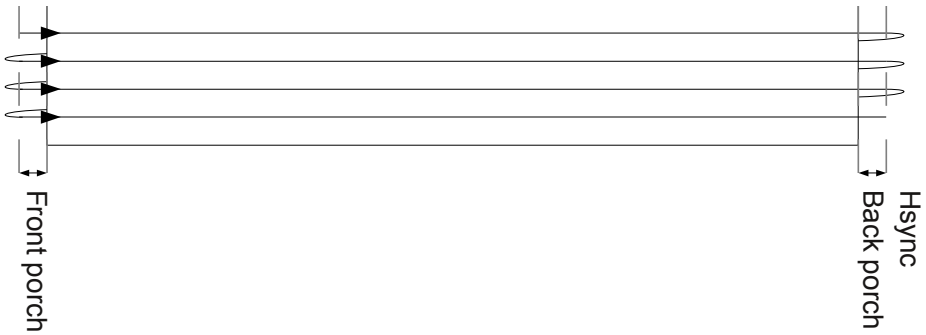


Figure 3: VGA scan to draw lines

which detects this timing and syncs with the incoming signal to convert each pixel to a pixel on the LCD panel.

3.4 RS232

For serial communication the RS232 protocol is used. It is an asynchronous protocol and needs that both sides have previously agreed on a set data rate to properly communicate (since clock speeds are otherwise out of sync). The actual serial transfer protocol is not relevant for the implementation and is handled by a readily available chip. Thus it is not described further here.

4 Hardware

4.1 VGA

The VGA controller is the only core that was custom created. It implements a $640x480$ VGA signal at 60 Hz using a pre-made VHDL module available from Digilent[4]. To generate characters from text a ROM is used to store a font with characters $8x16$ pixels wide. Each character is represented in this ROM using 16 bytes, one for each row of the character. Each bit then represents the column. The output is done by two components, a pixel generator and a copying module lifting data from the shared video memory. Simplified diagrams of these are available in appendix D.

4.1.1 Output generation

Since the counters operate on pixel positions it is necessary to translate this into two parts, the position of the current character and then the pixel in that character which must be drawn. As the characters have both a width and height as a power of 2, this is easily done by using the least significant

Output bit	Source bit(s)
Red 0	0
Red 1	red
Red 2	red & hi
Green 0	0
Green 1	green
Green 2	green & hi
Blue 0	blue
Blue 1	blue & hi

Table 1: Output colour conversion

bits as the pixel in the character and the rest of the bits as a position telling us which character position in the frame to use.

This makes the pipeline quite straight forward. A memory is used to store 16-bit words describing which characters to draw. This is split in colour and character number. The character value and row number is fed into the font ROM and from this the output pixel is selected using the current column.

The pixel value selects if either background or foreground colour is to be used, if reverse is on this is inverted so that foreground is background and vice versa and finally a component tests if the current position is equal to the set cursor position which inverts the choice once again.

The output of the VGA signal on the board is 8 bits and capable of generating 256 colours. However as part of the standard we only need 7 colours (and high intensity for the foreground colour) which halves the necessary bits for storing colour values. To convert these colours with 1 bit for each colour to 8 bits, some trial and error resulted in the conversion shown in table 1.

A simplified illustration of this pipeline can be seen in figure 8. In this figure, the output conversion as well as reversal of the colour has been removed to improve readability.

4.1.2 Buffering

When drawing to the screen, characters are drawn one pixel row at the time and all characters on that row are drawn before the next row of pixels are begun (as illustrated in figure 4). If the software changes the contents of the memory often this can lead to strange output on the screen while done. To prevent this, a buffering mechanism is needed. The two choices are the simple full buffering, which doubles the memory requirements, or row buffering, which is the minimum amount that can be buffered to avoid this problem. This design uses a row buffering model since the choice at design

time was to preserve memory in order to avoid the need for a redesign late in the project should it be shown that there were an insufficient amount of BRAMs in the FPGA. This result in a more complex set of states that needs to track both Vsync, Hsync, addresses to both buffers and when there is an output signal (as there are multiple Hsync signals when we are not outputting data on the top of the screen). This design is show in figure 7.

The general design of this copy logic is a direct copying of the current row from the shared RAM into the row buffer. This is driven by two counters, one for addressing each of the memories. Since the main memory is 32 bits wide a multiplexer is used to divide this into 16 bit words which the hardware works with. The main memory counter can be initialised to either address 0 which contains 4 8-bit registers that control the hardware, currently they define which row that is used as row 0 and the row and column for the current cursor position.

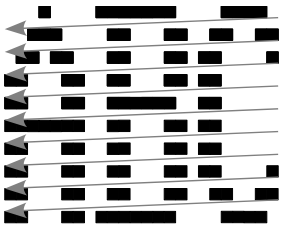


Figure 4: Drawing order for pixels on the display

The state machine initialises on Vsync and starts by loading address 0 in the main counter to load the registers. It then loads the start row and waits for Hsync. Hsync indicates that a new set of 80 words should be loaded into the buffer. Afterwards it blocks until it detects an output signal to the screen and returns to idle mode waiting for either Hsync or Vsync.

4.2 UART

For serial communication a 16550-compatible UART is used. This was chosen because its known features giving access to mainly FIFOs but also handshaking and flow control signals immediately in hardware. The latter features was shown not usable however, as the board does not connect these pins to the FPGA. The UART in this project is provided by Xilinx and connects to the MicroBlaze via the PLB-bus.[9]

4.3 PS/2

The PS/2 interface is provided by a pre-made core from Xilinx which connects to the PLB bus.[6]

5 Software

The software was written in C and can be split into three major parts. Initialisation, interrupts and the ECMA parsing which can be seen in figure 5.

The initialization sets up the hardware, and prints a welcome screen. The interrupts are triggered whenever the PS/2 keyboard sends a scancode. These are then decoded, keeping track of special characters and a few states, such as caps lock. Once decoded the corresponding commands or characters are sent to the host over the serial port.

The ECMA routine decodes the response of the host and updates the screen output accordingly. As seen in figure 5 we handle three different kinds of input: normal characters, special characters and escape codes.

Normal characters are just printed at the correct position on the screen. Special characters such as newline and backspace are interpreted correctly, for example '\n' will cause the software to clear the next row, scroll the screen if necessary and put the cursor at the first position of that row.

An escape sequence is more complex, the sequence may have a number of parameters strings and/or a number of intermediary bytes before it ends with a final byte. In practice the escape sequence is a lot more complex than needed, no applications tested use intermediate bytes and the parameters only contains integers. This allows us to implement a simplified parsing of escape sequences handling a fixed number of integer parameters. For debugging reasons a single intermediate byte is interpreted and stored, however no function makes use of this.

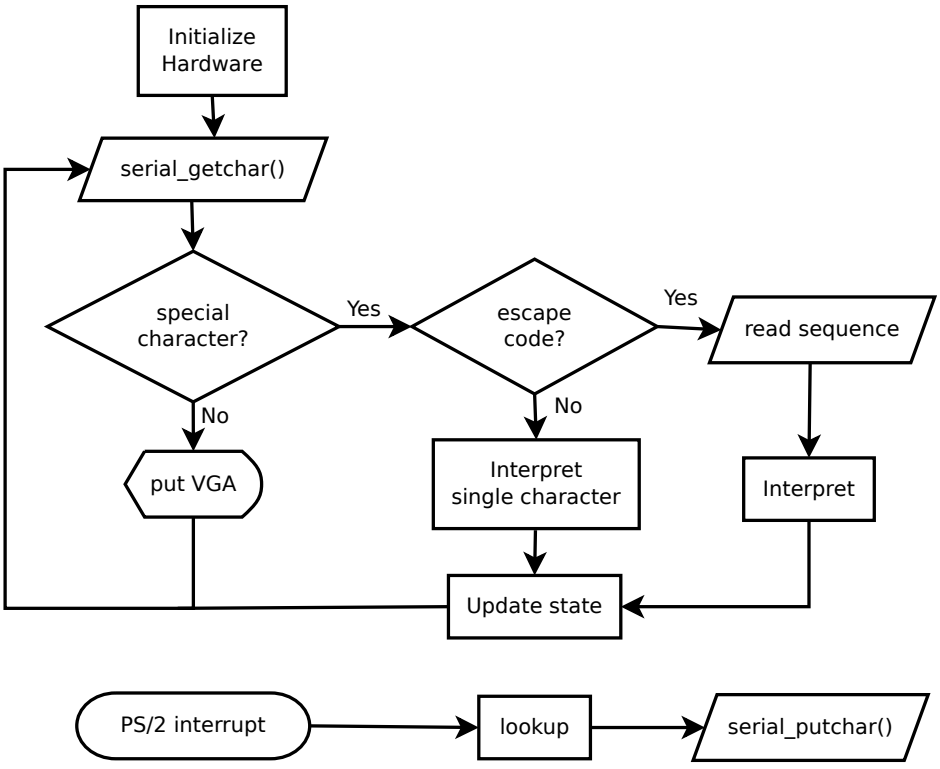


Figure 5: Drawing order for pixels on the display

6 Implementation problems

Design-wise our largest problem was is that we were too careful. Our design decisions were to forego a full double buffer of the video memory to avoid having to go off-chip for more memory. To further limit memory consumption we limited ourselves to 2 bytes per character and removed features such as blinking and underlined text for this reason. Given the current memory usage, 3 bytes would allow for more formatting and result in a minimal increase in device utilisation.

During the implementations we ran into a few hardware problems, where a slow point in the execution causes hardware buffers to overflow. This was particularly visible in the PS/2 controller as the Xilinx PS/2 core does not implement the flow control described in section 3.2 and only buffers a single byte. The biggest culprit causing these problems were the debugging as printing to the JTAG terminal (provided by the MicroBlaze debug module) could sometimes take seconds to transfer the text. We did not find a good solution to this since no part of our input hardware were able to do flow control and thus only could overflow.

The Xilinx Platform Studio also provided its fair share of grief when migrating an ISE project into XPS as an IP core. When creating a custom core as we did using netlists from the Xilinx LogiCORE wizard intermixed with VHDL and on top of that connecting it to a bus interface that is not creatable using the peripheral creation wizard in XPS one have to consult the reference manual for the files describing peripherals[8]. This format is extremely versatile, and allows for the reuse of modules from other cores. Thus if a copy is made of a core it does, by default, use the hardware design from the old core without warnings. Also, while there is a version number associated with each core, XPS does not allow you to easily replace a core with a newer version, even if their connections match exactly. This lead to what was expected to take 2-3 hours for integration, took a whole day of work. Therefore a workflow doing design in XPS, exporting to ISE to build the cores and then updating the software using the SDK would be a more streamlined method if possible. Any completed cores could then as a last step be ported into XPS.

7 Result

The finished prototype works well and has been tested with major software such as ncurses, vim, zshell as well as standalone hardware that utilises escape codes to format a user interface. Currently only a single known issue exists, however its cause is unknown. The bug does not crash the device but merely garbles the output until a clear or similar redraw of the screen is performed. The final device utilisation is shown in table 2. The software is

placed in a 32 KByte BRAM of which it uses 21.7 KByte. The final software has been compiled using GNU CC and using `-O2` optimisations and most debugging routines has been disabled (notably this removes the dependency on `xil_printf`).

8 Contributions

During the project each member had a generalized area of "expertise", PS/2 was done by Björn, ECMA by Andreas and VGA by Mattias. The full ECMA decoding was the largest part of the project. As it was done entirely in software it was trivial do solve on an as needed basis which resulted in gradual evolution by the team member that found out they needed some new instruction.

	Total	VGA core
Slices	3449	113
LUTs	3120	109
Flip Flops	2156	90
16K RAM Blocks	22	2
Multipliers	4	1

Table 2: FPGA device utilization

A Installation

Using a Nexys2-1200 board, connect a VGA display, a PS/2 keyboard and your serial device to the board. Make sure the device you are using is not transmitting on the serial port (if possible, turn it off).

Using a Nexys2-1200 FPGA board, download the bit file to your board. Use `vt100_CCLK.bit` for programming the PROM and `vt100_JTAGCLK.bit` for downloading directly to the FPGA chip. If programming the PROM, power cycle the board after completion. You will now see an instruction screen helping you get started. The baud rate is configurable using the switches on the board; each switch is mapped to a unique speed. No switches enabled mean 1200 baud and then each switch is mapped in order to 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 baud. If multiple baud rates are selected the highest will be used. As you select a baud rate it will be presented to you on the screen if you are unsure. *Be advised that selecting 230400 baud has not been proven safe and might lead to unexpected lockups.*

When satisfied, start your device using the same baud rate. The baud rate will now be locked for the rest of the session (you must press BTN0 to reset the board and tell it to load a new setting).

B Downloadable resources

`http://users.student.lth.se/~dt06mj4/vt100/vt100_CCLK.bit`
`http://users.student.lth.se/~dt06mj4/vt100/vt100_JTAGCLK.bit`

Bit files for programming the Nexys2-board, see also appendix A.

`http://users.student.lth.se/~dt06mj4/vt100/report.zip`

Report source, with sources for graphics used. Permission granted to use and modify these files under CC-by-SA 3.0.

`http://users.student.lth.se/~dt06mj4/vt100/vt100.zip`

Xilinx Platform Studio hardware project and external tools. The directory structure is as follows:

`fpga/master` The main Xilinx Platform Studio project

`fpga/master/VT100` C code running on the MicroBlaze

`fpga/vga_ise` Development project for the VGA hardware using Xilinx ISE

`support/vgafont` Converter tools for PSF1 font files (Linux console fonts)

C XPS Block diagram

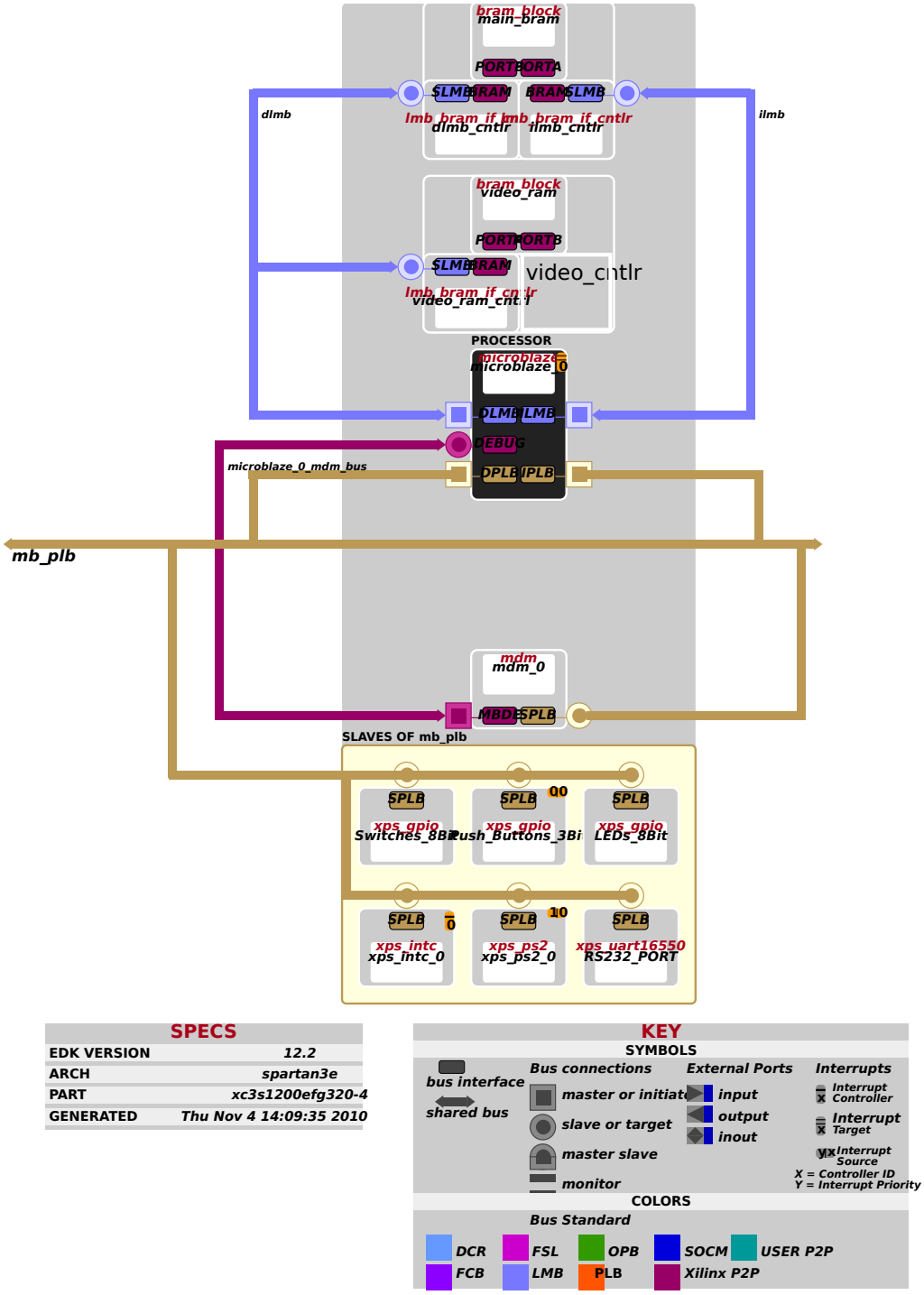


Figure 6: Block diagram

D Hardware diagrams

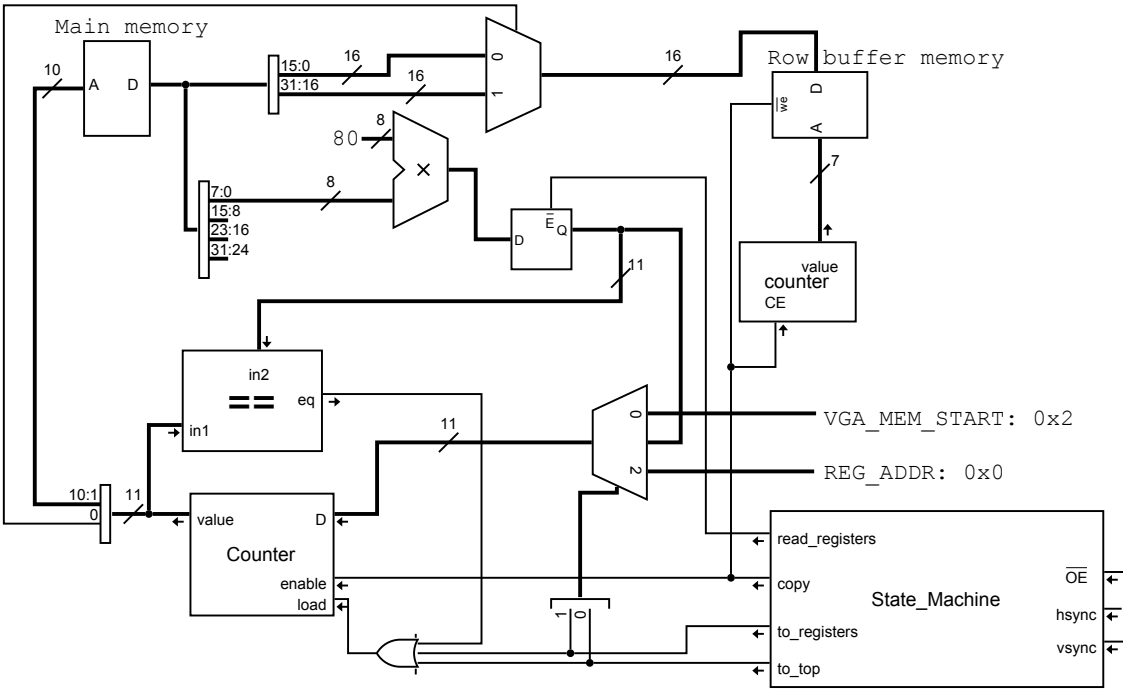


Figure 7: VGA row buffering component

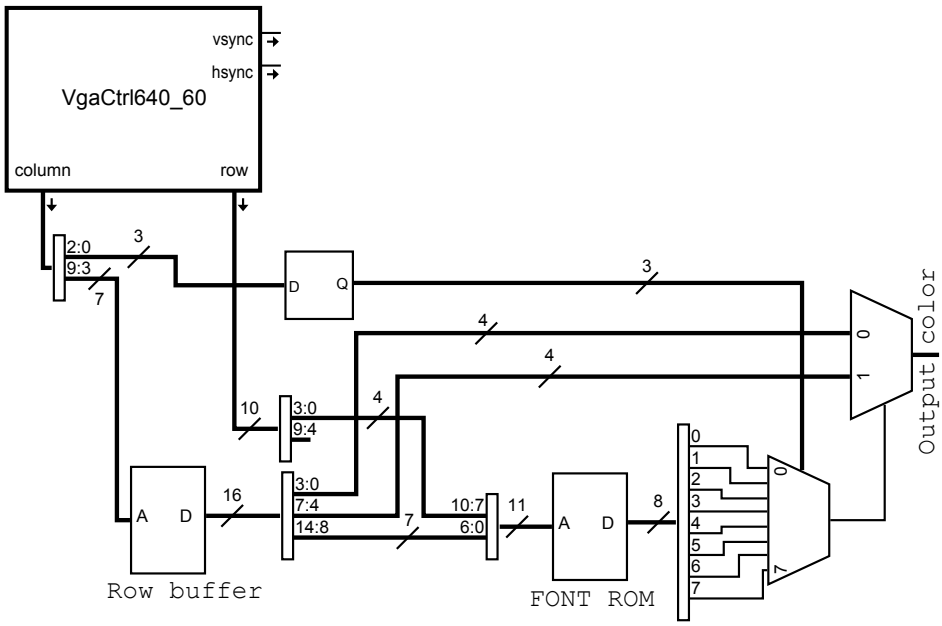
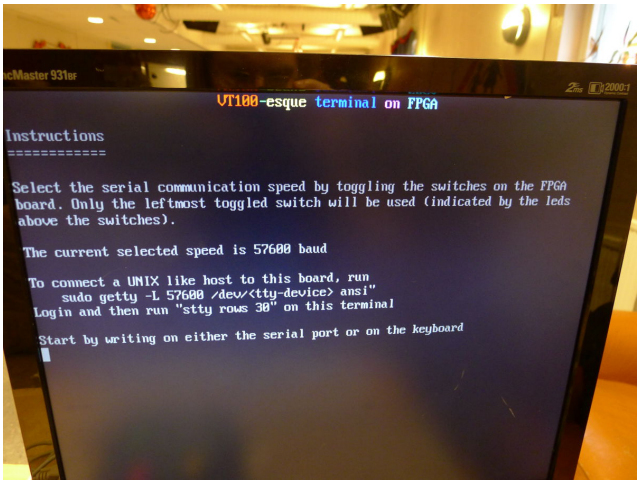
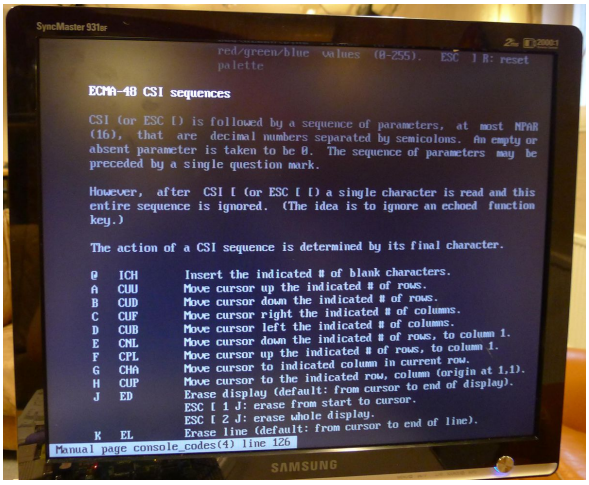


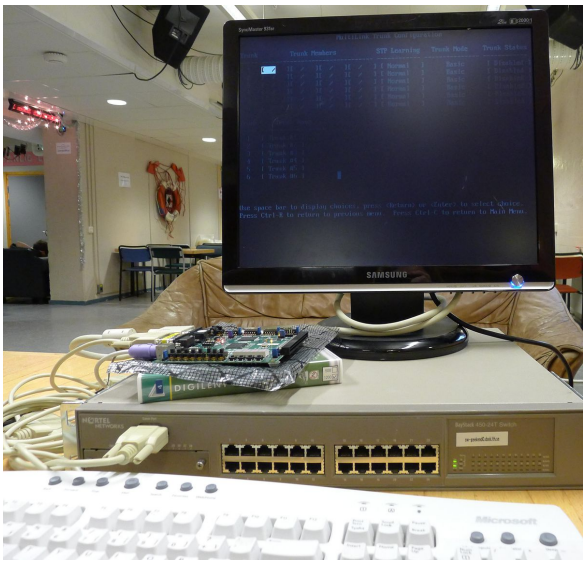
Figure 8: VGA output component

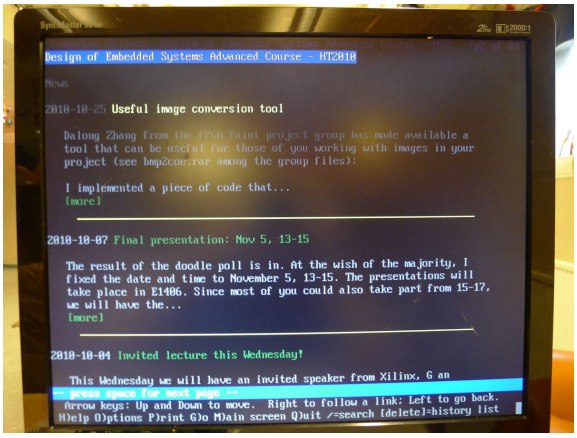
E Gallery

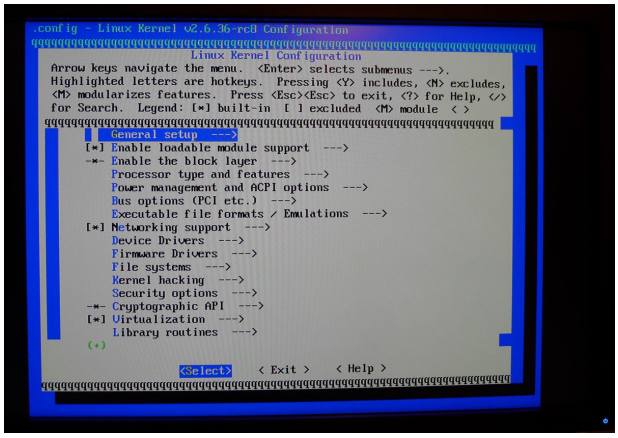

VT100-esque terminal on FPGA
Instructions
Select the serial communication speed by toggling the switches on the FPGA board. Only the leftmost toggled switch will be used (indicated by the leds above the switches).
The current selected speed is 57600 baud
To connect a UNIX like host to this board, run
sudo getty -L 57600 /dev/<tty-device> ansi"
Login and then run "stty rows 30" on this terminal
Start by writing on either the serial port or on the keyboard


ECMA-48 CSI sequences
CSI (or ESC [) is followed by a sequence of parameters, at most NPAR (16), that are decimal numbers separated by semicolons. An empty or absent parameter is taken to be 0. The sequence of parameters may be preceded by a single question mark.
However, after CSI [(or ESC [[) a single character is read and this entire sequence is ignored. (The idea is to ignore an echoed function key.)
The action of a CSI sequence is determined by its final character.
@ ICH Insert the indicated # of blank characters.
A CUU Move cursor up the indicated # of rows.
B CUD Move cursor down the indicated # of rows.
C CUF Move cursor right the indicated # of columns.
D CUB Move cursor left the indicated # of columns.
E CNL Move cursor down the indicated # of rows, to column 1.
F CPL Move cursor up the indicated # of rows, to column 1.
G CHA Move cursor to indicated column in current row.
H CUP Move cursor to the indicated row, column (origin at 1,1).
J ED Erase display (default: from cursor to end of display).
ESC [1 J: erase from start to cursor.
ESC [2 J: erase whole display.
K EL Erase line (default: from cursor to end of line).
Manual page console_codes(4) line 126
SAMSUNG


Switch Password:
Enter Password:


SAMSUNG


Design of Embedded Systems Advanced Course - HT2010
News
2010-10-25 Useful image conversion tool
Dalong Zhang from the FPGA Paint project group has made available a tool that can be useful for those of you working with images in your project (see bmp2coe.rar among the group files):
I implemented a piece of code that...
[more]
2010-10-07 Final presentation: Nov 5, 13-15
The result of the doodle poll is in. At the wish of the majority, I fixed the date and time to November 5, 13-15. The presentations will take place in E1406. Since most of you could also take part from 15-17, we will have the...
[more]
2010-10-04 Invited lecture this Wednesday!
This Wednesday we will have an invited speaker from Xilinx, G an
-- press space for next page --
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list


.config - Linux Kernel v2.6.36-rc8 Configuration
Linux Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ] excluded <M> module < >
General setup --->
[*] Enable loadable module support --->
-*- Enable the block layer --->
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Executable file formats / Emulations --->
[*] Networking support --->
Device Drivers --->
Firmware Drivers --->
File systems --->
Kernel hacking --->
Security options --->
-*- Cryptographic API --->
[*] Virtualization --->
Library routines --->
(+)
<Select> < Exit > < Help >

References

[1] Digilent Inc. *Digilent Plug-in for Xilinx 12.x Tools User Manual*. http://fileadmin.cs.lth.se/cs/Education/EDA385/HT10/documents/chipscope_digilent/Digilent_Plug-in_Xilinx_v12.pdf.

[2] ECMA. *ECMA-48: Control Functions for Coded Character Sets*. ECMA (European Association for Standardizing Information and Communication Systems), pub-ECMA:adr, fifth edition, June 1991.

[3] Digilent Inc. Master ucf file for the nexys2-1200. http://www.digilentinc.com/Data/Products/NEXYS2/Nexys2_1200General.zip.

[4] Digilent Inc. Vga controller reference design. http://www.digilentinc.com/Data/Documents/Reference%20Designs/VGA%20RefComp.zip.

[5] The Linux man-page project. *console_codes(4)*.

[6] Xilinx. *LogiCORE IP XPS PS2 Controller (v1.01b)*, april 2010. DS707.

[7] Xilinx. *MicroBlaze Processor Reference Guide*, jul 2010. UG081 v11.1.

[8] Xilinx. *Platform Specification Format Reference Manual (EDK 12.2)*, jul 2010. UG642.

[9] Xilinx. *XPS 16550 UART (v3.00a)*, may 2010. DS577.