Control Display Unit For A HI-4422 Electric Field Probe
EE 552 Project Design Lab

The design elements of this project are entirely the original work of the authors, except as follows:

- Clock Divider code "clkdiv.vhd" was modified from reference [5]
- VGA code “countxy.vhd”, “syncgen.vhd”, “title.vhd”, “char_set.mif”, “ic_test.mif” was modified from reference [6]

Signatures:

John-Michael Carolan Graeme Fricke

Christopher Kowalski

Name:	**E-mail address:**
John-Michael Carolan	jcarolan@ee.ualberta.ca
Graeme Fricke	gfricke@freenet.edmonton.ab.ca
Christopher Kowalski	cmk4@ualberta.ca

Abstract

The Government of Canada requires that human exposure to radio frequency electromagnetic fields be limited (see www.hc-sc.gc.ca/ehp/ehd/catalogue/rpb_pubs/99ehd237.htm). It is therefore necessary to determine the strength of these fields in cases where safety may be compromised. The purpose of this project is to design and construct a control/display unit (CDU) for the Holaday HI-4422 isotropic electric field probe. The CDU takes readings in from the probe and display the results onto a VGA monitor. As well a safety device comprising of a strobe light and a two-tone siren built by the Aerospace Engineering Test Establishment (AETE), is activated if the electric field is above a predefined tolerance limit.

Table of Contents

Achievements	1
Description of Operation	2
FPGA Data Sheet	8
Resource Requirements	10
Experimental Results	10
References	11
Diagram of Design Hierarchy	12

1.0 Achievements

A control and display unit for a Holaday HI-4422 electric field probe has been successfully built. The following are the achievements required to accomplish this:

A functional RS-232 / HI-4422 probe controller with parity checking, handshaking signals and code conversion has been successfully coded and tested in simulation and in hardware. A signal can be correctly sent to the probe and the signal sent back from the probe is properly received. This was tested by sending various commands to the probe and checking if one of the desired characters was returned.

The ability to get input from a PS/2 keyboard and convert the received scan code has been fully coded and tested in both simulation and in hardware. Test code, keydisplay.vhd, was used to display the scan code sent from the keyboard upon a key press on two 7-segment LED displays. The scan code displayed was the hexadecimal representation of the scan code.

That the monitor output is what I designed it to be is something that I regard as an achievement, as this is something that I haven't attempted before. Although I have based the code upon previous EE552 projects (the logic analyzer and the dice race), I'm using a higher resolution and more variable output, so I'm pleased that the underlying logic works properly.

In order to reduce the number of logic cells used by this project, it was necessary to carry out some optimization. I was able to achieve this by reducing the number of conditions required to determine the display, by using case statements where previous groups had used if-then statements and by minimizing the size of the additions used to calculate ROM addresses from screen coordinates. Since this was a divergence from previous code and since it produced significant results, I consider this an achievement, prosaic though it may be.

In creating the Command Display Unit (CmdDU), two major factors needed to be considered: time, and size. A major achievement was meeting these two goals. The ideal design of the Command Display Unit would be a custom microprocessor core, which would run instructions from memory. A problem with this is time. To construct a microprocessor with a custom instruction set and architecture would require more time than what was available. Existing microprocessor cores were available, but used so many logic cells that the remainder of the project would not fit on a FLEX10K20 FPGA. A tri-state bus could be used, but would make the flow of sending and receiving data difficult to ascertain without extensive handshaking signals. Finally, there is the tried and true state machine method. Using a giant state machine to send and receive information to the probe, and prepare it for use in the VGA display. This proved to be the most flexible solution, and gave the best chance of fulfilling the time and size requirements.

2.0 Description of Operation

2.1 General Overview:

The Control-Display Unit (CDU) is designed to interface with a Holaday HI-4422 Isotropic Electric Field Probe, receive a reading from the probe, and display it to a VGA compatible monitor. External control of the system is accomplished through a PS/2 compatible AT-keyboard or keypad. The probe is capable of a number of different functions which the user can control. An explanation of these function will follow.

2.2 The HI-4422 Probe

The following is a general explanation of the HI-4422 probe's operation taken from the User's manual (see reference 2):

The HI-4422 operates as a controller mode device. That is it only sends data in response to a command received. The information that the probe sends and receives is in the RS-232 standard with the following parameters:

Word Length:	7 bit
Parity:	Odd
Stop Bits:	1
Data Rate:	9600 baud

Commands sent to the probe consist of:
1) Command letter
2) Parameters if required
3) A terminating carriage return

Commands from the probe consist of:
1) A ":" start character
2) Command letter
3) Data if required
4) A terminating carriage return

The CDU is capable of sending the following commands to the probe (see the user's manual, reference 2, for a complete listing of all commands for the probe and the exact form they are in):

Axis enable/disable: The X, Y, and Z axis antenna readings can be read independently or simultaneously in any combination. This command allows the user to choose which of the axis(es) to take a reading from.

Read Battery Voltage: The HI-4422 has an internal rechargeable battery, this command tells the probe to send the voltage level.

Read probe data: The probe data is requested from the probe. The data is read in the "long form" only. (See received signals for information on what is received).

Set range: The probe has four internal ranges. This command selects the desired range.

Read Temperature: This command tells the probe to return the external air temperature in degrees Celsius.

Zero: The HI-4422 needs to be calibrated with a "zero" reference level upon power up. The command tells the probe that the current reading is to set as the zero reference level.

The CDU is capable of receiving the following commands from the probe (see the user's manual, reference 2, for a complete listing of all commands for the probe and the exact form they are in):

Bxx.xx: The battery voltage where xx.xx is the voltage.

Dxx.xxuuurrrobaaa: The reading from the probe where:
xx.xx is the reading. The position of the decimal point is dependent upon the range.

uuu is the units the probe is sending the reading in.
_V_ = V/m; mW2 = mW/cm^2; _V2 = $[V/m]^2$. (Underscore indicates a space)

rrr is not used in this design

o is the over range indicator. N = o.k.; O = over range

b is the battery status. N = safe operating level; W = warning level; F = fail level

aaa is the axis enable information. E = enabled; D = disabled; the axis order is X,Y,Z

Rx:	The range the probe is in where x is the numerical value
Txxx:	The temperature, for this design only the temperature in Centigrade is displayed.

2.3 RS-232 controller

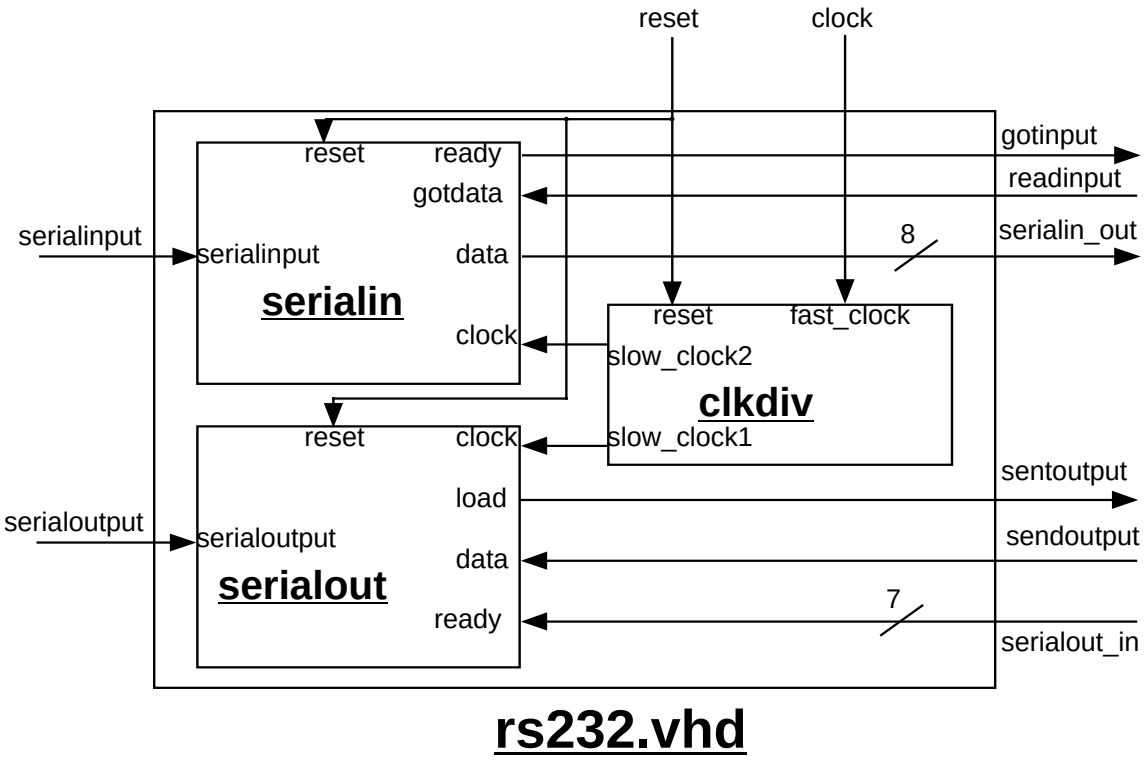
The RS-232 controller consists of three separately coded components. A serial input port, a serial output port, and a clock scaler.

The serial input port takes in the serial data one bit at a time and stores the data in a register. Once one data word has been received the data, with out the start and stop bits, is sent out a parallel bus. At the same time a “ready” line is raised high, and remains high until an other input line, “gotdata” is asserted at which time the “ready” line goes low until another world has been received in.

The serial output port takes in a 7-bit ASCII word in parallel, calculates the correct parity bit, and then sends the data out serially with a start and stop bit. The data is loaded when the “load” line is asserted and sent once the "load" signal is lowered. The signal “ready” goes high once all the data has been sent.

The clock scaler produces two clock signals slower then the system clock. One clock signal at 9600 Hz is used by the serial output port, and the other clock signal, which is 16 times faster then the clock signal, is used by the serial input port. When reset is asserted, the two clock signals produced, match the system clock to allow the two serial ports to reset.

The three above components are connected as follows:



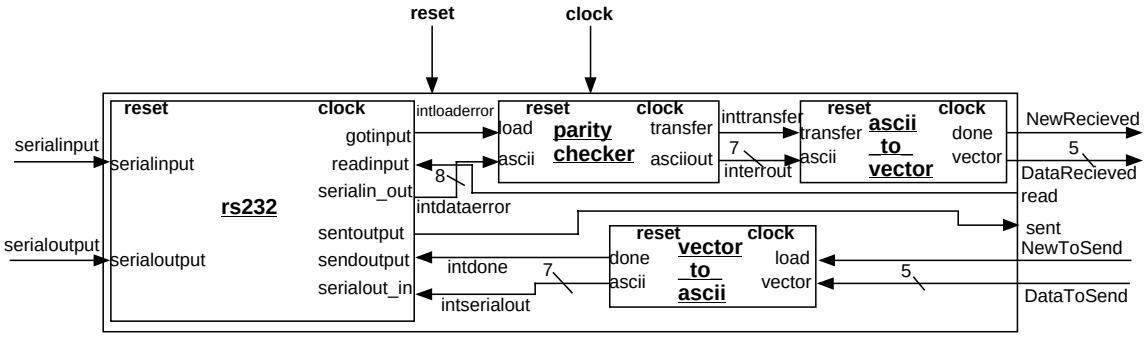
2.4 Probe I/O controller

The probe Input / Output (I/O) controller consists of four different components. The RS-232 component described above, a parity error checking component, a ASCII-to-vector converter, and a vector-to-ASCII converter.

The parity checker compares the parity bit received from the serial input port to what parity should have been received. It takes is a 8-bit vector in parallel and sends out a 7-bit vector. If the parity is correct the 7-bits sent correspond to the ASCII word received by the probe I/O controller. If the parity is incorrect then the 7-bits sent is all ones.

Both the ASCII-to-vector and the vector-to-ASCII components function in the same manner, only they are inverses of each other. A 7-bit ASCII character is received/sent and a 5-bit vector is send/received. The two components perform the translation between the ASCII and the 5-bit vector or vise versa. Please see the VHDL code ascii_to_vector.vhd for a chart showing the conversion done.

The components are connected as follows:



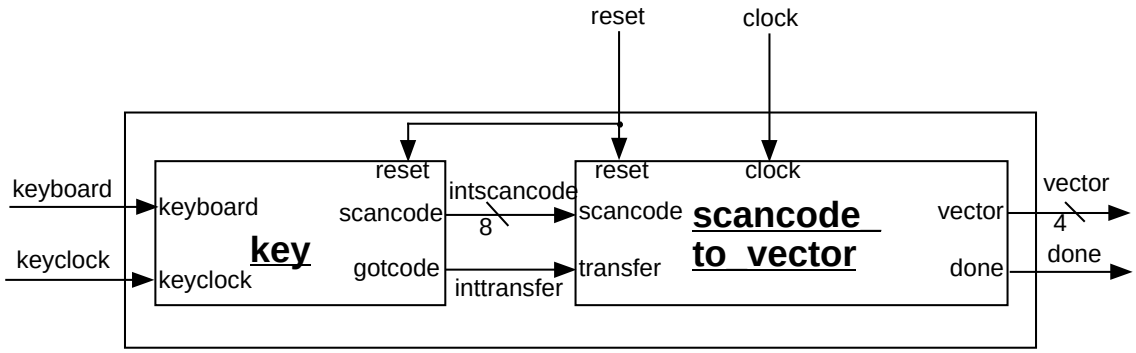
2.5 Keyboard controller

The keyboard controller consists of two separate components: a key reader, and a scan code-to-vector converter.

The key reader reads in the 11-bit scan code corresponding to the key that was pressed on the keyboard. Once all the data has been received the 8-bits representing the key's scan code are send out in parallel, and the signal "gotcode" is asserted. The start, stop, and parity bits are ignored and are not passed out of the key reader.

The scan code-to-vector converter reads in the 8-bit scan code from the key reader when the signal "transfer" is raised high. The 8-bit code is then converted into a 4-bit vector. If the received scan code is not one of the desired codes programmed into the scan code-to-vector unit, then a vector of all ones is sent out. Please see the VHDL code scancode_to_vector.vhd for a chart showing the conversion done.

The components are connected as follows:



2.6 VGA Display

The operation of a VGA monitor is governed by five signals. Two of these signals, the horizontal and vertical synchronization signals, control the location of output data on-

screen. The horizontal synchronization signal tells the monitor when to begin displaying a row of pixels and when to reset itself to the beginning of the row while the vertical synchronization signal tells the monitor when to change rows or reset itself to the top left-hand corner of the screen. The other three, the colours red, green and blue, determine the colour of each pixel. However, the synchronization signals act independently of the colour signals, so it is necessary for the colour signals to be properly timed - otherwise, the wrong data will be displayed.

The file probedisplay.vhd incorporates both the timing and data aspects of the VGA display. This proved to be the most convenient approach to the problem of controlling the monitor, as it eliminated the need to track signals across multiple entities. Within probedisplay.vhd, the synchronize process ensures that the synchronization signals are properly timed; it also generates the horizontal and vertical counters which are used to determine which pixel is currently active. The rest of the program consists of routines which produce the appropriate colour for each pixel, depending on the counters.

The 640 x 480 pixel VGA display can be subdivided into 80 x 60 square blocks of 64 pixels, working with the origin at the top left-hand corner of the screen. Consequently, character display is carried out by defining each block as containing a character or a blank. The col_address and row_address vectors, which omit the least significant three bits of the horizontal and vertical counters, determine the current block, while the least significant three bits of each determine the current pixel within the block.

Pixel colouring is handled through the use of two ROMs. The first of these, chars.mif, stores characters as 8 by 8 arrays of bits. Each address in the ROM refers to an 8-bit vector; eight of these vectors defines a character. The second ROM, screenwords.mif, stores the locations of sequences of characters within chars.mif. The program uses the row_address and col_address vectors to determine within screenwords.mif which character to look up in chars.mif; chars.mif, in conjunction with the pixel count within the current character block, then indicates whether a pixel should be on or off.

The on-screen location is also used to determine the colour of a pixel. Each coordinate is defined as either white (if the pixel is off) or some specific colour (if the pixel is on). This is independent of what is actually being written on the screen, and uses a separate process. A handy side-effect of the white background is that while colours like red and green must be defined by location, black characters don't require any such code - defining an off pixel as white automatically defines an on pixel as black. This cuts dramatically the number of conditional statements necessary for colour definitions, which in turn reduces the number of logic cells required.

The variable displays (such as the temperature value and the range indicator) are for the most part handled in this fashion. However, the memory addresses used to retrieve data from the ROMs are determined by the states of incoming external signals, so a number of possible characters are available to each relevant character block on screen. Thus, for example, any number from 0 to 9 may be displayed in the character block which was assigned to the first digit of the temperature, dependent on the input signal temp_value1.

The bar graph is handled in a similar manner, although its output is determined on a pixel-by-pixel basis rather than by initially using character blocks. The current probe reading (rounded down to the nearest integer) is calculated from the four signal vectors which specify the four reading digits (in the process varybar). For the appropriate rows of pixels, if the x-coordinate of a pixel is larger than that of the y-axis but smaller than the x-coordinate of the y-axis plus the probe reading then that pixel is coloured blue. The procedure is simple, but the multipliers required to construct the probe reading from its digits add a significant number of logic cycles (at least 5%) to the project.

probedisplay.vhd, then, outputs three types of data to the VGA monitor:

1. Fixed labels, whose pixel coordinates and addresses in memory are predetermined;

2. Variable values, whose pixel coordinates are fixed but which access different memory locations depending on input signals; and

3. The bar graph, whose pixel coordinates are variable but which doesn't access the ROMs for its colour.

Although three previous projects (Arkanoid, Dice Race and Logic Analyzer) all implemented some form of VGA display, their coordinate and synchronization counters failed when they were incorporated into probedisplay.vhd. It is necessary that the synchronization signals initially fire at the same time as the first red, green and blue signals, but simply counting up from zero as in previous projects caused these signals to be staggered. By delaying the synchronization signals for two clock cycles, the five signals were moved back into phase with each other.

A more significant problem was that the horizontal counter countX was initially too slow, resulting in pixels from the start of one character overwriting the first few pixels of the following character. This problem was solved by implementing a second counter which was equal to countX + 2 and which took over when countX = 6. Thanks are owed to John Koob for spotting the solution to this problem.

2.7 Command Display Unit

The Command Display Unit operates in a manner as follows:

1. Resets all registers and control signals on start or reset
2. Sends commands to get information from the probe, and receives and interprets that information – storing it in appropriate registers for use in the VGA display.
3. Checks for any keys pressed, and send commands to service them accordingly.

3. Size of the code (in logic blocks and percentage)

The size of the code as compile on MaxPlusII v9.6 is 407 logic cells, or 35% of the FLEX10K20.

4. Any experiments you did and their outcome (I.E. reducing states, adders, compiling upstairs vrs. downstairs)

Speed wasn't a large consideration, as 1MHz would be sufficient to read information from the ProbeIO module, however, space was a consideration. When compiling on MaxPlus2 v9.3, it was found that the amount of spaced used on a FLEX10K20 was 80% of the available logic cells, which is 35% more than what was available – fitting other modules on the FPGA. By compiling on v9.6, this was reduced by 45%, and speed was increased by an average of 6x. Reducing more states further increased speed, but did little to reduce overall area used on the FPGA.

3.0 FPGA Data Sheet

The CDU was built on an Altera UP1 Education board. Both the EPM7128SLC84-7 and the EPF10K20RC240-4 FPGAs were utilized in this design. The design incorporates the use of a VGA monitor via the on board VGA port, a PS/2 keyboard via the on board PS/2 port, a RS-232 connection wired externally from the board, and a strobe / siren box controlled via an external relay, transistor switch network.

3.1 PS/2 Keyboard

The PS/2 keyboard is connected to the UP1 board via the built in PS/2 connector. The following table shows the function associated with each key:

User Command	**Mapped Key**	**Description and Notes**
Axis Enable/Disable: X	7	This is a toggle command. It toggles between turning on and off the X-axis measurment.
Axis Enable/Disable: Y	8	This is a toggle command. It toggles between turning on and off the Y-axis measurment
Axis Enable/Disable: Z	9	This is a toggle command. It toggles between turning on and off the Z-axis measurment
Range Select: 1	1	This keys tells the probe to send the reading in the range upto 10 V/m
Range Select: 2	2	This keys tells the probe to send the reading in

		the range upto 30 V/m
Range Select: 3	3	This keys tells the probe to send the reading in the range upto 100 V/m
Range Select: 4	0	This keys tells the probe to send the reading in the range upto 300 V/m
Range Select: Next	. (Decimal)	This keys tells the probe to send the reading in the next highest range
Zero the Probe	+	This set the current reading of the probe to be a zero reference.

Two signals are received from the keyboard: Data, and the keyboard clock. See 3.5 for a table of pin connections.

3.2 VGA Monitor

The VGA monitor is connected to the UP1 board via the built in VGA connector. The signals Red, Green, Blue, Horizontal sync, and Vertical sync are sent to the monitor via the VGA connector. See 3.5 for a table of pin connections.

3.3 RS-232

One RS-232 signal is received by, and sent by the UP1 board. A serial output is sent on one wire, and a serial input is received on one wire. Signals to and from the RS-232 / Probe I/O unit are also sent to and from the EPM7128SLC84-7 (the FPGA were the probe I/O resides) to and from the EPF10K20RC240-4. Six different signals on 14 wires were used. The signals sent from the EPM7128S are "DataReceived" (5 wires), "NewReceived" (1 wire), and "sent" (1 wire). The signals received are: "DataToSend" (5 wires), "NewToSend" (1 wire), and "read" (1 wire). See 3.5 for a table of pin connections.

3.4 Strobe / Siren Box

Only one signals is sent from the UP1 board to the strobe / siren box; "noise". See 3.5 for a table of pin connections.

3.5 Pin Connections

The following table shows the location of connections to the UP1 board.

Name	EPM7128S Pin	EPF10K20 Pin	UP1 Pin	Description

Reset	45	66 (Expan A: 30)	MAX_PB1	A global reset.
keyboard	NC	31	PS/2: 3	Serial data from the keyboard
keyclock	NC	30	PS/2: 1	Clock from the keyboard
vga_red	NC	236	VGA: 1	Control of the red gun on the VGA monitor
vga_green	NC	237	VGA: 2	Control of the green gun on the VGA monitor
vga_blue	NC	238	VGA: 3	Control of the blue gun on the VGA monitor
HSync	NC	13	VGA: 13	Control of the horizontal sync on the VGA monitor
VSyn	NC	14	VGA: 14	Control of the vertical sync on the VGA monitor
serialinput	12	NC	P2	Serial input from the external RS-232 circuit
serialoutput	30	NC	P2	Serial output from the external RS-232 circuit
DataToSend0	80	231	Expan_C: 56	Data to be sent out serially. Bit 0 of 4.
DataToSend1	4	229	Expan_C: 54	Data to be sent out serially. Bit 1 of 4.
DataToSend2	6	227	Expan_C: 52	Data to be sent out serially. Bit 2 of 4.
DataToSend3	8	225	Expan_C: 50	Data to be sent out serially. Bit 3 of 4.
DataToSend4	10	222	Expan_C: 48	Data to be sent out serially. Bit 4 of 4.
NewToSend	9	220	Expan_C: 46	Handshaking control signal
Sent	11	218	Expan_C: 44	Handshaking control signal
DataRecieved0	44	46	Expan_A: 16	Data received serially. Bit 0 of 4.
DataRecieved1	46	49	Expan_A: 18	Data received serially. Bit 1 of 4.
DataRecieved2	48	51	Expan_A: 20	Data received serially. Bit 2 of 4.
DataRecieved3	50	54	Expan_A: 22	Data received serially. Bit 3 of 4.
DataRecieved4	52	56	Expan_A: 24	Data received serially. Bit 4 of 4.
Read	51	62	Expan_A: 26	Handshaking control signal

NewRecieved	49	64	Expan_A: 28	Handshaking control signal
Noise	NC	68	Expan_A: 32	Signal to activate the strobe / siren box.

4.0 FPGA Resource Requirements

The design is going to be split over the two FPGAs of the UP1 board. The probe IO is on the MAX7000 FPGA and the controller, VGA display, and the keyboard input are on the FLEX10K FPGA.

On the MAX7000 FPGA the probe IO component takes up 120/128 logic cells or 93% of the total number of logic cells.

On the FLEX10K FPGA the breakdown of logic cells required is as follows:

Component Name	**Number of Logic Cells**
Keyboard Control	73/1152 or 6%
Command Control unit	408/1152 or 35%
VGA Output	535/1152 or 46%
Clock divider	001/1152 or 0%
Total	1016/1151 or 88%

5.0 Experimental Results

5.1 Keyboard Implementation

A number of different experiments were preformed with the PS/2 keyboard. First the correct functioning of the keyboard and keyboard control VHDL code was verified by the code keydisplay.vhd. This code allowed the scan code sent from the keyboard to be displayed on two 7-segment LEDs. The following characteristics of the keyboard were observed:

1. The typematic delay for the keyboard in use has a default value. I.E. The keyboard will send multiple signals if a key is held down for a certain length of time, even if no delay time is sent to the keyboard
2. When two keys are pressed one after the other two different results can occur. If key "a" is pressed and then key "b". The scan code for "a" is sent and then for "b". If key "b" is released first the keyboard sends F0 (hex) and then the scan code for "b". If, however, "a" is released first, the keyboard sends F0 (hex), the scan code for "a", and then the can code for "b".

5.2 RS-232 / Fiber Optic Modem (Holaday part HI-4413P)

The HI-4413P was designed to be used by a computer, as such a few modifications had to be made. The DTR, RTS, and CTS lines are required to provide power to the modem. If was found that they had to have a logic 0 (TTL: 0V, RS-232: +3 - +25) on them, not a logic 1. It was also discovered that the HI-4413P's software was incompatible with the Windows NT boxes in the lab and could therefore the probe and the HI-4413P could not be tested by this manner.

5.3 VGA display

I experimented with two different approaches to the VGA counters. The first required three separate programs - two handled the synchronization signals and the pixel counters and the third looked after the graphics display. This proved to be unwieldy and difficult to debug, so I tried incorporating all of the code into a single program. The counter and synchronization code is quite small relative to the combined code, so I can now see no reason to use the first method. In fact, I recommend against it - the “complete in one program” approach works well

5.4 Hyperterminal tests.

A number of tests were preformed with hyperterminal, all of which were unsuccessful due to the fact that we were unable to get hyperterminal working properly

6.0 References

1. Health Canada: http://www.hc-sc.gc.ca/ehp/ehd/catalogue/rpb_pubs/99ehd237.htm

2. Holaday Industries, HI-4422 Isotropic Electric Field Probe User's Manual, Reversion A: August 94, Holaday Industries Inc.

3. Beyond Logic Organization: http://www.beyondlogic.org/

4. Altera UP1 Board Data sheet

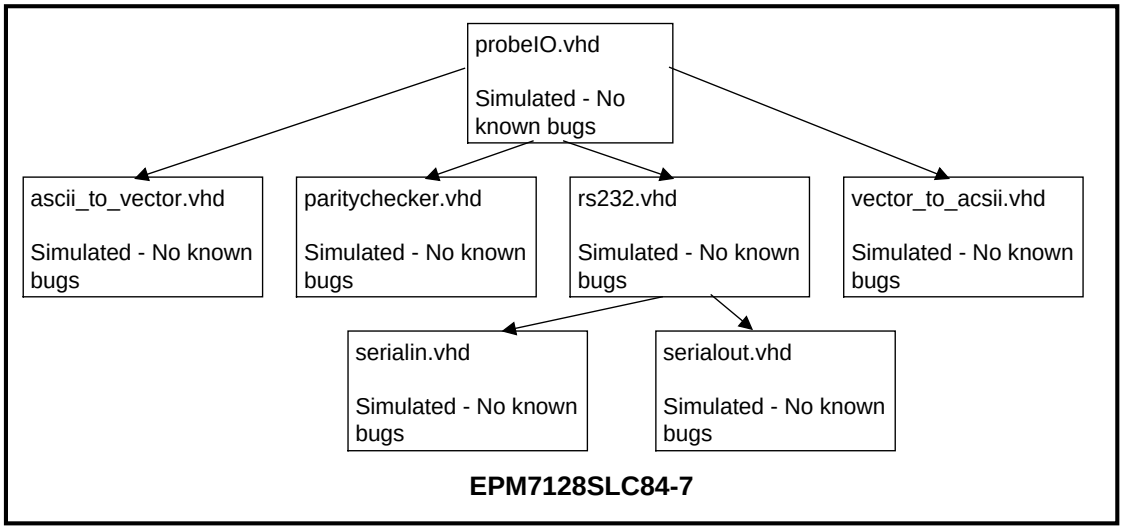
5. EE 552 Class Notes

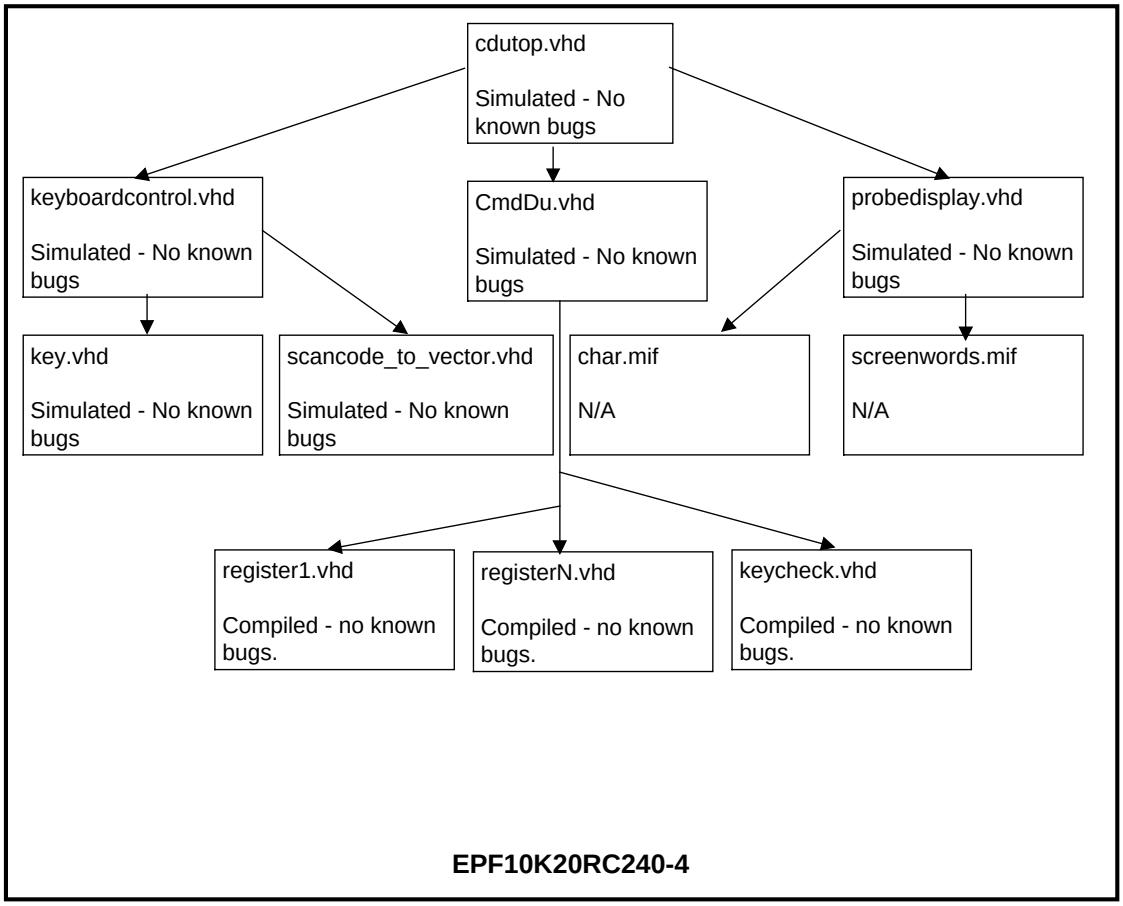
6. EE 552 Past Project: Logic Analyzer http://www.ee.ualberta.ca/~elliott/ee552/projects/1999_w/logic_analyzer/final_report.htm

7. Application note provided by Tyler Brandon, Chris Blasko and Kevin Lister
www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1998f/framebuffVGA/appnot.txt

8. EE 552 application note provided by Hao Luan, Bo Liu and Albert Chan
www.ee.ualberta.ca/~elliott/ee552/studentAppNotes/1998_w/dicerace_video_display

10.0 Diagram of Design Hierarchy


probeIO.vhd
Simulated - No known bugs
ascii_to_vector.vhd
Simulated - No known bugs
paritychecker.vhd
Simulated - No known bugs
rs232.vhd
Simulated - No known bugs
vector_to_acsii.vhd
Simulated - No known bugs
serialin.vhd
Simulated - No known bugs
serialout.vhd
Simulated - No known bugs
EPM7128SLC84-7


cdutop.vhd
Simulated - No known bugs
keyboardcontrol.vhd
Simulated - No known bugs
CmdDu.vhd
Simulated - No known bugs
probedisplay.vhd
Simulated - No known bugs
key.vhd
Simulated - No known bugs
scancode_to_vector.vhd
Simulated - No known bugs
char.mif
N/A
screenwords.mif
N/A
register1.vhd
Compiled - no known bugs.
registerN.vhd
Compiled - no known bugs.
keycheck.vhd
Compiled - no known bugs.
EPF10K20RC240-4