# ECE 484 – Remote Terminal Laboratory Assignment

Last Updated: 1 March 2013

## Lab Overview

In this lab, you will interface a number of peripherals to a simple PicoBlaze processor. By using a USB-to-UART bridge, you will create a program that can take a command written over your computer’s serial port (i.e., remote terminal) and read or write to any one of your input or output peripherals. Specifically, you will need to control the following on your development board: LEDs and switches. All characters typed by the user must be echoed.

In the second part of the lab, you will implement the same logic, but use the MicroBlaze processor instead. You will also add a VGA output peripheral.

## System Overview

Your software will read in three digit commands along with optional parameters. The list of commands you must implement is provided in Table 1. The commands are executed by your code as soon as the user finished typing (i.e., they do not have to press “Enter”).

|  |  |
| --- | --- |
| **Command** | **Description** |
| led ## | Write the hex value “##” to the LEDs |
| swt | Read in the current switch value and write to the terminal as a two hex values. |

Table : List of commands that must be implemented in this laboratory assignment. See Figure 1 for an example terminal session.

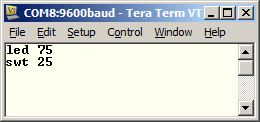


Figure : Sample terminal session that shows all the features needed for this lab. This session set the LEDs to 0x75 and shows that the switches are currently set to the value 0x25.

## Prelab Assignment

Create a PicoBlaze design that meets the following requirements:

1. Defines the following ports:
   1. Port 0xAF – Read switch inputs
   2. Port 0x07 – Read push button inputs
   3. Port 0x07 – write button values to upper-four LEDs, and lower-four switch values to the lower-four LEDs.
2. Constantly read in the push buttons and lower-four bits of the switches and writes those values out to the LEDs.

Turn in a hard copy of your software and VHDL code along with simulation screenshots to demonstrate your design works correctly.

You should use the openPICIDE software to write and simulate your assembly code, as shown in class. Make sure you have the following settings:

* Spartan 6 FPGA
* Memory Bank Size of 1024 instructions
* Select the VHDL PicoBlaze ROM template from the course website as your “VHDL Template.”

## PicoBlaze Implementation

You must design your hardware and software implementation for this lab.

## MicroBlaze Implementation

With MicroBlaze, recreate the same functionality as in the first part of this lab. However, you must also add a vga command that allows the user to specify the background color to send to the VGA monitor.

*Note:* Never connect any of your ports to the “GCLK” signal. Or you will get a PAR error and your design will not work correctly.

## Lab Hints

* Define your ports as constants in your PicoBlaze and VHDL code. This will make your code much more readable.
* Tera Term is freeware software that can communicate over your computer’s serial ports. You can download it from the course website.
  + Configure the speed to Tera Term by using the “Setup” → “Serial Port” menu option
  + Select the USB-UART serial port (check your computer Device Manager if you are unsure of the port).
  + The baud rate should be 9600 for this lab

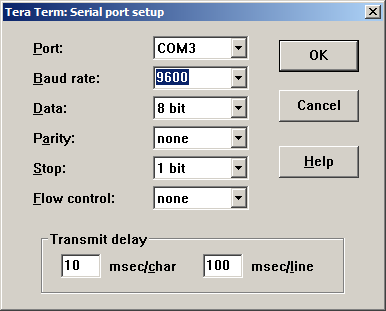


Figure - Example serial configuration for Tera Term.

* Note: Your “serial in” – i.e., where the computer is sending you data for the FPGA to read – pin is A16. Look at the datasheet to find the “serial out” pin location.
* To avoid crossing clock domains, *all* your synchronous elements should be based on a 25 MHz clock.
* Take advantage of the simulator in the openPICIDE.
* Build converter modules (e.g., ascii\_to\_nibble, nibble\_to\_ascii, etc.).
* Use the Xilinx-provided (available on the course website) uart\_tx6 and uart\_rx6 modules rather than writing your own UART controller.
  + The reference manual for these modules is available on the course website.
  + You must create a module, clk\_to\_baud, that generates an enable signal that pulses high for one clock cycle. Based on the time between these pulses, your baud rate is 16 times that value. For example, if you have an enable pulse once every 1 ms, your baud rate is .
  + You will probably need to manually create the write\_buffer signal rather than using PicoBlaze’s write\_strobe signal due to timing issues. The same applies for the write\_strobe signal.
* Start small and work larger. Here is a sample approach to this problem:
  1. Ensure the UART configuration is correct on your FPGA and drivers are installed on your computer. Do a hardware loopback on the UART module (serial\_out <= serial\_in;).
  2. Write your clk\_to\_baud module. A good example is provided in the Xilinx UART manual on the course website. Simulate this design to ensure it runs correctly.
  3. Connect the uart\_tx6 and uart\_rx6 modules to a simple PicoBlaze program that takes the serial input (if available) and writes it to the serial output.
  4. Expand your PicoBlaze code to process the swt command.
  5. Expand your PicoBlaze code to process the led ## command.

## Extra Credit

Add additional features to this lab for extra credit. Here are a few ideas, but you can come up with your own as well:

* Add a command-line prompt similar to what you see when you are using cmd.exe in windows.
* Add error checking to the input. Display an error message if the command is not valid.
* Add another unique (i.e., non-trivial – so just using the push-buttons as another input would not count) peripheral to your design. For example, you could use the character generator module created earlier in the course. Another, perhaps easier, option is to add the seven-segment displays an output.

|  |
| --- |
| **entity** ascii\_to\_nibble **is**  **port** **(** ascii **:** **in** std\_logic\_vector**(**7 **downto** 0**);**  nibble **:** **out** std\_logic\_vector**(**3 **downto** 0**)**  **);**  **end** ascii\_to\_nibble**;**  **entity** nibble\_to\_ascii **is**  **port** **(** nibble **:** **in** std\_logic\_vector**(**3 **downto** 0**);**  ascii **:** **out** std\_logic\_vector**(**7 **downto** 0**)**  **);**  **end** nibble\_to\_ascii**;**  **entity** clk\_to\_baud **is**  **port** **(** clk **:** **in** std\_logic**;** -- 25 MHz  reset **:** **in** std\_logic**;**  baud\_16x\_en **:** **out** std\_logic -- 16\*9.6 kHz  **);**  **end** clk\_to\_baud**;**  **entity** atlys\_remote\_terminal\_pb **is**  **port** **(**  clk **:** **in** std\_logic**;**  reset **:** **in** std\_logic**;**  serial\_in **:** **in** std\_logic**;**  serial\_out **:** **out** std\_logic**;**  switch **:** **in** std\_logic\_vector**(**7 **downto** 0**);**  led **:** **out** std\_logic\_vector**(**7 **downto** 0**)**  **);**  **end** atlys\_remote\_terminal\_pb**;** |

Code Listing 1 - Entity templates for the lab to ensure consistency between student designs.

|  |  |
| --- | --- |
| Remote Terminal Cut Sheet | **Name:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Instructor:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Section:** \_\_\_\_\_\_\_\_ |

**Number of hours spent on this lab:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (no points associated with this unless you leave it blank)

**Suggestions to improve this lab in future years:** (use blank space below)