Development of Methods and Means of Configuration Data Transfer For Use in an FPGA Based Trigger Controller Device

Kelton D. Stefan
Office of Science, Science Undergraduate Laboratory Internship Program

Purdue University

SLAC National Accelerator Laboratory
Menlo Park, California

August 14th, 2009

Prepared in partial fulfillment of the requirement of the Office of Science, Department of Energy's Science Undergraduate Laboratory Internship under the direction of Ronald Akre in the Klystron/Microwave Department at SLAC National Accelerator Laboratory .

Participant: _____
Signature

Research Adviser: _____
Signature

Work supported in part by US Department of Energy contract DE-AC02-76SF00515.

Table of Contents:

Abstract:

To determine if klystrons will perform to the specifications of the LCLS (Linac Coherent Light Source) project, a new digital trigger controller is needed for the Klystron/Microwave Department Test Laboratory. The controller needed to be programmed and Windows based user interface software needed to be written to interface with the device over a USB (Universal Serial Bus). Programming the device consisted of writing logic in VHDL (VHSIC (Very High Speed Integrated Circuits) hardware description language), and the Windows interface software was written in C++. Xilinx ISE (Integrated Software Environment) was used to compile the VHDL code and program the device, and Microsoft Visual Studio 2005 was used to compile the C++ based Windows software. The device was programmed in such a way as to easily allow read/write operations to it using a simple addressing model, and Windows software was developed to interface with the device over a USB connection. A method of setting configuration registers in the trigger device is absolutely necessary to the development of a new triggering system, and the method developed will fulfill this need adequately. More work is needed before the new trigger system is ready for use. The configuration registers in the device need to be fully integrated with the logic that will generate the RF signals, and this system will need to be tested extensively to determine if it meets the requirements for low noise trigger outputs.

Introduction:

Control is a very important part of almost anything going on at the SLAC National Accelerator Laboratory. This certainly holds true for the Klystron/Microwave Department Test Laboratory where new and reworked klystrons are constantly undergoing burn-in and testing routines. In an effort to improve the control system of each station in the test lab, a new digital FPGA (Field Programmable Gate Array) device was built and programmed to replace the old mostly analog systems. This upgrade was to meet the need of determining if klystrons will perform to the specifications of the LCLS project. Precise measurement of phase requires a low noise source. This new FPGA device will fulfill this requirement as well as introduce the possibility of synchronizing multiple test stations so that multiple klystrons can be operated in phase.

The FPGA device will control the various pieces of equipment at a test station using trigger signals. These trigger signals are precisely timed signals that have various parameters (see Figure 2). These parameters include rep rate (repetition rate), pulse delay, pulse width, and time slot. Rep rate is simply the frequency of the trigger pulses measured in Hz. Pulse delay is the time between the start of the signal to the rising edge of the pulse measured in ns. Pulse width is the time between the rising and the falling edges of the signal measured in ns. Finally, time slot is one of several possible clock pulses within a block of the 360Hz signal defined by the desired rep rate. For example, if you require a 60Hz rep rate, there are 360/60 = 6 different time slots you have the freedom to operate on.

Central to the new trigger device is a Xilinx Spartan FPGA (Field Programmable Gate Array) [1] that will be programmed to do everything required for triggering. There are four inputs, three of which we have plans of using. There are eight output channels, all of which are used in the design. These eight channels are divided into a six channel group and a two channel

group. Each group can be operated on a different time slot.

The triggering logic in the FPGA will get its operating parameters from a set of registers. These parameters will be set from a computer. A USB port is used to connect this device to the computer. Computer software was written to interface with the device, so that all operating parameters could easily be adjusted using a GUI (graphical user interface). My assignment was to write the computer software to interface with this device, as well as to write the logic for the FPGA to read and write data through the USB connection and to store and retrieve this data from registers.

The USB interface is facilitated by a small stamp called USBMOD4 [2]. It has an on-board USB interpreter, so that knowledge of the USB protocol is not required. This stamp is bundled with a DLL (Dynamic Linked Library) that can be easily incorporated into a another program. This software had to have several input fields and drop down menus. It allows the user to select the global rep rate, then the time slot for each group of output channels, and finally the delay and width of the pulse trigger for each channel output. When the "Enter" key is pressed, the data in the selected field is sent to the appropriate register. The data is then read back and compared to the original to make sure that there was no corruption during the transfer.

Methods and Materials:

The device itself (See Figure 1) is constructed on a custom board designed at SLAC. The heart of the device is a Xilinx Spartan XC3S500E FPGA [1]. This chip is relatively low cost, yet has more than enough functionality for our current project. The board has voltage regulators and routs power from the input power jack to each active component. The board also connects the Xilinx chip to twelve BNC (Bayonet Neill-Concelman) connections through buffers. Eight of

these twelve ports are output ports, and the other four are input ports. Two of these input ports are for RF signals only as they are magnetically coupled whereas the other two are direct connections. Another key to this device and hence the project was the USB interface. This function was performed by the stamp USBMOD4 by ELEXOL [2]. This stamp is based around the FTDI FT245BM USB FIFO (First In First Out) IC (integrated circuit) [3]. This chip handles the USB protocol, and comes with a DLL [4] that can be integrated into one of several commonly used software programming languages. My use of this device and the DLL [4] in my software made the software programming side of things quit simple.

Aside from the device itself, I had an assortment of other resources available to me for my project. I had all of the equipment in the lab of the second floor of building 44 (Klystron/Microwave Department Test Laboratory) available to me. The tools I used regularly from this lab were a soldering iron, a portable oscilloscope with the related probes and cables, a parallel JTAG (Joint Test Action Group) cable, a small 5VDC power supply, and an assortment of small hand tools such as screwdrivers, precision cutting instruments, etc. Some of these items, the oscilloscope for instance, was relocated to my office for convenience. In my office, probably the one object used most to facilitate the completion of this project was the computer workstation assigned to me. Of course it seems natural that this was the case considering the nature of my assignment. The software applications most used were: Microsoft Visual Studio 2005, Xilinx ISE 11 suite, and Mozilla Firefox.

The online references proved to be of the greatest assistance in this project. I very regularly referred to several different internet sources [1-18] for information regarding the programming languages used (C++ and VHDL) as well as for the technical documents required to adequately understand the hardware I was working with. I referenced several data sheets regularly. These included the data sheet for the Xilinx FPGA [1], USBMOD4 stamp [2], and the

FT245BM USB IC [3]. Aside from the references I used for the hardware pieces I was working with, a reference [4] on the DLL for the USB interface was vital to the rapid development of first console based test software, and then finally to the finished GUI based software.

As mentioned previously, my assignment was to get certain device setting parameters from a GUI on a computer workstation through a USB interface to certain registers on the device and then be able to read these registers back to verify error free data transfer. The methodology of this data movement evolved considerably during the initial work on this project. I considered several options. At first I considered writing all data in one large chunk but finally, I decided that some sort of addressing was the best option. I decided on a method based around five octet words. The first octet would be an address octet and the four octets after would be data octets. At first I was thinking that it would work well to automatically send back to the computer the data that was written to the device. Directly after five octets were written to the device, those same five octets would be written back to the computer without any command from the computer. Although this might work alright, it would not allow the configuration settings to be read back from the FPGA during the software startup. It was decided that the best way to setup the system was to be able to read from or write to any of the four octet registers in the device at any time. To write a register, an address octet and four data octets would be sent to the device. To read from the device, only the address octet would be sent. This idea for the system seemed to be the best at the time, and so I decided to implement it in my design.

To accomplish this required a count selector, an intermediate five octet register, a 128 octet register, and some additional control logic (see Figure 3). Each piece was written in VHDL. The count selector basically just counts to five while selecting different registers to either read from or write to. The intermediate five octet register takes the octet data stream from the USB card and writes it into parallel accessible registers. Finally the big register stores these four octets

in one of 32 four octet divisions. The control logic simply coordinates steps taken by each of these other pieces. In Figure 3, the control logic has been combined with the count selector and labeled “Timing Controller”.

After programming the FPGA to read and write data to the registers in the way described, I wrote the software that the users would use in the test lab on their workstations. This software was written in C++ and used the FLTK (Fast Light Tool Kit) library for the GUI, the FTD2XX DLL for USB communication, and the open-source functions IniReader and IniWriter for working with the INI file that holds configuration data. The GUI (see Figure 5) has several different settings available. There are settings for the rep rate, time slots for two separate groups of channels and the delay and width times. When a user enters the desired settings, the software sends the data to the device and writes it to the INI file for later use. After that, the software sends only the address octets of the previously sent data. The data associated with that address is sent back to the computer for transfer verification. If the received data matches the sent data, the setting entered is displayed to the right of the input box. If the data received does not match what was sent, then an error message will appear in the window warning the user about the fault.

Results:

The efforts put forth for this particular project have been very well rewarded. Although, the objectives themselves evolved slightly as I became more fully acquainted with the obstacles I was facing, I was able to satisfy most all of the initial design objectives.

Using the VHDL hardware description language I was successfully able to program the FPGA used in this project to interface with the USBMOD4 module. The FPGA accepts incoming data from the USBMOD4 module in the form of five octet words, stores the last four octets in

one of 32 four octet registers based upon the first (address) octet, and finally can read back the data to the USBMOD4 module after accepting a single octet address. This set of registers has a synchronization input to enable the synchronization of the registers to the 119MHz clock that clocks the rest of the logic on the FPGA.

I was also able to write the computer software to read and write this register data. I wrote this software in C++ along with the FTD2XX DLL, the FLTK libraries and some open-source INI functions. The code that I wrote myself ended up being ~1000 lines long. It is based around a graphical user interface (see Figures 4 and 5) with several "Widgets" that make up the controls available to the user.

The GUI has a nine option drop-down menu for the rep-rate selection and two value entries for the slot selections for the first and second groups of trigger channels. It has a value entry for the delay and width times of each channel. It even has a channel description text entry for each channel. When values are entered for the delay and width of a channel, these values are converted into start and stop times, rounded to the nearest 8.4ns division integer, and written to the FPGA through the USB port. The software then reads back the values for that channel and displays them under the header "Current Operation". When the software first starts up, all of the settings stored in the FPGA are read and displayed under the "Current Operation" heading and all of the previous time values and channel descriptions are read back from an INI file and put in their respective places. If there is any problem opening the USB communications during the software startup process, a message is presented to the user which gives the options "Retry" or "Quit". After the software is first loaded, if there is any problem with USB communications, an error message is presented to the user that states the presence of a USB error, and the necessity of re-opening the software. The only option is for the user to click the button "Close" at which time the software quits.

Discussion and Conclusions:

When examining the results of this project and comparing them with the objectives at the outset, I think it is reasonable to conclude that I had much success. Although I was not able to test my design with the trigger generating logic to the extent initially desired, I believe that my contributions to this new triggering system for the klystron test stations, will prove to be very beneficial to SLAC in general, and the Klystron/Microwave Department Test Laboratory in specific. Although the logic and software I wrote were aimed at a very specific purpose, they could be very easily adapted to other applications relating to control using digital logic.

Looking back, I can see now that I would do things differently if given the opportunity. One thing I would probably investigate doing differently is the basic method of USB communication. The way I set up the USB communication was that when the computer sends five octets of data, the FPGA assumes that it is supposed to write it to a register block based on the first octet which is treated as an address. If one octet is sent, the FPGA assumes that it is supposed to read back data from the register block. These assumptions forced me to divide the clock frequency of the read/write logic of the FPGA to ~1MHz to reduce the chances of a data under-run. If I were to do the same or a similar project again, I would investigate alternative methods of USB control, and would choose one that would eliminate the need for reducing the clock frequency. Also my VHDL structure could definitely use some improvement. Although it performs the job required of it, the structure is in some respects disorganized. In all, I am using seven different .vhd files. Some of these files are several hundred lines of code whereas some of them are under 50. The logic contained in these files could be more evenly and logically distributed resulting in more maintainable source code.

There are also several improvements that could be made on the computer software that

enables a user to read and write to the FPGA. One thing would be to have the software write and maintain a log file. This file would simply be a record of the settings being entered into the FPGA according to date and time. This would be useful for the operators to be able to look back and see exactly when they or someone else made a change to the operating parameters. Other improvements could be made in the general area of usability. It would be nice to have a button and keyboard shortcut to load all time entries. It would also be nice if when the user tabbed from one input field to the next, that the field jumped from would be written to the FPGA. The error handling could also be improved. Right now, if something goes wrong during a USB read or write operation, the only option available to the user is to close the software and reopen it. Although this is not that much of annoyance, it could be remedied by writing the error messages in such a way as to allow the user to retry instead of simply exiting.

Once again, with the C++ code that I wrote this software in, many structural improvements could be made. I know that in many ways, it is inefficient and not according to good coding practice. Given more time, this code could be re-factored and made easier to read.

I was able to contribute something significant to the Klystron/Microwave Department Test Laboratory. I am confident that my results will be very useful and I am optimistic about this project's future.

Acknowledgments:

Special Thanks to :

U. S. Department of Energy & DOE Office of Science
For providing the *Science Undergraduate Laboratory Internship* program.
National Science Foundation
For funding this internship.
SLAC National Laboratory
For hosting my internship.
Ron Akre
For electing to be a mentor to me, and for giving assistance and direction.
Jeff Olson
For assistance in writing the the VHDL code for the FPGA.
Greg Dalit
For soldering components on my device board, and finding parts.
Darius Grey
For technical and moral support while acting as my partner in this project.

References:

[1] Xilinx, Inc., General Products Division (GPD), "Xilinx DS312 Spartan-3E FPGA Family Data Sheet," [Online Document], 2008 Apr 18 (v3.7), [cited 2009 Aug 10], Available HTTP http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[2] Jason Stolzenberg, "USBMOD4 Datasheet," [Online Document], 2005 Apr 7 (v1.1), [cited 2009 Aug 10], Available HTTP http://www.elexol.com/Download/documents/USBMOD4DS2.PDF

[3] Fred Dart, "FT245BM USB UART (USB - Serial) I.C.," [Online Document], 2005 Nov 30(v1.7), [cited 2009 Aug 10], Available HTTP http://www.ftdichip.com/Documents/DataSheets/DS_FT245BM.pdf

[4] Future Technology Devices International Ltd., "D2XX Programmer's Guide," [Online Document], 2007 Jul 7, [cited 2009 Aug 10], Available HTTP http://www.ftdichip.com/Documents/ProgramGuides/D2XXPG34.pdf

[5] Roger Traylor, "ECE 474 - VLSI System Design," [Online Source], [cited 2009 Aug 10], Available HTTP http://classes.engr.oregonstate.edu/eecs/winter2007/ece474/

[6] "VHDL reference material," [Online Source], 2009 Feb 5, [cited 2009 Aug 10], Available HTTP http://www.csee.umbc.edu/help/VHDL/

[7] Auburn University Department of Electrical & Computer Engineering, "VHDL Tutorial,"

[Online Source], [cited 2009 Aug 10], Available HTTP http://www.eng.auburn.edu/department/ee/mgc/vhdl.html

[8] Sudhakar Yalamanchili, VHDL Starters Guide, 2nd edition [Online Book], Prentice Hall, 2005, [cited 2009 Aug 10], Available HTTP http://users.ece.gatech.edu/~sudha/book/starters-guide/

[9] "Clock divider," [Online Source], 2008 May 10, [cited 2009 Aug 10], Available HTTP http://snipplr.com/view/6173/clock-divider/

[10] "C++ Library Reference," [Online Source], [cited 2009 Aug 10], Available HTTP http://www.cplusplus.com/reference/

[11] Alex, "C++ Tutorial," [Online Source], [cited 2009 Aug 10], Available HTTP http://www.learncpp.com/cpp-tutorial/

[12] F. Costantini, D. Gibson, M. Melcher, A. Schlosser, B. Spitzak and M. Sweet, "FLTK Programming Manual," [Online Manual], 2009, [cited 2009 Aug 10], Available HTTP http://www.fltk.org/doc-1.3/index.html

[13] Robert Arkiletian, "Beginner FLTK Tutorial," [Online Document], 2005 Jan 4, [cited 2009 Aug 10], Available HTTP http://www3.telus.net/public/robark/

[14] "Convert array of char[] to octet[] and vice versa? C++," [Online Forum], 2009 May 18, [cited 2009 Aug 10], Available HTTP http://stackoverflow.com/questions/876213/convert-array-of-char-to-octet-and-vice-versa-c

[15] "Hexadecimal variable help!," [Online Forum], 2008 Feb 27, [cited 2009 Aug 10], Available HTTP http://www.daniweb.com/forums/thread111263.html#

[16] Todd A. Gibson, "Tips and tricks for using C++ I/O (input/output)," [Online Document], 2007 Mar 1, [cited 2009 Aug 10], Available HTTP http://augustcouncil.com/~tgibson/tutorial/iotips.html#prepare

[17] Dean DeFino, "Using Type Casting to "Round Off" a Value in Memory," [Online Document], [cited 2009 Aug 10], Available HTTP http://faculty.salisbury.edu/~dxdefino/RoundOff.htm

[18] Xiangxiong Jian, "A Small Class to Read INI File," [Online C++ Source Code], 2005 Jun 27, [cited 2009 Aug 10], Available HTTP http://www.codeproject.com/KB/cpp/IniReader.aspx

Tables and Figures:

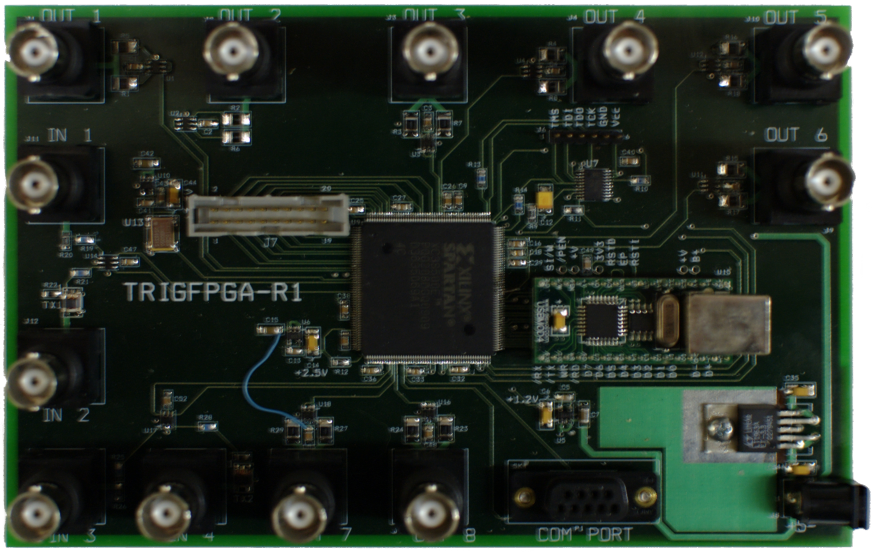

Figure 1: The custom FPGA based trigger device used in this project

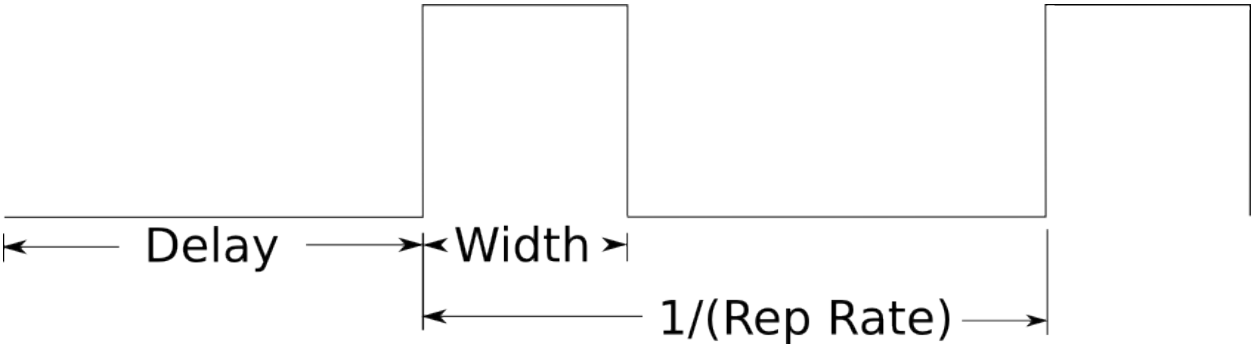

Figure 2: Trigger signal features

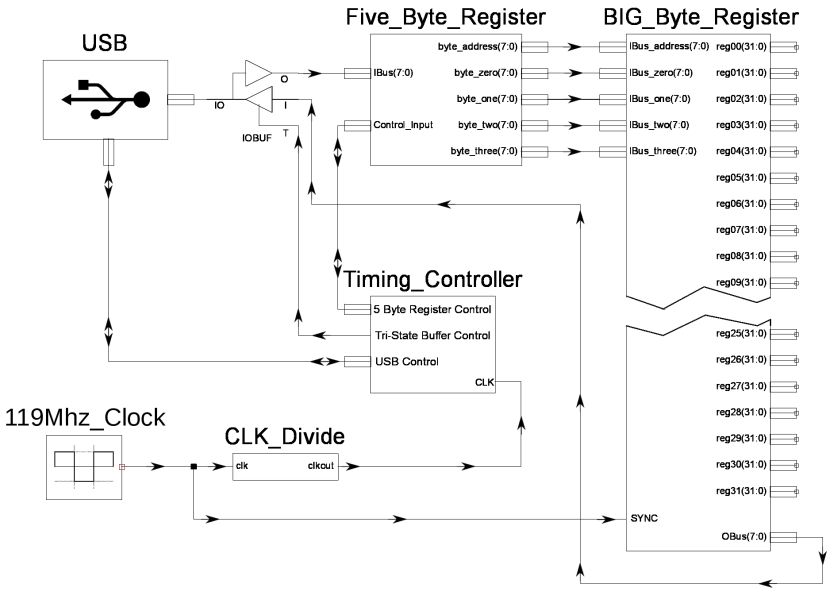

Figure 3: Simplified diagram of data flow from the USB port to the registers

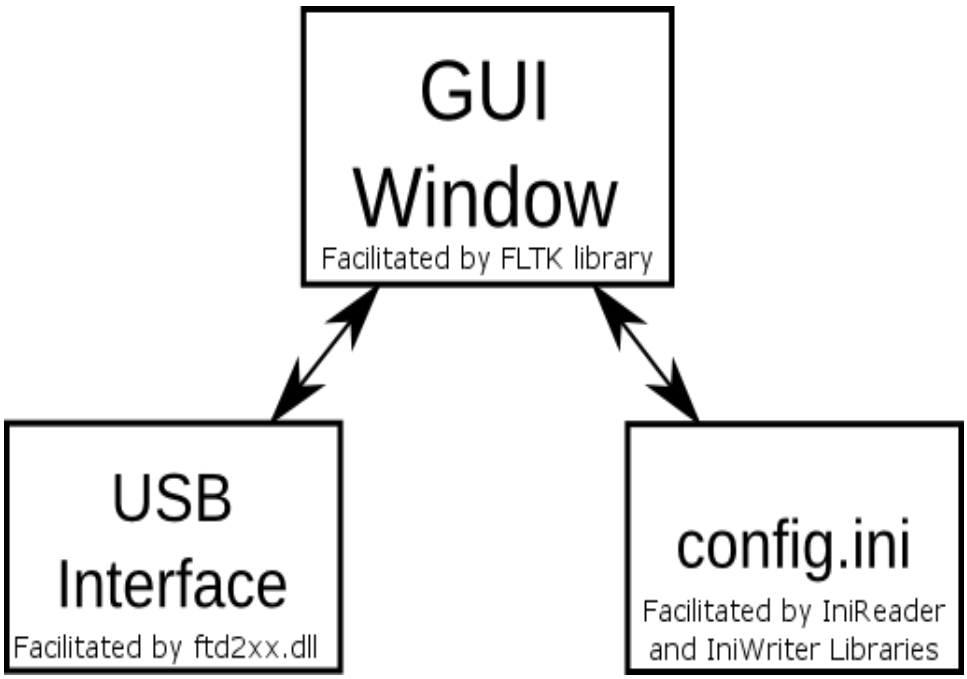

Figure 4: Block diagram showing the features of the Trigger Control software in relation

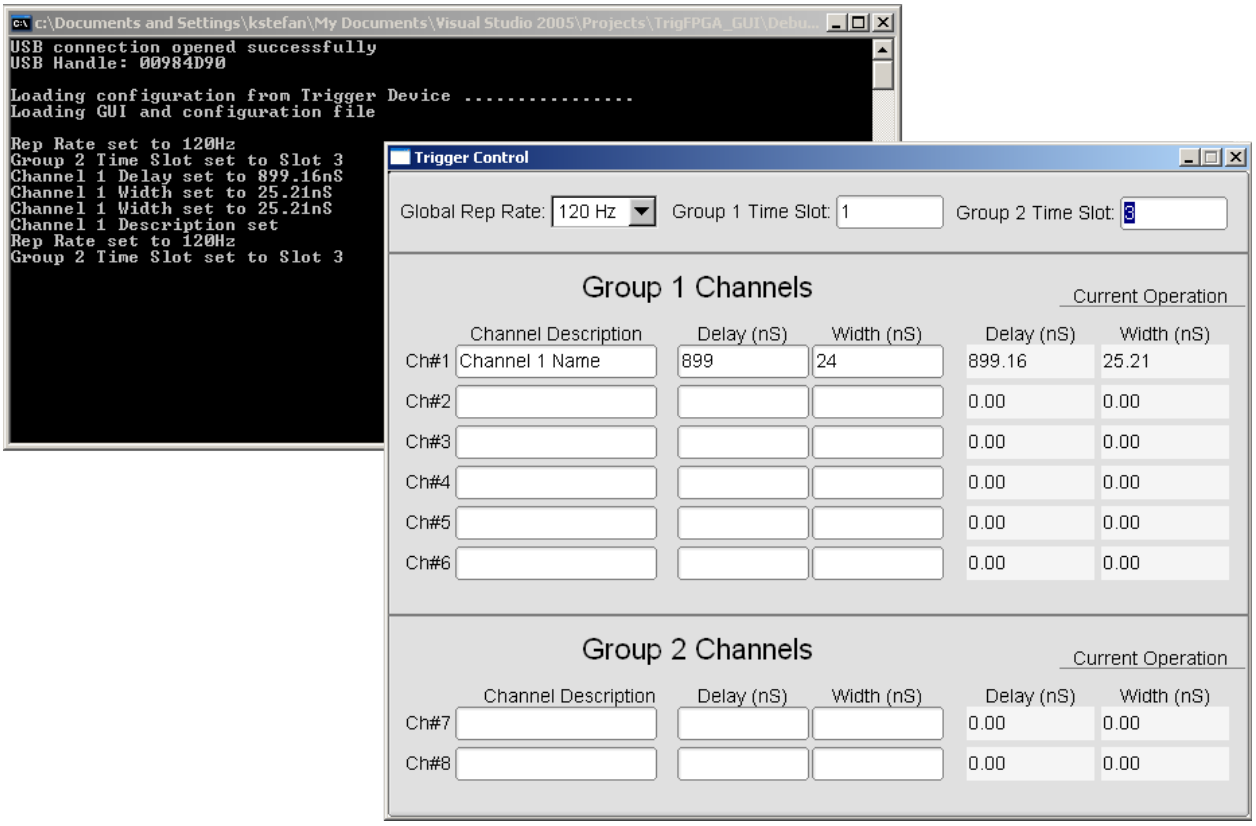

Figure 5: Trigger Control software user interface with the console messages in background