Optimization of the Data Acquisition Software

(PxSuite DAQ)

for the Si-Strip Telescope at the FTBF

Clifford R. Denis
Ramapo College of New Jersey
SIST Intern, Summer 2014
Fermi National Accelerator Laboratory, Batavia, IL 60510
Supervisor: Dr. Lorenzo Uplegger
August 4, 2014

Abstract

The Si-Strip detector was built to provide tracking information at the Fermilab Test Beam Facility (FTBF). The Data Acquisition software (DAQ) called PxSuite was designed by a staff of scientists and engineers in the Detector Instrumentation Group (DIG) of the Scientific Computing Division. The PxSuite software communicates with the Si-Strip telescope located at the FTBF, therefore recording the impacts between strips and particles. To do so, it needs methods able to both interact with the telescope and make sure that the response given by the telescope is right. Throughout this document, I explain how we improved this software, making it both faster and easier to read. After this summer, the PxSuite software can now execute commands faster than before and is easier to read than it used to be.

Contents

1 Introduction

The Si-Strip telescope is a detector used to track particles trajectories. It is located at the Fermilab Test Beam Facility (FTBF). The telescope is made of fourteen Si-Strip planes. The planes are connected to the Data Acquisition (DAQ) hardware based on the CAPTAN (Compact And Programmable daTa Acquisition Node), which is an octagon shaped electronic board created by the Detector Instrumentation Group (DIG) at Fermilab. A station is made of two strip planes placed orthogonal to each other; one of them measures with high precision the x-coordinate of the particle, while the other measures with high precision the y-coordinate, with the beam defining the z-coordinate. The whole system is represented on Fig. 1.

The DAQ software, PxSuite, is a web-based application used to control the CAPTAN hardware and the strip detectors. When particles are passing through the detector, the recorded signals are sent to the CAPTAN, and then the data is sent to the DAQ computer through the network.

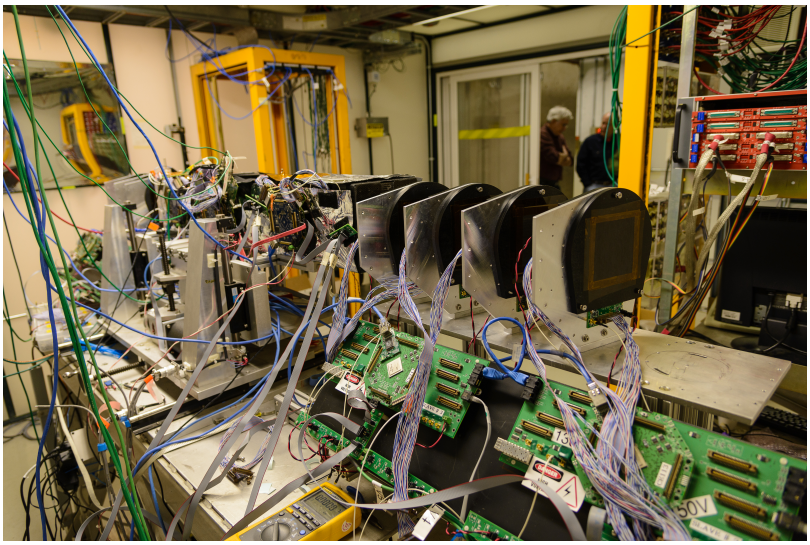

Fig. 1: **Overview of the Si-Strip telescope**

The DAQ software is already there. It has been created and works fine. The problem is that it was slow. Like many other programs, this one has to send commands, lots of them. The waiting time becomes a factor whenever the software has to wait for the hardware to respond to its requests, and this can really slow down the program. Another problem occurs when the data sent is not well understood by the CAPTAN. This can block the software for a while, or create issues further through the program.

Sometimes one might need to change the configuration of the software; disable some strips for testing purpose, or select different sources of the running clock. In order to do so, we need to go to the source code and change the information. Usually this might require other changes and forgetting one of them can cause the program not to compile. Worst case, the program might compile and act weird.

2 System Description

In this section I will describe everything necessary for the telescope to function normally. The telescope is physically made of CAPTAN boards and of tracking stations. The data coming from the telescope is sent to the DAQ computer through a gigabit Ethernet. The Px-Suite is the software that controls all the hardware in this chain. In the next few paragraphs I will explain the architecture and functionality of each of these components.

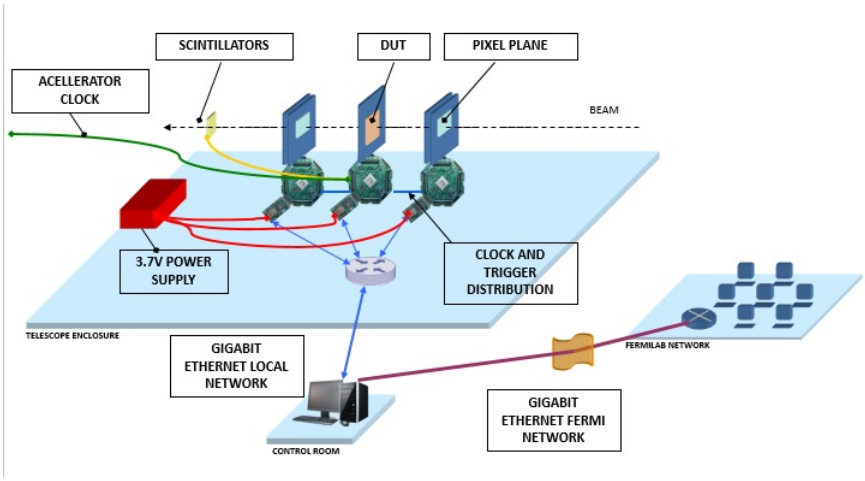

Fig. 2: **The Telescope**

2.1 The Strip Station

Each Si-Strip station (Fig. 4) is made of two Si-Strip planes placed orthogonally to each other in such way that one measures the x-coordinate and the other measures the y-coordinate. Each individual strip is 9cm long and 60 microns wide. The strip plane is made of 639 strips, which makes it approximately 4cm wide. Since the planes are perpendicular, the total usable area of a strip detector is around 4x4cm^2.

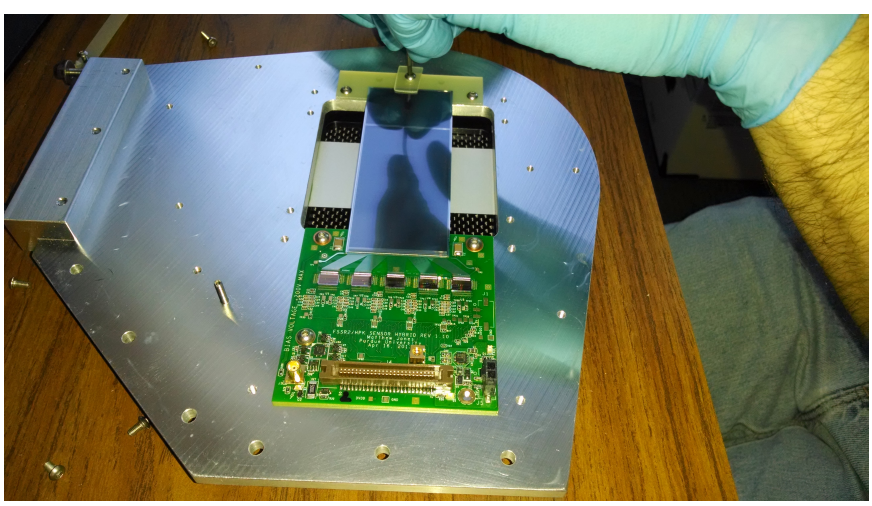

Fig. 3: **The Strip Station**

2.2 The CAPTAN Board

The CAPTAN (Compact And Programmable daTa Acquisition Node) (Fig. 4) is an octagon-shaped electronic board designed for data acquisition, processing and control within High Energy Physics. It is network-based and is made of a series of specifically designed circuit boards. It is very flexible as it allows the user to mount it with other electronic devices (including other CAPTAN boards) for particular and specific use. This property of the board makes it practical for small and large scale applications. Since it is network based, it includes an Ethernet in order to send data through the network.

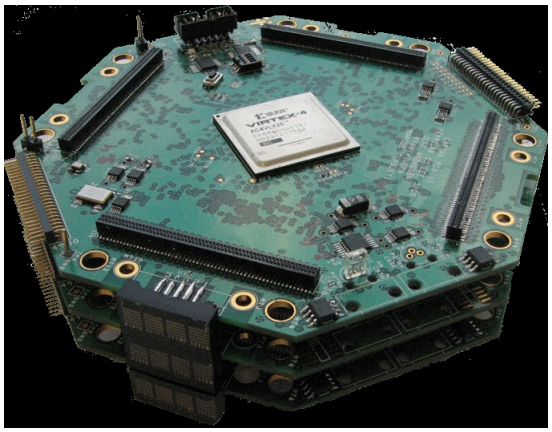

Fig. 4: **The CAPTAN**

2.3 The PxSuite Software

The PxSuite software is the interface between the user and the hardware. The user can interact and program the hardware through a web-based graphical user interface. It is written in C++, HTML and JavaScript.

From the PxSuite, the user can initialize, configure, start, pause, resume, stop or halt the telescope. All these commands are executed by the C++ source code. These same commands call on other functions that are supposed to prepare the telescope for the chosen state. By initializing the telescope, we only turn it on and make sure it is ready to work. Clicking on the Configure button applies all the settings chosen in the .xml files, like the state of the clock and its speed. The Start function does a lot of operations, like enabling the trigger, resetting the BCO, and finally starting the data stream. The Pause button only stops the data stream, while the Stop button resets everything, except the clock source selection. The Resume button only starts the data stream, since everything else should have stayed the same after choosing the Pause button.

2.4 The Configuration Files

There are also xml files that are used to configure the telescope and the DAQ. These files are used by the software whenever the Configure button is clicked. When changing the configuration, the user can set the desired speed he/she would like the clock to run at,

the clock source (internal or external), or even choose which strip should be collecting data. The configuration even has control over the threshold sensibility of the strips, making them capable of detecting background noise depending on the chosen setting.

3 Objectives

The program, although already working, needed to be fixed in order to run smoother. No one ever wants to modify their source code all the time, as one simple change can come with a chain of issues. This is why we needed to make sure that the program could be configured through an .xml file. That way, if there is ever a problem, we will know for sure that it comes from the way the suite was set up. Also, it can be tedious waiting 2-3 minutes for the detector to get configured. It is all about finding a code that is simple enough to make it work faster and better. Coming here, my goal was to optimize the PxSuite software both in terms of accessibility and performance.

In terms of accessibility, I had to make sure that the source code was clear. This includes replacing classes with new ones that can perform more operations, naming newly implemented methods well enough to give the reader a clear idea of their function, and creating intermediate methods that would decrease the confusion a new person would experience with program.

There was more work to be done on the performance level. In the programming world, when we say performance, we mean execution time. I had to make sure that the program would run faster at the end of the summer. To do so, I had to modify a lot of methods essential for the program to run well. To make sure that what we want is actually happening, I had to compare the commands and responses sent between the software and hardware.

4 The Project

One of the lengthiest and most difficult parts of working on a software is getting to know it. It personally took me 3 to 5 weeks to understand how the software works, and get accustomed to its layout and wording. In addition, there was a lot of new things to learn, things that are not taught at school. This all took a long time and by the time I was ready to start modifying the program, four weeks had already passed. However I was well prepared to tackle the program's issues that my supervisor would bring to my attention. These problems, as explained in section 4: Objectives, can be categorized into accessibility and performance issues. I will now detail everything that has been done this summer.

4.1 Accessibility

To make the program more accessible, and easier to understand and use, I had to create and modify a lot of methods. Most of the methods that I created can do specific operations on registers. Each register has all or some of its 32 bits associated to a specific operation.

Bits	Name	Access	Description
31	DCM_RESET	r/w	Reset the clock generator module
30	MCLK_LOCKED	r	MCLK frequency locked
29	BCO_LOCKED	r	BCO frequency locked
28	—		UNUSED
27	STREAM_ENABLE	r/w	Enables data stream over Ethernet
26	SEND_BCO	r/w	Include BCO number in data stream
25	SEND_TRIGNUM	r/w	Include trigger number in data stream
24	SEND_TRIG	r/w	Include trigger data in data stream
23	TRIG_ENABLE	r/w	Require trigger in order to read data
22	BCO_CLEAR	r/w	Resets BCO counter to zero
21	TRIGNUM_CLEAR	r/w	Resets trigger counter to zero
20	FLUSH	r/w	Flushes partially filled buffers
19	BCO_ENABLE	r/w	Enables BCO counter increment
17	FASTBCO	r/w	Doubles maximum possible BCO freq.
16	EXT_BCOCLK	r/w	Selects external BCO clock source
15-11	—		UNUSED
10-8	IDLE_COUNT	r/w	Time to wait before dumping idle packet
7-3	PACKET_SIZE	r/w	Number of 32-bit hit words per packet
2-0	N_CHANNELS	r	Number of channels defined in firmware

Table 1: **Bit Layout for the STRIP_CSR Register**

Table 1 shows the layout of STRIP_CSR, the register I mostly worked on this summer. The table also describes the role of each of the allocated bits.

The register can, depending on the selected bit, specify how many 32-bit words there are per packet, set the time to wait before getting rid of the packets, set the clock source, toggle between the normal and the fast clock frequency, enable the BCO (Beam Crossing number) clock, flush partially filled buffers, reset the trigger counter, reset the BCO counter, enable to trigger for data reading, send the trigger data, the trigger number, the BCO number, enable the data stream over Ethernet, or reset the clock generator. I also implemented methods that can check the currently chosen clock source, and set the frequency depending on the clock source. The registers are in binary codes, so while some functions only affect one bit, other can affect up to 6 bits.

After having methods taking care of basic register operations, I needed to create simpler classes that would handle the communication between the hardware and the firmware (a class that can act directly on the registers). The new and old classes do the same thing, the new ones directly call all the functions they need. As an example, in the FED (Front End Driver) interface, we enable the trigger, reset the BCO, and then enable the stream in the same start function. This puts everything together and makes it easier to locate an error in the program.

4.2 Performance

To increase the performance of a program, we usually need to make sure it can do as much as possible in a very small amount of time. The first step towards this objective was to decrease the excess time due to errors. This involved creating a method that would compare sent and received buffers in order to make sure that the hardware understood the commands sent to it. *compareSendAndReceive* was made a part of the firmware, could access both sent and received buffers and had to verify each one of the bits. In the case where some of these bits would not match, we would just resend the whole buffer until it worked.

Among the methods that I edited to optimize the performance of the software, there was *configureFEC* (Front End Controller) and *configuredetectors*. The first method can now receive the desired frequency and clock source right from the configuration files and pass it to the second aforementioned method, which would set the frequency from this information. To set the frequency, I created a method that can express the frequency as a fraction. The base frequency (66.667MHz for internal clock, 54MHz for external) is always divided by a certain factor, depending on the setting of the fast BCO. If fast BCO is enabled, the base frequency is divided by 4, making it higher than in normal cases (where it is only divided by 8), we must then multiply it by a fraction to obtain the file-inputted frequency. The numerator and denominator of this fraction are used to modify the registers that set the frequency.

The function that creates the DAC buffer was also edited, and this is probably the change that minimized the processing time the most. Like I already mentioned, when the software sends a buffer it needs to wait for a response from the hardware. This response tells us that the command was well-interpreted. Usually, buffers are sent one by one, but there can be a lot of them. This increases the time we have to wait until the whole command is executed. Since a buffer string can store up to around 1000 bits, it does not harm to send multiple commands using the same buffer. *makeDACBuffer* was edited so that one buffer string would take a many commands as possible. This meant less waiting time, so a better performance from the PxSuite.

While running, the PxSuite might not have the same information as the hardware. The information from the hardware (the CAPTAN) is the right one, since the CAPTAN is the one doing the real job. Thus we must synchronize the software with the CAPTAN often. Before every important step, such as starting the data acquisition, we would just look for a response from the hardware. To do so, we just send the current register, and read in the buffer that is returned. From this buffer, we can create a new register that will therefore assure that the software and hardware are working on the same thing. The method that executes all of the above was also used to check the validity of the returned value, since sometimes the hardware can misinterpret commands.

5 Results

The PxSuite is now much faster and the source code is easier to read. There are certainly more methods than before, but they help the programmer read the code easily. Sometimes it is better to add intermediate methods that can be used almost everywhere in the program than have to re-write them all the time. This saves the programmer time, and makes it easier to avoid mistakes that can take hours to fix.

The program can now be configured through the .xml files and no longer requires someone to go into the code and edit variables. This is really important as it makes testing much easier. In addition, there is no more need to re-compile the program every time a change is made.

6 Discussion and Conclusions

The software still has a lot of things that need to be worked on. I started the implementation of methods related to the direct manipulation of bits from other registers, such as STRIP_RESET, STRIP_SC and STRIP_ANALYSIS_CSR. The following tables give a layout of the bits for each of these registers.

Bits	Name	Access	Description
31	CHIP_RESET	w	Assert RESET signal on selected channels
		r	Indicates that RESET signal is asserted
30	CLEAR_FIFO	w	Clears link spy FIFO's
		r	Status of FIFO clear signal
29	CLEAR_ERRORS	w	Clears link error counters
		r	Status of link error counter reset signal
28	LINK_RESET	w	Causes the selected links to realign and synchronize
		r	Status of link reset signal
27	DAC_RESET	w	Asserts the reset signal to the DAC's (not masked)
		r	Status of the DAC reset signal
7...0	CHANNEL_MASK	r/w	Reset signals are sent to all channels with bits set in the CHANNEL_MASK field

Table 2: **Bit Layout for the STRIP_RESET Register**

Bits	Name	Access	Description
31	SC_BUSY	r	Set while a slow controls transaction is in progress
	INVOKE	w	Initiate the slow controls transaction
30	RAW	r/w	Set when raw data is being shifted out
29	BCO_SYNC	r/w	Synchronize rising edge of SHIFT with BCP zero
28	BCO_ZERO	r/w	Synchronize falling edge of SHIFT with BCP zero
26-24	LENGTH	r/w	Encodes the number of bits to send
23-16	CHANNEL_MASK	r/w	Set bits enable the channels to receive SCIN
15-13	READ_SELECT	r/w	Select channel from which to receive SCIN
12-10	INSTRUCTION	r/w	Slow-controls register instruction
9-5	ADDRESS	r/w	Slow-controls register address
4-0	CHIP_ID	r/w	Target chip ID

Table 3: **Bit Layout for the STRIP_SC Register**

Bits	Name	Access	Description
31	DONE	r	Indicates that the terminal BCO count has been reached
30	CLEAR	r/w	Resets all counters - not self clearing
18...16	CHANNEL_SELECT	r/w	Selects strip sensor channel for analysis
15...11	TERM_COUNT_BIT	r/w	$\log_2$ of BCO count at which to stop analysis
8...4	SET_NUMBER	r/w	Set number to match in data stream
3...0	STRIP_NUMBER	r/w	Strip number to match in data stream

Table 4: **Bit Layout for the STRIP_ANALYSIS_CSR Register**

Should the operations on these registers made easier and more readable in the future, the PxSuite would perform a lot better than it actually is.

7 Acknowledgments

Finally, I would like to express my gratitude to all of Fermilab's staff who has welcomed me here at Fermilab. First of all credits to Dianne Engram, Elliott McCrory and Linda Diepholz who have helped fund this internship and taken great care of us the duration of the whole program. My appreciation also goes to my supervisors Lorenzo Uplegger and Ryan Rivera without whom I would not have been able to complete this project. Their advices, help and great personalities helped me go through all the obstacles I have faced this summer; I have learned a great deal with them this summer and I hope to work with them again in the future. My final thanks go to my mentor Gustavo Cancelo for being ready to help at any time.

References

[1] *Strip telescope Interface Board Firmware.* November 11, 2013.

[2] Simon Kwan, CM Lei, Dario Menasce, Luigi Moroni, Jennifer Ngadiuba, Alan Prosser, Ryan Rivera, Stefano Terzo, Marcos Turqueti, Lorenzo Uplegger, Luigi Vigani. *The CMS Pixel Tracking Telescope at the Fermilab Test Beam Facility.* Fermi National Accelerator Laboratory, Batavia, IL, USA. Istituto Nazionale di Fisica Nucleare, Sezione di Milano Bicocca, and Universit degli Studi di Milano Bicocca, Piazza della Scienza 3, 20126 Milano, Italy.

[3] Marcos Turqueti, Ryan Rivera. *An overview of the CAPTAN network based pixel telescope readout architecture and data processing software state.* May 19, 2010.

[4] A. Kumar, S. Kwan, A. Prosser, R. Rivera, M. Turqueti, L. Uplegger. *CMS pixel telescope at MTEST.*