SOFTWARE FIRMWARE INFRASTRUCTURE FOR LLRF4 BASED SYSTEM*

G. Huang†, L. R. Doolittle, C. Serrano, LBNL, Berkeley, CA 94720, USA

Abstract

LLRF4 is a successfully designed FPGA based low noise signal processing board. The board has been used in several accelerators as a low level RF controller and timing system controller. The complexity of maintaining and supporting different versions of software and firmware increase as the number of application increases. This paper describes our attempt to abstract the software and firmware layer of support for LLRF4-based systems. For the software side, the infrastructure includes an FLTK based GUI, as well as providing an EPICS IOC driver. From the firmware side, the infrastructure separates board hardware dependent drivers, the common algorithm implementation, and project specific DSP. We also reserve the capability to expand to a UDP-based communication protocol for the next generation LLRF board.

INTRODUCTION

LLRF4 was originally designed in the context of ILC, expanding on ideas developed for the SNS LLRF system in 2005 [1,2]. The hardware design has been stable for several years. The board has demonstrated low phase noise and adequate data processing capability for a variety of applications. Known applications include accelerator low level RF control and timing system control. A software and firmware design platform is developing for LLRF4 board to make it easier to support different applications.

Software runs on the host computer, and the firmware runs on the LLRF4 board, as shown in Figure 1. The board communicates with the host computer through a Hi-speed USB 2.0 connection. A common register map file is used on both sides to keep the communication consistent.

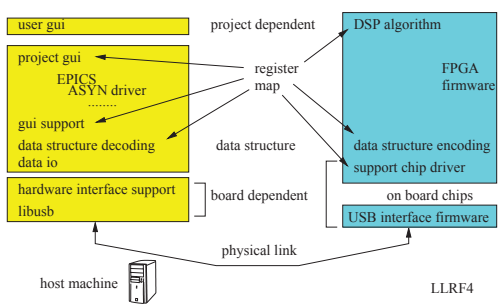

Figure 1: Software and firmware structure.

SOFTWARE STRUCTURE

The software side code contains three layers:

- The bottom layer is the board-dependent layer and implements the USB communication. The `libusb` library and the `usrp_usb library` wrap all the hardware dependencies and provide a transparent data communication interface.
- The intermediate layer receives data from the USB port and decodes the data encoded by its analog intermediate layer in the FPGA firmware. The output of this layer provides fundamental graphical user interface support, which consists of one development application interface, and two machine generated debugging interfaces. The first debugging interface is a FLTK-based interface called xgui, the second debugging interface is an EPICS IOC driver and a EDM screen.
- The top layer in the software side are user GUIs: end users need to develop the interface for daily operation and diagnostics. Some critical system level control registers should be hidden in this layer.

Software Class Description

The bottom and intermediate layers of the software side are coded in c++. The class design is shown in Figure 2.

The bottom layer is the class "usbio" which deal with the hardware read and write by calling usrp_basic functions and further calling libusb functions. This hardware read and write are run in a separate thread.

In order to match the usb read and memory map process speed, a bigger memory contain multiple pages are used, and a separate thread is used to assign the memory to be read, to be process. This is handled by rx_buf class.

The intermediate layer contains the mem_map class and an application interface "proj".

For control data from host to board, the application interface direct call the methods in usbio class and write to the hardware.

For the data acquired from board to host, the data is processed in the mem_map according to a the defined data structure. The application interface prepare register values and waveforms ready for upper layer to display.

The usb io is not always read from the start point of a "page"(will be explain later), the memory to be process has to be at least two usb read pages. In the memory map process, we need to check the integrity of the page before further process.

The results of memory map process are separate registers and waveforms. Each project have it's own list of

* Work supported by U.S. Department of Energy under contracts DE-AC02-05CH11231.

† ghuang@lbl.gov

ISBN 978-3-95450-115-1

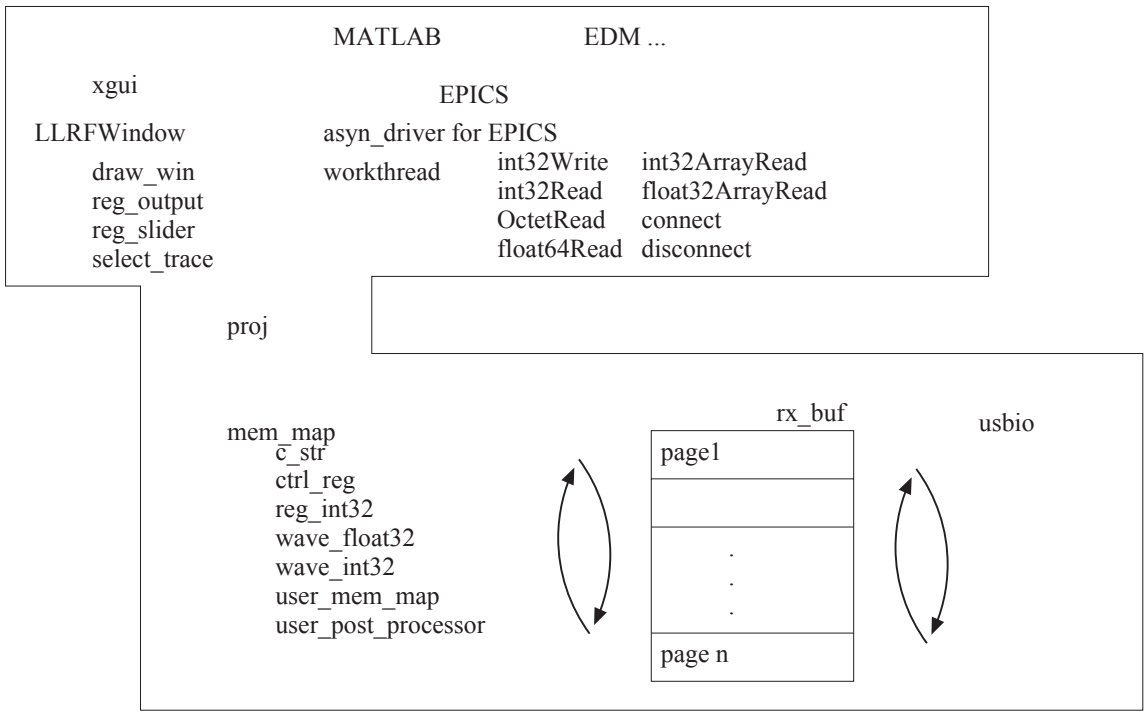

Figure 2: Software class description.

registers and waveforms, which will be defined in the human readable register map file and convert into a piece of c++ code and included in the memory map class. Also a set special functions can be defined in project space called user_postprocessor and user_postprocessor_caller. These functions are used to allow user define their own post processor to process the registers and waveforms in the memory maps and output them to an other registers or waveforms. All the internal elements of the memmap class are accessible from the user_postprocessor.

The application interface wrap the usbio, rx_buf and mem_map together, it open a thread for each of them. A single call of start_usb_process will start all three threads. Other functions in the application interface are update value for each kind of data type and request hardware register writes.

Two kinds of debug GUI are provided with the software structure. One is a FLTK [3] based graphical user interface, called xgui. The other is an EPICS driver based on AsynDriver together with an edm screen. Both GUIs are just a list of the widgets based on the register map provided through the application interface.

FIRMWARE STRUCTURE

The firmware side can be divided in to pieces: the USB interface chip firmware and the FPGA firmware. The USB interface chip (CY7C68013A) firmware and corresponding software usrp_usb library are inherited from the GNU radio project [4] with some modifications (Figure 3).

The FPGA firmware code also contains three layers:

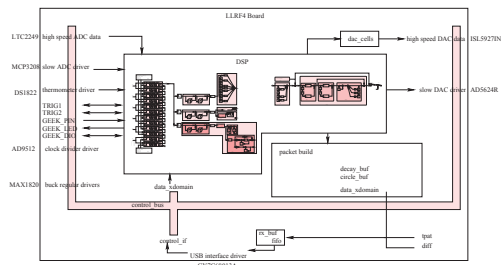

Figure 3: Firmware structure.

- The bottom layer is the board-dependent layer. In this layer, we implement all the hardware driver. The USB interface is converted into a local bus to implement data IO. The power regulator, thermometer and frequency divider drivers are run on USB clock domain, so they do not rely on the DSP implementation. The slow ADC and slow DAC can be run on either USB time domain or DSP time domain.
- The intermediate layer is the data structure designed for the host computer to decode. The LLRF4 communicate with the computer through USB port. The USB data structure we are using now is described below.
- The top layer is a hardware-independent DSP module implement the APEX-specific control algorithm.

USB Data Structure

The bottom layer of software and firmware provide the transparent data communication. The intermediate layer encodes and decodes the USB data structure. The data for-

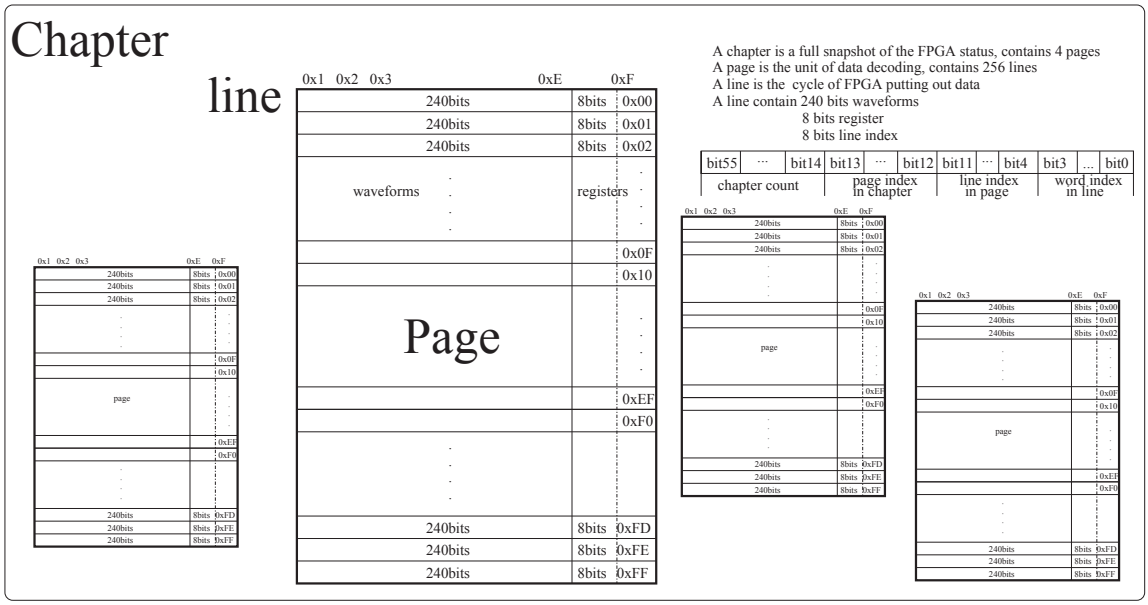

Figure 4: USB data structure.

mat used is illustrated in Figure 4. All the data cross domain happen here.

The unit of USB communication is 16-bit words. A data line contains 16 words, the first 15 are filled with waveform data, the upper byte of the last word is used for register data, and the lower byte of the last word is the line counter of the page. A page is the unit for the host side USB read and process. A page contains 256 lines, and the USB read page does not necessarily start from the line index 0x00. The software then needs to handle the difference of USB read page, find out the start line of each page, and only then send it to process. A chapter is the unit for a full snapshot of FPGA status including a mirror of control registers. A chapter contains 4 pages, Each line has 8 bits for register, so each page has 1024 bits for register and each chapter has 4096 bits for register. To index the register, we use the bit address of this 4096 bits to accommodate different register widths.

In the FPGA firmware, we use a counter address in the data stream. Bit 0 to bit 3 of that counter are the word index in a line, bit 4 to bit 11 are the line index in a page, bit 12 to bit 13 are the page index in a chapter, bit 14 and above are the chapter counter.

The register in line 0x0x and 0xFx area of each page are used for the page information, including the page ID, checksum, firmware version information, etc. The rest of area is left for the user to define. Again the user register map is defined in the register map file which is used in both software and firmware as mentioned earlier.

SUMMARY AND FUTURE PLAN

We have developed a software and firmware development platform for LLRF4 board based system. Both software and firmware are separated into different independent layers.

This infrastructure is applied to some of our ongoing development projects, including APEX [5] and SPX@APS [6]. We also plan to merge some of the previous developed firmware to this platform to increase the maintainability.

Furthermore, we plan to expend this platform to accommodate other boards and other communication protocols, i.e. UDP. and hopefully use this same platform for other boards we are developing.

REFERENCES

[1] L. Doolittle, “Conceptual design for ILC LLRF hardware”, LLRF2005.

[2] L. Doolittle, H. Ma, M. S. Champion. “Digital low level RF control using non-IQ sampling”, LINAC2006.

[3] http://www.fltk.org

[4] http://gnuradio.org

[5] G. Huang, L. Doolittle, K. M. Baptiste, F. Sannibale, J. Byrd “LLRF Control Algorithm for APEX”, IPAC2012.

[6] G. Huang, L. Doolittle, K. Campbell, J. Greer, J. Byrd, T. Berenc, H. Ma (ANL), “LLRF Control for SPX @ APS Demonstration Experiment”, IPAC2012.

ISBN 978-3-95450-115-1