Layer 1 CaloTrigger Online SW Tutorial

Bucky Badger

UW Madison

September 30, 2013

How To Make This Document

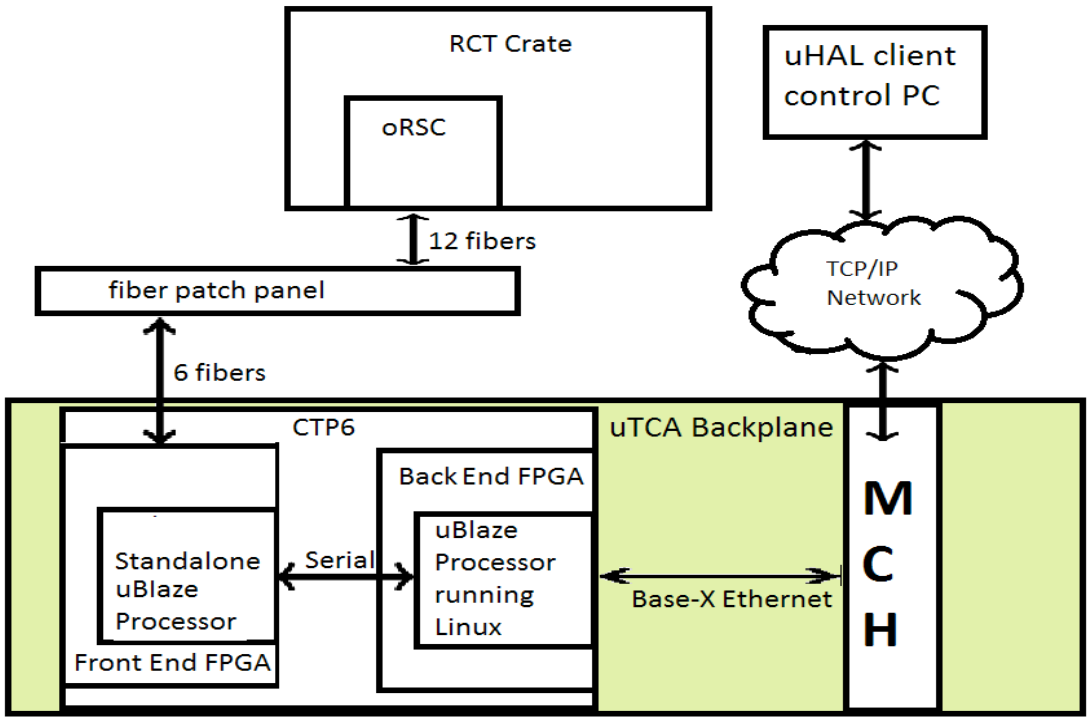
```
git clone git@github.com:uwcms/cms-calo-layer1.git
cd cms-calo-layer1/docs
make
```

Introduction

- This document gives a rough guide to the architecture of the Layer 1 CaloTrigger Online SW
- Outline
  - oRSC and CTP Architecture
  - IPBus
  - Our "soft" implementation of IPBus
  - VME and SPI byte-stream abstractions
  - Xilinx workflows
  - Existing standalone packages
  - Using XMD to upload and debug SW
  - ToDo list

Reminder: CTP6/oRSC Architecture

- CTP and oRSC have two FPGAs:
  - "Front end" is connected to the optical links
  - "Back end" is connected to the world (via VME or Ethernet)
  - FPGAs are connected together via serial bus: UART (CTP6) or SPI (oRSC)



Microblaze & Petalinux

- All of our FPGAs have "Microblaze" core
- This is an (ARM) CPU implemented in HDL
- On the CTP6 back end FPGA, Linux is installed, so you can SSH in, run multiple programs, etc
- On all other FPGAs are "standalone". This means you write a C program with a single `int main(void) {..}` function with an infinite loop and this runs everything on the chip

IPBus Protocol

- Common CMS protocol for interacting with Trigger HW
- Basically a specification for interpreting "transactions" from a byte stream:
  - READ *n* words (32-bit) from an address in device memory
  - WRITE *n* words (32-bit) into an address
  - READ-WRITE-MODIFY at word at an address
- Spec: https://github.com/uwcms/cms-calo-layer1/raw/master/docs/ipbus_protocol_v2_0.pdf
- Below: the specification for a READ request/response

3.2 Read transaction (Type ID = 0x0)

Request

	31 24	23 16	15 8	7 0	
Word 0	Version = 2	Transaction ID	Words = READ_SIZE	Type ID = 0	InfoCode = 0xf
Word 1	BASE_ADDRESS				

Response

	31 24	23 16	15 8	7 0	
Word 0	Version = 2	Transaction ID	Words = READ_SIZE	Type ID = 0	InfoCode = 0
Word 1	Data read from BASE_ADDRESS				
Word 2	Data read from BASE_ADDRESS + 1				
...	...				
Word n	Data read from BASE_ADDRESS + (READ_SIZE - 1)				

IPBus in C: softipbus

- Other groups have an HDL core to serve IPBus over UDP
- We have Microblaze/Petalinux, so we can write C(++) code
- (and can use the reliable TCP protocol for transport)
- Our "softipbus" implementation is in CACTUS SVN
  - functions to de-serialize a byte stream into IPBus transactions
  - functions to handle IPBus transactions (READ/WRITE/RWM)
  - functions to serialize the responses into a byte stream
  - a TCP server which runs on Linux and interprets the byte stream as IPBus
  - a TCP server which runs on Linux and forwards the byte stream along a file descriptor
- The code is written so it can run on Linux as a server or used as a library on the standalone devices

IPBus Clients

- The IPBus client μHAL is made by L1SW group
- C++ and Python bindings
- Device memory addresses are mapped to nice names via an XML file

```
<!-- Optical link capture rams -->
<node id="MGT0" address="0x60000000" size="1024"
    mode="incremental" permission="r"
    description="Capture RAM for link #0"/>
<node id="MGT1"...
```

- CTP6 FE map in CACTUS SVN: `cactuscore/softipbus/etc/ctp6_fe.xml`
- CTP6 Python Control Client: CTP6Commander package

The General Principles

Put a softipbus interpreter
to perform tasks on each FPGA

IPBus packets move in a stream of bytes

Abstract HW interfaces (VME, SPI, UART) into functions which transport bytestreams

Bytestream Abstractions

- We want to allow IPBus clients to connect to each FPGA
- Start with TCP server, which receives a bytestream over ethernet. This has to live on Linux.
  - TCP server on CTP backend
  - TCP server on "VME PC" connected to the oRSC's crate
- Transport the stream from the server to each FPGA:
  - CTP6 BE - easy, already there (done)
  - CTP6 FE - forward stream from BE server over UART `/dev/tty` (done)
  - oRSC BE - "VMEStream" abstracts stream over VME read/write
  - oRSC FE - VMEStream $\rightarrow$ oRSC BE $\rightarrow$ FE over SPI ("spi-stream")
- Put an IPBus interpreter on each standalone FPGA

VMEStream

- VME is protocol to READ/WRITE to address in crates - not streaming
- Goal: transparent continuous bidirectional bytestream over VME
- Linux VME PC interface:
  - named pipes on filesystem: `mkfifo txbuffer rxbuffer`
  - forward data to oRSC `./vme2fd txbuffer rxbuffer`
  - `echo "Go badgers" > txbuffer` sends "Go badgers"
  - `cat rxbuffer` prints oRSCs response to stdout

to forward IPBus over VME:

- `softipbus-forward txbuffer rxbuffer` spawns a IPBus TCP server on 60002 and forwards packets to oRSC
- exact same as on CTP6 BE→FE UART link
  `softipbus-forward /dev/ttyUS0 /dev/ttyUS0`

VMEStream Protocol

- Two RAMS and two registers in memory of oRSC BE
- Read/write on VME PC via CAEN controller
- Read/write in oRSC BE standalone code directly from memory
- Each RAM/register corresponds to a data flow direction
- When sender has data to send, check if register value is zero
- If zero, load n words into RAM, set register to n
- Receiver checks of register is > 0 - if so:
  - Reads n words of data from RAM
  - Sets register to zero

SPIStream

- oRSC backend and frontend are connected with a SPI
  - Serial Peripheral Interface
  - Master (BE) & Slave (FE) devices
  - Master always initiates the data interchange
  - Data size is fixed (by Mathias, 512 words)
- https://github.com/uwcms/spi-stream
- Master continuously initiates exchanges to ensure slave's data is read out

Xilinx Workflows

- Three components of a standalone program:
  - HW description XML file - describes HDL devices, memory, etc, built by Mathias
  - Board Support Package (BSP) - generated by Xilinx SDK from HW description XML file. Contains driver header files and libraries for the devices built into the FPGA HDL.
  - Your standalone program. `#includes` and links to the BSP
- You need to use the Windows (Linux version sorta sucks) Xilinx SDK GUI to generate new BSPs when the HW XML is updated.
- All other standalone programs are Makefile-ized to prevent tears of blood

Existing Standalone Programs

- The BSPs, HW, and standalone programs are in Github:
- https://github.com/uwcms/cms-calo-layer1

common	Trying to merge spi-stream with history	2 hours ago
ctp6_be_hw	Add HW + FE BSP. We need some other options in Xilinx to get the linu…	3 months ago
ctp6_fe_bsp	Set debug modules correctly	2 months ago
ctp6_fe_hw	Fix bug in BRAM size	2 months ago
ctp6_fe_uart_echo_test	More Makefile DRY	a month ago
ctp6_fe_uart_ipbus	Fix nondescript filename	a month ago
docs	Add IPBus protocol	2 hours ago
orsc_be_bsp	Add oRSC HW and BSP	3 months ago
orsc_be_hw	Add oRSC HW and BSP	3 months ago
orsc_be_spi_echo_test	Add oRSC BE spi echo Makefile	2 months ago
orsc_fe_bsp	Add oRSC FE HW + BSP	3 months ago
orsc_fe_hw	Add oRSC FE HW + BSP	3 months ago
orsc_fe_ipbus	Add copied-over IPBus implementation on the ORSC FE	2 months ago
orsc_fe_spi_echo_test	Move stuff into common Makefile	2 months ago

Uploading/Debugging via XMD

todo.

Outstanding Tasks

- Update oRSC frontend BSP?
- Validate VMEStream with echo test on oRSC backend
- Validate SPIStream with echo test between oRSC BE and FE
  - Code written, needs new BE bitfile(?)
- Validate IPBus to oRSC FE
- Port Tom's bitbanger oRSC backend clock config standalone code
- Define (with Mathias) the memory maps for CTP BE, oRSC BE & FE
- Determine what functionality is needed in Supervisor and implement
- Write interface for pattern test in oRSC