PFS Control Module Documentation
Deliverable for Caltech

October 6, 2017

Prepared by: Keith Rumney

JPL
Jet Propulsion Laboratory
California Institute of Technology

Table of Contents

1. Control Module Overview

2. FPGA Block Diagram

3. Source Code File Structure

4. FPGA Module Descriptions

5. C File Descriptions

6. Detailed Descriptions:
      a. TCP/IP + Ethernet
      b. Watchdog
      c. Board Counting
      d. Tracking Busy Boards
      e. Debug modes & UART port
      f. LED error flags

**7. Tools Used**

8. Firmware Parameters

1. Control Module Overview

The control module breaks down and implements commands sent over Ethernet into commands that are sent to driver boards over 6 Flex interfaces called sectors. The hardware consists of a Xilinx KC705 Development board which uses a Kintex 7 FPGA, and a custom designed daughtercard that controls the power supplies and provides communication via the flex connector for the 6 sectors. Each sector supports 14 driver boards chained together to support a total of 6*14 = 84 driver boards.

The control module is responsible for various commands defined in it's ICD with MPS (Movement Planning Software). These include SetFrequency, Run, Calibrate, Power, Diagnostic, and Housekeeping. To implement these the control module needs to send the relevant commands to the relevant boards, in some cases reading back status or in other cases driving motors and sending a response packet over Ethernet when the status is received or the motors completed their drive. For more information on these commands refer to the "PFI MPS Electronics ICD" document, most recently in Revision G.

The control Module FPGA design consists of primarily two blocks, one is the Xilinx provided IP for a Microblaze soft processor that runs C software, and the other is a block that handles control and communication for all 6 sectors and connects to the Microblaze via the Axi Lite Bus. There is also a USB connection for UART which prints out text to describe the operation of the Control Module and can come in handy for debugging.
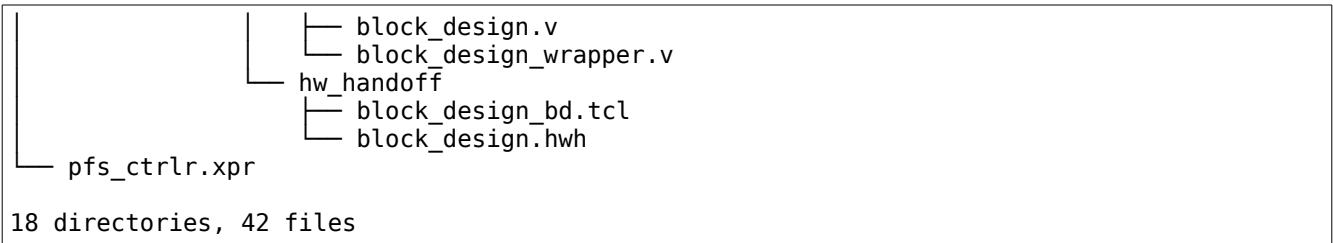
2. FPGA Block Diagram
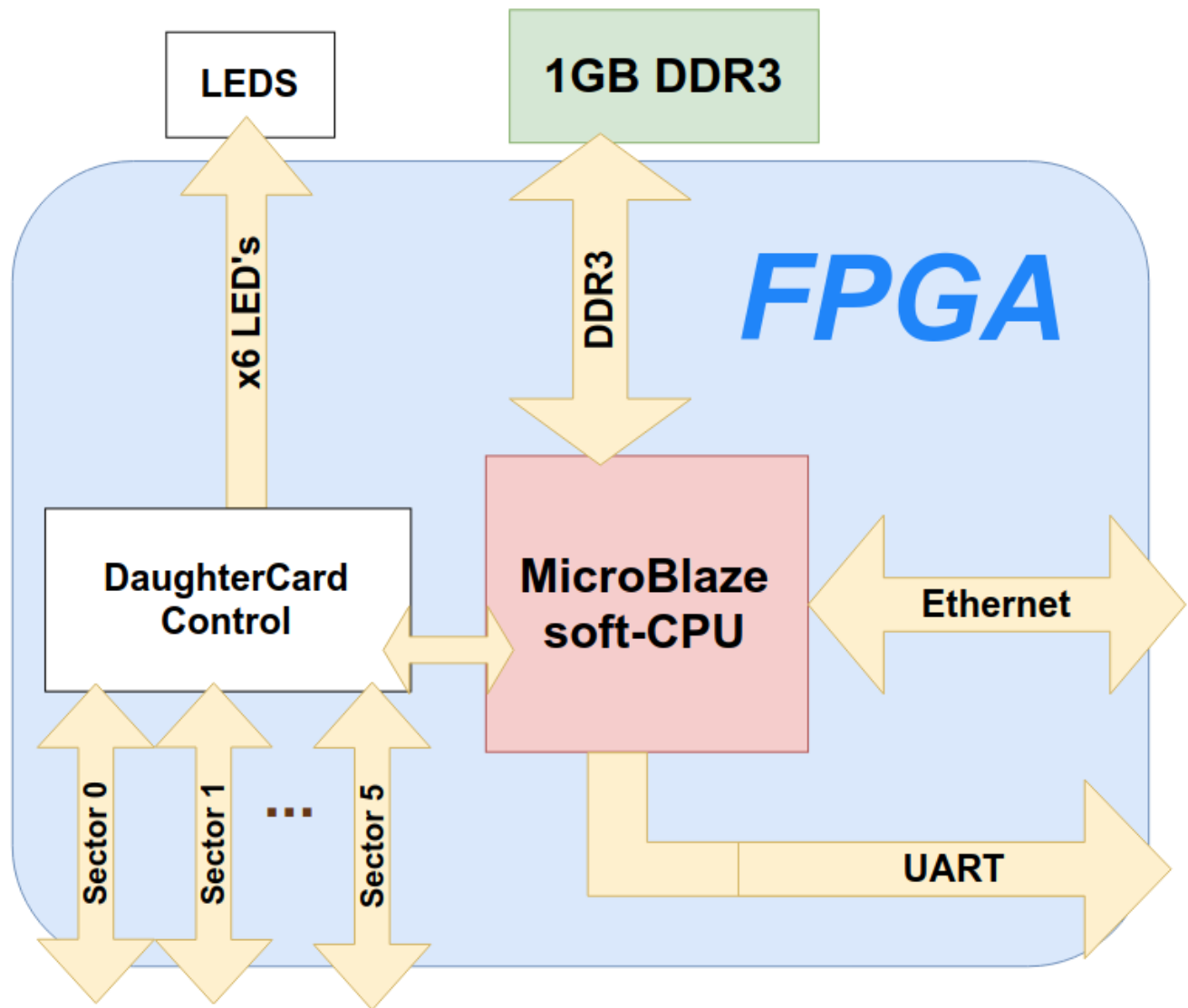
3. Source Code File Structure

File tree:

```
.
├── pfs_ctrlr.ip_user_files
│   ├── pfs_daughtercard
│   │   ├── component.xml
│   │   ├── deserialize.sv
│   │   ├── fifo.sv
│   │   ├── pfs_daughtercard.sv
│   │   ├── pfs_sector.sv
│   │   ├── serialize.sv
│   │   └── xgui
│   │       └── pfs_daughtercard_v1_0.tcl
│   └── simple_ip
│       ├── component.xml
│       ├── simple_ip.sv
│       └── xgui
│           └── simple_ip_v1_0.tcl
├── pfs_ctrlr.sdk
│   ├── block_design_wrapper_hw_platform_0
│   │   ├── block_design_wrapper.bit
│   │   ├── block_design_wrapper.mmi
│   │   ├── download.bit
│   │   └── system.hdf
│   ├── eth_bsp
│   │   └── system.mss
│   └── pfs_ctrlr
│       └── src
│           ├── lscript.ld
│           ├── pfs.c
│           ├── pfs_eth.c
│           ├── pfs_eth.h
│           ├── pfs_globals.h
│           ├── pfs_high_api.h
│           ├── pfs_low_api.h
│           ├── pfs_queues.c
│           ├── pfs_translate.c
│           ├── pfs_translate.h
│           ├── platform.c
│           ├── platform_config.h
│           ├── platform.h
│           ├── platform_mb.c
│           ├── project.c
│           ├── queue.c
│           └── queue.h
├── pfs_ctrlr.srcs
│   ├── constrs_1
│   │   └── new
│   │       ├── system_no_swap.xdc
│   │       ├── system.xdc
│   │       └── timing.xdc
│   └── sources_1
│       └── bd
│           └── block_design
│               ├── block_design.bd
│               ├── block_design.bmm
│               ├── hdl
```

```
                         ├── block_design.v
                         └── block_design_wrapper.v
                  └── hw_handoff
                         ├── block_design_bd.tcl
                         └── block_design.hwh
  └── pfs_ctrlr.xpr

18 directories, 42 files
```

This file structure follows the standard format created by the Xilinx Vivado toolkit.

pfs_ctrlr.ip_user_files/pfs_daughtercard/
- *.sv – System Verilog source code that connects the daughter-card to the Microblaze CPU, described further in section 4
- others: files generated by Vivado to package the source code into a re-usable IP

pfs_ctrlr.ip_user_files/simple_ip/
- all: A test IP that was created for debugging and left in the design

pfs_ctrlr.sdk/eth_bsp
- system.mss – board support package for the software, specifying settings for drivers

pfs_ctrlr.sdk/pfs_ctrlr/src/
- lscript.ld – Linker Script defines the sections of memory, maintained by Vivado
- *.c, *.h – Source code running on Microblaze software for the application, explained further in section 5

pfs_ctrlr.sdk/block_design_wrapper_hw_platform_0/
- block_design_wrapper.bit - Programming File imported from Vivado, runs microblaze boot-loop
- block_design_wrappper.mmi - The Memory mapping file, imported from Vivado
- download.bit - Created by Vivado SDK, this is the sole programming file which must be loaded into BPI Flash. It holds the software application in block ram.
- system.hdf - Hardware Definition File imported from Vivado

pfs_ctrlr.srcs/constrs_1/new/
- system_no_swap.xdc – Unused pinout file for the FPGA, this reflects the pinout of the schematic at the time, however the sectors were not ordered properly and this was solved via a new pinout
- system.xdc – The used pinout file for the FPGA, corrects the sector ordering so that sectors 1-6 enumerate from left to right physically on the daughtercard flex connectors
- timing.xdc – Timing file used primarily to add false paths in the pfs_daughtercard IP

pfs_ctrlr.srcs/sources_1/

- A collection of files that define the block design created using the Vivado block diagram editor, which implements the entire Microblaze system and the daughtercard IP. A top level file block_design_wrapper.v instantiates all the modules.

./

- pfs_ctrlr.xpr – Xilinx Vivado project file that can be opened to load the project

4. FPGA Modules Description

deserialize.sv
Deserializes incoming responses from a sector of driver boards. Also provides some status info such as number received, and any errors in parity/stop-bits.
- Adds bits for start, parity, and stop

fifo.sv
A wrapper for the xilinx Dual Port FIFO.

pfs_daughtercard.sv
Top level module for the daughtercard IP, implements an AXI Lite interface for the Microblaze and all the I/O to the daughtercard.
- Holds logic to implement the Slave interface of AXI Lite
- Breaks up AXI Lite bus addressing to the 6 different Sectors
- Has a 32-bit 1ms precision timer for use by software to track timeouts in operations
- Also has 6 bits for "rx_err" that goes to 6 LED's to flag errors in responses

pfs_sector.sv
The controller for each individual sector. Provides some registers and also uses the files "serialize.v", "deserialize.v" and "fifo.v" to implement the communication interface.
- A command bit initiates a reset down the chain of driver boards
- Note: The data width for communication transactions/packets is 34 bits, but the upper 2 bits are hard coded to zero since they are needed only if there are more than 16 boards per sector (there are 14), this was done so that the CPU can deal with transactions that are 32 bits in size
- Output rx_err flags high if any response comes in with a bad parity or stop bit.
- Fifo's exist to buffer commands and responses with the driver boards for the CPU

serialize.sv
Serializes an outgoing command for the driver boards.
- Adds bits for start, parity and stop

block_design.v
Auto-generated by Vivado, implements the entire system including DDR3 controllers, Ethernet, interrupts, etc. Preferably the *.bd file should be used with Vivado to explore the design.

block_design_wrapper.v
Top level HDL design file generated by Vivado.

5. C File Descriptions

lscript.ld
The linker script is setup so that the application is loaded into block-ram. Since the application can fit into block-ram this eliminates the need for any boot-loaders from external memory of the software since it will already be loaded upon the FPGA being programmed on power up.

pfs.c
Holds the low level functions for interfacing with the daughtercard controller. Also holds look-up tables within functions to generate bit-masks for cobra enables, cobra to motor conversion, etc. Uses parameters in "pfs_low_api.h".

pfs_eth.c
Used to control and setup the TCP/IP interface for the user to connect to. The bulk of this is done by the Xilinx library for TCP/IP called LWIP (Lightweight IP). Has all the functions to parse TCP commands sent by the user and generate the appropriate telemetry. Uses parameters in "pfs_high_api.h".

pfs_globals.h
A place to declare functions/variables that should be globally accessible.

pfs_high_api.h
The high level API that creates parameters and data-types for commands sent by the user.

pfs_low_api.h
The low level API that creates parameters and a data-type for commands sent to driver boards.

pfs_queues.c
Defining, creating, and allocating the queues, which are declared in "pfs_globals.h".

pfs_translate.c
Provides functions to take user command data types and translate them into low level driver board commands, which are put into a queue to be sent out at a later time.

platform.c platform_mb.c
Generated by Xilinx, provides functions for interrupts, timer control, etc.

project.c
The file with the main() call. Connects all the files together, performing initialization, and running the user loop. Several side programs are written and called in the user loop for things such as counting boards in a sector, detecting communication loss, tracking operation timeouts, etc.

queue.c
Implements the queue used throughout the project for managing user commands and driver board commands.

6. Detailed Descriptions

6-a. TCP/IP + Ethernet

IP Address: 128.149.77.24
Netmask: 255.255.255.0
Gateway: 128.149.77.1
TCP Port: 4001

The user should use an IP address of the type 128.149.77.x where x is any value other than 24. The ethernet on the control module uses the "auto-negotiate" setting for the speed, and a link speed of 10M/100M/1G is supported.

6-b. Watchdog

The watchdog sends a "ping" or read command out on each sector, to the last known board in the chain. The ping command is a read of command.version from the driver board. If a board was previously in the chain but has stopped responding the ping, the watchdog prints out over Uart "sector-board not responding." and then starts re-counting the number of boards in that sector.

The last board in the sector needs to respond to the ping within 5 seconds, otherwise the watchdog triggers.

6-c. Board Counting

Board counting can be run on one sector at a time, it is started under 3 conditions:
(1) The user issued a power command (all sectors)
(2) The Control Module software is initializing (all sectors)
(3) A watchdog timed out (only the relevant sector)

To count boards in a sector the detected board count is initialized to 0 for that sector, and a ping is issued to the 0th (0-indexed) board in the chain. If a reply comes then the board count is incremented and this process is repeated for the next board. Once a reply to the ping does not come the board count is finalized, and the pinging stops. The timeout period for a response to a ping command to arrive is 250miliseconds.

6-d. Tracking Busy Boards

All commands have a timeout field, by which point if the command is still running the control module will send it's second and final telemetry to terminate the command with a response code of 1 indicating timeout.

When the user sends a command for a "Run" or "Calibrate", a respective bit is set in the command.control register. The Control Module keeps it's own record of which boards are busy, and then continuously polls the command.control register to check if these bits are still set. When a reply comes back indicating these bits are low the control module updates it's record to show that board is no longer busy. This querying process takes place every 20miliseconds while the desired reply from a busy

board has not come back. Once no more boards are busy or the timeout triggers, the final telemetry is sent.

6-e. Debug Modes & UART Port

The UART runs at 115200 Baud with no parity and 1 stop bit. The receive side of UART is not utilized by the control module software.

There is a debug_level variable in "project.c" that specifies the level of debug print-out over UART. The debug level is AND'd with the "level" setting used for the DBG_PRINTF function. It is recommended to leave this value at 2 or 0 for suitable performance. Increasing this settings makes performance take a hit and in some cases can cause operations to timeout due to the processor printing massive amounts of text.

Note: Throughout the projects are mentions of something called USHELL, it is a user interface that can run over either TCP or UART. It was originally planned to be used in the design but was left out, and it's print statements of DBG_PRINTF and USHELL_PRINTF redirect to the standard xil_printf function. This definition occurs in "pfs_globals.h".

6-f. LED Error Flags

On the KC705 Board there is a strip of 8 LED's, two of which are left always high but the other 6 are for each sector on the daughtercard. In the event a malformed response is detected from any of the sectors by the control module, it flags the LED related to it's sector. The LED stays high until the control module is reset or re-powered.

The LEDs may set if power is cut off to the driver boards during a response. They may sometimes prove useful if a communication link is intermittent. However, they will only detect a flaw in the stop bit or parity bit, not in the data payload of the response.

7. Tools Used

Synthesis: Vivado 2017.2

Place and Route: Vivado 2017.2

Microblaze Development: Vivado 2017.2 SDK

Firmware Programming: Vivado 2017.2 SDK

8. Firmware Parameters

- Kintex-7 KC705 Evaluation Platform selected in Vivado (this handles almost all constraints including DDR3, Ethernet, etc)
- SW13 M[2:0] bits are set to '010' to cause the board to load firmware from the Master BPI flash.
- When flashing download.bit to flash memory with Vivado SDK select the Micron PC28F00AP30T part

For the delivered firmware (download.bit):

- Usage: 19% LUT, 65% of Block-Ram
- XC7K325T-2FFG900C with temperature range 0-85ºC
- Worst negative slack (WNS)
    - WNS on Setup: 0.148ns
    - WNS on hold 0.035ns

Note: Despite the FPGA part rating of 0-85ºC, part specialists at JPL have confirmed this device should not have trouble operating in the -5 to 25ºC rating for PFS.