



FC7 Rob User Manual

version 0.1

2014.02.25

DRAFT

FC7 project homepage:

Contact: Mark.Pesaresi@imperial.ac.uk & Paschalis.Vichoudis@cern.ch

Document History

- **V0.1, 2014.02.25:** First draft

Table of Contents

Document History	2
Table of Contents	3
1. INTRODUCTION	4
2. ARCHITECTURE	5
3. How to use the FC7.....	6
3.1 Hardware.....	6
3.2.1. Switches	6
3.2.2. Powering	7
3.2.3. Configuration	7
3.2.4. Jumpers.....	14
3.2.5. Resets.....	14
3.2.6. LEDs.....	14
3.2 Firmware	15
3.2.1. Requirements.....	15
3.2.2. Installation	15
3.3 Software	17
3.3.1. Requirements.....	17
3.3.2. Installation	17
3.3.3. Testing access to the FC7.....	18
4. REFERENCES.....	20
5. APPENDIX A	21
6. APPENDIX B.....	23

1.INTRODUCTION

To be completed



Figure 1-1: FC7-R0b top and bottom view

2.ARCHITECTURE

3. How to use the FC7

3.1 Hardware

3.2.1. Switches

CPLD CONFIG (J1)

The CPLD CONFIG (J1) dip switch provides the CPLD with the JTAG and FPGA boot options used for configuration of the board.

Switch numbering convention – a scheme in which the orientation of the switch (switches facing left or right) becomes inconsequential to the settings, as defined in this document, is used. The figure below defines the convention. Switch A is the switch closest to the BOTTOM of the board. Switches follow alphabetical order towards the TOP of the board with switch H being the switch closest to the TOP of the board.

Switch orientation picture needed.

The table below describes the switch options.

Table 3-1: CPLD CONFIG switch settings

Switch	Description	Default
A	Selects whether the FPGA is included in the JTAG chain or not. <ul style="list-style-type: none"> UP: not included DOWN: included 	DOWN
B	Allows configuration of the microcontroller using the JTAG chain. <ul style="list-style-type: none"> UP: not included DOWN: included <i>NOTE: programming the microcontroller using the JTAG chain precludes any other device from joining the chain. Consequently, switches A, C & D should be UP when attempting to configure the microcontroller.</i>	UP
C	Selects whether an FMC occupying the L8 slot is included in the JTAG chain or not. <ul style="list-style-type: none"> UP: not included DOWN: automatically included when an FMC is inserted into the L8 slot 	DOWN
D	Selects whether an FMC occupying the L12 slot is included in the JTAG chain or not. <ul style="list-style-type: none"> UP: not included DOWN: automatically included when an FMC is inserted into the L12 slot 	DOWN

E	<p>Selects whether the FPGA should be configured off the SPI Flash PROM or the microSD card on power on.</p> <ul style="list-style-type: none"> UP: FPGA is configured via SPI Flash PROM DOWN: FPGA is configured via microSD card <p><i>NOTE: switch should be set to DOWN if communication with the microSD card is required (e.g. for remote data uploading).</i></p>	DOWN
F	Reserved.	UP
G	<p>Allows direct SPI programming.</p> <ul style="list-style-type: none"> UP: SPI Flash programmed via FPGA (JTAG) DOWN: direct access to SPI Flash via SWITCHED JTAG header (J4) 	UP
H	<p>Selects JTAG source.</p> <ul style="list-style-type: none"> UP: 'local' SWITCHED JTAG header (J4) is selected DOWN: 'crate' AMC connector JTAG is selected 	UP

TO DO: IP address switch, debug header

3.2.2. Powering

TO DO: power jumper, switch

3.2.3. Configuration

JTAG connectors

The FC7 features two onboard JTAG headers.

- The CPLD JTAG (J3) header is for configuring the CPLD only.
- The SWITCHED JTAG (J4) connector is a JTAG header which allows for configuration of the FPGA or other onboard devices by use of the CPLD CONFIG dip switch (J1) as described in Table 3-1. Configuration options for the FPGA are described below.

JTAG connector figures needed.

An alternative option for accessing the on board JTAG chain (excluding CPLD) exists. By use of the appropriate switch setting (Table 3-1), one can select between using the local JTAG header (J4) or the AMC connector JTAG interface. Use of the AMC JTAG interface either requires a uTCA crate and an AMC card that support JTAG configuration over the backplane or a supported AMC extender card.

Direct programming of CPLD via JTAG

To update the CPLD with the latest binary:

1. Connect the JTAG programmer to the CPLD JTAG (J3) header.
2. Power the board on. **This programming procedure will only work in desktop power mode unless the microcontroller is already programmed.**
3. Using IMPACT, commence a Boundary Scan. Right click after completion and “Initialize Chain”.
4. When prompted to assign a new configuration file, navigate to the *top.jed* binary. Right click on the XC2C256 device, selecting “Program”.

Direct programming of Atmel microcontroller via JTAG

The Atmel microcontroller must be programmed if the user wants to take advantage of the advanced functionality available to the FC7 such as crate operation, remote power and network management, FPGA configuration off the SD card, and remote firmware loading.

NOTE: The programming procedure has been tested & validated on native (32/64bit) Linux and (32/64bit) Windows 7, using the JTAGICEII programmer tool under AVR32 Studio 4, and the JTAGICEII/III programmer tool under Atmel Studio 6.1 respectively.

To update the microcontroller with the latest MMC binary:

1. Connect the JTAGICEII or JTAGICEIII programmer tool to the SWITCHED JTAG (J4) header.
2. Set the CPLD CONFIG (J1) dip switch **B** to **DOWN**. Switches **A**, **C** and **D** must be **UP** (see Table 3-1).
3. Power the board on. **This programming procedure will only work in desktop power mode.**
4. Under AVR32 Studio 4 or Atmel Studio 6, connect to the tool. Exact instructions will depend on the version of software you are using.
 - In Studio 6, right click on the JTAGICE tool under “Available Tools” and select “Device Programming”. Make sure the correct tool (JTAGICEII or JTAGICEIII), device (AT32UC3A3256) and interface (JTAG) are being used before clicking “Apply”. Select the “Memories” tab and follow the instructions below.
 - In AVR32 Studio 4, find the JTAGICEII tool under “AVR Targets”. Right click and follow the instructions below.
5. Erase the chip. This will prevent any old User Page configuration data from being uploaded after the next step is run.
 - In Studio 6, under “Device”, select “Erase Chip” and run “Erase Now”.
 - In AVR32 Studio 4, select “Chip Erase”.

6. Follow this up by erasing the User Page. This will make sure that new User Page configuration data is updated once the new program code is uploaded and executed in the next step.
 - In Studio 6, under “Device”, select “Erase User Page” and run “Erase Now”.
 - In AVR32 Studio 4, select “Erase”. Select “Also erase the User Page” and click OK.
7. Finally program the microcontroller with the latest *fc7_mmc.hex* or *fc7_mmc.elf* file.
 - In Studio 6, under “Flash”, navigate to the desired hex file and click “Program”.
 - In AVR32 Studio 4, select “Program”, navigate to the desired elf file, check all the program options and click OK.

The FPGA can be both programmed directly via JTAG, or indirectly via non-volatile images stored either on an SPI Flash PROM or an optional SD card. Firmware images can either be stored directly to the SD card from a PC, or can be uploaded remotely via the FPGA using IPBus, providing the FPGA is already programmed with appropriate firmware.

Direct programming of FPGA via JTAG

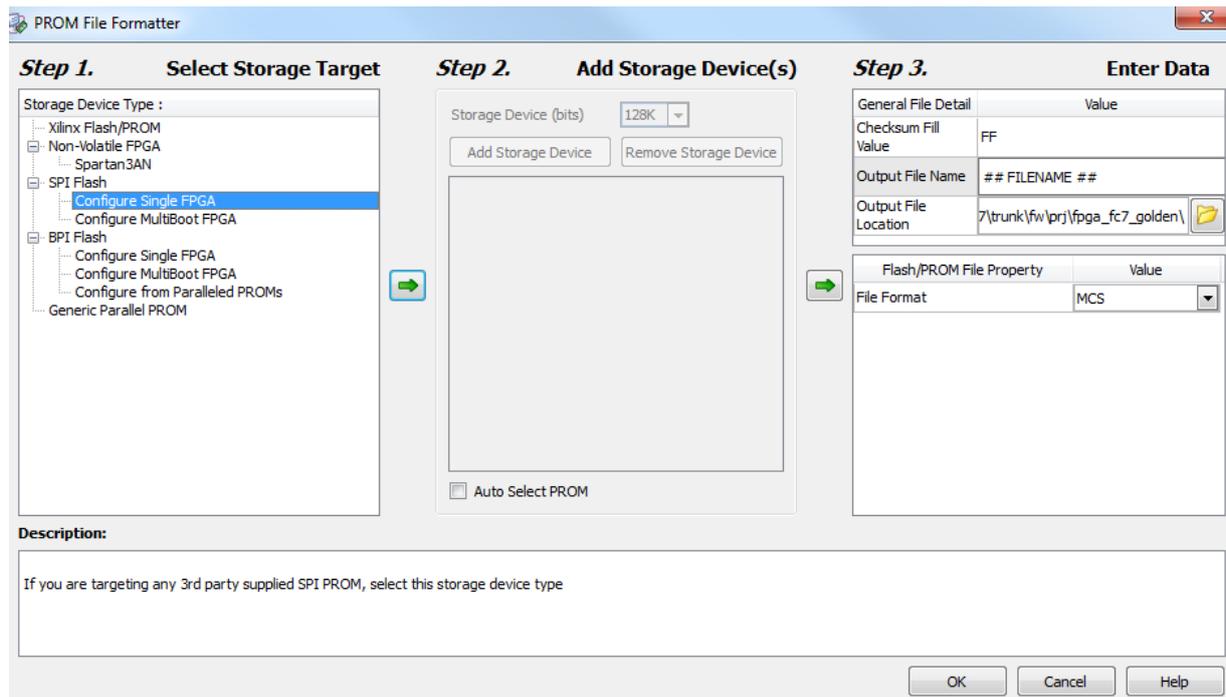
To program the FPGA directly:

1. Generate the firmware bit file. The bitgen option “-g Compress” can be set to **TRUE** to speed up the process.
2. Connect the JTAG programmer to the SWITCHED JTAG (J4) header.
3. Set the CPLD CONFIG (J1) dip switch **A** to **DOWN**. Switch **B** must be **UP** (see Table 3-1).
4. Power the board on.
5. Using IMPACT, commence a Boundary Scan. Right click after completion and “Initialize Chain”.
6. When prompted to assign new configuration file, navigate to the generated FPGA *.bit* file. Right click on the XC7K420 device, selecting “Program”.

Indirect programming of FPGA via SPI Flash PROM

1. Generate the firmware bit file. The bitgen option “-g Compress” should be set to **TRUE** as this vastly speeds up the programming process.

2. Connect the JTAG programmer to the SWITCHED JTAG (J4) header.
3. Set the CPLD CONFIG (J1) dip switch **A** to **DOWN**. Switches **B** and **E** must be **UP** (see Table 3-1).
4. Power the board on.
5. Using IMPACT, create a PROM file, using the parameters:
 - SPI Flash -> Configure Single FPGA
 - Auto Select PROM
 - File Format: MCS



6. Add the firmware bit file. Once complete, *“Generate File”*.
7. Commence a Boundary Scan. Right click after completion and *“Initialize Chain”*.
8. When prompted to assign new configuration file, cancel, but when prompted to add SPI/BPI Flash, accept and navigate to the generated MCS file.
9. Select SPI PROM N25Q256, Data Width 1.
10. Right click on the SPI/BPI Flash device, selecting *“Program”*.

Indirect programming of FPGA via microSD card

The MMC is able to configure the FPGA on power up if a valid programming file is available on a microSD card inserted into the FC7. Firmware images can be stored directly to the microSD card using a SD card writer and a SD file management utility called *imgtool*. The *imgtool* utility is currently supported as part of the CACTUS project. Any issues or bug reports related to the use of the *imgtool* utility to should be directed to the CACTUS TRAC

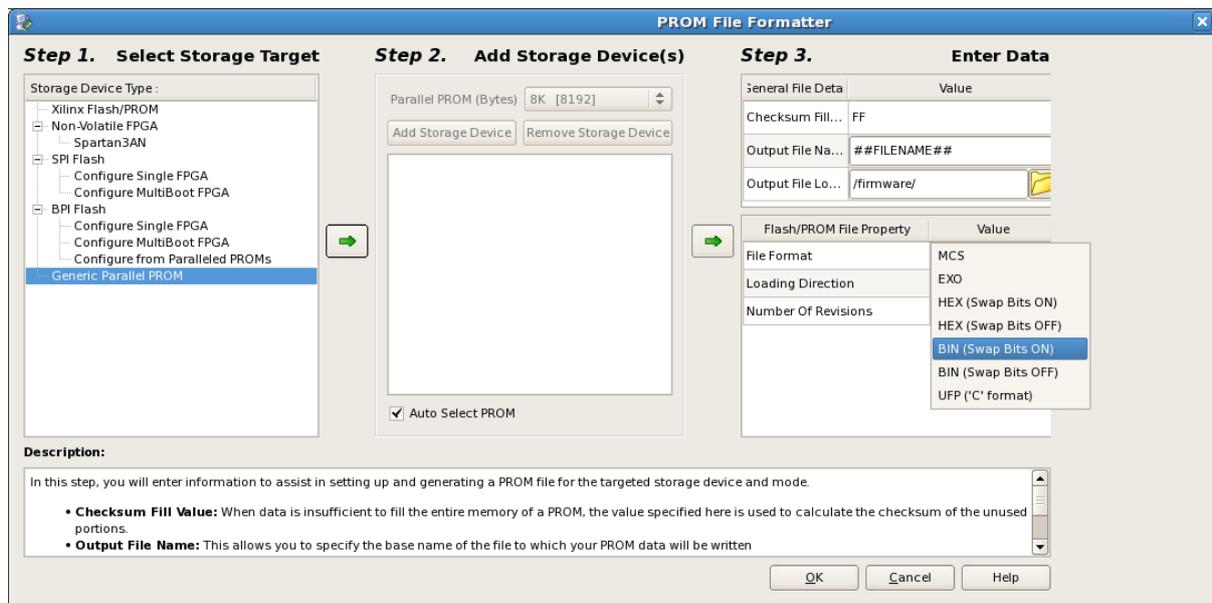
ticket system, <https://svnweb.cern.ch/trac/cactus/report/1>. Please check to see if your question has already been asked before submitting a new ticket.

NOTE: The *imgtool* procedure has been tested & validated on native (32/64bit) Linux SLC5 & SLC6 only

WARNING: Remote programming of the FPGA (see below) can only be guaranteed if the FC7 Golden Firmware Image “GoldenImage.bin” from the FC7 SVN repository is correctly stored on the microSD card.

To store firmware images to the microSD card directly:

2. Generate the firmware bit file making sure that the bitgen option “*-g Compress*” is set to **FALSE**
3. Using IMPACT, create a PROM file, using the parameters:
 - Generic Parallel PROM
 - Auto Select PROM
 - File Format: BIN (Swap Bits ON)



4. Add the firmware bit file. Once complete, “*Generate File*”.
5. From SVN, check out:

```
svn co https://svn.cern.ch/repos/cactus/tags/ic_mmc/ic_mmc_v1_6_0
```

6. Change into the “*imperial_mmc/tools/imgtool*” directory and run “*make Board=FC7_0*” to create the “*imgtool*” executable

7. If using an external card-reader, plug it into your linux PC **WITHOUT THE microSD CARD INSERTED.**
8. Insert the microSD card.
9. Run “`sudo /sbin/fdisk -l`”. There should be an entry that says “*Disk XXX doesn't contain a valid partition table*”. Note the name of this disk.
10. Run “`sudo chmod 777 XXX`”
11. The “`imgtool`” executable has several options. The usage options can be seen by running “`./imgtool ?`”:

Table 3-2: `imgtool` command options

Command	Description
<code>format <label></code>	Formats an image
<code>list</code>	List files in an image
<code>add <name> <file></code>	Adds a file to an image
<code>get <name> <file></code>	Gets a file from an image
<code>del <name></code>	Deletes a file from an image
<code>check <name></code>	Verifies the checksum of a file

12. Usage of “`imgtool`” is, then,

```
./imgtool XXX Command [parameters]
```

13. To prepare a microSD card do:

```
./imgtool XXX format Firmware
./imgtool XXX add YYY.bin ZZZ.bin
```

Where XXX is the name of the microSD card as reported by “`fdisk`” in step 8, YYY.bin is the name you wish the firmware to have on the microSD card and ZZZ.bin is the name of the PROM file created in step 2. This formats also the microSD card and gives it the volume name “*Firmware*”.

The name of the firmware image (YYY) from which the card is booted at power-up must always be “*GoldenImage.bin*”; if this file does not exist on the microSD card, the FPGA will not be programmed at power-up and so Ethernet access will not be available. Any number of additional firmware images can be stored to the microSD card using this method, provided there is enough space left on the volume (min 20MB). The FPGA can be re-configured with the desired firmware image via IPbus after power-up.

14. To list the contents of an existing SD card:

```
./imgtool XXX list
```

The conversion between the native endianness of the host system to big-endian, required for the Atmel UC3A3256, is handled automatically by the *imgtool* utility.

15. Insert the microSD card into the FC7 and power on, making sure that switch **E** on the CPLD CONFIG (J1) is set to the **DOWN** position beforehand (see Table 3-1). After a couple of seconds, the board should load the “*GoldenImage.bin*” file stored on the microSD card.

Remote programming of FPGA via microSD card and IPbus

The FC7 also allows for remote programming of the FPGA over IPbus, again by use of persistent storage of firmware images on the microSD card. The mechanism for remote programming is described in Appendix ?. Any number of firmware images can be stored on the microSD card, provided there is sufficient space on the volume, and the desired image to configure the FPGA can be selected via an IPBus command.

NOTE: New or non-SFWS formatted microSD cards must be formatted using the *imgtool* utility beforehand. It is highly recommended to follow the steps described in the section “*Indirect programming of FPGA via microSD card*” and add the SVN “*GoldenImage.bin*” to the microSD card first.

NOTE: The following procedure has only been tested & validated on native and virtual (32/64bit) Linux SLC5 & SLC6 so far.

Testing using uHAL for Windows to follow.

WARNING: Remote programming of the FPGA can only be guaranteed if the FC7 Golden Firmware Image “*GoldenImage.bin*” from the FC7 SVN repository is already correctly stored on the microSD card. At present, the MMC will allow the user to overwrite the Golden Firmware Image remotely for ease of debugging. The Golden Image should be inviolate to ensure IPBus access is always guaranteed so this feature will be removed in future.

To remotely program the FPGA over IPBus:

1. Generate the firmware bit file, making sure that the bitgen option “*-g Compress*” is set to **FALSE**
2. Set up the software environment according to Section 3.3
3. Insert the microSD card with a valid “*GoldenImage.bin*” file into the FC7 and power on, making sure that switch **E** on the CPLD CONFIG (J1) is set to the **DOWN** position beforehand (see Table 3-1). After a couple of seconds, the board should load the “*GoldenImage.bin*” file stored on the microSD card.

4. Run the following executable to reprogram the FPGA with the generated bitfile.

```
cd sw/fc7
source setup.sh
cd tests
./bin/fc7-userimage-reprogram.exe -i ip_address -f xilinx_bit_file
```

The executable will convert the provided bit file, generating a bin file which is both byte-swapped to handle the endianness of the host system, and bit-swapped for compatibility with selectMAP configuration of the FPGA. The generated file “*UserImage.bin*” is (over)written to the microSD card, before the instruction to reset the FPGA is transmitted.

5. Contained within the instruction to reset the FPGA is also the name of the firmware image to load, once the FPGA is ready to be configured. Two auxiliary executables are provided to reboot and reconfigure the FPGA; either with the “*GoldenImage.bin*” or the “*UserImage.bin*”.

```
./bin/fc7-boot-goldenimage.exe -i ip_address
./bin/fc7-boot-userimage.exe -i ip_address
```

If the executable fails to write a valid “*UserImage.bin*” file to the microSD card, or some other error occurs, the “*GoldenImage.bin*” is loaded automatically so that IPBus communication and remote access is always available. More details can be found in Appendix A.

3.2.4. [Jumpers](#)

TO DO: Port3, CLK in/out

3.2.5. [Resets](#)

TO DO: Reset switches

3.2.6. [LEDs](#)

TO DO: picture of LEDs and labels

Table 3-3: LED status codes (not all combinations included)

LED	COLOUR	STATUS
CPLD LED	RED	FPGA in reset
	GREEN (blink)	CPLD clock
	GREEN	FPGA configured
	BLUE	SPI PROM configuration mode

SYS LED TOP	RED	Golden Image Firmware loaded
	BLUE (blink)	IPBus 1Hz clock
SYS LED BOTTOM	RED	CDCE PLL not locked
	ORANGE	CDCE TTC clock out of phase
	GREEN	CDCE ok
MMC LED	GREEN	Loading firmware from SD

3.2 Firmware

3.2.1. Requirements

To build custom firmware for the FC7, the following packages are required:

IPBus 2.0v1 into [\[project_root\]/cactus/tags/ipbus_fw/ipbus_2_0_v1](#)

```
svn co https://svn.cern.ch/repos/cactus/tags/ipbus_fw/ipbus_2_0_v1
```

FC7 3.6.0 into [\[project_root\]/fc7/tags/fc7_3.6.0](#)

```
svn co https://svn.cern.ch/repos/ph-ese/be/fc7/tags/fc7_3.6.0
```

To build custom MMC software, the following package is required in addition:

IC_MMC v1.60 into [\[project_root\]/cactus/tags/ic_mmc/ic_mmc_v1_6_0](#)

```
svn co https://svn.cern.ch/repos/cactus/tags/ic_mmc/ic_mmc_v1_6_0
```

Xilinx ISE v14.6 is recommended for firmware development and implementation.

3.2.2. Installation

The folder tree should be as described in Figure 3-1.

To use the existing example code as is, compile the “*fpga_fc7_golden.xise*” project file under [fc7/tags/\[tag\]/fw/prj/fpga_fc7_golden](#) using Xilinx ISE.

When developing your own code, please keep in mind that in order to receive support, the files under [fc7/tags/\[tag\]/fw/src/sys](#) MUST remain unchanged. The files under [fc7/tags/\[tag\]/fw/src/usr](#) can be freely modified according to the user needs. The CPLD

firmware & MMC software source is provided for your reference only and we strongly suggest not modifying it.

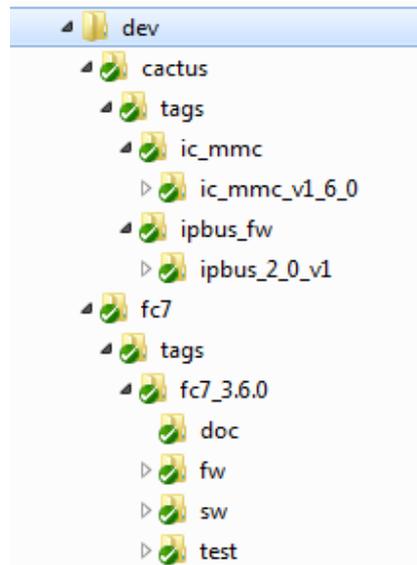


Figure 3-1: Folder Tree

Tagged binaries are provided that are guaranteed for cross-compatibility. These can be found in the following locations:

- CPLD “*top.jed*” in [fc7/tags/\[tag\]/fw/prj/cpld_fc7](#)
 - for installation, follow the guide “***Direct programming of CPLD via JTAG***” in Section 3.2.3 above.
- MMC “*fc7_mmc.hex*” or “*fc7_mmc.elf*” in [fc7/tags/\[tag\]/fw/prj/mmc_fc7/RunFC7](#)
 - for installation, follow the guide “***Direct programming of Atmel microcontroller via JTAG***” in Section 3.2.3 above.
- FPGA “*GoldenImage.bin*” in [fc7/tags/\[tag\]/fw/prj/fpga_fc7_golden](#)
 - for installation, follow the guide “***Indirect programming of FPGA via microSD card***” in Section 3.2.3 above.

3.3 Software

3.3.1. Requirements

- Linux SL6-64 bit (recommended) or Linux SL5-32/64 bit (deprecated), native or virtual.
- Gigabit Ethernet drivers.
- IPBus suite version 2.2 (installation instructions available at <https://svnweb.cern.ch/trac/cactus/wiki/uhalQuickTutorial#HowtoInstalltheIPbusSuite>)

TO DO: Testing uHAL for Windows.

3.3.2. Installation

Currently the FC7 software package is available via SVN only and needs to be compiled to be used.

The FC7 software package is organized into 2 sub-packages:

- **fc7/fc7**: C++ driver library
- **fc7/tests**: test programs and scripts

The driver library contains the high level functions. The tests folder contains collected binaries and scripts for testing FC7s. For running python scripts based on the PyChips library, the library source is provided alongside the top level fc7 folder as **pychips/**.

TO DO: Python bindings for C++ to allow old pychips scripts to eventually run in uHAL framework.

To install the software:

1. Check out from SVN

```
svn co https://svn.cern.ch/repos/ph-ese/be/fc7/tags/fc7_3.6.0/sw
```

2. Set up the environment using the setup script, editing as appropriate

```
cd sw/fc7
source setup.sh
```

3. Compile

```
make
```

3.3.3. Testing access to the FC7

Before starting, make sure the setup.sh script has been sourced.

```
cd sw/fc7
source setup.sh
```

In the **fc7/tests** folder, the following are available:

- **bin**: test executables
- **scripts**: test python scripts – tested with Python 2.4.3

WARNING: Currently the test python scripts under fc7/tests/scripts use PyChips, which is now deprecated. These will eventually be replaced with scripts that use the supported uHAL/python bound uHAL. PyChips is NOT compatible with Python 3.x

4. REFERENCES

5.APPENDIX A

TO DO: clean this up with diagrams etc.

MMC & Transactor are IPBus masters, FPGA is IPBus slave.

IPBus transport layer between MMC master and FPGA slave is an SPI x1 interface (MMC is SPI master). Four communication lines; MOSI (data to FPGA), MISO (data from FPGA), Master CLK, CS.

Additional 16bit DMA bus for data transfers between MMC and FPGA, controlled by MMC. Two communication lines; NRD (read data bus on falling edge, increment memory address), NWE (write to data bus on falling edge, increment memory address).

FPGA IPBus slave:

Data/commands to be interpreted by the MMC (e.g. load firmware to sd, delete firmware from sd, reboot fpga) are received from the standard IPBus Transactor and transparently packed into a 32bit load FIFO on the FPGA (FPGAtoMMC). A separate return FIFO (MMCToFPGA) is also available to read from, containing data from the MMC. The FIFOs have the IPBus register address 0x0402, each individually accessed by means of the IPBus R/W flag.

Pointer records for the two FIFOs are kept in the slave, accessible in the IPBus register space. The FPGAtoMMC FIFO record is on 0x0400, and the MMCToFPGA FIFO record is on 0x0401.

The MMC regularly reads the FPGAtoMMC pointer record on the FPGA via the SPI IPBus. If data are detected in the FIFO (write pointer>=read pointer+2), the MMC initiates a Direct Memory Access on the FIFO in the FPGA, reading two 16bit words. The first word is an instruction command, the second indicates the length of the payload to be read. The FIFO is continually read via DMA until the payload is fully received and the appropriate instructions are executed (e.g. write payload to SD card, set boot image filename to be loaded).

Following this, the MMC also loads a reply into the MMCToFPGA FIFO provided there is adequate space available. This is checked by querying the MMCToFPGA FIFO record via IPBus.

Firmware upload to FPGA from microSD card

The SD-card is formatted with a file-system known as simple firmware file-system (SFWFS). The storage medium is divided into “slots” around the size of a firmware image, guaranteeing an image can be stored without fragmentation. An index table at the front of the disk stores whether a slot is in use, a file name, file size and checksum, which allows access by name. A full library of SFWFS operations is included. The image files are stored in the blocks after the header. They do not require their own header and just start at the appropriate block (slot) and end before the next image. Any unused space in the slot should be padded with 0xFFFFFFFF if the images are being used to configure the FPGA.

6.APPENDIX B

TO DO: Clean this up with diagrams etc.

IPMI management using ipmitool

This guide assumes that the crate MCH is on the network at address 192.168.0.41 and has a blank username/password. To address a specific card, its slot number must be converted to an IPMB-L address, as indicated in the lookup table below.

Table 6-1: Lookup Table (uTCA.0 SPEC R1.0 Page 3-12)

Slot Number	IPMB-L Address
1	0x72
2	0x74
3	0x76
4	0x78
5	0x7A
6	0x7C
7	0x7E
8	0x80
9	0x82
10	0x84
11	0x86
12	0x88

Network configuration via the MMC

The IP and MAC address of the board can be optionally configured using IPMI commands to the MMC. However, in order for IPBus to pick up changes to the network configuration, the IPBus module `ipbus_ctrl` should be configured to use the 'internally' provided mac and ip addresses beforehand. To do this, the project needs to be regenerated with the following modification to `system_core.vhd` under `fc7/tags/[tag]/fw/src/sys/sys`:

```

-----
ipb: entity work.ipbus_ctrl
-----
generic map
(
  mac_cfg           => internal,
  ip_cfg            => internal,
  n_oob             => 1
)

```

...

TO DO: make this default configuration in golden?

With this firmware, the network configuration can be set as follows:

To set the MAC address,

```
ipmitool -H 192.168.0.41 -P "" -B 0 -T 0x82 -b 7 -t ipmb_address raw
0x30 0x02 0x00 0x11 0x22 0x33 0x44 0x55
```

will configure the board with the MAC address 00 : 11 : 22 : 33 : 44 : 55

To set the IP address,

```
ipmitool -H 192.168.0.41 -P "" -B 0 -T 0x82 -b 7 -t ipmb_address raw
0x30 0x03 0xC0 0xA8 0x00 0x7B
```

will configure the board with the IP address 192.168.0.123

To save the new configuration to the EEPROM,

```
ipmitool -H 192.168.0.41 -P "" -B 0 -T 0x82 -b 7 -t ipmb_address raw
0x30 0x01 0xFE 0xEF
```

otherwise any configuration applied will only be valid until the MMC is power cycled.

To read back the current network settings,

```
ipmitool -H 192.168.0.41 -P "" -B 0 -T 0x82 -b 7 -t ipmb_address raw
0x30 0x05
```

will return the MAC and IP addresses in the form,

```
IP[0] IP[1] IP[2] IP[3] MAC[0] MAC[1] MAC[2] MAC [3] MAC[4] MAC[5] FLAGS
```

e.g. "192 168 0 123 00 11 22 33 44 55 00" for the above example configuration. The flags (returned as 0x00 in the example), is a bitmask. The only bit that is currently used is 0x80

and if this is set then it means the network parameters have been changed but have not been copied to EEPROM.

TO DO: Will change when CMS wide definition of this procedure using new NetFns is implemented. Eventually best to describe this section using the helper script below.

Power cycling the board

The FC7 can be remotely reset using an IPMI command that only applies to the board itself (i.e. a local reset) without changing the module power state in the crate. All power rails except the external 12V payload power and 3.3V management power is reset in this process.

To reset the board,

```
ipmitool -H 192.168.0.41 -P "" -B 0 -T 0x82 -b 7 -t ipmb_address raw
0x30 0xFF 0xDE 0xAD
```

Sending raw user defined OEM commands

User defined IPMI commands can be implemented in the MMC code for extended purposes (see ??). The OEM group extension 0x2E is used as the NetFunction (NetFN) code to begin the transaction.

For example,

```
ipmitool -H 192.168.0.41 -P "" -B 0 -T 0x82 -b 7 -t ipmb_address raw
0x2E raw_commands
```

IPMI python script

A helper script exists that can be used to send the commands described above, and more, to the board. The script "*ipmi_helper.py*" is found under [fc7/tags/\[tag\]/sw/fc7/test/scripts](#) and is invoked under python. Slot number conversion is performed automatically.

For more information try,

```
python ipmi_helper.py help
```

For example, to retrieve the current network configuration and sensor data from the SDR, try,

```
python ipmi_helper.py -i -m 192.168.0.41 -s 11
```

assuming an MCH with IP 192.168.0.41 with null username and password, and a board in slot 11.

WARNING: A bug exists in ipmitool that reports some sensor values in the SDR as “Disabled” even though they are not. Using ipmiutil always reports the correct sensor values.