DSM / IRFU / SEDI
CEA Saclay
91191 Gif-sur-Yvette CEDEX
Tel: (33).1.69.08.17.96 Fax: (33).1.69.08.31.47
Email: Irakli.MANDJAVIDZE@cea.fr
Web: http://www-irfu.cea.fr

The TCC48 Remote Firmware Programming Tools:

User Guide

Summary:

This document describes firmware programming tools for the TCC48 board.

Author:

Name	Affiliation - Email
MANDJAVIDZE Irakli	Irakli.MANDJAVIDZE@cea.fr

Approved:

Name	Affiliation - Email
ROMANTEAU Thierry	romanteau@llr.in2p3.fr
PAGANINI Pascal	pascal.paganini@cern.ch

Destination:

Name	Affiliation - Email

History:

Date	Modifications	Version
20-07-2009	Creation of the document	DRAFT
13-10-2009	First draft ready for distribution	DRAFT
04-07-2010	Update of the locations of the Xilinx and Altera programming tools	1.0

OUTLINE

1. INTRODUCTION

There are three FPGA devices on the TCC48 board [1]: two xc4vlx100-10C-FF1513 devices from the Xilinx Virtex4 family and an xc2vp20-7C-FF896 device from the Xilinx Virtex2Pro family. There are also up to four Synchronization and Link Board (SLB) daughter cards [2] each equipped with two Altera Cyclone EP1C6 EPLD devices.

The xc2vp20 Virtex2Pro FPGA is historically referred as PuSrf (Processing Unit – Selective Readout Formatter). Among its other functions, this device implements a VME64x compliant interface, interfaces with the DCC and SRP boards and interfaces with the TTC and sTTS systems. The PuSrf FPGA is coupled with a Xilinx xcf32p Platform Flash PROM device that keeps up to four PuSrf firmware revisions. Any of the revisions can be selected as a default. The default revision is loaded in the PuSrf device at power-up or after the user initiated programming sequence. The xc2vp20 FPGA and the xcf32p FPROM form one of the three JTAG chains of an on-board ScanSTA111 Boundary Scan Bridge from National Semiconductor [3]. The latter is accessible via the VME MTM sub-bus giving the possibility of remote firmware programming and upgrade operations.

The xc4vlx100 Virtex4 devices are referred as TPG1 and TPG2 (Trigger Primitive Generator 1 and 2). For each bunch crossing they calculate trigger primitives out of the dedicated data from the endcap calorimeter front-ends and provide the primitives to the calorimeter L1 trigger via the SLB daughter cards. For each L1 accepted event they send to the PuSrf device the corresponding trigger primitive and trigger tower flag data to be delivered to the DCC and the SRP boards. The TPG1 and TPG2 devices are associated with the Xilinx SystemACE controller [4] and a Compact FLASH memory device that keeps up to 8 firmware revisions of both FPGAs. The SystemACE controller and the Compact FLASH are accessible via the TCC48 VME interface giving a possibility of remote firmware programming and upgrade operations for the TPG1 and TPG2 FPGA devices.

The SLB daughter boards provide an interface to the Regional Calorimeter Trigger. The main controller and synchronisation functionalities are implemented on two Altera Cyclone EP1C6 EPLDs. The EPLDs are associated with an EPC2 configuration device from Altera. The EPLDs and the configuration device together with the Vitesse Link circuit VSC7126 form the JTAG chain of SLB. All SLBs on the TCC48 board are joined in a single JTAG chain of the on-board ScanSTA111 Boundary Scan Bridge. As in the case of the xcf32p FPROM, the EPC2 configuration devices can be programmed or updated with a new firmware via the MTM sub-bus of the VME64x backplane.

In the following sections the TCC48 firmware update procedures for the PuSrf, TPG1 and TPG2 sub-systems and for the SLBs will be described.

2. THE TCC48 JTAG PROGRAMMING TOOLS

2.1 TCC48 JTAG chains

For simplicity only those JTAG features of the TCC48 card will be described which are relevant to the TCC48 firmware programming. The simplified scheme of a part of the TCC48 JTAG chains is shown on Fig. 1.

The TCC48 JTAG chains are accessible through the on-board ScanSTA111 Boundary Scan Bridge from National Semiconductor. The TCK, TMS, TDI, TDO, *TRST pins of the ScanSTA111 master JTAG port are connected:

a) To the corresponding pins of the VME64x MTM sub-bus of the P1 connector;

b) To a set of dedicated user-defined pins of the P2 connector.

Therefore, the master JTAG port of the ScanSTA111 Bridge can be exercised either via a JTAG controller that has an access to the MTM bus or via a JTAG controller placed on a transition board connected to the J2 connector on the rear of the VME crate.

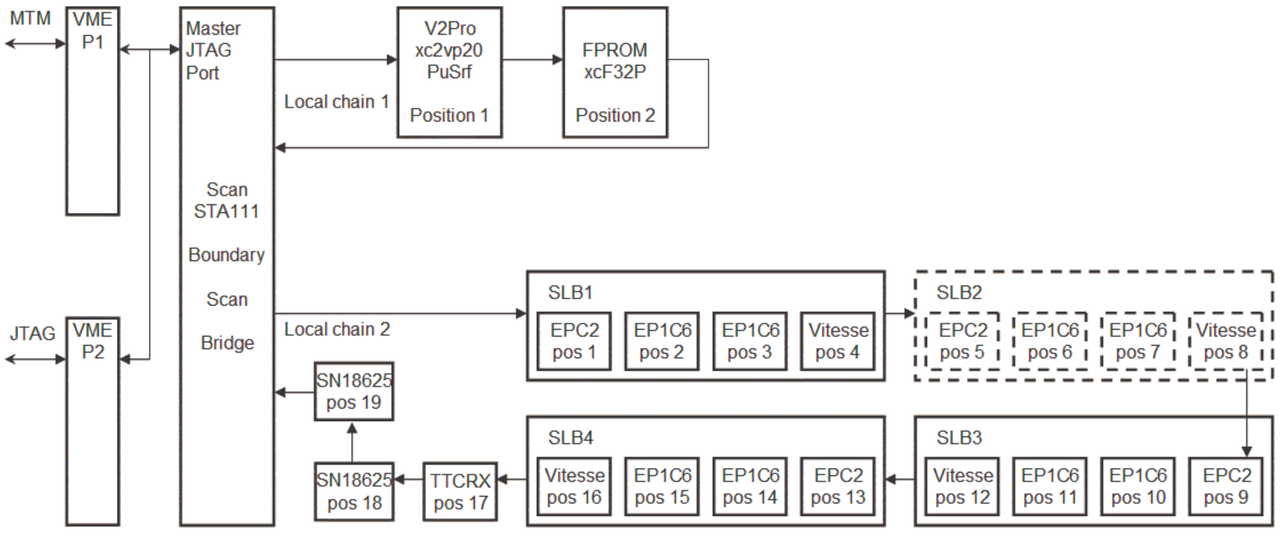

Fig. 1. Simplified view of a part of the TCC48 JTAG chains

An important feature of the TCC48 board is its capability to provide a passive connectivity between the JTAG signals on its P2 connector and the corresponding MTM signals on its P1 connector. A JTAG controller placed on a transition board behind a given TCC48 has an access to the MTM bus of the entire VME crate. It thus can control not only the JTAG chains on the TCC48 card to which it is directly connected (Fig. 2, blue path), but also the JTAG chains of all TCC48 cards in the VME crate (Fig. 2, red path).

The ScanSTA111 Bridge can be instructed through JTAG to establish a transparent connection between its master JTAG port and one of the three local JTAG ports. In this way the selected local JTAG chain can be controlled by the JTAG controller. A particular bridge on the MTM bus can be selected according to the ID of the slot that the corresponding TCC48 occupies. The geographical address pins of the VME64x bus are routed towards the address pins of the ScanSTA111 Bridge. The JTAG instruction that brings the bridge into the transparent mode carries an address field. This address field is decoded by all bridges on the MTM bus and is compared to the address encoded on their address pins. Only the ScanSTA111 Bridge for which the match occurs enters the transparent mode. For reliable JTAG operations only one bridge must be kept in the transparent mode at a time. Once the JTAG operations with the selected chain are finished the corresponding bridge must be brought to its default, non-transparent state to allow operation with other bridges and chains. This is accomplished asserting the MTM Reset signal.

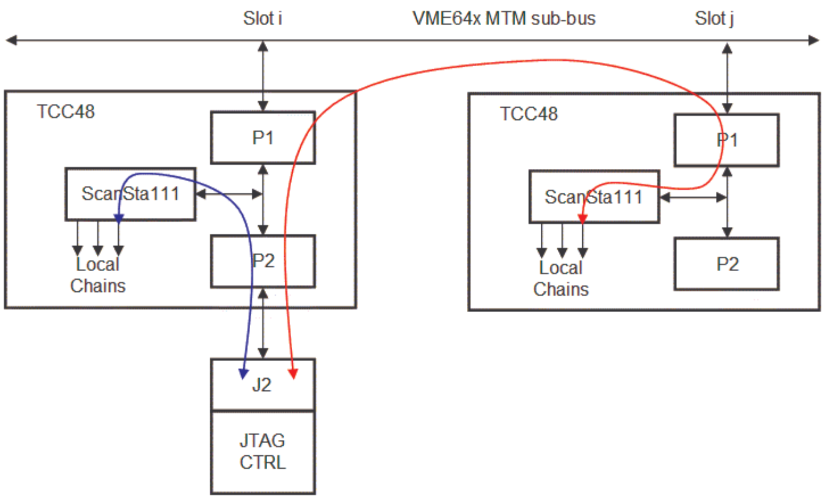

Fig. 2. External JTAG controller's access paths to the TCC48 JTAG chains

2.1.1 Local Chain "0"

The local chain 0 of the TCC48 ScanSTA111 Bridge (not shown on Fig. 1) is composed of a single device: the Xilinx SystemACE controller. This chain is not relevant for further discussion and will not be described.

2.1.2 Local Chain "1"

The local chain 1 of the TCC48 ScanSTA111 Bridge is composed of two devices: an xc2vp20-7C-FF896 FPGA from the Xilinx Virtex2Pro family (PuSrf) and associated Xilinx xcf32p Platform Flash PROM device. This chain is relatively simple the FPGA having the first position in the chain and the Flash PROM having the second (see Fig. 1).

2.1.3 Local Chain "2"

The local chain 2 of the TCC48 ScanSTA111 Bridge is composed of up to 19 devices. Depending on the Inner or Outer flavour of TCC48 it is equipped with 4 or 3 SLBs. In the outer TCC48 the SLB2 is not installed. Instead a passive piggy-back card is mounted in order to keep the continuity of the JTAG chain. Apart from devices that constitute the JTAG chains of individual SLBs, the local chain 2 comprises also a TTCrx ASIC and two SN18625 buffer chips. Therefore, the local chain 2 of an Inner TCC48 is constituted from 19 devices (as shown on Fig. 1) and the local chain 2 of an Outer TCC48 – from 15 devices. The SLB chips occupy first positions of the chain, the TTCrx and the two SN18625 buffers being at the last 3 positions.

In order to avoid important initial power consumption that characterise the SLB devices, the SLBs are kept switched off after power-up. Therefore immediately after power-up of a TCC48 there is no continuity on its local chain 2. Prior to any operation on the local chain 2 the SLBs must be switched on (the corresponding bits must be set in two dedicated configuration registers via VME write operations).

2.2 JTAG controller for the TCC48 boards

Both Xilinx and Altera propose JTAG programmer devices for their FPGA and CPLD devices. Currently the most popular Xilinx solution is the "Platform Cable USB II" [5]. The programmer connects to a USB port of a control PC that runs the Xilinx iMPACT configuration software suite [6]. The software suite and the programmer both support JTAG (boundary scan – BS) configuration modes. A dedicated flat ribbon cable is provided with the programmer and corresponding recommendations are given to interface the cable with the custom user-developed boards. The similar situation is in the case of Altera which proposes the "ByteBlaster II Cable" (parallel) and/or "USB-Blaster Download Cable" programmers [7] and the Quartus II configuration software suite [8]. As in the case of Xilinx, Altera provides a ribbon flat cable and corresponding guidelines to interface user-developed hardware with the programmers. Both the Xilinx and the Altera programming solutions allow for JTAG communications with the third party devices, such as ScanSTA111 Bridge for example, by executing sequences of JTAG commands placed in SVF (Serial Vector Format) files [9].

In order to program the on-board TCC48 FPGAs and Platform Flash PROMs from Xilinx, as well as the EPC2 configuration devices from Altera on the TCC48 daughterboard SLBs, a dedicated card called BSCAN has been developed [10]. The BSCAN card among other circuitry provides interfaces to the Xilinx and the Altera programmers via the on-board connectors that accept respective flat ribbon cables.

The BSCAN card is placed behind the VME backplane and is connected to a TCC48 via the J2 transition connector (see Fig. 3). As discussed in Section 2.1, this provides JTAG connectivity to any TCC48 card in the VME crate. The ScanSTA111 Bridges can be put in a transparent mode to connect to either local chain 1 with Xilinx devices or to local chain 2 with Altera devices and using the corresponding software suite program them.

When devising the JTAG programming scheme for the TCC48 boards the main objective was to provide remote programming capability for the on-board Xilinx FPGAs and Platform Flash PROMs. The SLB firmware being much more stable, rare *in-situ* programming sessions seemed to be acceptable. However, it was commonly admitted that configuration of either Xilinx or Altera devices should not necessitate neither removing the TCC48 card from the crate nor the power-cut of the crate to change the position of the BSCAN card.

Fig. 3 shows the adopted solution for the TCC48 JTAG controller.

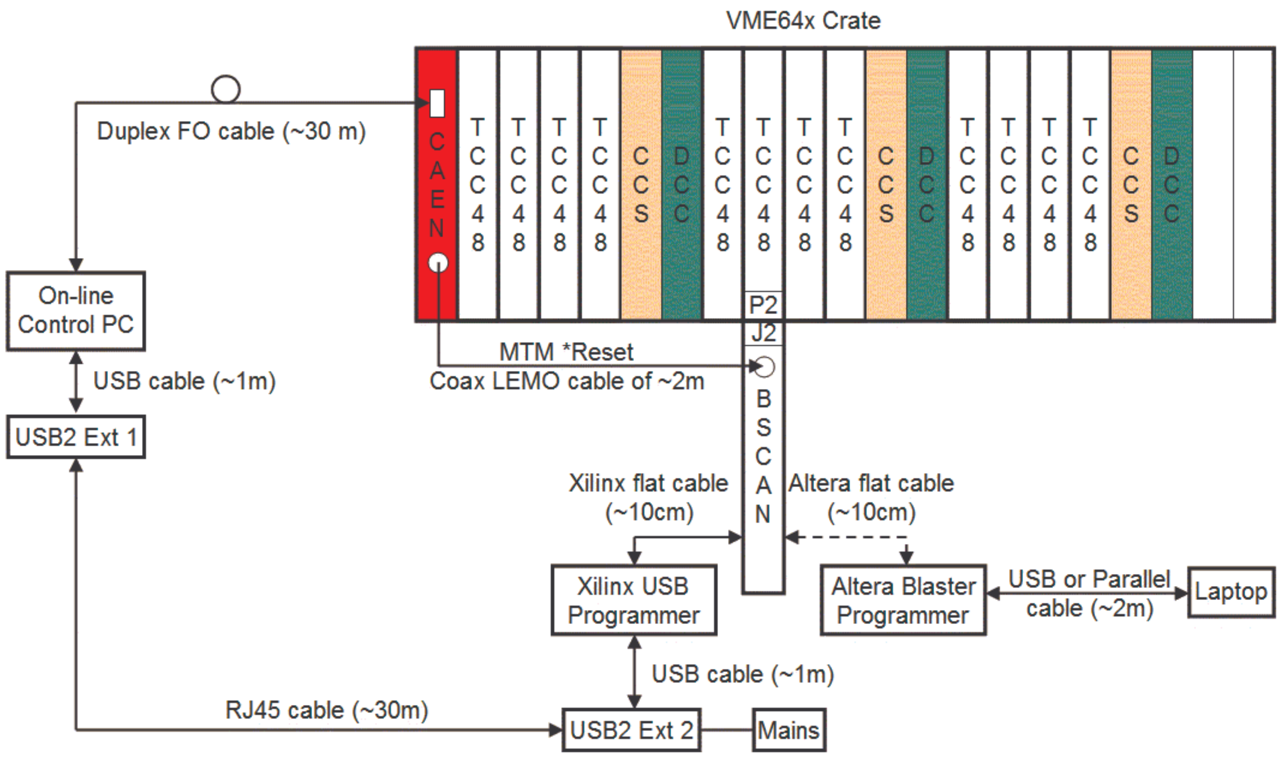

Fig. 3. Simplified view of the TCC48 JTAG controller organisation

For each VME crate with the ECAL endcap Off-detector electronics, a Xilinx USB programmer is attached to its control PC from the CMS on-line cluster. The USB programmer must be placed nearby the VME crate to allow connection to the BSCAN board via the short flat ribbon cable from Xilinx. To cover the distance between the control PC and the crate the USB2 / RJ45 extender is used [11]. The association between the endcap VME crates and the control PCs is given in Table 1.

The Xilinx iMPACT configuration software is installed on disc space shared between all control PCs (few configuration files but not drivers need to be installed locally in the `/etc` zone of each PC, however). A custom suite of configuration bash scripts and programs have been developed and installed on a shared disc space. This custom configuration software calls iMPACT utilities in batch mode in order to scan the VME crate and discover which slots are populated with the

TCC48 boards, what is the firmware revision that is loaded in the PuSrf FPGA on each board, what are the firmware revisions currently programmed in the Platform FLASH PROMs on each TCC48 board, to program the PuSrf FPGAs directly, to program the Platform FLASH PROMs with up to four firmware revisions, to set default firmware revision, to reload the PuSrf FPGAs with the selected default firmware revision, etc.

Table 1. On-line control PC and endcap VME crate association

Complete Name	Short Name	Crate	Type
vmepcS2F19-01	ecalod-01	ecalod-S2D04l	EE-
vmepcS2F19-02	ecalod-02	ecalod-S2D03l	
vmepcS2F19-03	ecalod-03	ecalod-S2D08l	
vmepcS2F19-16	ecalod-16	ecalod-S2D06l	EE+
vmepcS2F19-17	ecalod-17	ecalod-S2D02l	
vmepcS2F19-18	ecalod-18	ecalod-S2D07l	

As discussed in Section 2.1 prior to perform any operation with the JTAG chain of a particular TCC48 board, its ScanSTA111 Bridge has to be made "transparent" towards the desired chain. This is accomplished by the custom configuration software sending special JTAG command sequences over the MTM bus. Once JTAG operations on the selected chain of a TCC48 board are all done, the corresponding ScanSTA111 Bridge has to be reverted into its normal (default) "non-transparent" state. The only way to bring back the bridge to its default state is to assert the JTAG reset TRST* signal active. Neither Xilinx nor Altera programmers provide support for the optional JTAG reset TRST* signal. This signal is simply absent on the programmers.

In the adopted solution for the TCC48 JTAG programming, the TRST* signal is asserted and de-asserted by the on-line control PC through the CAEN PCI/VME adapter [12]. Addressing certain internal registers of the adapter, it is possible to generate pulses of a desired duration and polarity on the "Programmable Outputs" on its front panel. The signalling standard for the front panel LEMO outputs must be set to TTL by the "I/O" DIP-switch on the V2718 board of the adapter. A simple "C" program `JtagRstByCaen` that generates a pulse on the upmost "Programmable Output 0" is a part of the TCC48 JTAG configuration software suite. A coaxial cable connects the adapter's "Programmable Output 0" to the TRST* LEMO input of the BSCAN board (see Fig. 3). A two-transistor network conveys the TRST* pulse onto the corresponding line of the MTM sub-bus. The two-transistor network can be configured to have an open-collector output allowing for presence of yet another JTAG controller board in the crate with the similar functionalities as of BSCAN (*e.g.* the DCC programming board). To summarise, in order to assert the JTAG reset signal on the MTM sub-bus and bring all present ScanSTA111 bridges in their default "non-transparent" state, the bash scripts from the custom JTAG programming suite call the pulse generation "C" program `JtagRstByCaen` when appropriate.

The default configuration of the JTAG controller allows for remote operations with the "local chain 1" of the TCC48 boards composed of two Xilinx devices: an xc2vp20-7C-FF896 Virtex2Pro FPGA (PuSrf) and the associated xcf32p Platform Flash PROM (see Fig. 1 and Section 2.1.2). Thus the adopted programming solution permits for remote updates of the PuSrf firmware.

In rare cases, however, reprogramming of the SLB firmware might be necessary in future. The adopted solution allows for *in-situ* SLB firmware updates without removing the TCC48 cards from the crate and without power-cuts of the crate for moving the BSCAN board from slot to slot. A laptop under the Windows XP operating system is used for the firmware upgrade operations of the barrel TCC68 boards. The Altera Quartus software suite has been installed on the laptop. When a new firmware needs to be downloaded on the TCC48 SLBs, the Xilinx flat ribbon cable is disconnected from the corresponding BSCAN board and the Altera USB or Parallel Blaster cable is connected instead. To make the ScanSTA111 Bridge on a selected TCC48 board "transparent" towards the "local chain 2" with the SLB devices (see Fig. 1 and Section 2.1.3) a dedicated sequence of JTAG commands from a file on the laptop PC is executed in a command line mode of the Quartus software. Next, the Quartus is run in the GUI mode and the SLBs are programmed as if they were connected directly to the Blaster programmer. Once the upgrade operations on a TCC48 board are finished the JTAG reset is generated on the MTM bus by manually calling the CAEN pulse generation program `JtagRstByCaen` on the corresponding on-line control PC through an open remote SSH shell. Working with the SLBs on another TCC48 board in the crate can then be started. Once the firmware on the SLBs on all TCC48 boards in a crate is upgraded, the Altera Blaster programmer has to be disconnected from the BSCAN board and the flat ribbon cable of the Xilinx USB programmer has to be re-connected.

Currently programming of SLBs is somewhat tedious and time consuming task needing a permanent human attention and intervention. There might be two ways to automate the operations. One possible way is to design the configuration suites similar to the ones developed for the PuSrf Xilinx devices. This can be done for the Quartus suite under the Windows XP and using the *ad-hoc* laptop PC to run the scripts (bat file). Alternative and more attractive solution would be to install the Linux version of the Quartus suite on the shared disk space of the on-line control PCs (as it is done for the Xilinx iMPACT software). The second and the most attractive solution would be to produce in Quartus the SVF files with the SLB firmware and to use the Xilinx USB programmer and the corresponding on-line PC to play the SVF files in iMPACT. Preliminary tests of this solution were encouraging but at time being the efforts were not pursued as there were other more urgent priorities.

2.3 JTAG configuration software for the TCC48 boards

The JTAG configuration and programming software installed on the CMS on-line cluster mostly serves for the PuSrf firmware management of the Xilinx xc2vp20-7C-FF8 Virtex2Pro FPGA and the associated xcf32p Platform Flash PROM. It consists of:

a) the Xilinx iMPACT programming tool

b) the custom suite of configuration bash scripts and programs

The programming software for the SLB devices is installed on an *ad-hoc* laptop PC and will be discussed later.

2.3.1 The Xilinx iMPACT programming tool: 10.1.03 revision

For the moment of writing the tool is installed under the `/data/ecalod-disk01/daq-sw/Xilinx/ISE`[1]. The link `Xilinx` under the `/nfshome0/ecaldev` points to the `/data/ecalod-disk01/daq-sw/Xilinx` directory. Only the iMPACT standalone tool of the Xilinx Integrated Software Environment (ISE) revision 10.1 Service Pack 3 has been installed. It occupies 2.5 Gbyte of the shared disk space (In case of a fine-tuning installation one could probably liberate about a half of this space). The *ecaldev* user login has been used for most of the installation process.

The 10.1 version of the iMPACT tool can make use of the `libusb` open source software present on the Scientific Linux distribution of the CMS on-line cluster (for the moment of writing this is Scientific Linux CERN SLC release 4.4 with the kernel of 2.6.9-55.EL.cernsmp SMP Thu May 10 18:16:29 CEST 2007). The use of the `libusb` software has the following advantage: there is no need to install additional device drivers on each of the control PC. To instruct the iMPACT tool that it has to rely on the `libusb` software (rather then the `windrvr6` driver) the `XIL_IMPACT_USE_LIBUSB` environment variable must be set. The `settings32.sh` script under the `/nfshome0/ecaldev/Xilinx/ISE` directory has been modified adding the following two lines to the script:

```
XIL_IMPACT_USE_LIBUSB=1
export XIL_IMPACT_USE_LIBUSB
```

[1] For the test bench setup at building 904 the Xilinx iMPACT distribution directory is `/ecalsoft/ecaldev/Xilinix/10.1/ISE` on the `ecal904pc1` machine (`lxcms112.cern.ch`)

The `settings32.csh` script under the same directory has also been modified adding the following line:

```
setenv XIL_IMPACT_USE_LIBUSB 1
```

If the iMPACT tool has to be used manually in its GUI or batch modes the scripts must be executed once. The `settings32.sh` file is automatically called by the custom configuration scripts assuring that all the environment variables are correctly set.

Though there is no need to deploy additional drivers on the on-line cluster PCs, the iMPACT tool installation process still needs some actions to be performed with the root privileges on each of the control PCs. The `setup_pcusb` script under the `/nfshome0/ecaldev/Xilinx/ISE/bin/lin` directory must be executed (in order to avoid any attempt of update system files the original script has been modified adding the `exit` command at a place where the necessity for such an update is checked). The `setup_pcusb` script checks if the `usb` directory exists under the `/etc/hotplug` directory. If not it is created. The following structure is copied to the `usb` directory:

```
usb/:
-rwxrwxr-x  1 ecaldev ecaldev 1971 Feb 11  2008 xusbdfwu
drwxr-xr-x  2 root    root    4096 Mar 17  2009 xusbdfwu.fw
-rwxrwxr-x  1 ecaldev ecaldev  913 Feb 11  2008 xusb_emb
-rwxrwxr-x  1 ecaldev ecaldev  913 Feb 11  2008 xusb_xlp
-rwxrwxr-x  1 ecaldev ecaldev  913 Feb 11  2008 xusb_xp2
-rwxrwxr-x  1 ecaldev ecaldev  913 Feb 11  2008 xusb_xpr
-rwxrwxr-x  1 ecaldev ecaldev  913 Feb 11  2008 xusb_xse
-rwxrwxr-x  1 ecaldev ecaldev  913 Feb 11  2008 xusb_xup
usb/xusbdfwu.fw:
-rw-r--r--  1 ecaldev ecaldev 21666 Feb 11  2008 xusbdfwu.hex
-rw-r--r--  1 ecaldev ecaldev 21708 Feb 11  2008 xusb_emb.hex
-rw-r--r--  1 ecaldev ecaldev 21708 Feb 11  2008 xusb_xlp.hex
-rw-r--r--  1 ecaldev ecaldev 22956 Feb 11  2008 xusb_xp2.hex
-rw-r--r--  1 ecaldev ecaldev 20740 Feb 11  2008 xusb_xpr.hex
-rw-r--r--  1 ecaldev ecaldev 22956 Feb 11  2008 xusb_xse.hex
-rw-r--r--  1 ecaldev ecaldev 21666 Feb 11  2008 xusb_xup.hex
```

Fig. 4. The `/etc/hotplug/usb` directory structure after execution of the `setup_pcusb` script

In addition the following lines are added to the /etc/hotplug/usb.usermap file:

```
Xusbdfwu 0x0003 0x03fd 0x0007 0x0000 0x0000 0x00 0x00 0x00 0x00 0x00 0x00 0x00000000
Xusbdfwu 0x0003 0x03fd 0x0009 0x0000 0x0000 0x00 0x00 0x00 0x00 0x00 0x00 0x00000000
Xusbdfwu 0x0003 0x03fd 0x000d 0x0000 0x0000 0x00 0x00 0x00 0x00 0x00 0x00 0x00000000
Xusbdfwu 0x0003 0x03fd 0x000f 0x0000 0x0000 0x00 0x00 0x00 0x00 0x00 0x00 0x00000000
Xusbdfwu 0x0003 0x03fd 0x0013 0x0000 0x0000 0x00 0x00 0x00 0x00 0x00 0x00 0x00000000
Xusbdfwu 0x0003 0x03fd 0x0015 0x0000 0x0000 0x00 0x00 0x00 0x00 0x00 0x00 0x00000000
Xusbdfwu 0x0003 0x03fd 0x0008 0x0000 0x0000 0x00 0x00 0x00 0x00 0x00 0x00 0x00000000
```

Fig. 5. The lines added by the `setup_pcusb` script to the `/etc/hotplug/usb.usermap` file

It is important to note that the `setup_pcusb` script must be executed after every OS upgrade on the on-line cluster if the upgrade removes the `/etc/hotplug/usb` directory or its content.

2.3.2 The custom suite of configuration bash scripts and programs

The TCC48 configuration software, both the JTAG based for the PuSrf device and the SystemACE based for the TPG1 and TPG2 devices, can be installed anywhere on a shared disk space of the CMS on-line cluster. For the moment of writing it is installed under the following directory:

```
nfshome0/ecaldev/DAQ/ECAL/Devel/ecal/ecalTCC/Tcc48ConfTools
```

The rest of the document will refer to the root directory of the configuration software as `${Tcc48Tools}`. The subdirectories `${Tcc48Tools}/Distribution` and `${Tcc48Tools}/Implementation` contain respectively the source files of various tools and the corresponding executables and scripts. In principle, the `${Tcc48Tools}/Implementation` directory can be deleted and subsequently rebuilt from `${Tcc48Tools}/Distribution`. It can be installed in any other place by setting corresponding variables. The user interface executables and scripts of all tools are placed under the `${Tcc48Tools}/Implementation/bin` directory. Some expert executables and scripts as well as various log files are placed in the tool specific directories under `${Tcc48Tools}/Implementation`.

The `${Tcc48Tools}/Distribution/JtagConf` directory contains various files specific to the JTAG configuration tools.

The `${Tcc48Tools}/Implementation/bin` directory contains 5 main JTAG configuration scripts (`scanScript.sh`, `progScript.sh`, `setDefRevScript.sh`, `setAnyChain.sh`, `setSlbChain.sh`) that will be described in detail later. There is also an auxiliary script `script_env.sh` that the five main scripts rely on. It is called at the beginning of each of the main scripts to define various functions and environment variables. The main scripts are also extensively using the `JtagRstByCaen` executable under the JTAG configuration software specific directory `${Tcc48Tools}/Implementation/JtagConf/bin`. As discussed in the previous section the executable is needed to generate the JTAG TRST* signal through the CAEN VME/PCI adapter.

The 5 user interface JTAG configuration scripts produce temporary iMPACT command and various SVF files and invoke the iMPACT tool in its batch mode passing it the temporary command files. During execution the iMPACT programming tool produces log files. The command and log files are kept for further analysis in case of errors. The temporary files are stored under the `${Tcc48Tools}/Implementation/JtagConf/Config/$HOSTNAME` directories, where the `$HOSTNAME` environment variable is set to the complete name of the endcap control PC that executes the script (see Table 1).

The JTAG scripts can run simultaneously on all of the 6 endcap control PCs reducing almost by a factor of 6 the time needed to program the 72 TCC48 boards. However, only one script at a time can be executed on a given control PC. To prevent concurrent script executions on a same PC, at start-up the scripts check if a temporary lock file `Tcc48JtagConfig.lock` is present under the `${Tcc48Tools}/Implementation/JtagConf/Config/$HOSTNAME` directory. If the lock file exists the script stops execution with an appropriate message. If the lock file does not exist, the script creates it, pursues execution and removes the lock file at the end.

In case of faults the scripts output a message indicating a possible cause of encountered errors. They also indicate the log files that can be consulted for more detailed information.

2.3.3 PuSrf FPGA bitstream and associated Flash PROM MCS files

It is possible to program through JTAG the PuSrf FPGA directly. This is handy for test purposes as programming of the FPGA takes only few seconds while programming of its associated Flash PROM takes few minutes. The PuSrf firmware bitstream files are usually kept under the `${Tcc48Tools}/Implementation/Bitstreams` directory.

2.3.3.1 Naming conventions for the PuSrf FPGA firmware releases

The naming convention for the PuSrf bitstream files is as follows: the filename is composed of the "`pusrfv3_`" prefix string followed by a revision id code and the file extension is "`.bit`", *i.e.* `pusrfv3_<revid>.bit`.

By convention the PuSrf firmware revision is coded by an 8-digit value where the first 6 digits are the firmware production date in the `ddmmyy` format and the last two digits represent the major and the minor revision numbers of the firmware in the *Mm* format. For example, the code `29060921` would mean that the PuSrf firmware revision 2.1 was produced on June 29th, 2009. The filename with the firmware release will be `pusrfv3_29060921.bit`.

Note that when a firmware release is downloaded into the PuSrf FPGA, its user code register keeps the 8-digit firmware revision code. The user code register can be consulted through JTAG and therefore the firmware release loaded in the PuSrf device can be determined at any time. For the firmware release in the `pusrfv3_29060921.bit` example file the value of the user code register will be `29060921`.

2.3.3.2 Naming conventions for the PuSrf Flash PROM files

The Platform Flash PROM device associated with the PuSrf FPGA can contain up to four PuSrf firmware revisions with one of them being a default revision that is loaded in the PuSrf FPGA at start-up or at user initiated reload procedure. There are four memory blocks in the Flash PROM. The size of each block is enough to keep one complete PuSrf firmware. Prior to program the PuSrf Flash PROM the set of desired firmware releases must be grouped in a file of the Intel Micro-Controller System MCS format (file extension `.mcs`). Apart from the MCS file the iMPACT tool needs three more files for the Flash PROM programming: a configuration description (file extension `.cfi`), a signature (`.sig`) and a PROM map (`.prm`).

The naming convention for the TCC48 PuSrf Flash PROM files is as follows:

```
Tcc48PuSrf_<revnum>_<revids>_<date>.{mcs, cfi, sig, prm}
```

where

`<revnum>` is a digit in the [1-4] range indicating the number of PuSrf firmware revisions; `<revids>` is an 8-character string in a format of $M_1m_1M_2m_2M_3m_3M_4m_4$, where M_i and m_i are the major and the minor IDs of the ith revision. If fewer than four versions are present in the Flash PROM the "`ef`" code (for empty field) is used to indicate the unavailable firmware revisions; `<date>` is a 6-digits string indicating the file set production date in the yymmdd format.

For example, the following set of files `Tcc48PuSrf_2_2120efef_090629.{mcs, cfi, sig, prm}` was produced on June 29th, 2009 and it contains two PuSrf firmware releases with the revision IDs 2.1 and 2.0.

The `fpromScrip.sh` script in the `${Tcc48Tools}/Implementation/bin` directory is used to produce the sets of the Flash PROM files out of a desired number of PuSrf firmware releases. The usage of the script is as follows:

```
fpromScript.sh bitstream [bitstream]...
```

Here the "`bitstream`" is the PuSrf bitstream filename without the .bit extension. There can be up to four different bitstream files in the command line argument list. For example, the command:

```
fpromScript.sh pusrfv3_29060921 pusrfv3_09060920
```

will use the `pusrfv3_29060921.bit` and `pusrfv3_09060920.bit` firmware releases in the `${Tcc48Tools}/Implementation/Bitstreams` directory to prepare the `Tcc48PuSrf_2_2120efef_090629.{mcs, cfi, sig, prm}` set of files and place it in the `${Tcc48Tools}/Implementation/Bitstreams` directory.

In addition the `fpromScrip.sh` script produces yet another file with the same Flash PROM filename convention but with the `.usr` file extension (for user codes): *i.e.* `Tcc48PuSrf_<revnum>_<revids>_<date>.usr`. The Flash PROM has a user code register and a customer code register per memory block. The registers can be accessed from JTAG. Each customer code register is used to keep the revision code of the firmware contained in the memory block. The user code register is used to keep the 8-character `<revids>` field as presented in the filenames of the Flash PROM file sets. The information in the `Tcc48PuSrf_<revnum>_<revids>_<date>.usr` file is used by the JTAG programming script to program the user code and the customer code registers of the Flash PROM.

To summarise, in order to program the PuSrf Flash PROM a set of 5 files (`.mcs, .cfi, .sig, .prm, .usr`) with the filename convention described above must be present in the `${Tcc48Tools}/Implementation/Bitstreams` directory. The `fpromScript.sh` script under the `${Tcc48Tools}/Implementation/bin` directory must be used to produce an appropriate 5-file set from a desired number of PuSrf firmware releases.

2.3.4 The scanScript.sh

The scrip is used to scan slot-by-slot a given VME crate and to determine in which slots are TCC48 boards accessible for JTAG operations, which firmware revisions are loaded in the PuSrf

FPGA devices and how many and which PuSrf firmware revisions are available in the corresponding Platform Flash PROMs.

When scanning a VME crate the scanScript.sh script attempts to read the user code register of the PuSrf FPGA to determine the firmware revision that is loaded in the FPGA. If `–l` option (for long output) is passed to the script it reads the user code register of the Flash PROM to determine the number of active revisions and their major and minor IDs. It also reads the customer codes of the memory blocks containing active firmware releases to determine full revision codes including firmware production dates. The script interprets the contents of the registers according to the coding conventions described in the previous sections and produces a user readable output.

The following is few examples of the scanScript.sh script usage:

- to get help

```
scanScript.sh –h
```

- To scan the whole crate:

```
scanScript.sh [-l]
```

- To scan a range of crate

```
scanScript.sh [-l] -s [slot_min [slot_max]]
```

- To scan a range of crate followed by some individual slots

```
scanScript.sh [-l] -s [slot_min  slot_max] slot_id [slot_id ...]
```

- To scan individual slots

```
scanScript.sh [-l] slot_id [slot_id ...]
```

The Tcc48 boards in a fully populated endcap VME crate occupy the following slots: 2, 3, 4, 5, 8, 9, 10, 11, 14, 15, 16 and 17 (see Fig. 3). A typical example of a scanScript.sh invocation would be:

```
scanScript.sh –l  2 3 4 5  8 9 10 11  14 15 16 17
```

2.3.5 The progScript.sh

The script is used to program either the PuSrf FPGAs or the Platform Flash PROM devices in a given set of TCC48 boards within a VME crate.

2.3.5.1 Programming the PuSrf FPGA

When programming the FPGA the following command line parameters are passed to the script:

```
progScript.sh -b bit_file [-s slot_min  [slot_max]] [slot_id [slot_id]...]
```

The `bit_file` parameter is the bitstream filename without the `.bit` extension. The script will look for `bit_file.bit` in the `${Tcc48Tools}/Implementation/Bitstreams` directory. The slot ID parameters follow the same convention as for the `scanScript.sh` script (see section 2.3.4). The script accepts a range of consecutive slots, a list of arbitrary slots or both.

A typical example would be:

```
progScript.sh -b pusrfv3_29060921 2
```

(Program the PuSrf FPGA of the TCC48 board in slot 2 with the bitstream `pusrfv3_29060921.bit` from the `${Tcc48Tools}/Implementation/Bitstreams` directory.)

2.3.5.2 Programming the PuSrf Flash PROM

When programming the Flash PROM the following command line parameters are passed to the script:

```
progScript.sh –f mcs_file_base_name -r rev [-l] [-s slot_min [slot_max]]
[slot_id [slot_id]...]
```

The mcs_file_base_name parameter is the base name of Flash PROM programming files. The script will look for the files `mcs_file_base_name.{mcs, cfi, sig, prm, usr}` in the `${Tcc48Tools}/Implementation/Bitstreams` directory.

The `rev` parameter indicates the firmware revision to be set as a default one. It takes values from 0 to 3.

The optional `–l` parameter, when given, forces the script to reload the PuSrf FPGA with the default firmware revision from the newly programmed Flash PROM.

The slot ID parameters follow the same convention as for the `scanScript.sh` script (see section 2.3.4). The script accepts a range of consecutive slots, a list of arbitrary slots or both.

A typical example would be:

```
progScript.sh -f Tcc48PuSrf_2_2120efef_090629 –r 0  2 3 4 5  8 9 10 11  14 15 16 17
```

(Program the PuSrf Flash PROMs in all TCC48 boards in a crate with the MCS file `Tcc48PuSrf_2_2120efef_090629.mcs` from the `${Tcc48Tools}/Implementation/Bitstreams` directory, set the firmware revision 2.1 in the block memory 0 of the Flash PROMs as a default one.)

2.3.6 The setDefRevScript.sh

The script can be used to change a default revision within the PuSrf Flash PROM and/or to reload the PuSrf FPGA with the default revision.

2.3.6.1 Setting a new default revision

When setting a new default revision in the PuSrf Flash PROM the following command line parameters are passed to the script:

```
setDefRevScript.sh -r rev [-l] [-s slot_min [slot_max]] [slot_id [slot_id]...]
```

The `rev` parameter indicates the firmware revision to be set as a default one. It takes values from 0 to 3.

The optional `–l` parameter, when given, forces the script to reload the PuSrf FPGA with the newly set default firmware revision.

The slot ID parameters follow the same convention as for the `scanScript.sh` script (see section 2.3.4). The script accepts a range of consecutive slots, a list of arbitrary slots or both.

Usually the `scanScript.sh` script with its long detailed output option is used first to identify the firmware revisions present in the Flash PROM. The script will list firmware releases present in each of the four memory blocks of the Flash PROM. The user then chooses the block ID with the desired firmware release and passes it as the `rev` parameter to the `setDefRevScript.sh` script.

A typical example would be:

```
setDefRevScript.sh –r 1 -l –s 2 5
```

(For TCC48 boards in slots from 2 to 5 set the PuSrf firmware release present in the Flash PROM’s memory block 1 as the default one and reload the PuSrf FPGA with the newly set default firmware release.)

2.3.6.2 Reload PuSrf FPAG with a default revision

To reload the PuSrf FPGA the following command line parameters are passed to the script:

```
setDefRevScript.sh -l [-s slot_min [slot_max]] [slot_id [slot_id]...]
```

The slot ID parameters follow the same convention as for the `scanScript.sh` script (see section 2.3.4). The script accepts a range of consecutive slots, a list of arbitrary slots or both.

A typical example would be:

```
setDefRevScript.sh –l  2 3 4 5  8 9 10 11  14 15 16 17
```

(Reload PuSrf FPGAs in all TCC48 boards in a crate.)

2.3.7 The setSlbChain.sh

The script is used to set the ScanSTA111 Bridge of a single TCC48 board in a transparent mode towards the board's JTAG local chain 2 with SLBs (see Section 2.1.3). This is used mainly for test purposes. The command line is as follows:

```
setSlbChain.sh slot_id
```

2.3.8 The setAnyChain.sh

The script is used to set the ScanSTA111 Bridge of a single TCC48 board in a transparent mode towards a desired local JTAG chain. This is used mainly for test purposes. The command line is:

```
setAnyChain.sh slot_id chain
```

where the `chain` parameter can take one of the following values:

- `0` or `Ace` to attain the local chain 0 (see Section 2.1.1)
- `1` or Xilinx to attain the local chain 1 (see Section 2.1.2)
- `2` or Slb to attain the local chain 2 (see Section 2.1.3)

The actions of the `setAnyChain.sh slot_id Slb` command and the `setSlbChain.sh slot_id` command are equivalent.

If for some reasons a user wants to run the Xilinx iMPACT programming tool in its GUI mode, the command `setAnyChain.sh slot_id Xilinx` has to be issued first. This will make the ScanSTA111 Bridge of the TCC48 board in slot slot_id "transparent" towards its local JTAG chain 1 with the PuSrf FPGA and Flash PROM devices.

2.4 Programming of the TCC48 SLB devices

As discussed in Section 2.2 in rear cases the firmware of the Altera devices on the SLB daughter-boards might need an update. The Xilinx USB parallel cable has to be disconnected from the BSCAN board and the Altera Blaster programmer must be connected instead. Currently the Altera blaster programmer is controlled by an *ad-hoc* laptop PC that has the Altera Quartus 8.2 programming software installed. The laptop PC is running under the Windows XP operating system. It is registered at CERN as `M60-tcc`.

Prior to attempt any connection to the TCC48 local JTAG chain 2 with SLBs, one has to assure that the SLBs are powered-up. Probably one of the simplest ways is to execute the `Tcc48SlbTest` program under the `${Tcc48Tools}/Implementation/bin` directory. To power-up the SLBs on a TCC48 board in a slot, it is enough to call the `Tcc48SlbTest` passing it the slot ID of the board as a command line option.

```
Tcc48SlbTest slot_id
```

Note that the `Tcc48SlbTest` must be executed on the CMS on-line cluster PC that controls the VME crate (see Table 1).

As in the case of the Xilinx FPGA and Flash PROM devices, the ScanSTA111 Bridge on a TCC48 board must be set to a transparent mode towards the corresponding local JTAG chain: the local chain 2 in case of the SLBs. This is accomplished by executing a JAM program with the `quartus-jli.exe` player included in the Quartus software distribution. For each slot a specific JAM program has been developed. The programs are stored in the `scansta111_slot<xx>_slb.jam` files, where the 2-digit `<xx>` field takes values from "`00`" to "`21`". The JAM program files are placed in the same directory as the `quartus-jli.exe` executable: `c:\EAO\altera\81\qprogammer\bin`.

Before executing the JAM programs the port number of the Altera Blaster programmer must be determined. For this the `quartus-jli.exe` must be executed with the `–n` command line option:

```
quartus-jli.exe –n
```

The output enumerates all Blaster programmers connected to the PC and their port numbers (Usually there is only one Blaster programmer). To set the ScanSTA111 Bridge in to the local chain 2 (SLB) transparent mode on a TCC48 the following command must be typed at the command line prompt:

```
quartus-jli.exe –c port scansta111_slot<xx>_slb.jam –a JAM
```

where `port` is the port number returned with the `-n` option and the `<xx>` field indicates the slot ID of the TCC48 biard.

Starting from this point the Altera Quartus software can be used in its GUI mode and programming of the TCC48 SLB devices can follow a standard procedure (for details refer to the Altera Quartus User Guide [8]).

At the end of the programming operations it is important to bring the selected ScanSTA111 Bridge in its normal non-transparent state (to allow SLBs on other TCC48 to be programmed). This is achieved running the `JtagRstByCaen` executable under the `${Tcc48Tools}/Implementation/JtagConf/bin` directory on the CMS on-line cluster PC that controls the VME crate.

The following are the summary of actions to be performed in order to download a firmware release in the SLB devices of the TCC48 boards:

1) Attach the Altera Blaster programmer to the `M60-tcc` laptop PC
2) Open a window with the command interpreter CMD tool on the `M60-tcc` laptop PC and change directory to the Altera Quartus installation directory: `c:\EAO\altera\81\qprogammer\bin`
3) Run `quartus-jli.exe –n` in the CMD tool and note the Altera blaster port number
4) Start the Quartus tool in GUI mode

Per VME crate:

a. Replace on the BSCAN board the Xilinx USB programmer cable by the Altera Blaster programmer cable
b. On the `M60-tcc` laptop PC open a remote shell to the CMS on-line cluster PC that controls the VME crate
c. Run in the remote shell the `JtagRstByCaen` executable under the `${Tcc48Tools}/Implementation/JtagConf/bin` directory

   Per TCC48 board

   i. Run in the remote shell the `Tcc48SlbTest` executable under the `${Tcc48Tools}/Implementation/bin` directory passing it the board's slot ID as the command line parameter: `Tcc48SlbTest slot_id`
   ii. In the CMD tool on the `M60-tcc` laptop PC, run the following: `quartus-jli.exe –c port scansta111_slot<xx>_slb.jam –a JAM`, where the port is the port number returned with the `-n` option and the 2-digit `<xx>` field indicates the slot ID of the TCC48 board.
   iii. In the Quartus GUI detect the SLB chain of the TCC48 board, assign the firmware release to the PROMs, program them
   iv. Run in the remote shell of the crate controller PC the `JtagRstByCaen` executable under the `${Tcc48Tools}/Implementation/JtagConf/bin` directory

d. Repeat the steps i. – iv. for other TCC48 boards in the crate
e. Replace on the BSCAN board the Altera Blaster programmer cable by the Xilinx USB programmer cable

5) Repeat the steps a – e for other endcap VME crates

There are several ways to simplify the SLB programming operations on the TCC48 boards. For example, automation of the per VME crate actions that does not require human intervention (such as replacing cables on the BSCAN boards) should be possible. The Linux version of the Altera Quartus programming tool can probably be installed on the shared disk space of the CMS on-line PC cluster and the programming bash scripts can be developed similar to the ones that have been devised for the Xilinx FPGA and Flash PROM devices.

2.4.1 Running of the Altera Quartus tool on a remote Blaster programmer server

The Altera Quartus installation on the `M60-tcc` laptop PC is configured to be a remote server for a Blaster programmer cable attached to the machine. It is therefore possible to attach the `M60-tcc` laptop PC to the BSCAN card of an endcap VME crate and to perform all Altera Quartus related operations on a distant user's machine installed in a more comfortable environment (*e.g.* underground control room). Running with the remote server has to be allowed in the Quartus installation on the user's machine (see [8] for how to declare remote servers). In such a configuration the `quartus-jli.exe –n` command on the user's machine will return the name of the remote server (`M60-tcc`) and the port number of the Blaster programmer attached to the server. When running the JAM programs on the local user's machine the command line arguments to the `quartus-jli.exe` executable must be as follows:

```
quartus-jli.exe –c port –c server escansta111_slot<xx>_slb.jam –a JAM
```

where the `port` and the `server` are the port number and the server name (`M60-tcc`) returned with the `-n` option.

3. THE TCC48 SYSTEMACE PROGRAMMING TOOLS

As has been mention in the Introduction section the two xc4vlx100 Virtex4 devices, referred as TPG1 and TPG2, are associated with the Xilinx SystemACE controller [4] and a Compact FLASH memory device. The later can keep up to 8 firmware revisions of both FPGAs. The SystemACE controller and the Compact FLASH are accessible via the TCC48 VME interface giving the possibility of remote firmware programming and upgrade operations for the TPG1 and TPG2 devices.

The file system supported by the SystemACE Compact FLASH is FAT. Each of the 8 possible firmware revisions is kept in a separate directory. The directories are indexed from 0 to 7. According with the TCC48 design, the default firmware revision is fixed to the one placed in the directory with the index “0”. The default firmware revision is loaded in the TPG1 and TPG2 devices at power-up or at SystemACE controller reset. When a TCC48 board is up and running, the TPGx FPGAs can be programmed with other revisions on the Compact FLASH. The operation is initiated through VME. This is useful for test purposes.

3.1 Naming conventions for the TPGx FPGA firmware releases

The TPG1 and TPG2 firmware is stored in .ace files with the following convention for their filenames

`ddmmMm`$_1$`Mm`$_2$`.ace`
where dd and mm 2-digit fields code respectively the date and months when the ACE file was produced and `Mm`$_1$ and `Mm`$_2$ 2-digit fields code the major and the minor revision IDs of TPG1 and TPG2 firmware respectively. For example, the file 06102121.ace was produced on October 6, (2009) and contains 2.1 release of both, TPG1 and TPG2, firmware.

It is foreseen that in future, when the TPGx firmware will be more stable, the naming convention will change to `yymmMm`$_1$`Mm`$_2$`.ace`, where `yy` and `mm` fields will code the year and the month of the ACE file production date.

The size of a valid TCC48 .ace file must be 7709567 bytes.

Note that due to the known limitations of the FAT system the .ace file names are restricted to the maximum of 8 characters.

3.2 File system structure of the TCC48 Compact FLASH disks

The file system structure is shown on Fig. 6. The root must contain a directory called `Initial` and an ASCII file called `xilinx.sys`. The `Initial` directory must contain 8 sub-directories: `Rev0` through `Rev7`. The `rev0` sub-directory must contain an .ace file that will be used at power-up to program the TPG1 and TPG2 FPGAs. The other `Rev{x}` directories may or may not contain .ace files. The ACE files in these directories contain optional TPG1/TPG2 firmware that can be downloaded in the FPGAs for test or debugging purposes.

```
/(root)
      Initial
            Rev0
                  ddmmMm$_1$Mm$_2$.ace
            Rev1
            Rev2
            Rev3
            Rev4
            Rev5
            Rev6
            Rev7
      xilinx.sys
```

Fig. 6. The TCC48 Compact FLASH file system structure

The content of the `xilinx.sys` file is shown in Fig. 7. It instructs the SystemACE controller that there are 8 firmware revision sub-directories in the `Initial` directory and associates an address to each of the sub-directory. The SystemACE controller downloads the firmware revision into FPGAs connected to it depending on the address presented on its external pins or the address programmed in one of its internal registers. On the TCC48 boards the address 0 is hardcoded on the external pins of the SystemACE controller. That is why the firmware in the `Rev0` sub-directory is a default one. However, it is possible to program a specific internal register of the controller with a desired address and initiate the TPG1/TPG2 firmware download process from the corresponding sub-directory.

```
#Automatically generated. PLEASE DO NOT MODIFY.
dir = Initial;
cfgaddr0 = Rev0;
cfgaddr1 = Rev1;
cfgaddr2 = Rev2;
cfgaddr3 = Rev3;
cfgaddr4 = Rev4;
cfgaddr5 = Rev5;
cfgaddr6 = Rev6;
cfgaddr7 = Rev7;
```

Fig. 7. Content of the `xilinx.sys` file

3.3 The SystemACE configuration Software

A set of standalone "C" programs and a bash script form the TCC48 TPG1/TPG2 remote programming software suite. The most commonly used executable `Tcc48AceConf` can be found under the `${Tcc48Tools}/Implementation/bin` directory. The rescue tools are placed under the `${Tcc48Tools}/Implementation/SystemAce/bin` directory. Only one rescue tool, the `Tcc48AceRawCopy` executable, will be described.

3.3.1 The `Tcc48AceConf` tool

Executing the `Tcc48AceConf` program with the `–h` command line option gives a detailed comprehensive help output with a set of the most common use case examples (Fig. 8).

The program allows to view the content of the Compact Flash, to copy one or several .ace files to desired revision directories, to load TPG1/TPG2 FPGAs with a selected firmware revision, to reset the SystemACE controller forcing the reload of the TPG1/TPG2 FPGAs with a default revision, or to perform any combination of the above actions. The `Tcc48AceConf` program takes as a command line parameter a list of VME slot IDs so that the same actions will be consequently performed on all of the TCC48 boards from the list. This greatly reduces the need of human intervention in the firmware upgrade process. Refer to Fig. 8 for more details.

The typical use of the `Tcc48AceConf` tool will be upgrade of the default TPG1/TPG2 firmware on all of the 12 TCC48 boards in a VME crate. The command line parameters in this case will be as follows:

```
Tcc48AceConf -0 path/acefile.ace -S 0 -l 0  2 3 4 5  8 9 10 11  14 15 16 17
```

For TCC48 boards in slots 2, 3, 4, 5, 8, 9, 10, 11, 14, 15, 16 and 17 the tool will copy the file `acefile.ace` under the `path` to the Rev0 sub-directory on their Compact FLASH disks (`-0 path/acefile.ace`), will print the contents of the directory once the copy operation is finished (`-S 0`) and will reboot the TPG1 and TPG2 FPGAs with the newly upgraded firmware in the directory (`-l 0`).

The progress of the copy operation is shown on the screen as an increasing horizontal bar so that the user can monitor the activity of the tool. The duration and the speed of the copy operation are measured as well. In general it takes about 50 seconds to copy a TCC48 .ace file on a Compact FLASH.

For monitoring purposes during the firmware reload the tool displays the TPG1 and TPG2 firmware revisions before and after the operation. There is a conservative 5 second delay between the order to reload the firmware and the new firmware verification action.

```
./Tcc48AceConf -h

Usage: Tcc48AceConf [-<n> ace_file] [-f] [-S a,x,<n>] [-l load_rev_num] [-R] [-v [-v]] [-d] [-h]
slotId [slotId]

-<n> file [-<n> file] - copy ace file to rev<n> directory n=[0,7]
-f                    - force flag allows creation of revision directories if does not exist
-S a,x,<n>            - show SysAce CF contents: a-all,x-xilinx.sys, <n>-rev<n> n=[0,7]
-l <n>                - load FPGAs with rev<n> n=[0,7]
-R                    - Reset SysAce and reload default revision 0
-v [-v]               - once forces debug output, twice forces verbose VME accesses
-d                    - force dummy VME accesses
-h                    - help

slotId [slotId]       - a list of board slot Ids in range [2,21]

Typical examples
  View the CF contents of the TCC48 board in the slot 7:
   Tcc48AceConf -S a 7

  View the CF contents of the Rev1 of TCC48 boards in the slots 7 11 16:
   Tcc48AceConf -S 1 7 11 16

  Download an ACE file in the revision 0:
   Tcc48AceConf -0 ../Bitstreams/08101a1b.ace 7

  Download in Rev1 and reload in slot 16:
   Tcc48AceConf -1 ../Bitstreams/08101a1b.ace -l 1 16

  Download in Rev1 and Rev2 in slot 16:
   Tcc48AceConf -1 ../Bitstreams/08101a1b.ace -2 ../Bitstreams/08101b1c 16

  Reload from Rev2 in slot 11:
   Tcc48AceConf -l 2 11

  Download to, show and reload from Rev0 in al 12 TCC48s:
   Tcc48AceConf -0 ../Bitstreams/08101a1b.ace -S 0 -l 0  2 3 4 5  8 9 10 11  14 15 16 17
```

Fig. 8. On-line help output of the `Tcc48AceConf` tool with some common examples

In average the firmware upgrade procedure takes about a minute per TCC48 board and not more that 15 minutes per an endcap VME crate. Upgrades in the 6 endcap VME crates can be performed in parallel.

3.3.2 The `Tcc48AceRawCopy` rescue tool

Usually the Compact FLASH disks are preformatted and an initial directory structure is created on a Windows or Linux PC. After that the disks are installed on the TCC48 boards. It is not possible to reformat the Compact FLASH disk *in-situ*. If for some reasons the FAT system becomes damaged the front-panel cables on the TCC48 must be disconnected, the board unplugged from the crate, the Compact FLASH removed and programmed on some PC. The whole operation is extremely delicate.

The `Tcc48AceRawCopy` rescue tool was developed to restore the damaged FAT systems of the Compact FLASH disks *in-situ* without the need of them being removed from the TCC48 boards. The program allows to a) perform a byte-by-byte copy of the Compact FLASH content from a TCC48 card into a file and b) to perform a byte-byte copy of a Compact FLASH image from a file to the Compact FLASH on a TCC48 card.

Using the `Tcc48AceRawCopy` rescue tool one can create a master copy of a healthy Compact FLASH. The master image than can be copied to the Compact FLASH with the damaged FAT system. The byte-by-byte copy operation may restore the FAT system of the Compact FLASH.

The tool can be found under the `${Tcc48Tools}/Implementation/SystemAce/bin` directory. Fig. 9 shows its on-line help output. By default, when `–w` command line option is not given, the tool reads a desired number of sectors from the Compact FLASH disk of a TCC48 board determined by the `slotId` parameter. The `–n nb_of_sectors` command line option determines the desired number of sectors. The size of one sector in the FAT system is 512 bytes. The number of sectors needed to keep a complete TCC48 .ace file is 15058 (the size of a TCC48 .ace file is 7709567 bytes – see Section 3.1). Initial sectors on the TCC48 compact FLASH keep the Xilinx SystemACE specific information, the file structure descriptor and allocation tables and the `xilinx.sys` file (see Section 3.2). Therefore reading the first 16k sectors is enough to crate a Compact FLASH image with a correct file system structure and a default TPG1/TPG2 firmware revision kept in the `Initial/Rev0` directory. Reading the first 32k, 64k or 128k sectors allows creating of Compact FLASH images with 2, 4 or all 8 firmware revisions respectively.

```
./Tcc48AceRawCopy -h

Usage: Tcc48AceRawCopy -f filename [-n nb_of_sectors] [-w] [-v [-v]] [-d] [-h] slotId

-w                    - write to CF, default is read from CF
-f filename           - file on disk
-n nb_of_sectors      - defines # of CF sectors to be treated: 1
-v [-v]               - once forces debug output, twice forces verbose VME accesses
-d                    - force dummy VME accesses
-h                    - help

slotId                - VME slot Id in range [2,21]
```

Fig. 9. On-line help output of the `Tcc48AceRawCopy` tool

A typical example of the master image creation would be:

```
./Tcc48AceRawCopy -f Slot05.raw -n 65536 5
```

Read the first 64k sectors of the Compact FLASH disk on the TCC48 board in the VME slot `5` and store the obtained image in the `Slot05.raw` file. The size of the `Slot05.raw` file must be 33554432 bytes (64k*512).

An attempt can be made to write the obtained master image to a Compact FLASH with damaged FAT system. For this the `Tcc48AceRawCopy` tool must be invoked with the `–w` command line option. As in the case of reads the number of initial sectors to be written is determined by the `–n` option.

A typical example of the Compact FLASH restore operation would be:

```
./Tcc48AceRawCopy -f CfMasterImage.raw –w -n 65536 4
```

Write the first 64k sectors from the `CfMasterImage.raw` file to the Compact FLASH disk on the TCC48 board in the VME slot `4`. If the write operation succeeds, it will create on the Compact FLASH the directory structure of the master image and will fill the first four `Initial/Rev{x}` directories with the data from the master image.

The raw read and write operations are significantly slower compared to the performance of the `Tcc48AceConf` tool. Therefore, it might be more convenient to restore only part of the 8 revisions with the `Tcc48AceRawCopy` program. Restoring of the first 16k (32k) sectors with only one (two) firmware revision(s) might be enough. The other firmware revisions can be added later on using the faster `Tcc48AceConf` tool.

4. REFERENCES

1. TCC48 [To be defined]
2. N. Almeida, P.Silva, J. C. Silva, J. Varela, "Description of the Synchronization and Link Board ECAL and HCAL Interface to the Regional Calorimeter Trigger", Version 3.3 (SLB-S), CMS IN 2005/007
3. National Semiconductor, "ScanSTA111 Enhanced SCAN bridge Multi-drop Addressable IEEE 1149.1 (JTAG) Port", April 2004
4. Xilinx, "System ACE Compact Flash Solution", DS080 (v2.0) October 1, 2008
5. Xilinx, "Platform Cable USB II", DS593 (v1.2) June 9, 2008
6. Xilinx, "iMPACT User Guide", Revision 4.1
7. Altera, "ByteBlaster II Download Cable", User Guide, Revision 1.4, July 2008
8. Altera, "Introduction to the Quartus II Software", Version 9.0
9. ASSET InterTech, Inc, Texas Instruments Inc, "Serial Vector Format Specification", Revision E, 8 March 1999
10. BSCAN [To be defined]
11. Icron Technologies Corporation, "USB 2.0 Ranger 2101, USB 2.0 Hi-Speed 100m Cat 5 Extender", Datasheet, 2008
12. CAEN, "MOD. V2718 – VX2718 – N2738, VME – PCI OPITCAL LINK Bridge", MANUAL REV. 6, 31 January 2006

5. APPENDIX I : SOFTWARE DISTRIBUTION

The root directory of the TCC48 configuration tools is called `Tcc48ConfTools`. It contains the `Distribution` sub-directory with the source files specific to the JTAG configuration tools. There are four TCC48 software tools in the distribution: The JTAG configuration for the PuSrf FPGA, the SLB tester software, the PuSrf functionality tester software called SpyTest, and the SystemACE configuration software for the TPG1/TPG2 FPGA devices. There is a sub-directory for each of the tools. The directory structure of the software distribution is as follows:

```
Distribution:
      JtagConf:
            Config:
            Make:
      SlbTests:
            include:
            src:
      SpyTests:
            include:
            src:
      SystemAce:
            SysAce:
                  include
                  src
            Tools:
                  include
                  src
            vme:
                  include
                  src
            XilCommon:
                  Include
                  src
            XilFatFs:
                  include
                  src
```

Fig. 10. Directory structure of the TCC48 configuration software distribution

Currently there is no common Makefile for the Distribution. Each software tool has to be installed manually by entering its top level sub-directory and executing the `make` utility. This creates the `Tcc48ConfTools/Implementation` directory and the tool specific sub-directory structure (with `bin`, `lib` and `tmp` directories). This also creates `Tcc48ConfTools/Implementation/bin` subdirectory and places there only most commonly used programs and scripts of the tool, the rest being kept in the tool specific `bin` directory.

5.1 Installation of the SlbTests tool

Change to the `Tcc48ConfTools/Distribution/SlbTests` directory and type `make`.

5.2 Installation of the SpyTests tool

Change to the `Tcc48ConfTools/Distribution/SpyTests` directory and type `make`.

5.3 Installation of the SystemAce tool

Change to the `Tcc48ConfTools/Distribution/SystemAce` directory and type `make`. Copy the `Tools/setAceStruct.sh` script to the `Tcc48ConfTools/Implementation/SystemAce/bin` directory.

5.4 Installation of the JtagConf tool

a) Change to the `Tcc48ConfTools/Distribution/JtagConf/Make` directory

b) Execute the `JtagRstScript.sh` script: `./JtagRstScript.sh` (this will create the `JtagRstByCaen` executable under the `Tcc48ConfTools/Implementation/JtagConf/bin` directory)
c) Copy the `fpromScript.sh` script to the `Tcc48ConfTools/Implementation/bin` directory
d) Change to the `Tcc48ConfTools/Distribution/JtagConf/Config` directory
e) Copy the `progScript.sh`, `scanScript.sh`, `script_env.sh`, `setAnyChain.sh`, `setDefRevScript.sh`, `setSlbChain.sh` scripts to the `Tcc48ConfTools/Implementation/bin` directory

Clearly the installation steps have to be automated and be included into the standard distribution procedure of the TCC48/TCC68 software.

6. APPENDIX II: QUICK START GUIDE

This section is intended as a brief reminder of the most common firmware upgrade operations. For more detailed discussions refer to corresponding sections. When applicable the command line parameters are set so that actions are performed on all TCC48 boards in a VME crate.

6.1 Preparing PuSrf Flash PROM files

a) Log on the PC that controls the VME crate and type `dev` to become the `ecaldev` user
b) Go to
`/nfshome0/ecaldev/DAQ/ECAL/Devel/ecal/ecalTCC/Tcc48ConfTools/Implementation/bin`
c) Verify that the `../Bitstreams` directory contains the desired set of the PuSrf firmware releases to be included in the Flash PROM file (files with `.bit` extension)
d) Type:
`./fpromScript bitstream0 [bitstream1 [bitstream2 [bitstream3]]]`
where the `bitstream{x}` are the base names of the bit stream files without the .bit extension. The command should produce in the `../Bitstreams` directory a set of 5 files with extensions (.mcs, .cfi, .sig, .prm, .usr) and with a base name starting by Tcc48PuSrf prefix and ending by the current date string.

6.2 PuSrf JTAG chain verification

a) Check that the USB extender power LED is on (behind the rack)
b) Check that the BSCAN board is inserted behind existent TCC48 board (behind the rack)
c) Check that the Xilinx USB programmer is attached to the BSCAN board and the Status LED is green (behind the rack)
d) Check that the "Programmable Output 0" on the CAEN VME adapter's front panel (in front of the rack) is connected to the reset input of the BSCAN board via a coaxial cable (behind the rack)
e) Important: the I/O DIP switch on the CAEN VME adapter must be set to TTL
f) Log on the PC that controls the VME crate and type `dev` to become the `ecaldev` user
g) Go to
`/nfshome0/ecaldev/DAQ/ECAL/Devel/ecal/ecalTCC/Tcc48ConfTools/Implementation/bin`
h) Type:
```
./scanScript.sh 0  2 22 3 23 4 24 5 25  8 28 9 29 10 30 11 31  14 34 15 35 16 36
17 37
```
The valid output must be of the type:

```
Slot  0 FPGA: Failed to get firmware info
Slot  2 FPGA: Date 290609 Revision 2.1
Slot 22 FPGA: Failed to get firmware info
Slot  3 FPGA: Date 290609 Revision 2.1
Slot 23 FPGA: Failed to get firmware info
Slot  4 FPGA: Date 290609 Revision 2.1
Slot 24 FPGA: Failed to get firmware info
Slot  5 FPGA: Date 290609 Revision 2.1
Slot 25 FPGA: Failed to get firmware info
Slot  8 FPGA: Date 290609 Revision 2.1
Slot 28 FPGA: Failed to get firmware info
Slot  9 FPGA: Date 290609 Revision 2.1
Slot 29 FPGA: Failed to get firmware info
Slot 10 FPGA: Date 290609 Revision 2.1
Slot 30 FPGA: Failed to get firmware info
Slot 11 FPGA: Date 290609 Revision 2.1
Slot 31 FPGA: Failed to get firmware info
Slot 14 FPGA: Date 290609 Revision 2.1
Slot 34 FPGA: Failed to get firmware info
Slot 15 FPGA: Date 290609 Revision 2.1
Slot 35 FPGA: Failed to get firmware info
Slot 16 FPGA: Date 290609 Revision 2.1
Slot 36 FPGA: Failed to get firmware info
Slot 17 FPGA: Date 290609 Revision 2.1
Slot 37 FPGA: Failed to get firmware info
```

i) In case of errors check the contents of the proposed log files. Certain type of errors requires reboot of the control PC.

6.3 PuSrf remote firmware upgrade

a) Validate the PuSrf JTAG chain as described in Section 6.2
b) If possible ensure that there is no periodic VME activity on the crate. Stop programs susceptible to perform periodic VME read/write cycles
c) Verify that the `../Bitstreams` directory contains the desired set of the PuSrf Flash PROM configuration files with the following extensions: `.mcs`, `.cfi`, `.sig`, `.prm`, `.usr` (if the files do not exist create them as described in Section 6.1)
d) Type:
```
progScript.sh -f mcs_file_base_name –r 0  2 3 4 5  8 9 10 11  14 15 16 17
```
This sets the firmware revision in the Flash PROM memory block 0 as default. The value after the `–r` command line option, that can be set to 0, 1, 2 or 3, defines the Flash PROM memory block that contains the default firmware revision. The association can be found in the `.usr` and `.prm` files. If the PuSrf FPGA must be reloaded with the new default revision add the `–l` command line option

6.4 Getting contents of the PuSrf Flash PROMs

a) Validate the PuSrf JTAG chain as described in Section 6.2
b) Type:
```
./scanScript.sh –l 2 3 4 5  8 9 10 11  14 15 16 17
```

6.5 Setting a default PuSrf firmware revision

a) Get contents of the PuSrf Flash PROMs as described in Section 6.4
b) Note the Id of the Flash PROM Block (0 through 3) that contains the firmware revision to be set as default (`def_rev_blok_id`)
c) Type:
```
./setDefRevScript.sh –r def_rev_blok_id  2 3 4 5  8 9 10 11  14 15 16 17
```
If the PuSrf FPGA must be reloaded with the new default revision add the `–l` option

6.6 Reloading the PuSrf FPAGs with the default revision

a) Validate the PuSrf JTAG chain as described in Section 6.2
b) Type:
```
./setDefRevScript.sh –l  2 3 4 5  8 9 10 11  14 15 16 17
```

6.7 TPG1/TPG2 Remote Firmware Upgrade

a) Log on the PC that controls the VME crate and type `dev` to become the `ecaldev` user
b) Go to
```
/nfshome0/ecaldev/DAQ/ECAL/Devel/ecal/ecalTCC/Tcc48ConfTools/Implementation/bin
```
c) Verify that the `../Bitstreams` directory contains the desired `.ace` file
d) Type:
```
Tcc48AceConf -0 ../Bitstreams/acefile.ace -S 0 2 3 4 5  8 9 10 11  14 15
16 17
```
If the TPG1 and TPG2 FPGAs must be reloaded with the new firmware add `–l 0` option

6.8 Getting Contents of the TPG1/TPG2 Compact Flash

a) Log on the PC that controls the VME crate and type `dev` to become the `ecaldev` user
b) Go to
```
/nfshome0/ecaldev/DAQ/ECAL/Devel/ecal/ecalTCC/Tcc48ConfTools/Implementation/bin
```
c) Type:
```
Tcc48AceConf -S a  2 3 4 5  8 9 10 11  14 15 16 17
```

6.9 Reloading of the TPG1/TPG2 with the default revision

a) Log on the PC that controls the VME crate and type `dev` to become the `ecaldev` user
b) Go to
```
/nfshome0/ecaldev/DAQ/ECAL/Devel/ecal/ecalTCC/Tcc48ConfTools/Implementation/bin
```
c) Type:
```
Tcc48AceConf -R 2 3 4 5  8 9 10 11  14 15 16 17
```

6.10 Programming of the SLB devices

1) Validate the PuSrf JTAG chain as described in Section 6.2
2) Attach the Altera Blaster programmer to the `M60-tcc` laptop PC
3) Open a window with the command interpreter CMD tool on the `M60-tcc` laptop PC and change directory to the Altera Quartus installation directory: `c:\EAO\altera\81\qprogammer\bin`
4) Run `quartus-jli.exe –n` in the CMD tool and note the Altera blaster port number
5) Start the Quartus tool in GUI mode

Per VME crate:

a. Replace on the BSCAN board the Xilinx USB programmer cable by the Altera Blaster programmer cable

b. On the `M60-tcc` laptop PC open a remote shell to the CMS on-line cluster PC that controls the VME crate and go to: `/nfshome0/ecaldev/DAQ/ECAL/Devel/ecal/ecalTCC/Tcc48ConfTools/Implementation/bin`

c. Run in the remote shell: `../JtagConf/bin/JtagRstByCaen`

   Per TCC48 board in slot `slot_id`

   i. Run in the remote shell: `./Tcc48SlbTest slot_id`

   ii. In the CMD tool on the `M60-tcc` laptop PC, run the following: `quartus-jli.exe –c port scansta111_slot<xx>_slb.jam –a JAM` where the `port` is the port number returned with the `-n` option and the 2-digit <xx> field indicates the slot ID of the TCC48 board.

   iii. In the Quartus GUI detect the SLB chain of the TCC48 board, assign the firmware release to the PROMs, program them

   iv. Run in the remote shell: `../JtagConf/bin/JtagRstByCaen`

d. Repeat the steps i. – iv. for other TCC48 boards in the crate

e. Replace on the BSCAN board the Altera Blaster programmer cable by the Xilinx USB programmer cable