LPHE 2005-07

Creditcard PC softwareguide

Internal Note

Issue: 1
Revision: 0

Reference: LHCb 2004-058 ECS
Created: July 25, 2004
Last modified: July 27, 2004

Prepared by: Niko Neufeld, EPFL & CERN

Abstract

This document describes the available standard libraries and programming tools for the LHCb CCPC glue-card. It gives instructions on how to compile one's own projects for the CCPC.

Please check for updates of this document on the official web-page http://lhcbonline.cern.ch/ccpc/development

Document Status Sheet

1. Document Title: Creditcard PC softwareguide			
2. Document Reference Number: LHCb 2004-058 ECS			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
1	0	July 26, 2004	Initial release
Draft	0	July 25, 2004	First version

Contents

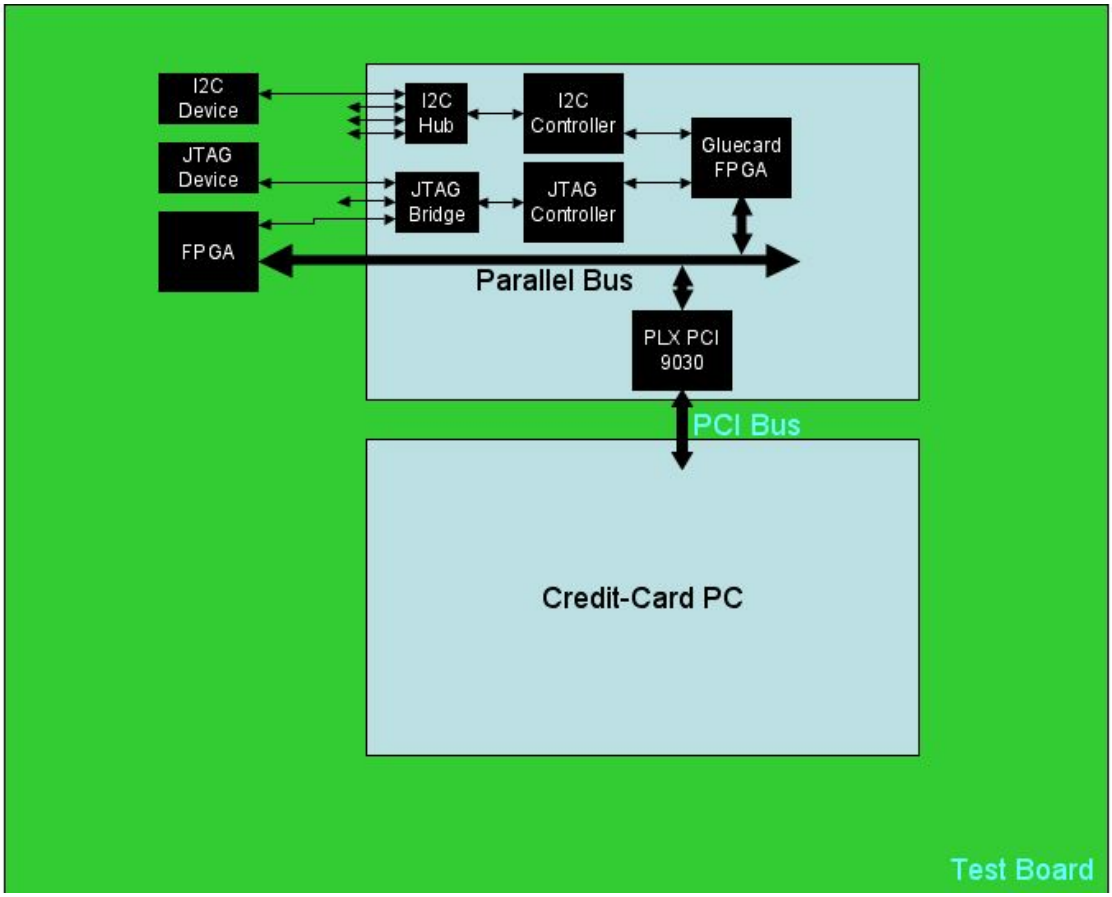

Figure 1 Schematics of the CCPC hardware and glue-card

1 Introduction

This document describes programming the CCPC and glue-card. It describes the API for the various functionalities and some command line utilities. The description in this note pertains only to the "Genoa" glue-card, not to the tiny glue-card used in the TFC system. For this please refer to [5]. Manuals referred to in this guide can be found on the CCPC home page, starting from `http://lhcbonline.cern.ch`.

1.1 The hardware

The hardware consists of the CCPC which is a commercial embedded PC produced by Digitallogic, Switzerland [6]. This PC is designed around the Elan 520 Micro-controller [2]. This chip is a i486 compatible processor, including a PCI bus, an Intel Ethernet chip, standard legacy devices (keyboard, mouse, serial line) and a hardware watchdog. The CCPC is accessed via its LAN connection. Details on the implementation of the CCPC can be found in [4]. The interface to electronics devices found on the carrier-board is provided by the "glue-card". The glue-card uses a PLX-PCI9030 bridge to create a local bus from PCI. The PLX provides in addition support for interrupts and several GPIO lines. A small FPGA is used to map the control registers of a JTAG controller and a I^2C-controller into the address space of the local bus. This FPGA can be reprogrammed via the parallel port of the CCPC. Details can be found in 3. The glue-card contains a JTAG and an I^2C-hub to provide 3 independent JTAG chains and 4 independent I^2C-buses respectively. A schematic view of the hardware is shown in Figure 1. The libraries described in this guide, allow accessing all the resources mentioned above.

1.2 Basesystem

The basesystem of the CCPC is Linux. It is a stripped-down version of the CERN customized Redhat distribution 7.3.4. The installation instructions for the basesytem can be found on http://lhcbonline.cern.ch/baseinstall. The CCPC boots via DHCP and TFTP and then proceeds to mount its root filesystem via NFS. The system is preconfigured to program the glue-card FPGA and to start the device driver for the PLX chip, which is the basic interface to all resources on the glue-card.

Software distribution for the basesystem is done using a central web-server at CERN: http://lhcbonline.cern.ch. The automatic package distribution is based on yum http://linux.duke.edu/projects/yum.

The CCPCs boot via the network and since they are disk-less their entire root file-system is served via NFS from a server. In the following we will call this server the *host*. The host also contains all the files for running the glue-card software. While compilation is typically done on the host, the programs obviously must be run on the CCPC.

1.3 Conventions in this manual

On several places in this manual examples are given to do things on the command line. Some things must be done on the host, which houses all files. Examples for commands to be given to the host will look like this:

```
host% make clean
host% make
```

Sometimes root privileges are necessary this is indicated by a # sign. Example:

```
host# make install
```

Commands to be used on the CCPC itself are marked like this:

```
ccpc% i2cscan
ccpc% ./mytest
```

Again commands requiring root priveledges will be marked with a #.

```
ccpc# /etc/init.d/plx restart
```

Sometimes references is made to standard UNIX commands like this `ls(1)`. The number in brackets refers to the section of manual. You would in this case type:

```
ccpc% man 1 ls
```

2 The PLX device driver

The plx device driver is started automatically at boot-time. It is a kernel module called `plx.o`. Currently it is the only driver module. As more functionality will be moved into drivers, be careful always to stick to the API provided by the libraries and do not to try to use the device driver directly.

2.1 Reloading the PLX device driver

To reset the PLX bridge and reset all mappings done by the driver, it can be useful to reload the driver.

```
ccpc# /etc/init.d/plx restart
```

Note that this will not work as long as any process is using the driver. You can find (and kill) these processes using `fuser(1)`.

3 The Gluecard FPGA

The glue-card FPGA is Xilinx XCS05. It interfaces to the I^2C controller, the JTAG controller and the bus-switch. The XCS05 needs a special firmware which is usually automatically loaded at startup of the PC. In case of doubt it can be useful to re-flash the FPGA to force a full reset. The reset is done using GPIO 8. *GPIO must therefore not be used in any user application*. Reset is done using

```
ccpc# /etc/init.d/xilinx restart
```

The FPGA is reprogrammed automatically at each reboot. It should actually rarely be necessary to touch it. Please avoid writing and reading the first page on the local bus `0x000 to 0xfff`, since this is decoded by the glue-card. This page is always mapped to local space 0 and must not be touched.

4 Libraries

The libraries have a layered structure. The basic library is the local bus library `lblib`. It interfaces directly to the device driver plx. It offers functions to program the PLX registers to read and write local bus addresses and to remap local memory regions into user space.

Closely linked with lblib is glue-card library, `gluelib`. It offers basic functions to initialize the glue-card, write and read the GPIO lines and to load a carrier board library. A carrier-board library interfaces the generic tools to the specific carrier board (for example to enable an I^2C-bus a carrier-board could first need to write to a specific register on an FPGA using the local bus). This is described in 4.3.

The i2c library `i2culib` offers functions to read, write and scan the 4 I^2C-buses. It interfaces to the PCF8584 I^2C-controller of the glue-card.

The JTAG library allows to do JTAG operations like IRSCAN and DRSCAN on the 3 JTAG chains. You can do boundary scan operations using the TI and the SCANSTA 111 JTAG hub of the glue-card. The layering of the libraries is shown in Figure 2.

4.1 Local bus library: lblib

The local bus is created by the PLX PCI9030 bridge described in [3]. The PLX creates a 24-bit address 32 bit data bus, which can be operated in in 8, 16 or 32 bit mode. The whole address space is a 256 MB flat range of byte addresses. Up to four address ranges (windows) can be defined, which can have different properties. In addition there are four chip selects, which can be associated with an address space. An important limitation of the PLX is, that the starting address of a local address space must an integer multiple of its size. Address spaces may overlap, as may chip select ranges, care for correct address decoding in such a configuration is up to the user. The library will try to check that these constraints are satisfied and return an error, but this is not 100% fool-proof.

An address space is initialized by calling lb_init_s(). Currently a local address space cannot be re-initialized. This requires starting and stopping the driver 2.

As an example lets see how to initialize the local bus and then read and write a few words. The example is taken from the initialization of the TELL1 board. This card uses the address range from 0x20000 to 0x0fffffff to read data from the FGPAs and buffers. Due to the restrictions of the PLX mentioned above we must map a bus of this size to start at local bus address 0x00000. However we tell the driver to not pass accesses to addresses below 0x20000 to the memory mapped to *this* area of the local bus.

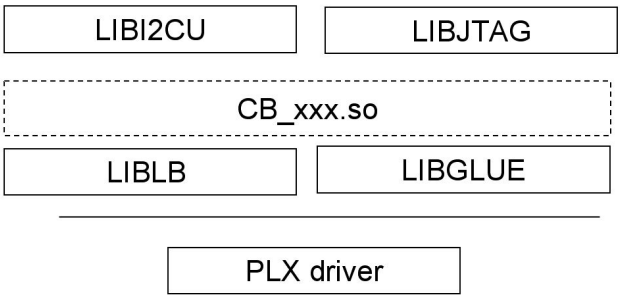

Figure 2 Layering of the CCPC libraries. The carrier board library typically also configures all local address spaces except 0, which is configured by the glue library.

Each local address space has also some bus protocol characteristics which we pass directly as flags to the device driver. The description of these flags can be found in [3].

```
4  #include <gluelib.h>
5  #include <lblib.h>
6  #define SL  0x1000000    /* base of the SyncLink FPGA on the Tell1 */
7  #define AUTO_RESET_REG_SL  124 /* offset of reset reg of SyncLink */
```

We need also a function from the glue library because the glue-card has a bus-switch to isolate the localbus from the carrier-board. This bus-switch must be closed before any access to local bus addresses outside the glue-card are done.

```
8     lb_desc_t bd;
9     u_int32_t w;
10    int rc = 0;
11    /* map TELL1 FPGAs to local address 0x0000000 */
12    bd.space = 1; bd.local_address = 0;
13    bd.local_effaddr = 0x20000;   /* ignore accesses to the first
14    0x20000 bytes on this bus */
15    bd.size = 0x10000000; bd.width = LB_WIDTH_DWORD;
16    bd.chipselect = 1; bd.phys_addr = 0xe0000000;
17    bd.flags = LB_DESC_READY_ENABLE;
18    if (lb_init_s(&bd)) return -1;
19    if (glue_enable_lb()) return -1;  /* close the bus-switch */
```

```
20    rc = lb_write_word(SL + AUTO_RESET_REG_SL, 0x1); /* tell FPGA to
21    reset */
22    rc |= lb_read_word(SL + AUTO_RESET_REG_SL, &w);
23    /* the register should now be 0 */
24    return rc;
```

Typically code like this will be executed only once during the life-cycle of a program. Moreover it is very likely that it is part of the carrier-board library, see below 4.3. In that case it will be implicitly done by a call to either `cb_init()` or more generally `glue_default_init()`.

4.2 Gluecard library: gluelib

The glue-card is controlled via a Xilinx FPGA, which is mapped to a 8-bit wide region of 256 bytes at the bottom of the local address space. The glue-card FPGA uses chip select 0, which can therefore not be used by any other application. Conventionally the glue-card is mapped to the first 4 kB (addresses 0x000 to 0xfff) of the local bus address space.

The glue-card must be initialized before first using. This is usually done by calling `glue_default_init()`. This will tri-state the I^2C and JTAG controllers and open the local bus switch, thus severing any link between the carrier-board and the glue-card. Therefore, before the local bus can be used the local bus must be enabled, i.e. the bus switch closed, this is done by calling `glue_enable_lb()`. If the initial closed state of the glue-card hardware should be reentered `glue_close_all()` must be called.

Calling `glue_default_init` will also initialized the carrier-board and load the corresponding library, if the environment variable GLUE_CBLIB is defined. Thus the following code is sufficient to start working with the glue-card.

```
1  #include <gluelib.h>
2
3  if (glue_default_init()) exit(1); /* can't work without glue-card */
4  glue_enable_lb();
```

The library has also functions to selectively enable the JTAG and I^2C-controllers. See the reference for further information.

4.3 Carrier-board libraries

The CCPC software provides some generic tools like i2cscan, jam, etc.... Different carrierboards might use different ways to activate the resources, which these commands are expecting on the carrierboard. Example: a testboard is using one of the GPIOs to enable an I2C device. This GPIO has to be enabled *before* attempting to scan the I2C bus. The glue library cannot do this, because this might differ from board to board. For this reason carrierboard libraries are introduced. They implement a simple API of four functions and are loaded dynamically by the generic tools. The tools look for the library at a place pointed to by the GLUE_CBLIB environment variable. The API for the carrier libraries is described in the appendix. We give an example for using the carrier library for the TELL1 board here:

```
1  \#include <gluelib.h>
2  cbload(NULL); /* library will be loaded at name define in GLUE_CBLIB */
   if ((*cb_init)() {
4    fprintf(stderr, ''Failed to init carrier board'');
     exit(1);
6  }
   (*cb_init_i2c)(n); /* make i2c bus number n accessible */
8
```

Normally it is not necessary to call these functions directly. Once the carrier-board library is defined, it is sufficient to have it implicitly called by `glue_default_init()`.

The cb_tell1 library actually ships with the gluelib package. It is hence sufficient to do like this:

```
ccpc% export GLUE_CBLIB=/usr/local/lib/cb_tell1.so
ccpc% i2cscan
```

4.4 I2C library: i2culib

The I2C controller on the glue-card can drive 4 different I2C channels, which in principle can be operated at different voltages (for details see [4]). The I2C library allows to write, read and scan. It also allows doing combined write and reads (separated by a restart condition on the bus). It must be initialised by i2c_init().

As an example assume you have an 8-bit I^2C-EEPROM sitting at address 0x50, bus 0. You want to write and then re-read 3 bytes. In the typical I^2C way you would write first the internal address and then the bytes to be written from this internal address on (the internal state-machine will update the write-pointer automatically until a stop condition is issued). Likewise for reading you would first write the internal address then issue a repeated start condition and then read the bytes, again the internal state machine will update the read-pointer until the stop condition is issued. In "C" this would look like this:

```
1  #include <i2c.h>
2
   /* assume glue-card and carrier board already initialised */
4  u_int8_t addr;
   u_int8_t wbuf[3], rbuf[3];
6
   wbuf[0] = 0x1; /* the internal address */
8  wbuf[1] = 0xcc; wbuf[2] = 0xce; /* some data */
   if (i2c_init(0)) return -1; /* if the glue-card initialised properly
10                                the return-code need not be checked */
   i2c_write(I2C_ADDR(0, 0x50), wbuf, 3);
12 i2c_writeread(I2C_ADDR(0, 0x50), wbuf, 1, rbuf, 3); /* we write only
                                                          one byte: the internal a
```

There are similar functions to read and scan[1]. Most of the functionality exists also as command line tools 6.

4.5 JTAG library: JTAGlib

The JTAGlib is a C library which make possible to send data to the JTAG Controller LVT8980. This Library exposes all of the controller function, and other function in order to control controller registers.

4.5.1 Configuration Function

Init the JTAG Controller	:	int JTAGInit(void)
Select Discrete Mode	:	void JTAGDiscrete(void)
Select Gated Mode	:	void JTAGGated(void)
Select a chain	:	int SelectChain(int chain)
Reset a chain	:	LbJTAG_ChainReset(int chain)

[1]Scanning means putting the address on the bus and waiting for an acknoledgement. Regardless of the result a stop condition will be issued, nothing will be written or read

4.5.2 Discrete Mode Function

Send a value : void JTAGDiscrVal(unsigned char val)
Read a value : int JTAG_TDIVal(void)

4.5.3 Gated Mode Function

All the controller operation can be done with this function in gated mode:

```
int JTAGOperation
    (
    int opcode, int endstate,
    int nbitsOut,unsigned char *bitsOut,
    int nbitsIn, unsigned char *bitsIn
    );
```

opcode — give the operation code

endstate — give the final state after the operation

nbitsOut — the number of bits out

bitsOuts — a pointer on an array of bits

nbitsIn — the number of bits in

bitsIn — a pointer on an array of bits

List of operations :

OPCODE	EXECUTE	USES
OPCode_NULL	NULL	N/A
OPCode_EXEC_RUN_TEST	run test	counter
OPCode_ASPScan_Input	input only ASP scan	counter, TDI
OPCode_ASPScan	ASP scan	counter, TDI, TDO
OPCode_ASPScan_Output	output only ASP scan	counter, TDO
OPCode_StateMove	state move	N/A
OPCode_StateJump	state jump	N/A
OPCode_IRScan	instruction register scan	counter, TDI, TDO
OPCode_DRScan	data register scan	counter, TDI, TDO
OPCode_IRScan_Input	input only IRScan	counter, TDI
OPCode_DRScan_Input	input only DRScan	counter, TDI
OPCode_IRScan_Output	output only IRScan	counter, TDO
OPCode_DRScan_Output	output only DRScan	counter, TDO
OPCode_IRScan_ReCirc	recirculate IRScan	counter, TDI
OPCode_DRScan_ReCirc	recirculate DRScan	counter, TDI

Is also possible, in gated mode, to do IRSCAN or DRSCAN selecting the good chain with:

int LbJTAG_IRScan

(
int chain, int nbitsOut,
unsigned char *bitsOut,
unsigned char *bitsIn
)

int LbJTAG_DRScan

(
int chain, int nbitsOut,
unsigned char *bitsOut,
unsigned char *bitsIn
)

4.5.4 jamlib: a library to program FPGAs

jamlib is a modified version of the ALTERA jam player [1], which is used to program FPGAs. It builds on top of JTAGlib. This library makes it possible to use jam player in C programs with the function jam_prog_fpga in cluding *jam.h*. this function sends a Jam STAPL program to the JTAG controller.

```
jam\_prog\_fpga(char * JAM\_FILE\_PATH, char * ACTION, int CHAIN);\\
```

example: ./jam_prog_fpga(”idcode.jam”,”read_idcode”,1)

This function will execute the jam program by calling the function read_idcode.

You can also use the binary version of the library by including *jbi.h*

```
jbi\_prog\_fpga(char * JBC\_FILE\_PATH, char * ACTION, int CHAIN);\\
```

example: ./jbi_prog_fpga(”idcode.jbc”,”read_idcode”,1)

These tools exist also on the command line. This is described below 6.

5 Software development

Software development with the libraries described in this guide is easy. It can be done either directly on the CCPC or on the NFS server where the software is located. The include files for the libraries are installed along with the binaries[2]. All software is located per default under /usr/local on the Creditcard PC. The first thing to do is to define the environment variable CCPCROOT to point to the root file system of the CCPC. Assume this is done and our program is called mytest.c and uses the i2c and localbus libraries. The makefile would look like this:

[2]This is a slight violation of the general principle to split the packages in pure binaries and development versions.

```
1  CC=$(CCPCROOT)/usr/bin/cc
2  LD=$(CCPCROOT)/usr/bin/ld
   SOURCES:= mytest.c
4  OBJECTS:= $(patsubst %.c,%.o,$(SOURCES))
   INC=-I. -I$(CCPCROOT)/usr/local/include  ,
6

8  CFLAGS+=-Wall -Wstrict-prototypes -g  -march=i486 \$(INC)
   LIBDIR=$(CCPCROOT)/usr/local/lib
10 LD_LIBRARY_PATH+=$(LIBDIR)
   export LD_RUN_PATH:=$(LIBDIR)
12 LDFLAGS=-L$(LIBDIR) -lglue -li2c -llb

14 all: mytest

16 mytest: $(OBJECTS)
           $(LD) $< -L$(LIBDIR) -lglue -li2c -llb
```

The first two definitions ensure that even when you compile on the host (much faster) you will use the right compiler and linker. The next three lines are standard and ensure that you use the includes from the CCPC root. The CFLGAS must include the -march=i486 option, to generate the correct binaries for the CCPC. The next three lines ensure that the linker will pick up only the CCPC libraries. Finally in the LDFLAGS line the necessary libraries for mytest.c are included.

The program itself could look something like this:

```
1  #include <lb.h>
2  #inlcude <gluelib.h>
   #include <i2c.h>
4
   glue_default_init();
6  glue_enable_lb();
   glue_enable_i2c();
8  if (i2c_scan(I2C_ADDR(0, 0x50), I2C_SCAN_WRITE) == I2C_OK) {
       printf(''Found device at address 0x50'');
10 }
   lb_read_word(0x1000, &w);
12 printf(''I read \%x at local bus address 0x1000'', w);
```

Uninteresting detail has been omitted.

Complete files can be found on hrefhttp://lhcbonline.cern.ch/developmenthttp://lhcbonline.cern.ch/ccpc/de

5.1 Using the standard libraries

It is recommended to use the standard libraries. Like this future changes, in particular in the low-level parts, will not affect your programs. Using the libraries needs just including the appropriate header file and afterwards linking against the correct library. The default commands for doing so are given in Table 1

The header files would be used typicall using `# include <lb.h>` and the libraries using `-llb`[3].

A detailed HTML reference of the libraries can be found on http://lhcbonline.cern.ch/ccpc/doc.

[3]For special purposes also non-shared libraries exist, which can be used for static linking

Package name	header file	library file
lblib	`lb.h`	`liblb.so`
gluelib	`glue.h`	`libglue.so`
i2culib	`i2c.h`	`libi2cu.so`
JTAGlib	`JTAG.h`	`libJTAG.so`
carrier lib	`cblib.h`	`cb_xxx.so`

Table 1 Package-names, header files and library files for programming the CCPC

5.2 Working with development versions of standard software

If a development version of some package is used it should be installed in the home directory of the cc user. The tarballs of the packages can be found on http://lhcbonline.cern.ch/ccpc/export

Example: assume you want to use a development version of the i2culib. The tarball you downloaded from lhcbonline is called i2culib-5.1.3.tar.bz2. The steps to do are now the following:

```
host% wget lhcbonline.cern.ch://export/i2culib-5.1.3.tar.bz2
host% tar -jx < i2culib-5.1.3.tar.bz2  # unpacks the archive
host% cd i2culib-5.1.3
host% make
```

optionally, if you are sure it works :-),

```
host# make install
```

If you omit the last step, you have to modify the makefile of your own project such that it takes the include files from the new directory. Typically something like

```
-I~/i2culib-5.1.3
```

must be added in the makefile. Likewise to use the new libraries you must add

```
export LD_LIBRARY_PATH=~/i2culib-5.1.3:$LD_LIBRARY_PATH
```

so that the dynamic linker will use the new version.

6 Commandline utilities

Commandline utilities are little programs (usually) written in C, which provide access to basic operations on the CCPC from the command line. They can be used for simple tests or to incorporate them in scripts.

Most of them display a help message when entered without any arguments. Manpages are still missing.

6.1 General gluecard utilities

6.1.1 setupglue

Initialises and resets the glue-card and opens the busswitch. The glue-card is then electrically disconnected from the carrier board.

```
Usage:
cpcpc% setupglue
```

6.1.2 gpio_enable, gpio_disable, gpio_read

Set a specific gpio to a certain level or disable it. This usually means setting it back to its other function[4]

```
Usage:
ccpc% gpio_enable 0 1 1
```

This will enable GPIO 0 as an output and set the level to 1.

6.2 I2C utilities

The I^2C take the address and data arguments in hexadecimal. The number of bytes to read and the bus number are in decimal.

6.2.1 i2cread, i2cwrite

Read/write from a specific i2c address

```
ccpc% i2cwrite 0 0x50 0xcc
ccpc% i2cread 0 0x50 1
0xcc
```

Write 0xcc to the device at address 0x50 on bus0 and then read back one byte from this address.

6.2.2 i2cwread

Many I^2C devices use indirect addressing. That means that to read one first writes an internal address and then reads from the register. The two operations are separated by a *repeated start* condition. This cannot be emulated by a i2cwrite followed by an i2cread because both are terminated by a stop condition on the I^2C bus. For writing one uses i2cwrite as above.

```
ccpc% i2cwrite 0 0x54 0x10 0xfe
ccpc% i2cwread 0 0x54 0x10 1
```

Write 0xfe to the internal address 0x10 of device at I^2C address 0x54 bus 0 and then convince yourself that the write worked.

Note: it is not possible to write more than one byte before the repeated start condition (the 0x10 above). If that is required you have to modify i2cwread.

6.2.3 i2cscan

Just check if a device is responding. Do not actually read or write something. Mind that buggy I^2C devices can be confused by this. Complain to the manufacturer if you find something like this!

```
ccpc% i2cscan
Scanning devices bus 0 device 0x50
.
.
Found 10 devices
```

[4]The PLX PCI9030 has 9 GPIOs in total. However most of the are used either as GPIO or for other (control)-signals. Details can be found in [3].

6.3 Local bus utilities

These tools rely of course on the carrier board libraries 4.3, because they assume that the bus characteristics are setup correctly there.

6.3.1 lbread and lbwrite

```
ccpc% lbwrite 32 0x1000 0xfeedbabe
write 0xfeedbabe to 0x1000
ccpc% lbread  32 0x1000
Read 0xfeedbabe from 0x1000
```

Obviously this will only work if your board has a 32-bit register at 0x1000. The first argument (32) is the width of the access in bits. It can be also 8 or 16.

6.4 JTAG utilities

The only tools for JTAG until now are `jtagscan` and `jam`.

6.4.1 jtagscan

Scans all three JTAG chains, prints out the number of devices and, if supported by the device, its IDCODE.

6.4.2 jam and jbc

Allows to program FPGAs on the selected chain by sending a STAPL file. STAPL files contain a number of *actions*. One action must be defined using the -a switch of `jam` or `jbc`[5]. At the time of this writing there is a problem with some of the compiled STAPL files, which makes the programming fail. This has been observed with ALTERA EPC16 devices. In this case you have to fall back to the ASCII version and use `jam`:

```
cccp% jam -c1 -w1 -aPROGRAM tell1.jam
```

The Creditcard PC is not very fast, you will need some patience. . . .

[5] completely equivalent, except that the byte-code compiled version of the STAPL files is used

7 References

[1] ALTERA. Jam player documentation.

[2] AMD. *ELAN 520 manual*. http://www.amd.com.

[3] PLX Corporation. *PCI 9030 Data Book*, 2003.

[4] Flavio Fontanelli, Beat Jost, Giuseppe Mini, Niko Neufeld, Ramy Abdel-Rahman, Kuno Rolli, and Mario Sannino. Cc-pc gluecard application and user's guide. LHCb note LHCb 2003-098, CERN, 2003.

[5] Z. Guzik and R. Jacobsson. Glue-light - as simple programmable interface. Technical Report LHCb 2003-056.

[6] Digital Logic. *Smartmodule 520*.