LHCb 2003 – 056 Control
EDMS document #581368
March 11, 2005, revision 2

"Glue *light*"

A simple programmable interface between the Credit Card PC and board electronics

Z. Guzik and R. Jacobsson

ABSTRACT

The LHCb experiment control interface to electronics boards located in the counting rooms is based on a commercial Credit Card PC from Digital Logic. In order to interface the Credit Card PC to the board electronics, a simple intermediate mezzanine card "Glue *light*" has been designed. It is based on a single FPGA which emulates a PLX 9030 and, in addition to the standard PLX Local Bus, forms a JTAG and an I^2C bus.

The "Glue *light*" card has been successfully used on all the TFC modules since the beginning of the prototyping of the TFC system, and it is in use in the prototyping of the L0 muon trigger electronics.

This document describes the implementation of the "Glue *light*" board. It also serves as a technical reference.

LHCb Technical Note

Issue: LHCb 2003 – 056 Control, revision 2
EDMS: #581368
Reference: LHCb ECS
Created: 5 June 2003
Last modified: 11 March 2005

Prepared By: R. Jacobsson, CERN, Geneva, Switzerland
Z. Guzik, IPJ, Swierk, Poland
LHCb Online Group

INDEX

1 Introduction

The electronics boards in LHCb will all be controlled from the LHCb Experiment Control System (ECS)[1]. In order to access the actual board resources, an ECS interface will be located on each board. The ECS interface is connected to the control network and performs directly all programming, configuration, control and monitoring of each electronics board. Three ECS interfaces have been proposed for different applications. The SPECS (Serial Protocol for the Experiment Control System)[2] and the ELMB (Embedded Local Monitor Board) [3] will be used for electronics boards situated in the radiation area close to the experiment. For the electronics situated in the counting rooms behind the shielding wall, the ECS interface is based on a commercial Credit Card PC (CCPC) with Ethernet from Digital Logic, AG, Switzerland [4].

In order to facilitate implementing the Credit Card PC and interfacing to it, it has been proposed to develop a small intermediate mezzanine card "Glue Card" [4]. The glue card would be connected to the CCPC PCI bus [5] and form a standard set of simple interfaces needed by all the different electronics boards, such as a parallel board bus, and I^2C [6] and JTAG [7] busses. The standard set of interfaces have been defined in terms of a connector with a specified pin configuration.

Following this idea, a simple programmable "Glue *light*" board has been designed. The board is entirely based on a single FPGA which is interfaced with the PCI bus of the CCPC and emulates a PLX 9030 [8]. Besides providing the standard PLX Local Bus, the FPGA also generates an I^2C and a JTAG bus.

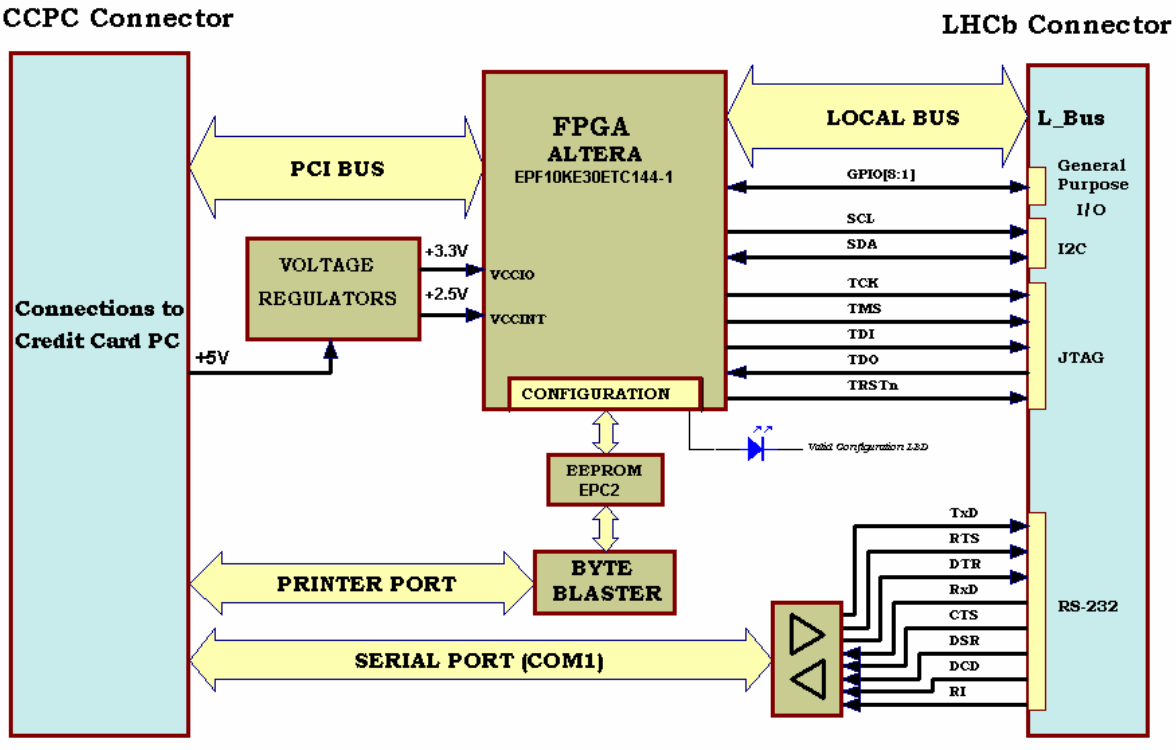


Figure 1: Logical block diagram of the "Glue *light*".

2 Implementation

2.1 Architecture and physical layout

Figure 1 shows a logical block diagram of the "Glue *light*" board. The architecture is centred around an FPGA from Altera (currently a FLEX *EPF10KE30ETC144-1*). The tasks of the FPGA are:

- providing a PCI 33 MHz target instantiation
- generating the PLX Local Bus functions, hereafter referred to as "local bus"
- instantiating an I2C bus
- providing a JTAG bus
- providing eight General Purpose I/O lines

Since the "Glue *light*" is based on a programmable device, its functionality may be tailored to specific needs.

The only supply voltage of the board is +5V. All other voltages are made on the board using regulators. The FPGA core is powered by +2.5V, and +3.3V is used for all I/O interfaces of the FPGA.

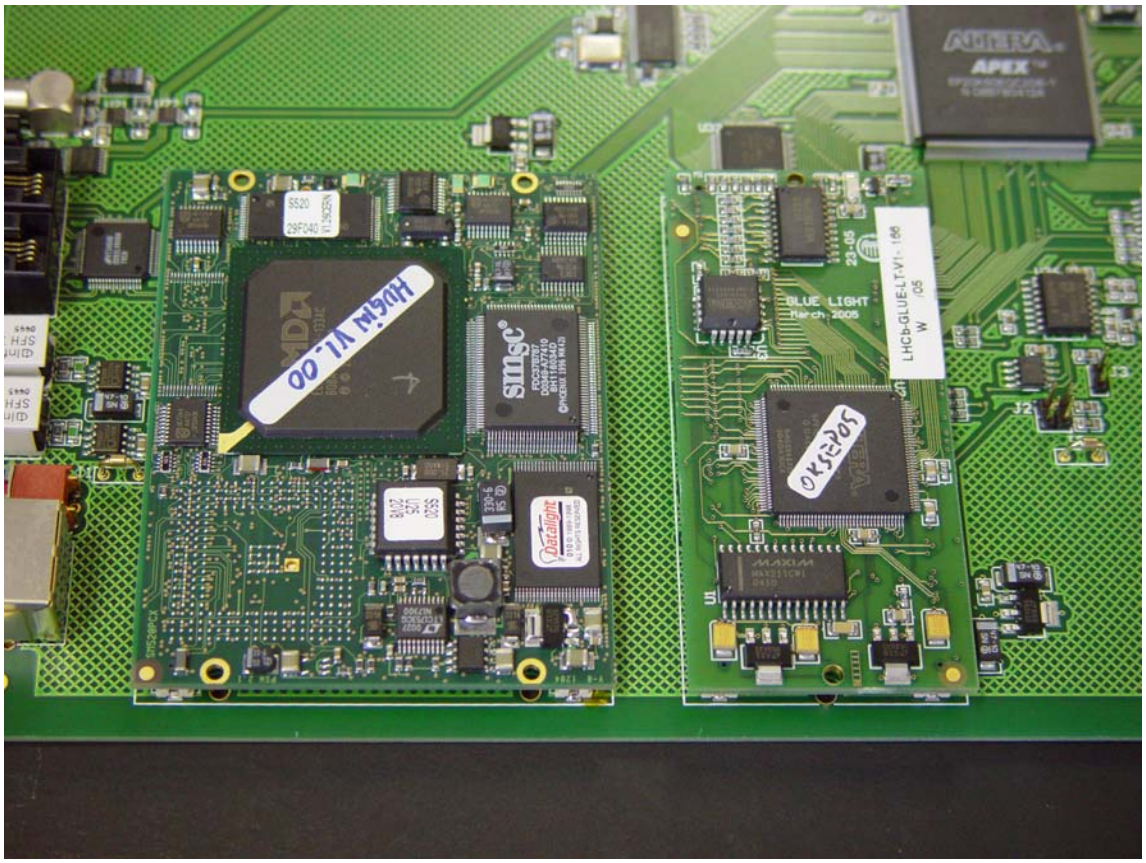


Figure 2 : Picture of the "Glue *light*" board (right) together with the Credit Card PC (left).

Figure 2 shows a picture of the "Glue *light*" together with the Credit Card PC. The actual dimensions of the board are shown in Figure 3. The board is plugged to the motherboard with two Molex connectors, each having two rows of 120 pins. Seen from top, the left connector (CCPC) is designated for connections to the Credit Card PC, while the other (LHCb) outputs the standard set of simple interfaces to the motherboard. The pin configuration of the LHCb connector and the CCPC connector are given in Appendix A and B, respectively.

The footprint is fully compatible with the one proposed in [4] with the exception of:

- the middle connector (J1) was dropped because it was found unnecessary
- the LHCb connector on the "Glue *light*" is currently slightly simplified in that it contains only one I^2C instead of the four proposed and one JTAG bus instead of three (Appendix A)
- only the necessary connections between the J2 connector of the CCPC and the "Glue *light*" have been routed (see Appendix B)

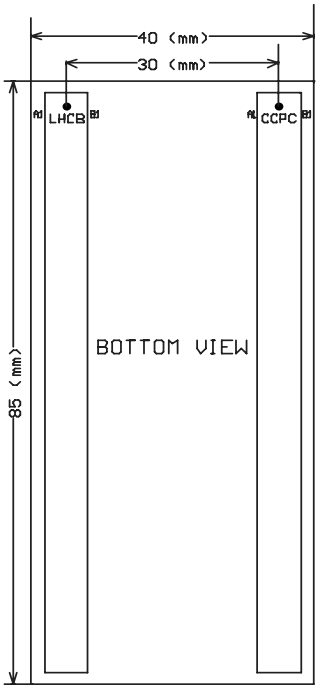


Figure 3 : Dimensions of the "Glue *light*" board.

Table 1 : Implemented PCI bus commands.

CBEN[3..0]	Bus Command Cycle	Note
0110	Memory Read	fully implemented
0111	Memory Write	fully implemented
1010	Configuration Read	fully implemented
1011	Configuration Write	fully implemented
1100	Memory Read Multiple	substituted by Memory Read
1110	Memory Read Line	substituted by Memory Read
1111	Memory Write and Invalidate	substituted by Memory Write

2.2 PCI Interface

The PCI interface implemented in the FPGA is compliant with the PCI Local Bus Specification Revision 2.2 [5]. The target supports a 33 MHz and 32-bit PCI bus.

Table 1 summarizes the PCI bus commands that are supported by the "Glue *light*" board. Table 2 presents the PCI Configuration Registers. These registers are accessed by the commands "Configuration Read" and "Configuration Write". Only the shaded fields are implemented. Read accesses to the other fields always return zero. The Configuration Registers can be accessed either in byte, word or double word transfers.

The first two Base Address Registers (BAR0 and BAR1) are reserved for system use. BAR2 holds the base address of the local resources described below. Table 3 shows the values assigned by default to the Configuration Registers.

Table 2 : PCI Configuration Registers. Shaded registers have been implemented.

Address	31 .. 24	23 .. 16	15 .. 8	7 .. 0
00h	Device ID		Vendor ID	
04h	Status Register		Command Register	
08h	Class Code			Revision ID
0Ch	BIST	Header Type	Latency Timer	Cache Line Size
10h	Base Address 0 (BAR0)			
14h	Base Address 1 (BAR1)			
18h	Base Address 2 (BAR2)			
1Ch	Base Address 3 (BAR3)			
20h	Base Address 4 (BAR4)			
24h	Base Address 5 (BAR5)			
28h	Cardbus CIS Pointer			
2Ch	Subsystem ID		Subsystem Vendor ID	
30h	Expansion ROM Base Address			
34h	Reserved			
38h	Reserved			
3Ch	Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line
40h - FCh	Reserved			

Table 3 : Default settings of the PCI Configuration Registers.

Name	Offset	Value
Vendor ID	00h	1172h (Altera ID)
Device ID	02h	0001h
Command Register	04h	Read/Write bits: Bit 1 – MEM_ENA Bit 6 – PERR_ENA Preset bits: Bit 7 – STEP_IMPL (forced to HIGH) Other bits hardwired to LOW
Status Register	06h	Updated Read/Write bits: Bit 27 – TAR_ABORT (Set when no slave response) Bit 31 – DET_PAR_ERR (Detected parity error) Preset bits: Bits 26..25 – DEVSEL_TIM (Set to 10) Other bits return LOW
Revision ID	08h	01h
Class Code	09h	118000h (Data acquisition + Other + None)

BAR0 – BAR2	10h	Bit 0 - MEM_IND (hardwired to LOW) Bit 2..1 - MEM_TYPE (hardwired to LOW) Bit 3 - PRE_FETCH (hardwired to LOW) Bits 15..4 - hardwired to LOW Bits 31..16 - r/w accessible
Subsystem Vendor ID	2Ch	201Ch
Subsystem ID	2Dh	0000h

2.3 Internal Registers

Seen from the PCI bus, the entire user memory occupies an address range of 64 KB (16 address lines). As shown in table 4, the block is divided into two regions: the first 64 bytes (16 double words) hold the Internal Registers and the rest of the space is used for the local bus accesses to the registers on the motherboard. Accesses to the Internal Registers and the local bus transfers may only be done as 32-bit double words via the Base Address Register 2 (BAR2) in the PCI Configuration Register.

Table 4 : Assignment in the local memory of the device.

Offset	31 .. 24	23 .. 16	15 .. 8	7 .. 0
00h				
...	Internal Registers (r/w) (Table 5)			
03Fh				
040				
...	Local Bus (r/w)			
FFFFh				

The Internal Registers of the "Glue *light*" controls three areas of activity: the JTAG and the I^2C busses and the General Purpose I/O lines. The usage is described in Section 2.5 – 2.7. In addition the Internal Registers contains a Control and Status Register (CSR) which allows configuring several functions of the board. A summary of all the Internal Registers is given in Table 5. All unused (not shadowed) registers return zero during read access.

In the current version of the "Glue *light*" board, the CSR (Table 6) allows enabling and disabling the JTAG and the I^2C bus. Disabling a bus puts the output pins in the high-impedance state. The CSR also allows selecting the width of the local bus and configuring the abort method when a time out occurs on the local bus. When a smaller local bus than 32 bits is used, it always starts from the least significant bit.

In addition to the control registers there is also a test register used in verification and debugging, and a version register with the date of the FPGA code. Read as a decimal number the version is YEAR|MO|DA|SQ where SQ is a sequential number, e.g. for the current document 2005 03 11 00.

Table 5 : Assignment of the Internal Registers.

Offset	31 .. 24	23 .. 16	15 .. 8	7 .. 0
00h				CSR (Table 6)
04h				JTAG_CON (Table 8)
08h				I2C_CON (Table 9)
0Ch				I2C_DAT
10h				GPIO (Table 10)
14h				GPIO_DIR (Table 10)

18h – 34h	Reserved	
38h		R/W Test register
3Ch		Version of FPGA code
3Fh	Reserved	

Table 6 : Bit assignment in the Control and Status Register (CSR) (Offset 00h in the Internal Registers).

Bits	Mnemonics	Description	Access
1 .. 0	BSIZE [1..0]	Selecting the size of the local bus: 00 – 8 bits bus 01 – 16 bits bus 10 -- 24 bits bus 11 – 32 bits bus	r/w
2	TMO_ABORT	When set Target Abort is generated on a time out, this is missing slave response. When zero a dummy cycle is generated	r/w
3	BRST_ENA	When set enables burst operation on the local bus, otherwise only single data cycles are performed	r/w
4	JTAG_ENA	Enables/disables the JTAG bus	r/w
5	Reserve	Reserve	r/w
6	I2C_ENA	Enables/disables the I2C bus	r/w
7	JTAG_TRST	JTAG reset line (active in low)	r/w

2.4 Local Bus

The Local Bus (L_BUS) provides a data path between the PCI Bus and non-PCI devices such as FPGAs, FIFO memories etc. The implemented local bus functionality is a subset of the PLX 9030 Local Bus. The "Glue *light*" board provides a 32-bit multiplexed synchronous bus that can only operate with a 32-bit width as seen from the PCI bus. At the back-end, the local bus width can be tailored to 8, 16, 24 or 32 bits, always occupying the least significant bits.

The "Glue *light*" board acts as a Local Bus Master with a permanent bus ownership. The memory range between address 0040h and FFFFh in BAR2 is designated for the local bus transfers.

Table 7 describes the Local Bus pins.

Table 7 : List of the Local Bus signals.

Signal	Name	Type	Function
LCLK	Local Bus Clock	O	Local bus clock, typically 20 MHz
LRESETn	Local Bus Reset Out	O	Asserted when the CCPC is reset
LAD[31..0]	Address/Data Bus	I/O	During the address phase, the bus carries the address of the device on the [15..2] lines. During the data phase the bus carries 32-bit data words

LADSn	Address Strobe	O	Indicates a start of a new bus access and valid address. Asserted during the first clock cycle of the bus transaction
LW_Rn	Write/Read	O	LOW for read accesses and HIGH for write accesses
LWAITn	Wait Out	O	Asserted by the master to insert wait states
LRDYn	Local Ready Input	I	Indicates valid read data on the bus in a read access or that write data is accepted in a write access. The signal is not sampled until LWAITn is deasserted
LLASTn	Burst Last Data	O	Asserted by the master to indicate the last data transfer in a bus access
LTERMn	Burst Terminate	I	Asserted by the slave to terminate a burst access

2.4.1 Bus Transactions

Four types of bus transactions can occur on the Local Bus:

- Read
- Write
- Read Burst
- Write Burst

A Local Bus transaction is bounded by the assertion of the **LADSn** (address strobe) at the beginning and by the de-assertion of the **LLASTn** (last data transfer) or by the de-assertion of **LTERMn** (terminate) at the end. The access consists of an address cycle followed by one or more data transfers. During each clock cycle of the access, the Local Bus is in one of the four basic states as defined below. A clock cycle consists of one **LCLK** clock period.

2.4.2 Local Bus Clock

The Local Bus clock **LCLK** is generated externally to the "Glue *light*" board. All Local Bus signals except for the **LRESETn** are driven and sampled on the rising edge of the **LCLK**.

With respect to the **LCLK**, the following time constraints must be observed:

$\mathbf{T_{setup}}$ ≤ 7 ns - setup time
$\mathbf{T_{hold}}$ ≥ 1 ns - hold time

The current version runs at a nominal frequency of 20 MHz which allows easy attachment of relatively distant devices located on 9U VME motherboards. This nominal frequency can be increased if needed by modifying the Local Bus implementation in the "Glue *light*" FPGA.

2.4.3 Basic Bus States

The four basic states are idle, address, data/wait, and recovery. Upon request from the PCI target, the Local Bus Master starts a bus access by entering the address state while presenting a valid address on the address/data bus and asserting the **LADSn** for one clock period. Data is subsequently transferred in the data/wait state. The **LRDYn** is used by the slave to indicate that data is written or that data is available for reading. It may also be used to insert wait states by temporarily deasserting the signal. The **LWAITn** may be asserted by the master to suspend the slave, which in this

condition should not sample data during a write access and should not update data during a read access. The **LLASTn** is asserted by the master to indicate the last data transfer of the access. To terminate a burst access the slave may assert the **LTERMn** signal instead of only deasserting the **LRDYn**.

The direction of the data transfer is determined by the **LW_Rn** signal, which is low for a read and high for a write access. The **LW_Rn** must be kept either asserted or deasserted during the entire transaction.

After the data transfer, the bus enters the recovery state to allow the bus device to recover. The bus then enters the idle state and waits for another access.

Write data are accepted by the slave on the rising edge of the **LCLK** when the **LW_Rn** is high, the **LWAITn** is high, and either the **LRDYn** or the **LTERMn** is low. The master accepts the read data on the rising edge of the **LCLK** when the **LW_Rn** is low, and either the **LRDYn** or the **LTERMn** is low. This process repeats until the transaction is terminated either by the master or by the slave.

2.4.4 Acknowledgement by the Slave

In a bus transaction, the presence of the slave is signalled to the master by a low level on the **LRDYn** line or the **LTERMn** line. When multiple slaves are attached to the local bus, the device selection is based on internal decoding of the address lines by all slaves. Only one slave at a time may drive the **LRDYn** and the **LTERMn**. Consequently the slaves which are not selected must drive the **LRDYn** line in a tri-state. The same rule applies to the **LTERMn**. Internal on-board pull-up resistors have been implemented on these two lines on the "Glue *light*" board.

When no slave responds in a predefined period of time with either the **LRDYn** or the **LTERMn** asserted, the PCI interface completes the transaction with either a TARGET ABORT or with a valid dummy cycle. The method depends on the configuration of the TMO_ABORT bit in the CSR (see Table 6). In the case of a dummy cycle nothing happens in a write access and a read access returns all zeros.

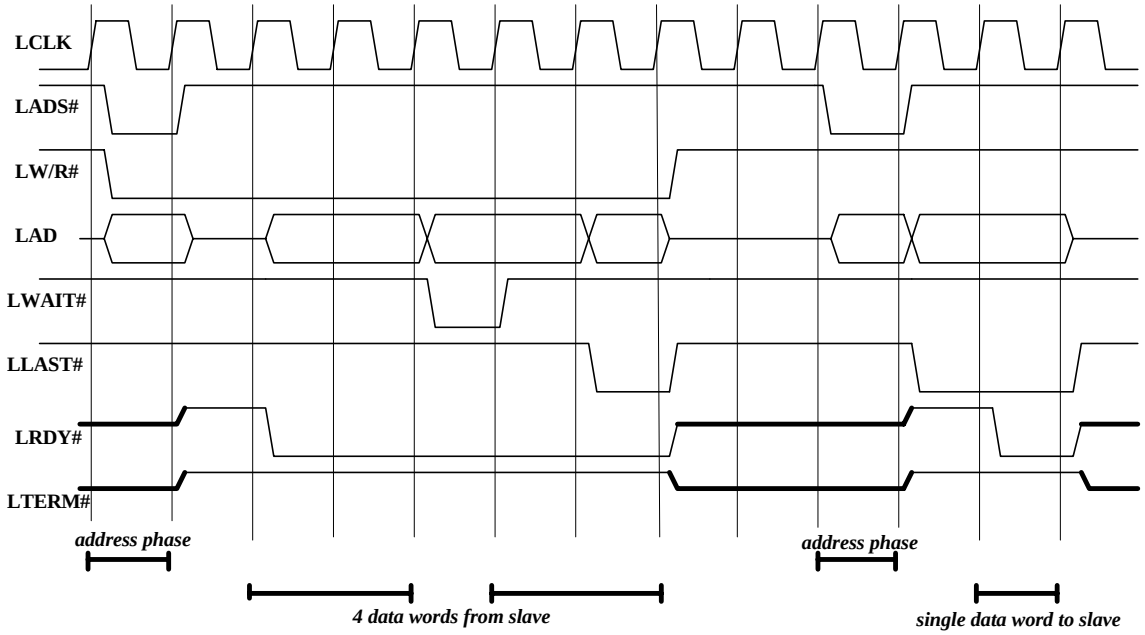


Figure 4 : Timing diagram showing an example of a burst read and a single write transaction.

2.4.5 Examples of Local Bus Timing

Figure 4 shows an example of two Local Bus transactions. The first transaction is a burst read access of four words with one wait state inserted by the master and one data phase delayed by the slave. The transaction is terminated by the master. The second transaction is a single write access which is also terminated by the master.

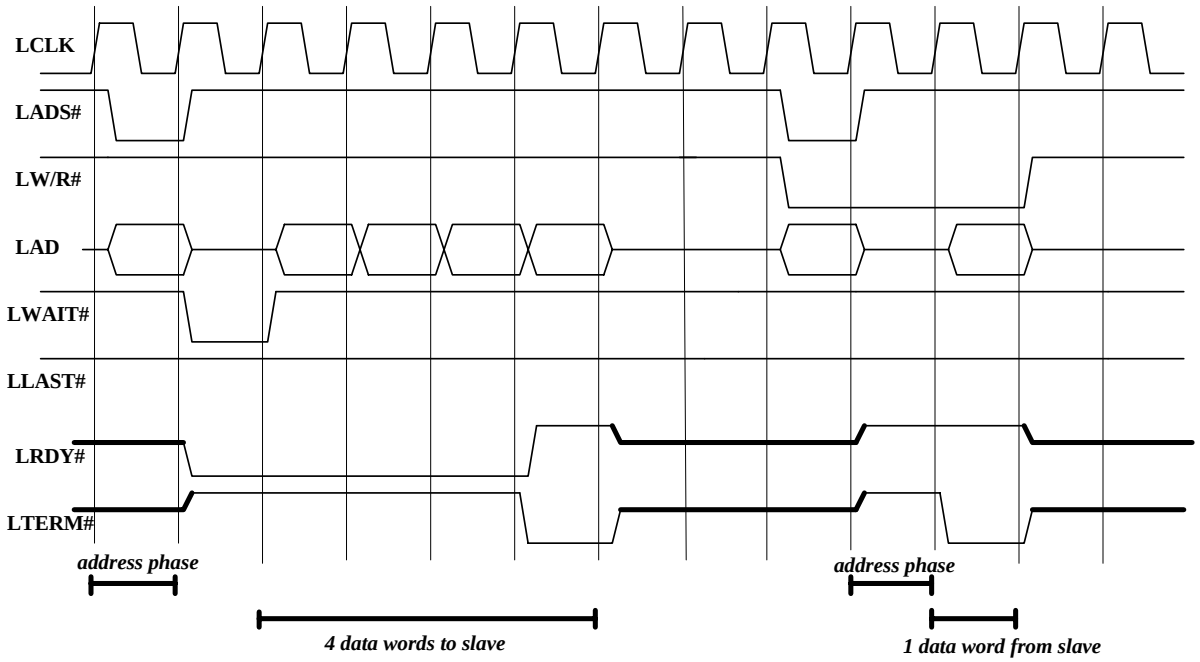


Figure 5 : Timing diagram showing an example of a burst write and a single read.

Figure 5 shows a burst write access of four words which is terminated by the slave. The second transaction is single word read access where the master attempts a burst read but the slave forces a single cycle by terminating the transaction.

2.5 JTAG Bus

The JTAG bus is typically used for making boundary scans and for downloading programming data to FPGAs. The bus consists of three output signals (**TCK**, **TMS**, **TDI**) and one input signal (**TDO**). It is enabled by the JTAG_ENA bit in the CSR register of the Internal Registers (Table 6). The bus is driven directly via the PCI bus by accessing the JTAG_CON register (Table 8) in the Internal Registers. In a read access the first four bits reflects the state of the JTAG bus lines. Transactions on the JTAG bus are performed by writing to the register. The states of the **TCK**, the **TDI** and the **TMS** output lines are set by writing commands to the three two-bit registers that in a write access are associated with the three lines. The set/clear operations are defined in Table 8. JTAG transactions are thus made by putting data on the **TMS** and the **TDI** lines and toggle the clock line **TCK**, and by reading the **TDO** line.

Table 8 : Bit assignment in the JTAG control register JTAG_CON (Offset 04h in the Internal Registers).

Bits	Read Access	Write Access	

0	Read state of **TCK** line	00 – no action 01 – selective clear **TCK** bit 10 – selective set **TCK** bit 11 – no action	
1	Read state of **TMS** line		
2	Read state of **TDI** line	00 – no action 01 – selective clear **TMS** bit 10 – selective set **TMS** bit 11 – no action	
3	Read state of **TDO** line		
4	0	00 – no action 01 – selective clear **TDI** bit 10 – selective set **TDI** bit 11 – no action	
5	0		
6	0	**TMS**	Generate bus sequence (see Note 1)
7	0	**TDI**	
Note 1:	When during write access all bits in the field 5..0 are set to "1", a four clock sequence is generated with the **TMS** and the **TDI** lines set according to bit 6 and 7, respectively.		

The "Glue *light*" board also allows making a full JTAG transaction for every PCI write access by generating a sequence of JTAG levels. The sequence is invoked when the six bits JTAG_CON[5..0] are all HIGH. The sequence consists of four clock cycles. At the first cycle the level of the clock line **TCK** is deasserted. At the second cycle the levels of the **TMS** and the **TDI** lines are set according to the state of bit 6 and 7, respectively, in the JTAG_CON register (Table 8). The **TCK** is asserted during the third and the fourth cycle, and is finally deasserted at the end of the fourth cycle. The states of the **TMS** and the **TDI** lines remain until they are changed.

Holding the **TMS** line in the high state during more than four clock cycles resets the TAP controller of the remote JTAG interface. After power up the **TMS** line is set to HIGH by default.

To configure or scan each JTAG device individually a JTAG bridge, e.g. SCANSTA111 from *National Semiconductors*, may be implemented on the motherboard. We use a small PLD for this purpose which is programmed to provide a 1:8 JTAG hub in a star configuration.

2.6 I^2C Bus

The I^2C bus is typically used for configuring certain devices and for transferring small amounts of data. It consists of a **SCL** (Serial Clock) and a **SDA** (Serial Data). The operation of the I^2C bus is controlled via the PCI bus by accessing two 8-bit wide registers in the Internal Registers: I2C_CON and I2C_DAT. The I2C_CON register provides an interface to the I^2C state machine implemented in the "Glue *light*" FPGA. The procedure to operate the bus is described below and the bit assignment in the I2C_CON register is shown in Table 9. Data are transferred to and from the I^2C bus by writing to and reading the I2C_DAT register. The register holds the address byte during the address phase, and a byte of data to be transferred in write mode or a received data byte in read mode. The least significant bit of the address byte contains the direction flag. A "0" means a write transfer and a '1' a read transfer.

Table 9 : Bit assignment in the I^2C control register I2C_CON (Offset 08h in the Internal Registers).

Bits	Mnemonic	Description	Access

0	**NOACC**	Error bit. Set after a bus request access failed (unsuccessfully writing **START**, **STOP** or data or reading data). Cleared after the first valid transfer. *Note:* Writing or reading the data register I2C_DAT when **DONE** is low will cause this error. Also reading data while in "sending" mode or writing data while in "receiving" mode will generate this error.	ro
1	**ACK**	The bit indicates that the slave acknowledged the address phase (for read or write). For write transfers **ACK** = '1' when byte is accepted.	ro
2	**DONE**	Set after completion of the start, the data or the stop phase . Cleared after setting the **START** or the **STOP** and reading or writing the register **I2C_DAT**.	ro
3	**BUSY**	The bit indicates that the I^2C bus is being used. It is set after setting the **START** bit and it is cleared when the stop condition occurs.	ro
4	**I2C_RESET**	When **I2C_RESET** is set, a reset of the entire I^2C controller is performed. The reset is a momentary action and extends only over two system clock cycles.	mom
5	**LAST**	Can be set only if **DONE** has been asserted. Forces no read acknowledge while sending data. Cleared after the stop condition occurs.	r/ws
6	**STOP**	Can only be set if **DONE** has been asserted. It generates a stop condition. It is cleared when the access has been completed.	r/ws
7	**START**	Can be set if either **BUSY** is LOW or **DONE** is HIGH. It generates the start condition. When **BUSY** is asserted, repeated start conditions may be generated. Cleared after the acknowledgment in the address phase	r/ws

Notes:
1. Bits 3..0 are read only
2. Bit 4 is a momentary write-only action bit and it doesn't retain its value.
3. Bits 7..5 are special action read/write bits. Writing a logic one to the bit sets the appropriate bit in the register. Writing a logic zero has no effect.

The procedure to write a multiple-byte data record is as follows:

1. Wait for BUSY = '0'
2. Set START = '1'
3. Write the peripheral address with the direction bit set to '0' (write) to I2C_DAT
4. Wait for DONE = '1'. If ACK = '0', terminate by setting STOP = '1'
5. Load I2C_DAT with a data byte
6. Wait for DONE = '1'. If ACK = '0', terminate by setting STOP = '1'
7. Repeat step 5 and 6 for each data byte until all bytes have been transferred
8. Set STOP = '1'

To procedure to read a multiple-byte data record is as follows:

1. Wait for BUSY = '0'
2. Set START = '1'
3. Write the peripheral address with the direction bit set to '1' (read) to I2C_DAT
4. Wait for DONE = '1'. If ACK = '0', terminate by setting STOP = '1'
5. Read I2C_DAT and discard the data. This initiates the first burst of nine SCL pulses to clock in the first byte from the slave

6. Wait for DONE = '1'
7. Read the data from I2C_DAT. This initiates another read transfer
8. Repeat steps 6 and 7 for each byte until ready to read the second to last byte
9. Before reading the second to last data byte, set LAST = '1'
10. Read the data from I2C_DAT. With LAST = '1', this initiates the final read transfer
11. Wait for DONE = '1'
12. Set STOP = '1'
13. Read the last byte from I2C_DAT

The bus frequency is approximately 100 kHz by default for compatibility. It can be configured to run four times faster for devices that support the higher speed.

Note that pull-up resistors are required outside the "Glue *light* :board on the **SCL** and the **SDA** lines. Each line should be pulled up via a 2 kohm resistor to +5V or +3.3V depending on the remote device.

2.7 General Purpose I/O

The General Purpose I/O (**GPIOx**) is an 8-bit port consisting of 8 bi-directional lines. The port is controlled via the PCI bus by accessing the GPIO and the GPIO_DIR registers in the Internal Registers (Table 10). The GPIO_DIR registers allows configuring individually the direction of each I/O pins using tri-state drivers. The GPIO register consists of a read and a write register, and reflects either the input value or the output value on the GPIO lines depending on the setting in the GPIO_DIR register. Setting a bit LOW in the GPIO_DIR register puts the corresponding line in a tri-state and puts the current value of the GPIO line in the read register of GPIO. If a bit is set HIGH in the GPIO_DIR register, the corresponding line will be driven to the current value in the write register of GPIO. For lines set to outputs, the GPIO read register returns during a read the value set in the GPIO write register. **Note, if the direction of a line is changed from in to out the value driven on the GPIO line will be the last value set in the GPIO write register.**

Table 10 : Description of the registers to control the General Purpose I/O port.

Register	Description	Access
GPIO[7..0]	Input and output port register bit 7..0	r/w
GPIO_DIR[7..0]	I/O direction register bit 7..0	r/w

2.8 Configuration of the FPGA

The FLEX 10K30E FPGA is configured from an Altera EPC2 configuration device using the native Altera programming interface. The EPC2 contains a reprogrammable Flash memory of 1,695,680 serial bits. It can programmed on-board using JTAG (see Figure 1). The parallel port of the CCPC is used to form a JTAG bus, which drives the JTAG interface of the EPC2 via a Byte Blaster. This configuration allows fast start-up with the immediate download from the EPC2, but it also allows changing the FPGA programming in situ.

3 Application in the TFC system

The "Glue *light*" board has been used in the TFC system[9][10]. All boards make use of the JTAG bus for downloading the programming data to the FPGAs on the motherboards. The programming method is based on the STAPL player [11], which has been modified to run over the PCI bus. The Local Bus is used to access all registers in the FPGAs. The Local Bus Slave has been written as a generic plug-in component. The I^2C bus is used for accessing an I^2C EEPROM, I^2C registers and the TTCrx. The I^2C driver has been implemented in a simple C-function which makes the bus transactions over the PCI bus. In the future the aim is to implement the large part of the I^2C state machine which now resides in a C-function directly in the Glue FPGA.

In order to control and monitor the TFC modules during the testing, a complete TFC Local Run Control system [12] with graphics user interfaces has been implemented using PVSS and DIM [13]. The three access methods are invoked from a DIM server running on the CCPC which receives commands and configuration data from the PVSS system via Ethernet. It also passes back status and counter information to the PVSS system for displaying on the user interfaces.

4 Conclusion

The "Glue *light*" is a mezzanine board used for interfacing the Credit Card PC with the motherboard devices via a simple set of interfaces. The "Glue *light*" board logic is based on an Altera FLEX 10K30E device. Besides providing the standard PLX Local Bus functions, it also provides an I^2C bus, a JTAG bus and eight general purpose I/O. The four interfaces are all driven via the PCI bus of the Credit Card PC.

Acknowledgement

We would like to thank Niko Neufeld who equipped the Credit Card PC with Linux and provided the PCI driver.

We would also like to thank Jean-Pierre Cachemiche and Pierre-Yves Duval who have been using the mezzanine card and have made useful suggestions to improve the local bus interface.

Many thanks to Arek Chlopik who helped in the preparation of the PCB.

References

[1] The LHCb Collaboration, *LHCb Online System TDR*, CERN/LHCC 2001-040, 19 December 2001.

[2] D. Breton and D. Charlet, *SPECS, the Serial Protocol for the Experiment Control System*, LHCb 2003-04 DAQ.

[3] Atlas collaboration, ELMB home page : http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/DCS/ELMB/ELMBhome.html.

[4] C. Gaspar, B. Jost, N. Neufeld, and S. Schmeling,*The Use of Credit Card-sized PCs for interfacing electronics boards to the LHCb ECS*, LHCb 2001-147.

[5] PCI Special Interest Group, *PCI Local Bus Specification*, revision 2.2, December 1998, (http://www.pcisig.com/).

[6] Philips Semiconductors, *The I2C-bus specification*, version 2.1, January 2000, (http://www.semiconductors.philips.com/buses/i2c/).

[7] IEEE, *IEEE Standard Test Access Port and Boundary Scan Architecture, "JTAG"*, IEEE 1149.1-2001.

[8] PLX Technology, *PCI 9030 Data Book*, version 1.4, May 2002 (http://www.plxtech.com/).

[9] R. Jacobsson, P. Koenig, Z. Guzik, A. Chlopik, *The Final LHCb Readout Supervisor "ODIN"*, proceedings of the 8th Workshop on Electronics for LHC Experiments, September 2002, Colmar, France.

[10] Z. Guzik, R. Jacobsson, B. Jost, *Driving the LHCb Front-End Readout*, proceedings of the Real Time 2003 conference, May 2003, Montreal, Canada. Submitted to IEEE TNS.

[11] Altera, *ISP & the Jam Standard Test and Programming Language (STAPL)*, (http://www.altera.com/support/devices/programming/jam/dev-isp_jam.html)

[12] R. Jacobsson, *Experience with PVSS and DIM in building a TFC Run Control*, to be submitted.

[13] C. Gaspar, Partitioning, *Automation and Error Recovery in the Control and Monitoring System of an LHC Experiment*, paper submitted to CHEP2001.

Appendix A: Pin configuration of the LHCb connector

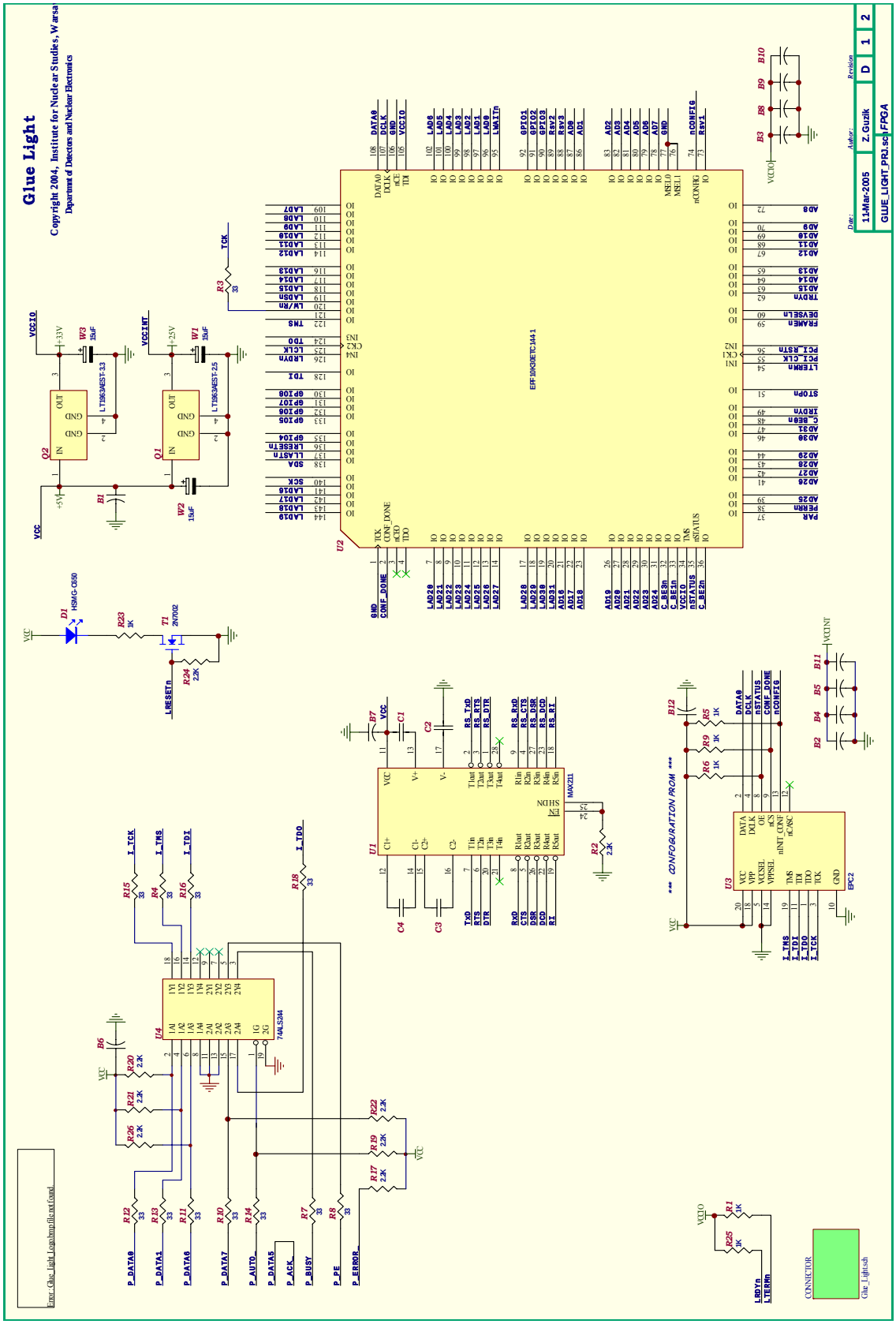
Class	Signal	Mode	Pin	Bus	Signal	Mode	Pin
L_BUS	**LCLK**	I	A32	**I2C**	**SDA**	external pull up	B46
	LRESn	O	A30		**SCL**	external pull up	B47
	LADSn	O	A21	**RS232**	**DCD**	I	B61
	LW/Rn	O	A22		**RxD**	I	B62
	LWAITn	O	A94		**TxD**	O	B63
	LLASTn	O	A31		**DTR**	O	B64
	LRDYn	I	A25		**DSR**	I	B66
	LTERMn	I	A108		**RTS**	O	B67
	LAD0	I/O	A3		**CTS**	I	B68
	LAD1	I/O	A4		**RI**	I	B69
	LAD2	I/O	A5	**JTAG**	**TCK**	O	B26
	LAD3	I/O	A6		**TDI**	O	B29
	LAD4	I/O	A7		**TDO**	I	B27
	LAD5	I/O	A8		**TMS**	O	B28
	LAD6	I/O	A9		**TRSTn**	O	B30
	LAD7	I/O	A10	**GPIO**	**GPIO1**	I/O	A105
	LAD8	I/O	A12		**GPIO2**	I/O	A106
	LAD9	I/O	A13		**GPIO3**	I/O	A107
	LAD10	I/O	A14		**GPIO4**	I/O	A49
	LAD11	I/O	A15		**GPIO5**	I/O	A50
	LAD12	I/O	A16		**GPIO6**	I/O	A51
	LAD13	I/O	A17		**GPIO7**	I/O	A52
	LAD14	I/O	A18		**GPIO8**	I/O	A53
	LAD15	I/O	A19	**MISC**	**PC_RES_INn**	I	B98
	LAD16	I/O	B2		**Rsv1 (ALE)**		A23
	LAD17	I/O	B3		**Rsv2 (RDn)**		A58
	LAD18	I/O	B4		**Rsv3 (WRn)**		A59
	LAD19	I/O	B5				
	LAD20	I/O	B6				
	LAD21	I/O	B7				
	LAD22	I/O	B8				
	LAD23	I/O	B9				
	LAD24	I/O	B11				
	LAD25	I/O	B12				
	LAD26	I/O	B13				
	LAD27	I/O	B14				
	LAD28	I/O	B15				
	LAD29	I/O	B16				
	LAD30	I/O	B17				
	LAD31	I/O	B18				
Power Supply	**+5V**	**-**	A1, A67, A101, A104,, B90, B92, B113				
	GND	**-**	A2, A11, A20, A29, A34, A41, A48, A55, A61, A66, A69, A86, A90, A92, A93, A102, A103, A112, A113, A114, A120, B1, B25, B85, B101, B102, B103, B112, B120				

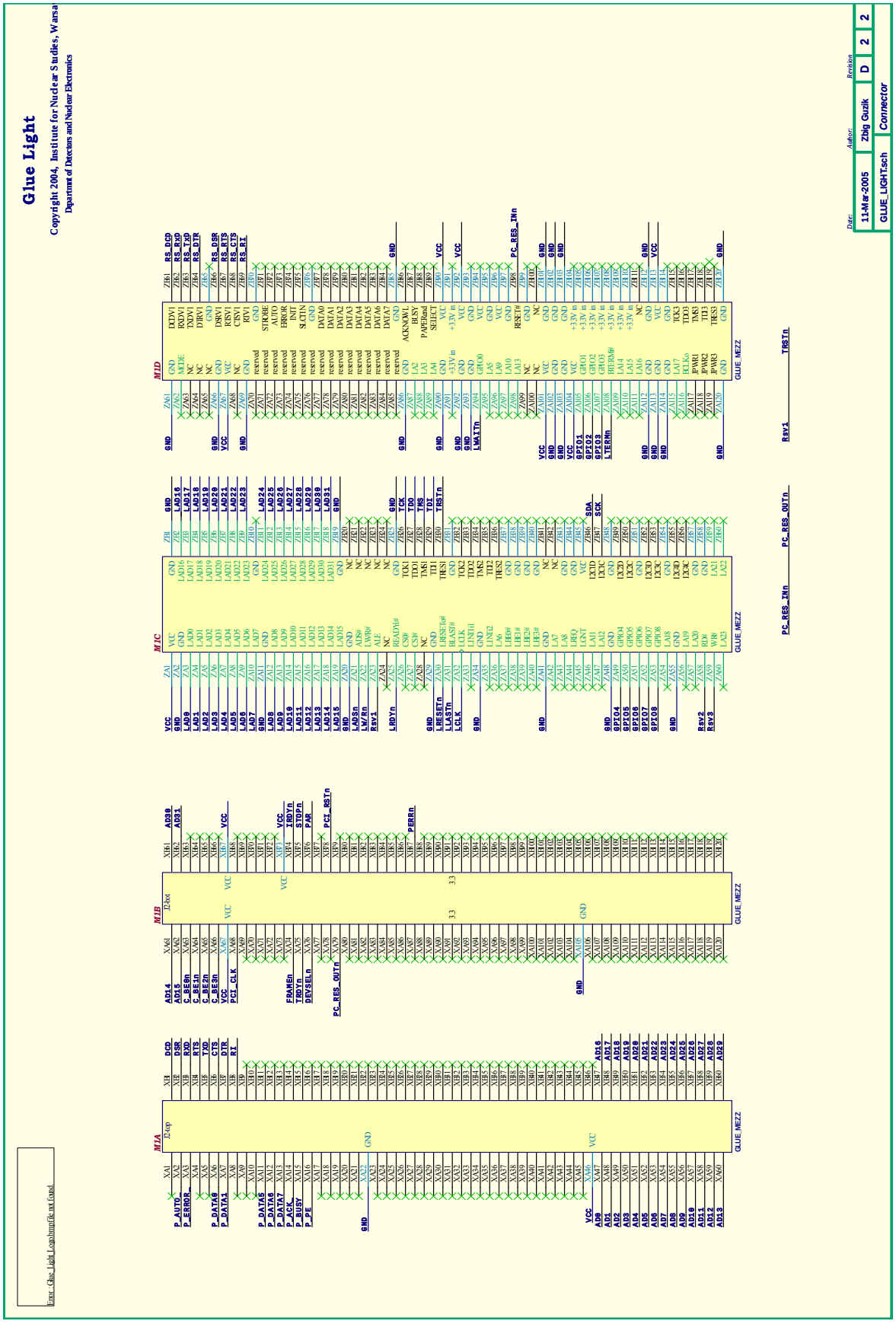
Appendix B: Pin configuration of the CCPC connector (J2)

Class	Signal	Mode	Pin	Class	Signal	Mode	Pin
POWER	+5V			PCI	AD0	I/O	A47
			A46		AD1	I/O	A48
			A67		AD2	I/O	A49
			B67		AD3	I/O	A50
			B73		AD4	I/O	A51
	GND				AD5	I/O	A52
					AD6	I/O	A53
					AD7	I/O	A54
					AD8	I/O	A55
					AD9	I/O	A56
					AD10	I/O	A57
					AD11	I/O	A58
					AD12	I/O	A59
			A22		AD13	I/O	A60
			A105		AD14	I/O	A61
		I/O			AD15	I/O	A62
		I/O			AD16	I/O	B47
		I/O			AD17	I/O	B48
		I/O			AD18	I/O	B49
PRINTER[1]	Strobe#	O	N.C		AD19	I/O	B50
	Auto#	O	A2		AD20	I/O	B51
	Error#	O	A3		AD21	I/O	B52
	Init#	O	N.C.		AD22	I/O	B53
	Slctin#	O	N.C.		AD23	I/O	B54
	Data 0 (TCK)	I/O	A6		AD24	I/O	B55
	Data 1 (TMS)	I/O	A7		AD25	I/O	B56
	Data 2	I/O	N.C.		AD26	I/O	B57
	Data 3	I/O	N.C.		AD27	I/O	B58
	Data 4	I/O	N.C.		AD28	I/O	B59
	Data 5	I/O	A11		AD29	I/O	B60
	Data 6 (TDI)	I/O	A12		AD30	I/O	B61
	Data 7	I/O	A13		AD31	I/O	B62
	Acknowledge#	I	A14		C-BE0#	O	A63
	Busy (TDO)	I	A15		C-BE1#	O	A64
	Paper End#	I	A16		C-BE2#	O	A65
	Select#	I	N.C.		C-BE3#	O	A66
COM1	DCD	I	B1		PCI-CLK	O	A68
	DSR	I	B2		FRAME#	I/O	A74
	RXD	I	B3		TRDY#	I/O	A75
	RTS	O	B4		DEVSEL#	I/O	A76
	TXD	O	B5		IRDY#	I/O	B74
	CTS	I	B6		STOP#	I/O	B75
	DTR	O	B7		PAR#	I/O	B76
	RI	I	B8		PCI_RESET#	I	B78
MISC	PC_RES_OUT	I	A79		PERR	I/O	B87

[1] The printer port is used for programming the programming device on the Glue card. The pin configuration is identical to a ByteBlasterMV cable.

Appendix C: Schematics




Glue Light
Copyright 2004, Institute for Nuclear Studies, Warsaw
Department of Detectors and Nuclear Electronics
Error: Glue_Light Logarithmatik not Found
Date: 11-Mar-2005
Author: Zbig Guzik
Revision: D
2
2
GLUE_LIGHTsch
Connector
M1A
M1B
M1C
M1D
GLUE_MEZZ
P_AUTO
P_ERROR
P_DATA6
P_DATA7
P_DATA6
P_DATA7
P_ACK
P_BUSY
P_PE
P_DATA5
P_DATA4
P_DATA3
P_DATA2
P_DATA1
P_DATA0
RS_DSR
RS_CTS
RS_RI
RS_DCD
RS_RXD
RS_TXD
RS_DTR
VCC
GND
PC_RES_INn
PC_RES_OUTn
Rsv1
T85Tn

Appendix E: Support software

There are several tools implemented for the Glue *light* card:

- jbi_pp

The LHCb PCI library (http://pclbonsrv01.cern.ch/ccpc/doc/lb/lb_8h.html) for the CCPC is fully supporting the Glue *light* card.