A Users Guide
to the TRB3
and FPGA-TDC Based Platforms

Grzegorz Korcyl, Ludwig Maier, Jan Michel, Andreas Neiser, Marek Palka, Manuel Penschuck, Pawel Strzempek, Michael Traxler, Cahit Ugur

August 10, 2018 - 14:47

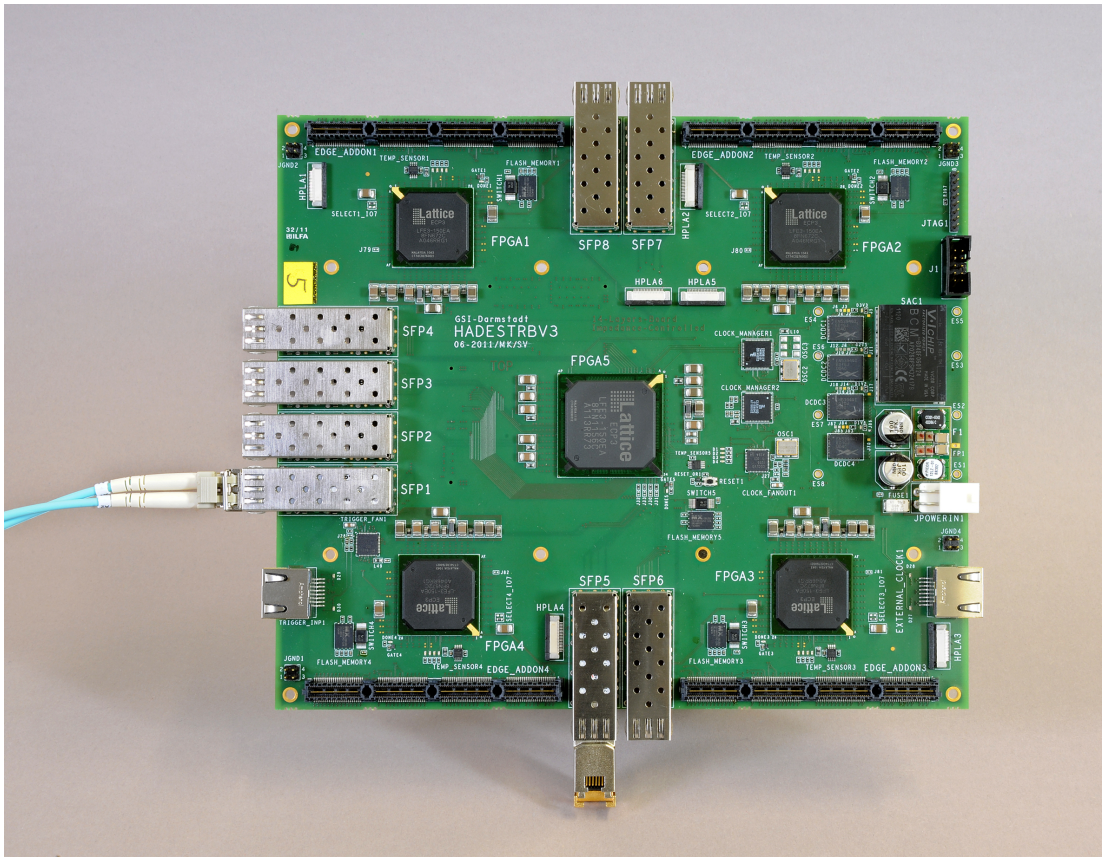


Figure 1: TRBv3 Board.

Contents

I. Resources 9

1. Code Repository 9
- 1.1. VHDL . . . 9
- 1.2. Software / Documentation . . . 9
- 1.3. Additional Resources . . . 9
- 1.4. Repository Notes . . . 10
  - 1.4.1. Some Hints for git contributors . . . 10
- 1.5. FPGA designs . . . 10
- 1.6. Coding Style . . . 11

II. General Information 13

2. General Remarks 13
- 2.1. System Overview for Beginners . . . 13
- 2.2. Beam Time Preparations . . . 14
- 2.3. Hardware Information . . . 14
- 2.4. Board Identification . . . 14
- 2.5. Flash Programming . . . 14
- 2.6. Design Identification . . . 15
- 2.7. Included Features . . . 16
- 2.8. Network Addresses . . . 18
- 2.9. Testing Procedure for New Trb3 Boards . . . 20
- 2.10. JTAG . . . 20
- 2.11. Data Unpacker . . . 21
- 2.12. Trigger & Clock Input . . . 22
- 2.13. Power Consumption . . . 22

3. Slow Control Registers 22

III. Hardware 23

4. Measurements 23
- 4.1. FPGA I/O Performance . . . 23

5. TRB3 Platform 24
- 5.1. Known Bugs and Limitations . . . 24

5.2. Clock and Trigger Distribution 24

6. Trb3sc **26**
6.1. Basics 26
6.1.1. Powering Schemes 26
6.1.2. Clock Inputs 26
6.1.3. Trigger Input/Output 27
6.1.4. Other I/O 27
6.1.5. Serial Links 28
6.1.6. Modifications 29
6.2. FPGA based TDC calibration 30

7. DiRich **32**

8. AddOns **32**
8.1. TDC AddOn 32
8.2. 32-Pin AddOn 32
8.3. Multi-Test-AddOn 32
8.3.1. Known bugs 33
8.4. Hub AddOn 33
8.5. MVD AddOn 33
8.6. CTS AddOn 33
8.7. General Purpose AddOn 33
8.8. ADC AddOn 35
8.8.1. Data Format 35
8.8.2. Slow Control Registers 37
8.9. Padiwa 38

9. Related Boards **38**
9.1. CBM-RICH 38
9.2. CBM-TOF 38

IV. Design Components **39**

10. New VHDL Project **39**

11. TDC **40**
11.1. Building Blocks 40
11.1.1. Fine Time Measurement 41
11.1.2. Fine Time Calibration 41

11.2. Features . . . 43
11.2.1. Trigger Window and Trigger Mode . . . 43
11.3. Data Format . . . 43
11.3.1. TIME DATA . . . 43
11.3.2. TDC HEADER . . . 44
11.3.3. DEBUG - Status Information . . . 45
11.3.4. EPOCH Counter . . . 47
11.3.5. TDC TRAILER (was RESERVED before tdc_v2.3) . . . 47
11.4. Slow Control Registers . . . 48
11.5. TDC Version Table . . . 52

12. Additional Modules **55**
12.1. DAC Programming . . . 55
12.2. Forward inputs for trigger . . . 57
12.3. Interfaces . . . 58
12.4. Flash programming . . . 58

13. GbE Data Read-out **61**
13.1. Data Readout . . . 61
13.2. Addressing . . . 62
13.3. Configuration . . . 62
13.4. Monitoring . . . 63
13.5. Building Blocks . . . 63
13.6. Slow Control Registers . . . 63

14. GbE Slow-Control **64**
14.1. Getting Started . . . 64
14.1.1. FPGA design . . . 64
14.1.2. Trbnetd . . . 64
14.1.3. Trbcmd server . . . 65
14.1.4. Usage . . . 66
14.1.5. Ping of Death . . . 66
14.2. Building Blocks . . . 66
14.3. Slow Control Registers . . . 66

15. CTS **67**
15.1. Features . . . 67
15.2. Getting Started . . . 67
15.2.1. The GUI . . . 67
15.3. Building Blocks . . . 68

15.4. CTS Network Logic . . . 68
15.4.1. SubSubEvent Data Format . . . 69
15.4.2. Multiple Event Builders . . . 69
15.5. Trigger Logic . . . 71
15.5.1. Input module . . . 71
15.5.2. AddOn Input module . . . 72
15.5.3. Triggers from Peripheral FPGAs . . . 73
15.5.4. Coincidence detection . . . 73
15.5.5. Pulsers . . . 74
15.5.6. External Trigger Logic . . . 74
15.5.7. Free-Running with Spill-Dependent Frequency . . . 74
15.5.8. Latency and Jitter . . . 75
15.6. Slow Control Registers . . . 76
15.7. Trigger Generation Options . . . 80
15.8. HowTo Implement an External Trigger Module . . . 80
15.8.1. The module's interface . . . 80
15.8.2. Obtaining a module id and registering the module . . . 83

16. Nxyter Read-out **85**
16.1. Design Blocks . . . 85
16.2. Data Format . . . 85
16.3. Slow Control . . . 85

17. Billboard **86**
17.1. Trigger Scheme . . . 86
17.2. Memory . . . 86
17.3. Slow Control . . . 87
17.4. SubSubEvent Format . . . 88

18. CBM-MBS Receiver **89**
18.1. Data Format . . . 89

19. CBMNet Bridge **90**
19.1. Synthesising the Bridge . . . 90
19.2. Read-Out via CBMNet . . . 90
19.3. Synchronisation with CBMNet . . . 92
19.3.1. Read-Out Format . . . 92

V. Experimental Setups and Configurations 95

20. Trigger Time vs Reference Time 95

VI. Software Quick Start 97

21. Distribution Related Notes 97
21.1. How to set up SUSE Tumbleweed (64bit) on a PC . . . 97
21.2. How to prepare a Debian distribution (and others) . . . 100

22. Software installation 100
22.1. User scripts . . . 101

23. Configuration 102
23.1. Preparing DHCP . . . 102
23.2. Preparing DNS . . . 104
23.3. dnsmasq as an alternative to ISC dhcpd and DNS over /etc/hosts . . . 105
23.4. Starting TRBnet . . . 106
23.5. Configuring TRB3 . . . 107
23.6. DAQ configuration . . . 108
23.7. CTS monitor configuration . . . 109

24. DAQ startup 109
24.1. Starting TRB3 . . . 109
24.2. CTS monitor . . . 110
24.3. Event builder . . . 110

25. Analysis Software 111
25.1. Mainz Unpacker . . . 112
25.2. DABC and go4 . . . 112
25.3. DABC documentation . . . 113

26. Web interface 113

27. Data File Format 114

VII. Synchronous TrbNet 117

28. Media Interfaces 117
28.1. Central Aspects . . . 117
28.2. Clock Measurements . . . 117

28.3. Media Interfaces 118

Appendices **121**

A. TDC Calibration **121**

Part I.
Resources

1. Code Repository

All code is available from a server in Frankfurt. There are two ways to access the repositories:

- `git://jspc29.x-matter.uni-frankfurt.de/projects/REPONAME.git` - read access is available from everywhere, no password or permission needed
- `git@jspc29.x-matter.uni-frankfurt.de:REPONAME` - gives full read/write access, but is only available on request when your ssh key is added to the server.
- `http://jspc29.x-matter.uni-frankfurt.de/git` - a web GUI to browse existing repositories.

1.1. VHDL

The main VHDL repositories are as follows, there are several more for specific sub-projects. Have a look to the web page to find them. For most projects, several repositories are needed because they depend on each other. As a general rule, repositories depend on each other from top to bottom in the following list. All repositories have to be cloned to a common subdirectory to make links between the repos work.

trbnet - the main TrbNet repository
vhdlbasics - some generic code fragments
trb3 - code for TRB3 boards
trb3sc - code for TRB3sc boards
tdc - the TDC code
dirich - code for the DiRich board family
padiwa - code for all Padiwa boards

1.2. Software / Documentation

daqdocu - the documentation, including this PDF
trbnettools - the basic software for any slow control communication
daqtools - all tools and scripts to control the TrbNet based system
labtools - controls for various lab equipment like power supplies

1.3. Additional Resources

DABC Eventbuilder

`http://hades-wiki.gsi.de/cgi-bin/view/DaqSlowControl/EventBuilderDabc`

1.4. Repository Notes

- The repositories with software and documentation (daqdocu, daqtools, trbnettools, daqdata) are supposed to reside in a common directory named trbsoft, e.g. in your home directory.
- If users want to put their own scripts to the git (recommended for education of others), there is a directory daqtools/user/. Here you can add a sub-directory with your project name and add all your files (tools, configuration etc.).

1.4.1. Some Hints for git contributors

Here are some (personal) hints for using git more effectively. Feel free to extend that list or provide useful links, but don't create a whole git tutorial here (this can be googled duckduckgo'ed).

- NEVER use `git push --force` or `git push -f` (most likely you will make everybody else angry, except you're the poor one who must fix a previous forced push)
- Have a look at `git rebase -i`. With interactive rebase, you can easily reword, split and reorder your local commits before pushing them, helping others to understand your changes quickly.
- Use `git add -p` if you made a lot of changes but you want to create several commits out of them (can be combined with interactive rebase).
- Use `git commit --author "My Name <your@email.com>"` if you're committing from the hadaq account (or similar group accounts). Then you don't need to mention your name or initials in the commit message.
- If you find out that somebody changed the repository on the server when trying to push your changes, try `git pull --rebase`, this saves you from having to do a merge commit. If you have uncommitted local changes, you have to do a `git stash` before and a `git stash pop` afterwards.

1.5. FPGA designs

- All information is collected on http://trb.gsi.de, some direct links are:
- A list of current FPGA design files can be obtained from http://jspc29.x-matter.uni-frankfurt.de/trbweb/?action=page&url=design-files
- All designs can be downloaded from http://jspc29.x-matter.uni-frankfurt.de/bitfiles
- Developers upload their files via scp to hadaq@jspc29.x-matter.uni-frankfurt.de:/srv/www/htdocs/bitfiles/

1.6. Coding Style

There is no mandatory coding style, but most code follows these simple rules to make it easier to read by others:

- Indentation is done using two spaces per level. If tabs are used, make sure the width is set to two.
- Keywords (if, process, case) are written in lower case.
- variable and signal names are lower case. Use prefixes (`buf\_`, `next\_`, `reg\_`) and suffixes (`\_i` for internal signals, `\_q` after a FF) and keep the base name the same if necessary.
- Inputs and outputs should be identifiable. Use `\_OUT` and `\_IN` as suffix, or use a very obvious naming.
- All blocks are named using capital letters. Use a common prefixing: `PROC\_XYZ` for processes, `GEN\_` for generate, `THE\_` for instances.
- Use speaking names, keep them precise but not too long. Use a common namespace for signals that belong to the same logical group.
- Each section of the code should get a title like

```
------------------------------------------------------------------
-- DescribeMe
------------------------------------------------------------------
```

- For synchronous processes, use the handy 'wait until' syntax which avoids one level of 'if', the possibility of asynchronous resets and all trouble with sensitivity lists:

```
PROC_DO : process begin
wait until rising_edge(CLK);
[...]
```

- Do not put component declarations into the archticture. Either have a separated package file with all components, or use `THE_ID : entity work.ENTITYNAME` syntax.
- Use indentation and alignment also within the code line, e.g. here are all signal names aligned and easier to read:

```
THE_MAIN_PLL : pll_in200_out100
  port map (
    CLK    => CLK_GPLL_LEFT,
    RESET  => '0',
    CLKOP  => clk_100_i,
    CLKOK  => clk_200_i,
    LOCK   => pll_lock
    );
```

- FSM coding style is up to each individual. Most code uses the single-process version,

using just one single, synchronous process. This reduces the number of signals needed and fully avoids any potential issue with latches due to unassigned signals.

- All reset signals are synchronous.

Part II.
General Information

2. General Remarks

2.1. System Overview for Beginners

When you start to use the TRB-Platform, you can be overwhelmed with the many acronyms and the basic setup of such a system. Here is a very short overview for a minimal setup of one TRB3 on a table.

The CTS (Central Trigger System) is a VHDL module in the central FPGA of *one* TRB3 in the system (could also be on a different hardware, but we can combine all on one TRB3). This module takes the external (and internal) trigger sources and generates out of them a timing signal and the needed internal TRBNet trigger, which is then transported to all slaves (which in your case are all on the same TRB3). They react on the trigger and extract the data from the front end and transport it to the central FPGA, which is your *special* case (only one TRB3) is the same FPGA as the CTS is running in. There the data is collected from all 4 peripheral FPGAs and then combined to a UDP-frame, which is then sent via many Ethernet-packets to the Eventbuilder. The Eventbuilder-PC linux is combining the packets to a UDP-frames which then goes to the eventbuilder process (dabc), which combines many sources (in your case only one source) to coherent events via the unique event-numbers the CTS has generated in the first place. The Evenbuilder can be on any computer in the same network (we use directly the MAC). For small setups this can be the same computer you run the slow control commands.

Other things you might want to know: The TRB3 is based on FPGAs, so we didn't implement TCP/IP in them, which is a *very* large effort. We only have implemented UDP/IP. And the sending part of the TRB3 also doesn't ask for the MAC of the receiver, we decided just to put this into registers (up to 16 receiver Eventbuilder MACs can be stored).

To get started with the startup scripts (after you read all the details described later in this manual): We recommend to copy an existing directory, e.g. daqtools/users/gsi_ee_trb84 to a name of your preference, e.g.

```
> cd daqtools/users
> cp -a gsi_ee_trb84 triumf_trb171
```

and base your script on the one which is in there:

```
> ./startup.sh
```

2.2. Beam Time Preparations

For a beam time, please be aware, that everything can happen! Any sorts of non experienced failures happen according to Murphy always when you really don't need them :-)... So, please don't go to a beam time with **one** TRB3 and no replacement module and always be sure that you have a Lattice Download Cable available to re-flash the TRB3 if something goes wrong in the fever of a beam time.

2.3. Hardware Information

- TRB3 Schematics
- Pin-out file for all FPGAs vary based on the AddOns in use and can be found in the git repository (*.lpf files)

2.4. Board Identification

The TRB3 boards are equipped with 5 temperature sensors, one for each FPGA. They contain a unique ID that is used to identify each FPGA. Additionally, the peripheral FPGAs have to bits identification (endpoint ID) corresponding to their position on the board. This information can be read out after the first initial programming of the FPGAs.

The board itself has a sticker with a three-digit serial number to identify it (we are looking into a bright future of the TRB3 ;-)). The combination of serial number and unique ids is given in the file `serials_trb3.db` available in the main directory of the cvs repository. For each board it contains five lines

```
#SID  Unique ID
0015  0x08000002e2e22b28
0010  0xa6000002e2e2df28
0011  0x51000002e2e22828
0012  0x72000002e2eb4628
0013  0xb0000002e311b928
```

The first three digits of the SID is the serial number as written on the board, the fourth digit is an identifier for the FPGA number as printed on the PCB (central FPGA is FPGA 5, the others are numbered 1 to 4, but mapped to 0 to 3 in the file). Based on this file, a second file `addresses.db` can be written for each individual set-up to assign each board the necessary network addresses.

2.5. Flash Programming

Typically only the first programming of a board is done with a JTAG cable, all later upgrades can be done directly via TrbNet to the Flash ROMs. The advantage is the increased speed

(about a factor 10) and that no physical access to the board is necessary. The software needs some settings in the FPGA code to function properly:

First, the name of the design has to contain a certain sub-string:

trb3_central or trb3_fpga5 if the design is targeted to the central FPGA

trb3_periph or trb3_fpga1234 if the design is targeted to either of the peripheral FPGA

trb3_fpgaN were N is a number between 1 and 4 if the design should be loaded to a special FPGA only

Second, the upper 16 Bit of the Hardware Version register as described below is checked.


WARNING

Always ensure that your trbflash accesses the TrbNet over RPC (i.e. trbnetd) and *not* locally. So check if `trbflash -V` mentions RPC. If it's not the case, fix your `$PATH`. If you don't access the TrbNet via trbnetd, a running CTS monitor in the background might disrupt your flash and hence corrupt your flash image!

2.6. Design Identification

The TrbNet endpoint has a generic setting REGIO_HARDWARE_VERSION (register 0x42) that has to be set according to the following rules: The upper 16 Bit are used by the software to identify the hardware before programming the Flash to prevent loading invalid designs and have to contain one of the following values. The last digit should be used to denote the hardware revision.

9000 design is for the central FPGA
9100 design is for either of the peripheral FPGAs
9110 design is for FPGA 1 only
9120 design is for FPGA 2 only
9130 design is for FPGA 3 only
9140 design is for FPGA 4 only
9200 design for CBM Rich
9300 design for CBM Tof
9500 design for Trb3sc
9600 design for DiRich
9700 design for DiRich Combiner module
9900 design for Munich Skyroc boards

The lower 16 Bit are used to identify the contents of the design and the AddOn boards they should be used with. Combine as many values as you like by logical or. Please note that these values have been partly superseded by the IncludedFeatures register described below. This register should only be used to describe hardware-related features, like the needed AddOn Board.

Central FPGA

cXX0 contains a CTS
cXX1 contains a CTS, use with AddOn for trigger signals
8XXX uses RX clock as main internal clock
0e00 contains a GbE link for slow control and read-out
0d00 contains a GbE link for read-out only
0010 accepts triggers from optical link SFP1
0020 accepts slow-control from optical link SFP1
0040 sends triggers to optical link SFP1
0080 sends slow-control to optical link SFP1

Peripheral FPGA (also CBM-RICH and other derivatives)

0XXX use with ADA adapter board AddOn version 1
1XXX use with ADA adapter board AddOn version 2
2XXX use with multi-purpose test AddOn
3XXX use with SFP hub AddOn
4XXX use with Padiwa adapter AddOn
5XXX use with General Purpose AddOn - with NIM connectors
6XXX use with Nxyter
7XXX use with 32PinAddOn
9XXX use with ADC AddOn
AXXX use with SFP AddOn v2
BXXX onboard connectors
X0nX contains 2^n TDC channels, single edge, n<8
X1nX contains 2^n TDC channels, double edge, n<8
X2XX contains a network hub
X4XX SPI interface on AddOn connector
X8XX Double edge TDC realized with two single edge channels
XX8X Non-TDC (because of bad choice of encoding)
XX9X for MVD converter board 2013
XXX1 uses RX clock as main internal clock

2.7. Included Features

Basic Information about included features should be put to a generic setting named Included-Features. This should be set in the config.vhd file of each project according to the following table. The values are available from registers 0x41 (lower 32 Bit) and 0x43 (upper 32 Bit) on TrbNet. The uppermost 8 Bit define a set of definitions that defines the meaning of the remaining 56 Bit in the word.

Fields marked as “version” should be increased after any bigger change in the design or

major bug fixes.

Table	Bit	Name	Description
0	0	Undefined	The feature table is not implemented in the design
1	“Central” - For a normal central FPGA design with Cts and/or GbE		
	3 – 0	ExtModule	Type of external trigger module (0: none, 1: CBM MBS, 2: Mainz M2)
	11 – 8	DoubleEdge	See table 2.
	15	TDC	The design contains a TDC module.
	16	GbeData	Event data is sent via GbE
	17	GbeCtrl	FPGA accepts slow-control messages via GbE
	19 – 18	GbeDataBuf	Size of the buffer for event data. 1: 64 kB
	21 – 20	GbeCtrlBuf	Size of the buffer for sctrl data. 1: 4 kB, 2: 64 kB
	22	GbeMultBuf	GbE sctrl data can be split to multiple packets
	23	GbE	Contains a GbE module
	27 – 24	Sfp	Number of SFP configured for TrbNet connections
	28	Backplane	TrbNet interface on backplane connector
	40	Lcd	LCD Information display included
	41	ReferenceTime	Reference Time Path 0: RJ-45 (default) 1: Through Clock Manager (cbmtof only)
	42	Spi	Contains SPI on all relevant I/Os depending on AddOn board design
	43	Uart	Uart on RJ45_CLOCK(4) (TTL) o HDR_IO
	44	InpMonitor	Monitoring of input signals.
	51 – 48	TrgModule	Type of trigger module 0: none, 1: simple or, 2: edge detect
	55 – 52	Clock	TRB3 - Main clock source: 0: onboard 200 MHz, 1: onboard 125 MHz, 2: 200 MHz RX clock on SFP1, 3: 125 MHz RX clock on SFP1, 4: external clock input 200 MHz, 5: external clock input 125 MHz TRB3sc - 52: 120 MHz instead of 100 MHz, 53: use RX clock, 54: external clock, 55: has 200 Mhz oscillator
2	“TDC” - For TDC designs. Detailed information about the TDC setup can be found in register 0xc8xx		

	7 – 0	Pinout	Which pin-out is being used for the TDC inputs. 0: flexible by multiplexers, 1: default 1-to-1, 2: every second input (e.g. Padiwa Amps fast-only), 3: every fourth input (HPTDC very high speed mode)
	11 – 8	DoubleEdge	Double edge setup: 0: single edge only, 1: same channel, 2: alternating channels, 3: same channel with stretcher
	14 – 12	RingBuffer	Ring Buffer size: 0:12 words, 1:44 words, 2:76 words, 3:108 words, 7:dynamic
	15	TDC	Contains a TDC
	18 – 16	ReadoutModule	Number of readout modules minus 1
	55 – 40		See table 1
3	"MVD" - For CBM-MVD designs.		
	7 – 0	Sensors	Number of sensor inputs
	11 – 8	Chains	Number of sensor chains
	16	Mode	Normal read-out (0), testmode (1) or selectable (2)
	23 – 20	Type	Type of sensor. 0: M26
	55 – 40		See table 1
4	"ADC" - For ADC AddOn designs.		
	7 – 0	Frequency	ADC sampling frequency in MHz
	11 – 8	Processing	Type of processing logic. 0: dummy readout. 1: simple preprocessing and trigger. 2: advanced filtering, 8:full feature extraction
	14	Baseline	Baseline determination
	15	Trigger	Trigger generation
	23 – 16	Channels	Number of channels
	55 – 40		See table 1

2.8. Network Addresses

The network addresses in a TRB3 set-up (not in HADES) follow a simple scheme:

All network addresses are of the form ABBC, where:

- C is the FPGA number. 0 for the central FPGA, 1-4 for the peripheral FPGAs.
- A denotes the function of the FPGA:
  C: CTS, 8: Hub, 0–1: TDC, D–E other purposes
- BB is a number identifying the TRB in the full system. BB is equal on all 5 FPGA of one board.

The FPGA with the CTS has address C000. For data unpacking schemes see also 2.11.

All boards of a given type are accessible by a broadcast address at the same time. This is set by BROADCAST_SPECIAL_ADDR in the TrbNet endpoint:

- 0x40 for the central FPGA
- 0x45 for the peripheral FPGA
- 0x48 TDC design for peripheral FPGA
- 0x49 peripheral FPGA with Nxyter AddOn
- 0x4a peripheral FPGA with General Purpose AddOn
- 0x4b peripheral FPGA with ADC AddOn
- 0x4c peripheral FPGA with ADDON_4CONN2 AddOn (for Padiwa)
- 0x4d peripheral FPGA for MAPS read-out
- 0x4e peripheral FPGA for Hades Start detector
- 0x4f peripheral FPGA for Panda Straw Tube
- 0x50 CBM-Rich
- 0x51 DiRich Frontend
- 0x52 DiRich Combiner
- 0x60 Trb3sc
- 0x61 Trb3sc Backplane Maser w/ GbE
- 0x62 Trb3sc CTS w/ SFP
- 0x63 Trb3sc CTS w/ Backplane
- 0x64 Trb3sc Hub no Backplane, no GbE
- 0x65 Trb3sc Hub w/ Backplane, no GbE
- 0x66 Trb3sc Hub no Backplane, w/ GbE
- 0x67 Trb3sc Hub w/ Backplane, w/ GbE
- 0x68 Trb3sc ADC AddOn
- 0x69 Trb3sc for trigger gen. & monitoring
- 0x70 Trb3sc TDC 4conn SFP
- 0x71 Trb3sc TDC 4conn Backplane
- 0x72 Trb3sc TDC KEL SFP

The initial address set with REGIO_INIT_ADDRESS can be chosen from the following set:

- 0xF300 for the central FPGA
- 0xF305 for the peripheral FPGA
- 0xF30n for a design for FPGA n only
- 0xF3C0 a design with CTS
- 0xF3CC slave TRB3sc
- 0xF3CD TRB3sc with hub AddOn
- 0xF3CE crate master TRB3sc
- 0xF3D1 Dirch
- 0xF3DC Dirich combiner

2.9. Testing Procedure for New Trb3 Boards

- Visual Inspection
- Add sticker with serial number
- Power-up - current should be around 0.25A at 48V
- Check if fixes (Flash 0R) are done
- Load set of configuration files (no Flash programming for FPGAs, only load design)
- Also note that you need to issue an `trbcmd reset` after JTAG programming in order to have the FPGAs obtain the correct local number identification reported by an info trbcmd broadcast (5 for central, 0-3 for peripheral FPGAs).
- Check basic TrbNet functionality
- Program Flash ROMs via TrbNet
- Reboot Board to see if all FPGA boot from Flash
- Add the five unique IDs to the `serials_trb3.db` in the cvs (see 2.4)
- Add board as tested to wiki page

2.10. JTAG

Since programming of FPGAs can be done via GbE, the JTAG connector is usually not used. In case of corrupted designs it provides the only access to reconfigure FPGAs. Loading a design directly to the FPGA is quite fast (25 s) but loading it to the Flash ROM is deadly slow (5 minutes) - better: first load the design to the FPGA via JTAG, then flash it using TrbNet.

The pin-out of the JTAG connector (1x8 pin-header near the power supply).

1 VCC (3.3V, red)
2 TMS (violet)
3 TCK (white)
4 TDI (orange)
5 TDO (brown)
6 GND (black)
7 GND (n/c)
8 GND (n/c)

Pin 1 is next to the 2x6 pin-header. Note that TDO and TDI are switched compared to the layout on all other boards. If you experience strange behaviour of the programming procedure and think you might have destroyed the cable: It's most likely a software issue - reboot your PC!

Note that the programming cable can only be used as root by default in most Linux flavours. You can either change permissions on the device in /dev/ by hand or put this (as one line!)
`SUBSYSTEMS=="usb", ATTRS{idVendor}=="1134", SYMLINK+="lattice_cable", GROUP="users", MODE="0666"'`
into

`/etc/udev/rules.d/91-usb-hardware.rules`
and run
`udevadm control --reload-rules; udevadm trigger`
`/etc/init.d/boot.udev restart`
Commands and files might be different on your machine.

There's also an alternative to the quite expensive Lattice Programming cable, if you have some time left, you can buy the FT2232H mini module and solder a cable yourself. This costs roughly 50 Euro and one hour soldering fun. If the programmer is not able to find a valid JTAG chain or errors are reported during programming, the problem may be related to the Linux ftdi_sio kernel module. You can unload it automatically using a udev-rule by adding (as one line)

```
SUBSYSTEM=="usb",ATTRS{idVendor}=="0403",ATTRS{idProduct}=="6010",MODE=="0666",
OWNER=="root",GROUP="users",RUN+="/bin/sh -c 'RESULT=$(basename %p);
echo $RESULT:1.0 >/sys/bus/usb/drivers/ftdi_sio/unbind"
```

into
`/etc/udev/rules.d/91-usb-hardware.rules`
and restart udev or unload it by hand with
`rmmod ftdi_sio`
after plugging in the cable.

2.11. Data Unpacker

The data stream contains SubEvent-IDs and SubSubEvent-IDs to identify the sender of each block of data. While SubSubEvent-IDs are equal to the network address of the corresponding board by design, the SubEvent-IDs can be chosen freely, but are defined to be equal to the network address of the board sending the SubEvent via Gigabit Ethernet. The first digit of each SubSubEvent-ID is sufficient to determine how the following data has to be handled:

0..1 Unpack as TDC data
5555 The only SubSubEvent-ID starting with 5 is 5555 at the end of each SubEvent, marking the beginning of the status information word for this SubEvent.
7 Special TDCs (like beam detectors, not part of the system under test)
8 Skip this SubSubEvent-Header. Note: Only the header, the next word should be another valid SubSubEvent-Header.
C000 Unpack as CTS data
D..E Other read-out formats, no TDC data included

The data format is shown in later sections.

2.12. Trigger & Clock Input

Both trigger and clock input (RJ-45 connectors) are equipped with a configurable fan-out chip. Its control signals can be set in slow-control register 0xd300 (currently in CTS only, but should also be present in other central FPGA).

Bit 0 Trigger select (local signal: 1, external signal: 0
Bit 8 Clock select (Input 0 or 1)
Bit 19 – 16 4 User signals on trigger fan-out
Bit 27 – 24 4 User signals on clock fan-out

2.13. Power Consumption

The TRB3 with linear supplies and typical TDC designs needs

- 3.3V - 1.25A
- 2.5V - 0.5A
- 1.2V - 8A

3. Slow Control Registers

Address	Name	Description
4000 – 40FF	Hub	Hub Config and status
7000 – 73FF	RDO	Readout status
8000 – 83FF	GbE	Ethernet registers
A000 – BFFF	FEE	Thresholds, Pedestals, Settings
B000 – B7FF	Serdes	Serializer status (on hubs)
C000 – CEFF	TDC	TDC Control and Status [11]
CF00 – CF7F	Trg	Trigger signal generation [12.2]
CF80 – CFFF	Inp	Input Monitoring [12.2]
D000 – D13F	Flash	Control for SPI Flash of FPGA [2.5]
D200	Rom	Flash Rom Switch
D300	TrgIn	Selection for trigger and clock input on CTS
D400 – D41F	SPI	SPI Interface for DAC and Padiwa
D480 – D4FF	Adc	On-board monitoring of voltages or currents
D500 – D5FF	SED	Soft Error Detection
D600 – D6FF	Uart	Serial Uart Interface
E000 – FFFF	Debugging	Memories and Registers for Debugging

Table 2: Register Map of the Slow Control Endpoint. Suggested usage of the address space.

Part III.
Hardware

4. Measurements

4.1. FPGA I/O Performance

5. TRB3 Platform

5.1. Known Bugs and Limitations

- The SFPs are missing LEDs - so no information about link status
- The outputs of the CLK5410 chips are not independent: Two outputs share some settings - regarding this the connection is far from optimal, e.g. the clock to the serdes from FPGA1 is not independent from clock to GPLL input on FPGA4 etc.
- SFP cages could have been distributed differently. Now, all TrbNet links (SFP1-4) are on one side and all GbE capable links on the other two (SFP5-8). Usually only few ports are in use, so a mix would be better.
- TRB_TO_ADDON_CLOCK is located on a port that does not support LVDS output
- All pins connected to a DLL and PLL are Input only and can not be used as outputs! On the central FPGA, these are pins FS_PE 4,5,7,8 and ADO_TTL 23,26,30,46
- SPARE_LINE between FPGA and AddOn-Connector are connected to Feedback input of PLL, not the normal input to PLL
- On peripheral FPGA lines 8 and 9 of DQUR0 are input only - they are removed from the LPF for now.
- Both GPLL inputs to peripheral FPGA are not usable - they are connected to Feedback input instead of normal input of PLL. They can still be used through normal routing, but source synchronous operation is not possible.
- The JTAG Connector is wrong. The line labelled TDI is TDO and vice versa.
- The 48V-to-6V converter gets quite hot without air-flow. When AddOns are mounted, the fan must be installed on the short side of the TRB, not on the long one.

5.2. Clock and Trigger Distribution

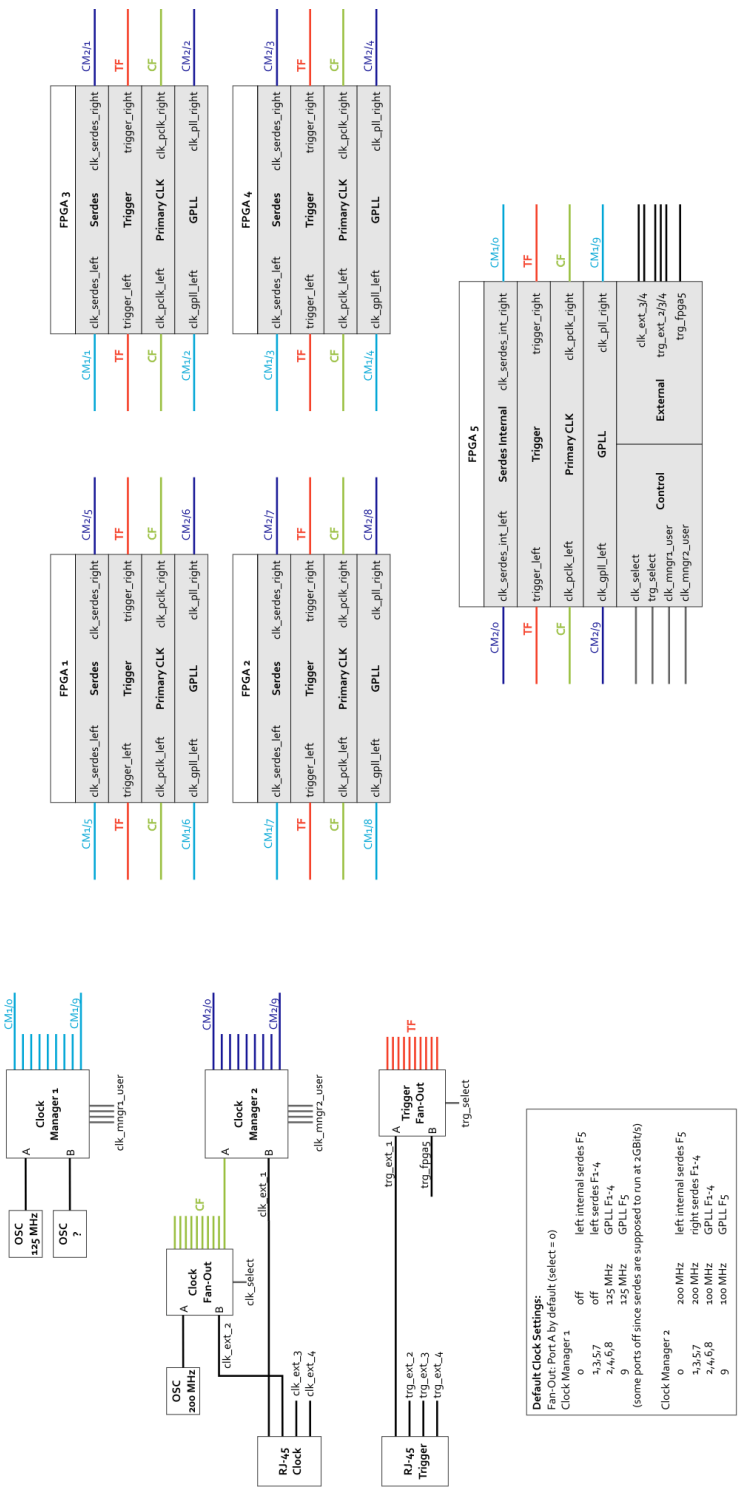


Figure 2: Clock and Trigger Distribution Network on TRB3.

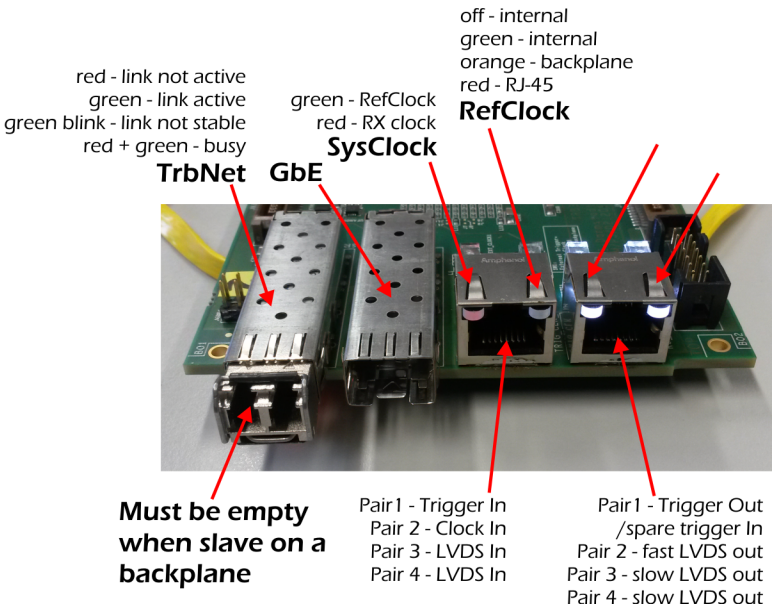


Figure 3: Connections and LEDs on the Trb3sc front-panel

6. Trb3sc

6.1. Basics

6.1.1. Powering Schemes

External 5V Connect a 1.5A power supply with not more than 5.5V to the black 3pin input.

Backplane 5V Connect a fitting power supply to the backplane. See the backplane description for further details. DC/DC converters are bypassed for lower noise. Power-LEDs will be off.

External 3.5V/1.4V Remove jumpers J12, J14 and J16. Connect a 1A power supply with 3.5-4.0V and 2A 1.4-1.7V to the black 4pin input. The 2.7V rail can be powered individually or connected to the 3.5V rail.

Backplane 3.5V/1.4V Remove jumpers J12, J14 and J16. Close fuses F1V2L, F3V3L and F2V5L. Connect a proper power supply to the backplane. DC/DC converters are bypassed for lower noise. Power-LEDs will be off.

6.1.2. Clock Inputs

The current system clock configuration is shown on two LEDs on the front-panel.

- The board has own 240 MHz and 125 MHz oscillators. The internal 240 MHz will be selected if there is no external clock available at power-up.

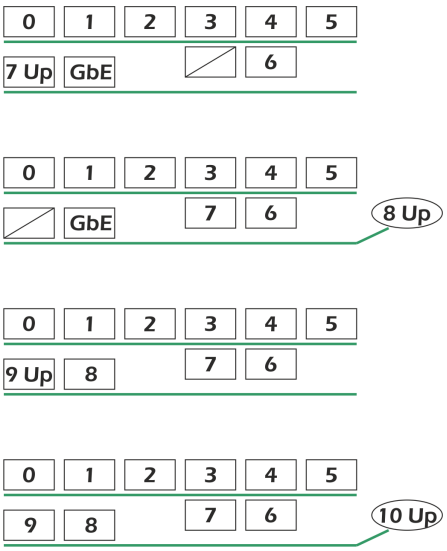


Figure 4: If the TRB3sc is used with SFP AddOn, these are the numbers of the SFPs as seen by the hub.

- A external clock can be fed in via the RJ-45 connector (left, pair 2) or from the backplane. The source is selected with a switch. At power-up the board searches for an external clock on the selected input. If none is found, the internal is used.
- The system clock can be recovered from the SFP1 input signal. This is selected at compile-time.

The native frequency of the board are 240 MHz, logic running at 120 MHz. Nevertheless, running at 200/100 MHz will be used for compatibility with existing setups. Most logic should be able to run at the higher speed when selected during compilation.

6.1.3. Trigger Input/Output

- The default trigger input is pair 1 on the left RJ-45 on the front-panel
- The front-panel trigger input can be rerouted to the second RJ-45 if separation of trigger and clock is required.
- The trigger can be supplied from the backpanel, selected by the switch (same as clock setting)

6.1.4. Other I/O

HDR_IO is available for any general purpose I/O. All lines are LVCMOS25. By default, SPI and UART are available:

Pin	Usage
1	UART TX
2	UART RX
3	SPI MOSI
4	SPI MISO
5	SPI CLK
6	SPI CE
7	(LCD DC)
8	(LCD Reset)
9	
10	
11	3.3V
12	3.3V
13	GND
14	GND

SPI channels 0 to 3 are linked to the AddOn connector (e.g. four Padiwa chains), channels 4 and 5 are used on additional KEL connectors. Channel 8 is reserved for HDR_IO.

Optionally, a LCD can be connected. In this case SPI channel 8 is not used.

6.1.5. Serial Links

By default, SFP1 is used for GbE, SFP2 for TrbNet. SFP2 must be removed if the board is to be used on a backplane as slave module. Removing the SFP selects the backpanel as TrbNet input.

By default, the use of Serdes channels is as follows:

Block	Channel	Usage
A	0	Backplane, Master: TrbNet to Slave 1, Slaves: TrbNet input from backplane
	1	Backplane, Master: TrbNet to Slave 2
	2	Backplane, Master: TrbNet to Slave 3
	3	Backplane, Master: TrbNet to Slave 4
B	0	AddOn Connector, Master: TrbNet to Slave 8
	1	AddOn Connector, Master: TrbNet to Slave 7
	2	AddOn Connector, Master: TrbNet to Slave 6
	3	SFP2 (TrbNet), can be re-routed to AddOn-Connector
C	0	AddOn Connector, Master: TrbNet to Slave 5
	1	AddOn Connector, Master: TrbNet to Slave 9
	2	AddOn Connector
	3	AddOn Connector
D	0	SFP1 (GbE)
	1	AddOn Connector, can be re-routed to SFP2
	2	not used
	3	not used

The TrbNet uplink (either Sfp or backplane) is synchronous, i.e. no clock tolerance compensation is enabled. Hence, both boards connected with the link must share a common clock source. For low-accuracy applications, the clock can be recovered from the optical link. In this case, no additional clock distribution is needed. If running in a crate, clock recovery is available, but not necessary as the backplane distributes a clock signal to all boards connected.

6.1.6. Modifications

The following changes compared to the original schematics are to be made:

R12, R14 270 Ohm, LED is too dim
R13, R15 680 Ohm, LED is too dim
R16, R17 680 Ohm, LED is too bright
Patchwire Disconnect R14 from 2.5V (rotate by 90°), patchwire to Pin 2 of switch - green LED shows status of clock select. See picture.
C21, C22, C24, C25 Replace by 0R (AC coupling for trigger signals)
R8, R9, R10, R11 Replace by proper 100 Ohm termination (AC coupling for trigger signals)

On the backplane, the AC coupling of the trigger signals needs to be removed as well:

C21, C22, C24, C25 Replace by 0R (AC coupling for trigger signals)
R8, R9, R10, R11 Replace by proper 100 Ohm termination (AC coupling for trigger signals)

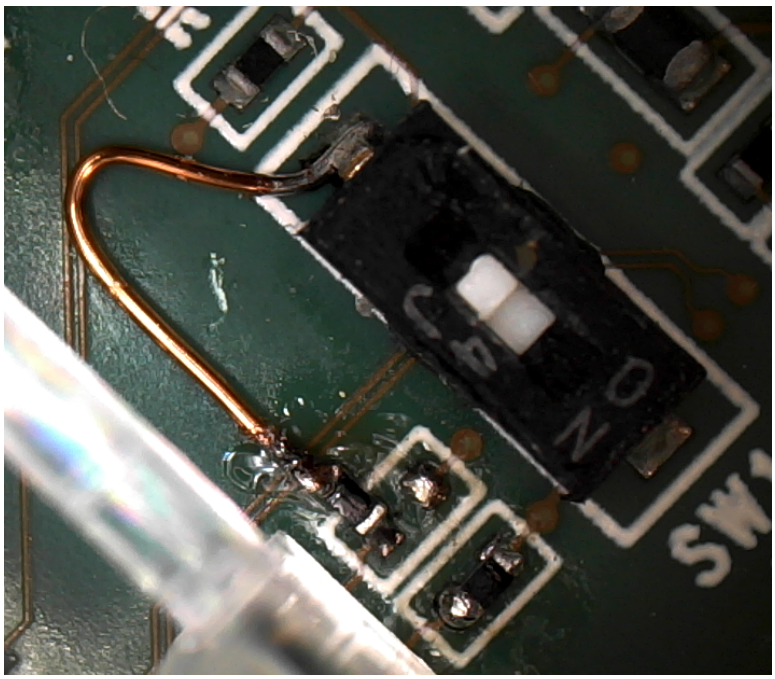


Figure 5: Trb3sc patch to display status of clock input switch

6.2. FPGA based TDC calibration

In order to calibrate the TDC data directly on a TrbSc, a FPGA based online TDC calibration was implemented. If this method is part of the FPGA design, the online calibration method can be activated (odd values) or deactivated (even values) with register `0xE000` of the TrbSc. If the online calibration is deactivated, the TrbSc can be used like a normal TrbSc.
The calibration method is the linear calibration. Each TDC channel is calibrated individually. The TDC data is used to generate a new linear calibration and it is directly calibrated.
The output format of the FPGA based calibration is slightly different to the software based calibration: The fineTime values are now fixed between 0 and 1000. 0 equals a fine time of 0 ns, 1000 equals 5ns. In order to get the fineTime in ns, the output has to be multiplied by 5. With Go4 it is possible to multiply by 5 if you set an additional linear calibration between 0 and 1000.

In case that a TDC data value is out of the calibration range, it is put to the value 1010 (value is too small). If it is greater than the range, the value is put to 1015. This has to be taken into account for further data processing. If Go4 is used, the values greater 1000 are set to 1000. This leads to incorrect data. This has to be changed directly in the go4 source code until now (10. January 2018).

The amount of TDC data that is necessary (statistics) to create a new calibration can be set by hand (Register `0xE001`). The value is set to 100.000 by default. The value has a lower limit

Register	Description
0xE000	activate (odd values) or deactivate (even values) the calibration
0xE001	amount of statistics of each channel
0xE007	default calibration values of channel 0
0xE008	default calibration values of channel 1
0xE009	default calibration values of channel 2
0xEXXX	default calibration values of channel XXX

of 10.000. It is not possible to go below this value.
If the FPGA is started, the first 100.000 TDC data values have no calibration. The calibration is done with standard values. There is the possibility to load values to the Flash memory. If the flash is loaded with values, they are loaded at startup and directly used for the first 100.000 calibrations of a channel.

The flash address for this feature is starting at 0xE007 for channel 0. For channel 3 it is 0xE00A and so on. The data written to this address consists of the lower fineTime limit and the upper fineTime limit. The lower limit is the smallest fineTime bin that is greater than 0. The upper limit is the greatest fineTime bin that is not equal 0. Bit 0-9 is filled with the lower value. Bit 10-19 is filled with the upper value.
The data is written to the flash by *flash_settings.pl* from *daqtools/tools*.

Currently (10. january 2018) only a 11 channel TDC design for the TrbSc is available and tested. More channels could not be tested due to missing manpower for TDC development.

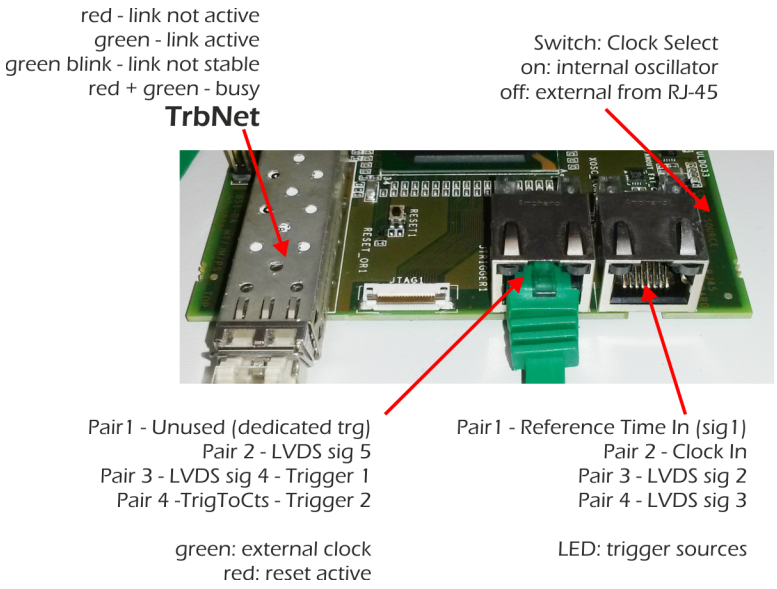


Figure 6: Connections and LEDs on the DiRich combiner front-panel

7. DiRich

8. AddOns

8.1. TDC AddOn

- Schematics ADDON1ADA1_alles.pdf
- Pin-out file for the FPGA cvsroot/trb3/base/trb3_periph_ada.lpf

8.2. 32-Pin AddOn

- Designed for time measurements with PADI FEE. Has 32 input pins and rising an falling edges are measured in 64 channels.
- Has also SPI interface for DAC configuration
- Schematics
- Has the same pin out as the TDC AddOn for the first 32 pins
- Pin-out file for the FPGA cvsroot/trb3/base/trb3_periph_ada.lpf

8.3. Multi-Test-AddOn

- Schematics MultiTestAddon1_alles.pdf
- Pin-out file for the FPGA cvsroot/trb3/base/trb3_periph_multitest.lpf

8.3.1. Known bugs

- The serial interface to ADC 1 is not usable - CSB and PDWN are input-only on the FPGA
- SDIO is missing a pull-down. This way the device runs in low-power mode on the digital outputs.
- LVDS_INP_2 is the only LVDS port that is terminated by an external resistor. Do not switch on corresponding LVDS termination in FPGA

8.4. Hub AddOn

- Schematics SFP-Addon1_alles.pdf
- Pin-out file for the FPGA cvsroot/trb3/base/trb3_periph_sfp.lpf

8.5. MVD AddOn

- Schematics
- Pin-out file for the FPGA

8.6. CTS AddOn

An AddOn for the central FPGA featuring some CTS I/O connections. Since it was not foreseen to actually use such an AddOn when the TRB3 was designed, there are several possible issues with the design:

- All LVDS outputs from the FPGA to the AddOn do not use standard LVDS-outputs, but emulated signals (FPGA produces differential TTL signals which are sent through three resistors to adjust signal heights and impedance. The performance might be somewhat lower than on dedicated LVDS lines (e.g. on the two RJ-45 connectors on the TRB3 itself)
- Most of the differential lines on the AddOn are not routed differentially on the TRB3 which might have an influence on the signal quality
- The connector JINLVDS can not be used as LVDS input - during layout the signals have been mixed up. Only TTL signals are available, despite on the last two pairs which might be used as LVDS but are not configured as LVDS by default.
- The board has two analog discriminator inputs, using a reference produced by the FPGA. The pin-out of the connector is as shown in table 4.

8.7. General Purpose AddOn

- Schematics GPIN_AddOn1_alles.pdf

Name	I/O	Type	Cnt	Description
ECLIN	ECL In	RJ-45	4	ECL Standard input. Routed as TTL signal on-board
NIMIN	NIM In	LEMO	2	NIM Standard input. Routed as TTL signal on-board
JIN1	LVDS In	RJ-45	4	
JIN2	LVDS In	RJ-45	4	
JINLVDS	TTL I/O	pinhead	16	first two and last two pins are GND.
COMPARATORIN	Analog In	pinhead	2	See extra table for pin-out
PWMOUT	TTL Out	pinhead	2	See extra table for pin-out
JOUT1	LVDS Out	RJ-45	4	
JOUT2	LVDS Out	RJ-45	4	
JOUTLVDS	LVDS Out	pinhead	8	first and last two pins are GND
JTTL	TTL I/O	pinhead	16	first and last two pins are GND
TRGFANOUTADDON	LVDS Out	–	4+8	trigger signal to fan-out chip. Available on RJ-45 and pinhead
LEDBANK			8	8 yellow LED in a row
LEDFAN*			4	4 colourful LED next to fan-out chip
LEDRJ			2x6	red and green LED for each RJ-45 connector. Order is JIn1, JIn2, JOut1, JOut2, JFan2, EclIn

Table 3: I/O connectors and devices on CTS AddOn

Pin	Signal
1	Input 0
2	Filtered DAC 0
3	GND
4	Raw DAC 0
5	GND
6	Raw DAC 1
7	Input 1
8	Filtered DAC 1

Table 4: Pinout of the discriminator input of the CTS AddOn

8.8. ADC AddOn

- Schematics AddOn_ADC1-ALL.pdf

8.8.1. Data Format

Word	Bits	Value	Description
HDR	31 – 28	0x4	ADC Header. Not required, e.g. can be omitted if the first word sent is DAT1
	23 – 20	0 – 11	ADC Number
	19 – 16	0 – 3, F	ADC Channel. If channel is 0xF, the words following (e.g. status information) belongs to all channels of this ADC.
	7 – 0		Number of data words from this channel
DAT1	31 – 28	0x0	Normal (verbose) Data
	23 – 20	0 – 11	ADC Number
	19 – 16	0 – 3	ADC Channel
	15 – 0		Data
STAT	31 – 28	0x1	Status Information word. E.g. to signal a broken channel
	27 – 24		Type
	23 – 0		Information
INFO	31 – 28	0x2	Configuration of the device. Information about ADC status. Only sent in trigger type 0xe events. Configuration words are sent for each ADC, the corresponding device is given in a HDR word.
	27 – 20		Word type, see separate table
	19 – 0		Data
DAT2	31	1	Compressed data word. Note the reduced bit width of 15 Bit. This type of word can only be sent after a HDR word.
	30 – 16		First data word
	14 – 0		Second data word

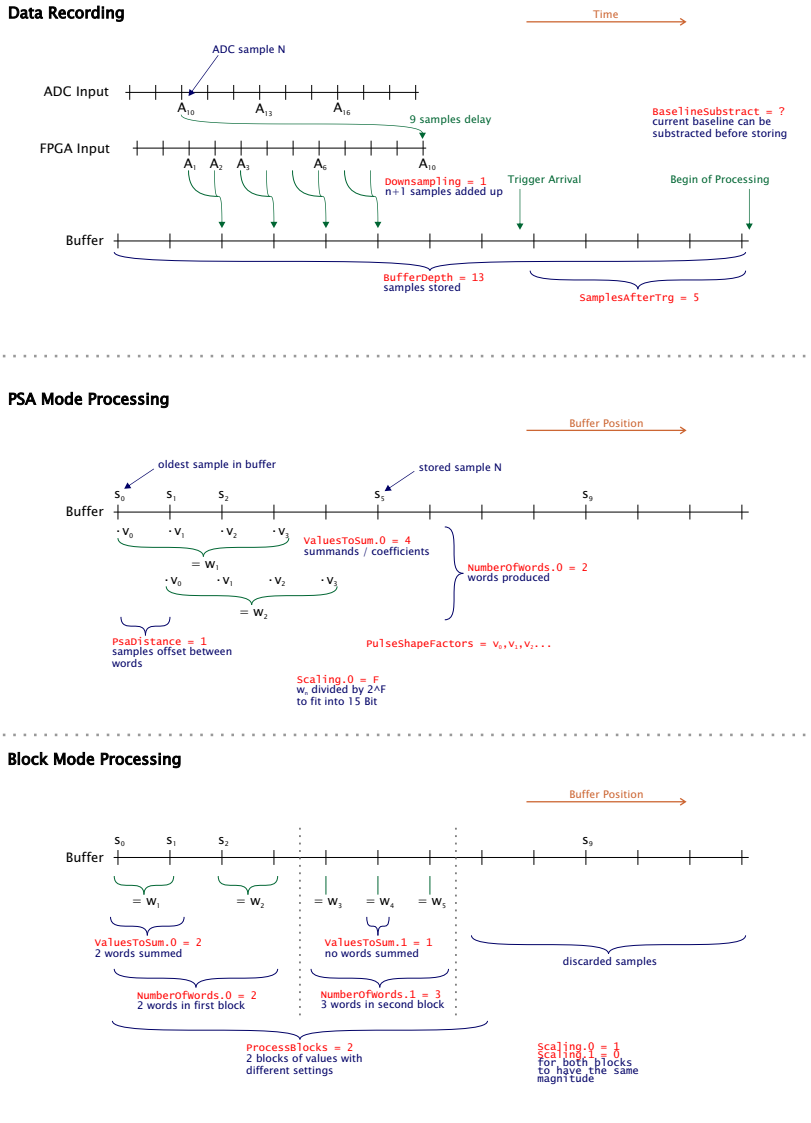


Figure 7: ADC data recording and processing

Type	Bits	Description
0x01	10 – 0	Configured buffer depth
0x02	10 – 0	Samples after trigger
0x03	17 – 0	Trigger threshold
0x04	17 – 0	Readout threshold
0x05	7 – 0	Number of ADC samples to sum before storing in Fifo
	11 – 8	Samples to average for baseline calculation (actual number is 2**N of given value)
	13 – 12	Number of blocks to process
0x06	7 – 0	Samples to sum before sending data (first block)
	15 – 8	Number of sums to be sent (first block)
	19 – 16	Scaling factor (result is divided by 2**N before sending (first block)
0x07		See 0x06, but for second block
0x08		See 0x06, but for third block
0x09		See 0x06, but for fourth block
0x10	15 – 0	Calculated baseline of first channel
0x11	15 – 0	See 0x10, but for second channel
0x12	15 – 0	See 0x10, but for third channel
0x13	15 – 0	See 0x10, but for fourth channel
0x14	23 – 0	Samples within last 100 ms for first channel
0x15	23 – 0	See 0x14, but for first channel
0x16	23 – 0	See 0x14, but for first channel
0x17	23 – 0	See 0x14, but for first channel

8.8.2. Slow Control Registers

An up-to-date list of registers can be generated in daqtools/xml-db by running

```
./xml-db2tex.pl -e ADC --pdf -o adc
```

8.9. Padiwa

Please see the separate documentation maintained in

```
git clone git://jspc29.x-matter.uni-frankfurt.de/projects/padiwadocu.git
```

A freshly compiled PDF can be downloaded here.

9. Related Boards

9.1. CBM-RICH

9.2. CBM-TOF

Part IV.
Design Components

10. New VHDL Project

A not complete list of steps how to create a new TRB3 VHDL project.

- Choose a projectname. It has to start with `trb3_periph_*` or `trb3_central_*`.
- Create a new subdirectory inside `./trb3/`. Choose a short, descriptive name for the project. Change to this directory.
- Create subdirectories

  `code` for your own vhd codes for this project
  `cores` for generated ipcores
  `sim` for the simulation project

- Copy necessary files from another project. Choose one using the same FPGA you want to create your project and if possible one that uses the same pinout.

  `compile*.pl` The main script that runs synthesis, map, par...
  `*.prj` The project settings and list of source files
  `*constraints.lpf` The file with constraints specific for a design
  `*.p2t` Settings for the place and route tool
  `*.vhd` The top-level vhd file as basis for the new design

- Edit `compile_constraints.pl` This script takes one optional parameter pointing to the workdir *relative to the script itself* (if omitted, `./workdir/` is assumed). The program is invoked by the other `compile*.pl` scripts and the `/base/create_project.pl` tool and has the following tasks:
  - Create the workdir if not existing
  - Execute `base/linkdesignfiles.sh` with the correct parameters to account for a varying number of `../` of the relative links generate and depending on the position of the workdir relative to the repositories root directory.
  - Combine global and design specific `*.lpf`-files into a single file `{workdir}/{topname}.lpf`.
  - Optional: Generate design specific script that are invoked in the build process.
- Edit `compile*.pl`
  - Set the `$projectname`
  - Check that all configuration options (the marked block in the beginning of the file) match your local environment.
- Edit `$projectname.prj`

- Set your projectname (four places in total)
- Add / Remove source files as necessary

- Try to run the compile script.
- Optional: run `base/create_project.pl` which will generate a diamond project from your `*.prj` file. Further, it executes the `compile_constraints.pl` tool to obtain the constraint and configuration files. You can rerun this program at any time – in this the project file are rewritten which may undo manual changes. Observe, that the script only extracts lpf/vhd/v files and outputs a warning if non-supported files are found in your `*.prj` file.

11. TDC

11.1. Building Blocks

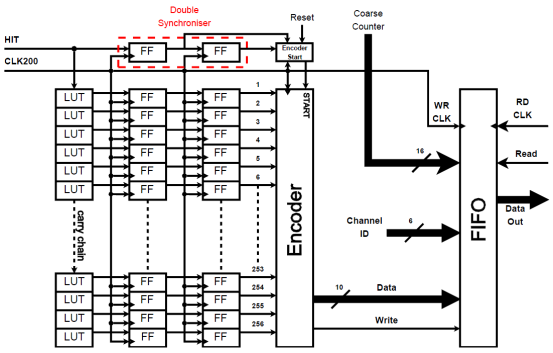
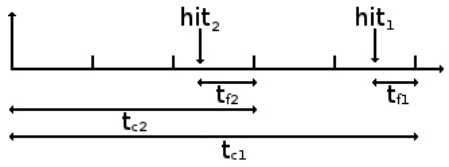


Figure 8: Diagram of a channel.

The architecture of the TDC consists of a fine time measurement block, a coarse counter with granularity of 5 ns, an encoder for the conversion of the result to binary number and a First-In-First-Out (FIFO) memory block for data storage. A block diagram of the designed TDC is shown in Figure 8.

In each TDC channel the measurement result of the fine time measurement block is converted to a binary number in the encoder and saved in the FIFO with a coarse time flag. The time interval between different signals measured at different channels can be calculated by simply taking the difference of the relevant measurement results. In Figure 9 an example of two signals, their coarse and fine time values and the calculation of the time interval between these signals are shown.



(a)

$$\Delta t = t_{hit_1} - t_{hit_2} = (t_{c1} - t_{f1}) - (t_{c2} - t_{f2})$$
$$= (t_{c1} - t_{c2}) - (t_{f1} - t_{f2})$$

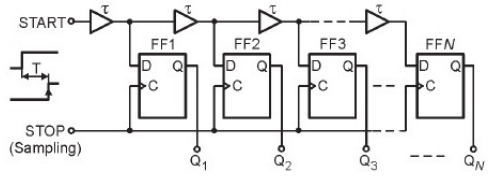
(b)

Figure 9: (a) Illustration of two time measurements and (b) calculation of the time interval between them.

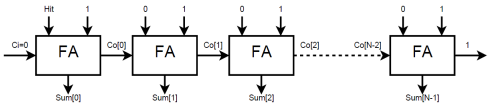
11.1.1. Fine Time Measurement

For fine time measurements the Tapped Delay Line (TDL) method is used. This method is based on a delay path with delay elements, which have similar propagation delays. With the start signal the propagation along the delay line starts and with the stop signal the output of the each delay element is latched (Figure 10a). The location of the propagating signal along the delay line defines the fine time between start and stop signals.

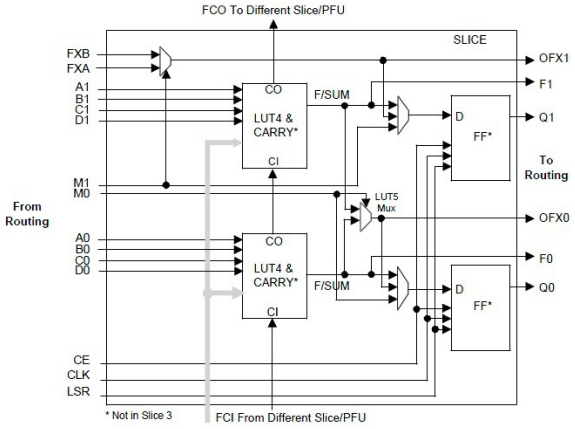
The delay line is realised on the dedicated carry chain structure of the Lattice FPGA using the 4-bit Look Up Tables (LUT) and the registers, as delay elements and as latches respectively. In Figure 10c the diagram of a slice with 2 LUTs and 2 registers is shown.



(a) Tapped Delay Line. [Kal04]



(b) Tapped Delay Line realised with full adders. [SAL06]



(c) Lattice FPGA slice diagram. [Lat09]

Figure 10: Look Up Tables programmed as full adders along the carry chain are used as delay elements of Tapped Delay Line and their outputs are registered at the registers located at the same slice.

In the designed TDC the stop signal is defined as the next rising edge of the system clock after the start signal. As the maximum time interval to be measured by the fine time counter is one clock cycle, the total propagation time of the carry signal, along the delay line, has to be longer than a clock period. Manual placement of the delay elements and the corresponding registers are done in order to achieve a uniform delay along the line.

The propagation delay of a delay cell depends on temperature and the consistency of the power supply. This dependency effects the resolution of the TDC. In order to overcome this problem, the output data of the TDC has to be calibrated.

11.1.2. Fine Time Calibration

For an FPGA TDC, digital calibration has to be applied to the raw data. Bin-by-bin calibration [SKP97] is suitable for this purpose. In this method a DNL histogram is created for a given number of hits. Assuming the hit signals are completely random and not correlated with the

clock signal, the hits should be equally distributed over the time interval of the fine interpolator, which is the clock period. Then the bin width can be calculated from,

$$BW = n \times \frac{T_o}{N} \tag{1}$$

where n is the actual number of hits of the bin and N is the total number of hits. Using this calculation and the DNL histogram, which is already calculated, a *Look-Up Table* (LUT)[1] is created to store the time values of each bin. The corresponding time value for each bin is the middle point of the bin width values. The time value of the first bin is the half of the bin width of the first bin. For the second bin, it is the summation of the bin width value of the first bin and half of the bin width value of the second bin, and so on. After creating the LUT this is used for subsequent measurements. An example of calibrated and uncalibrated-calibrated time values are shown in Figure 11. As may be seen from the graph, the quantisation levels of the calibrated data are distributed along the time more evenly than the uncalibrated-calibrated data quantisation steps. As these quantisation steps effect the non-linearities of the TDC, calibration has lowers the non-linearity values.

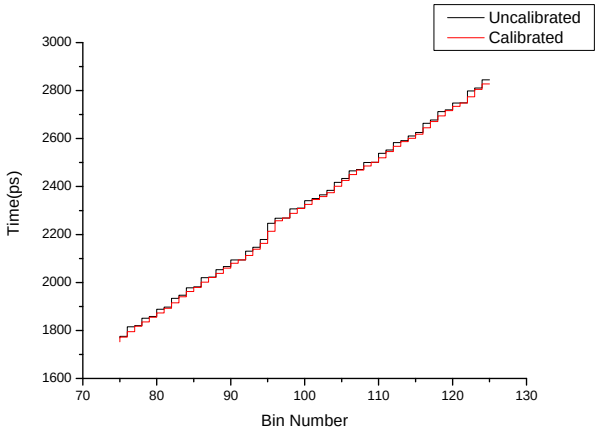


Figure 11: An example of a LUT created by the bin-by-bin calibration method (Adapted from [WS08])

This calibration method is correct for a given temperature and supply voltage values, and the calibration LUT has to be updated regularly during the offline analysis.

For channels, which don't get enough signals - statistics - for proper calibration, a calibration trigger functionality is implemented. The trigger type 0xd sent from the CTS is used to shoot every channel with the signals from the second oscillator on the board - uncorrelated with the oscillator used for fine time measurements - in order to have sufficient statistics for calibration.

[1] A lookup table is used to display information, which is recorded previously, corresponding to an individual input.

It is advised to separate the calibration data taking and physical event data taking, as the first event generated by the physical trigger after the calibration trigger might still have calibration data.
This problem is fixed with the TDC version 2.1.2

11.2. Features

11.2.1. Trigger Window and Trigger Mode

In order to reduce the data load on the DAQ a feature called trigger window is implemented. With this feature the user can define the interested time interval and filter the hits occurred in this time interval. An illustration of the trigger window feature is shown in Figure 12. The trigger window is relative to the rising edge of the reference time at the *TRIGGER_INP1* (see Figure 1) with the granularity of 5 ns. The *Pre-Trigger Window* and *Post-Trigger Window* widths can be set via slow control (Table 13).

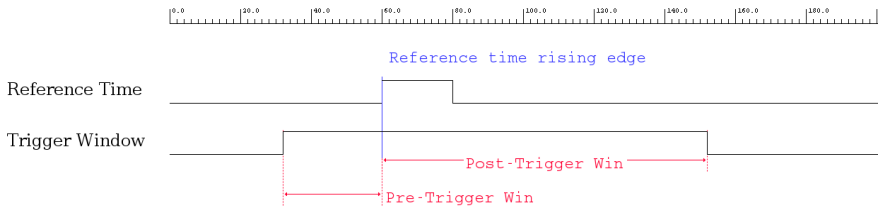


Figure 12: An illustration of trigger window relative to the reference time.

Trigger mode is controlled by register 0xc800 bit 12. If it is set to triggered mode ('1'), the epoch and coarse counters are reset after each trigger window. If this bit is set to trigger-less mode ('0'), the epoch and coarse counters are never reset, unless there is a system wide reset. They will run until they have an overflow. **This feature is disabled the after tdc version 2.0.0, as it is obsolete for the analysis software.**

11.3. Data Format

The TDC data consists of 4 different kinds of information: TDC header, time data, debug, epoch counter and reserved.

11.3.1. TIME DATA

The data format of the ***time data*** word is shown below:

Any word starting with the bit "1" indicates a time data word from the TDC in the system.

7 bits are reserved for indicating the channel number in the TDC. The first channel – channel "0000000" – is used to measure the reference time. All TDCs in the system measure the same reference time in this channel, so that they can be all synchronised.

Data Format	Bits	Description
0x0	31	Time Data Marker
	30-29	reserved
	28-22	channel number
	21-12	fine time - sum of the two transition of the WUL
	11	the type of the measured edge - ’1’ rising, ’0’ falling edge
	10-0	coarse time - 5ns granularity
0x1	31	Time Data Marker
	30-29	reserved
	28-22	channel number
	21	reserved
	20-12	fine time - individual values of the two transition of the WUL
	11	the type of the measured edge - ’1’ rising, ’0’ falling edge
	10-0	coarse time - 5ns granularity

Table 5: The data format of the *TIME DATA* word.

Three time informations are generated for each signal detected by each channel; epoch counter, coarse counter and fine counter. The epoch counter word is explained in 11.3.4. The coarse time information has the granularity of 5 ns (period of the system clock). The range of the coarse time is 10,24 us. The fine time has the range of 5 ns but doesn’t have a fixed granularity. The fine time information has to be calibrated using the statistic collected by the individual channel (for details see 11.1.1). The response efficiency of the TDC is 100%. So even if a proper fine time for a hit cannot be generated, the TDC will register the hit and write a time data word in the memory with a dummy fine time x"3FF". This hit should be excluded from the calibration and time calculations.

11.3.2. TDC HEADER

The data format of the ***header*** word is shown below:

Bits	Description
31-29	“001” TDC Header marker
28	reserved
27-24	TDC Data Format type
23-16	reserved
15-0	Error bits

Table 6: The data format of the *TDC HEADER* word.

Any word starting with the bits "001" indicates a header word from the TDC in the system. The TDC is defined with the previous word in the data stream – TDC network header.

The trigger type and trigger random codes from the TrbNet are moved to the trailer word since the tdc_v2.3.

The error bits are used to indicate any error might occurred in the TDC since the last trigger. The error bits coded in the header is given in Table 7.

Bit	Explanation
0	At least one of the channel ring buffers is overwritten.
1-15	Reserved.

Table 7: TDC Header Error bits.

11.3.3. DEBUG - Status Information

Various statistics and information of the TDC can be written to the data. This options can be enabled in two ways:

- via slow control register (0xc800 - bit 4)
- via status information trigger (0xE)

In the first case the status information will be sent out after the tdc time data with the physics trigger (0x1) or tdc calibration trigger (0xD). In the later case only the status information will be written out.

The data format of the ***debug*** word is shown in Table 8:

3 bits	5 bits	24 bits
"010"	debug mode	debug bits

Table 8: The data format of the *DEBUG* word.

"010"	3 bits	Debug marker
debug mode	5 bits	It is used to define the different debug bits
debug bits	24 bits	Debug information and statistics to the user

The debug information sent is given in Table 9.

The debug words sent with DAQ can be accessed also via slow control registers (see Table 14).

Debug Mode	Name	Explanation
"00000"	Trigger number	Number of valid triggers received
"00001"	Release Number	Number of release signals sent
"00010"	Valid timing trigger number	Number of valid timing triggers received
"00011"	Valid NOtiming trigger number	Number of valid triggers received which are not timing triggers
"00100"	Invalid trigger number	Number of invalid triggers received
"00101"	Multi timing trigger number	Number of multi timing triggers (triggers received before trigger is released) received
"00110"	Spurious trigger number	Number of spurious triggers received (in case of timing trigger is validated although it was a timing-trigger-less trigger)
"00111"	Wrong readout number	Number of wrong readouts due to spurious triggers
"01000"	Spike number	Number of spikes (pulses narrower than 40 ns) detected at the timing trigger input
"01001"	Idle time	Total time length, that the readout FSM waited in the idle state (with granularity of 10 ns)
"01010"	Wait time	Total time length, that the readout FSM waited in the wait states (with granularity of 10 ns)
"01011"	Total empty channels	Number of empty channels since the last reset signal
"01100"	Readout time	Total time length, that the readout occurred (with granularity of 10 ns)
"01101"	Timeout number	Total number of timeouts occured since the last reset
"01110"	Temperature	The temperature value read from the temperature sensor
"01111"		RESERVED
"10000"	Compile time 1	The first 16 bits of the compile time (the number of non-leap seconds since 01.01.1970)
"10001"	Compile time 2	The last 16 bits of the compile time (the number of non-leap seconds since 01.01.1970)

Table 9: Debug information word bitmap.

11.3.4. EPOCH Counter

As the global coarse counter has the time limit of ∼10 us, an overflow counter (EPOCH counter) is implemented in order to increase the measurement range. The data format of the ***EPOCH Counter*** word is shown below:

3 bits	1 bit	28 bits
"011"	reserved	EPOCH Counter

Table 10: The data format of the *EPOCH Counter* word.

The EPOCH counter is designed with 28 bits increasing the total measurement range up to ∼45,8 min. For each channel an individual EPOCH counter is implemented and they are incremented, when the coarse counter wraps around. The value of the EPOCH counter is kept in a register before it is written in the channel memory. It is written once per event per channel and only updated, if a time measurement takes place after the last increment of the EPOCH counter. In order to be on the safe side and not overflow the EPOCH counter, the readout trigger frequency can be set minimum to 24 Hz.

11.3.5. TDC TRAILER (was RESERVED before tdc_v2.3)

This data format is being used as the trailer word since the tdc_v2.3 to mark information, warning and errors about the status of the last event readout. The data format of the ***trailer*** word is shown below:

3 bits	1 bit	4 bits	8 bits	16 bits
"000"	reserved	trigger type	random code	error bits

Table 11: The data format of the *TRAILER* word.

"000"	3 bits	Trailer word marker
reserved	1 bit	Reserved for future use
trigger type	4 bits	The trigger type of the last event
random code	8 bits	The random trigger code of the last event from the endpoint
error bits	16 bit	Warnings and errors about the last event readout status

In case of any abnormal event readout the TDC readout will not stop the DAQ but rather mark this in the trailer word. The user should check the status of these bits for the analysis. The explanation of these bits are given in Table 12.

Bit	Explanation
0	Set if the trigger handler in TDC doesn't detect any reference time.
1	Set if a reference time precedes a non-timing trigger (case 3 in TrbNet Manual section 7.3)
2	Set if a timing trigger is delivered without a reference time (case 4 in TrbNet Manual section 7.3)
3	Set with the bit 2 to mark the missing reference time
4	Set if there are more than one detected reference times. The reference channel will generate time data for both of the pulses, unless they violate the dead time limit. (case 5 in TrbNet Manual section 7.3)
5	Set if the reference time was too short (<40 ns). The reference channel will still detect the pulse and generate a time data. (case 6 in TrbNet Manual section 7.3)
6	Set if no trigger validation arrives from the endpoint after a valid reference time. (case 7 in TrbNet Manual section 7.3)
7	Set if any timing trigger type except 0x1 is sent. The data will be readout normally.
8-15	Reserved.

Table 12: TDC Trailer Error bits.

11.4. Slow Control Registers

A set of control registers (0xc800) are assigned in order to access the basic controls, edit the features and debug information of the TDC. A detailed explanation of the control registers are given in Table 13.

Register		Addr	Bits	Description
BasicControl	(rw)	c800		Basic control for all channels
DebugOutput			0–3	Enables different signals to the HPLA* output for debugging with logic analyser
DebugMode			4	Enables the Debug Mode. Different statistics and debug words are sent after every trigger
LightMode			5	Enables the Light Mode. No header and reference channel information is sent if there are no recorded hits. Works only in the free streaming mode (trigger window off)
ResetCounters			8	Resets the internal counters
ResetCoarseCounter			13	Used to reset the coarse counters. Setting this bit signals for the coarse counter reset but the action will take place with the arrival of the next valid trigger in order to synchronise the coarse counters in a large system.

Continued on next page

Table 13 – *Continued from previous page*

Register	Addr	Bits	Description
CalibrationPrescaler		28–31	Used to divide the calibration hit frequency.
TriggerWindowConfig (rw)	c801		Configuration of the TriggerWindow feature
TriggerWindowBefore		0–10	Trigger window width BEFORE the trigger with granularity of 5 ns
TriggerWindowAfter		16–26	Trigger window width AFTER the trigger with granularity of 5 ns. ATTENTION: Minimum value is x"00f"!
TriggerWindowEnable		31	Trigger window enable
ChannelEnable.0 (rw)	c802	0–31	Enable signals/hits of the specific channel. LSB is channel 1.
ChannelEnable.1 (rw)	c803	0–31	Enable signals/hits of the specific channel. LSB is channel 1.
ChannelRingBufferSize (rw)	c804		Defines the size of the channel ring buffer size
MaxWords		0–6	Defines the size of the channel ring buffer size. Maximum value 124.
ChannelInvert.0 (rw)	c805	0–31	Inverts the polarity of the signals/hits of the specific channel. LSB is channel 1.
ChannelInvert.1 (rw)	c806	0–31	Inverts the polarity of the signals/hits of the specific channel. LSB is channel 1.

Table 13: The control registers of the TDC.

Some status information and statistics of the TDC can be accessed via the status registers (0xc100). The status registers of the TDC are explained in Table 14.

Register		Addr	Bits	Description
BasicStatus	(r)	c100		Basic config and status information for all channels
ReadoutFSM			0–3	Debug word of the TDC readout FSM
WriteoutFSM			4–7	Debug word of the TDC writeout FSM
ChannelCount			8–15	Number of implemented channels
RefTimePolarity			16	Reference time polarity
TdcVersion			17–27	TDC core version number
TriggerType			28–31	Trigger type
DebugRegister	(r)	c101		Various state machine states.
TriggerHandlerFSM			0–3	Debug word of the Trigger Handler FSM
TriggerTime	(r)	c102	0–31	Arrival time of the last valid timing trigger at the reference channel with granularity of 5 ns
TriggerWindowStatus (r)		c103		Status of the TriggerWindow feature
TriggerWindowBeforeValue			0–10	Trigger window width BEFORE the trigger with granularity of 5 ns
TriggerWindowAfterValue			16–26	Trigger window width AFTER the trigger with granularity of 5 ns
TriggerWindowEnabled			31	Trigger window enabled?
TriggerCounter	(r)	c104	0–23	Number of valid triggers received
TimingTriggerCounter (r)		c105	0–23	Number of valid timing triggers received
NoTimingTriggerCounter (r)		c106	0–23	Number of valid triggers received which are not timing triggers
InvalidTriggerCounter (r)		c107	0–23	Number of invalid triggers received
MultiTimingTriggerCounter (r)		c108	0–23	Number of multi timing triggers received (triggers received before trigger is released)
SpuriousTriggerCounter (r)		c109	0–23	Number of spurious triggers received (in case of timing trigger is validated although it was a timing-trigger-less trigger)
WrongReadoutsCounter (r)		c10a	0–23	Number of wrong readouts due to spurious triggers

Continued on next page

Table 14 – *Continued from previous page*

Register		Addr	Bits	Description
SpikesCounter	(r)	c10b	0–23	Number of spikes (pulses narrower than 40 ns) detected at the timing trigger input
IdleTime	(r)	c10c	0–23	Total time length, that the readout FSM waited in the idle state (with granularity of 10 ns)
WaitTime	(r)	c10d	0–23	Total time length, that the readout FSM waited in the wait states (with granularity of 10 ns)
TotalEmptyChannelsCounter	(r)	c10e	0–23	Total number of empty channels since the last reset signal
ReleaseCounter	(r)	c10f	0–23	Number of release signals sent
ReadoutTime	(r)	c110	0–23	Total time length of the readout process (with granularity of 10 ns)
TimeoutCounter	(r)	c111	0–23	Number of timeouts detected (too long delay after the timing trigger)
FinishedCounter	(r)	c112	0–23	Number of sent finished signals

Table 14: The status registers of the TDC.

11.5. TDC Version Table

Version	Release Date	Release Notes
tdc_v2.4*	08.03.2015	Faster clock (400 MHz) for the delay line is used.
tdc_v2.3	08.12.2015	Trailer word is introduced to mark some error bits. Temperature value is added to the data stream for status trigger (0xE). Some bugs are fixed for physics and status trigger mixture. Compile time vlaue is added to the data stream for status trigger (0xE).
tdc_v2.2	07.10.2015	The delay line size is decreased to 288 from 304. The trigger window end and coarse counter reset signals are distributed via SECONDARY clock nets A bug in the semi-asynchronous stretcher (combinatorial reset signal caused blockage) is removed.
tdc_v2.1.6	06.08.2015	Updated the codes with record based bus lines.
tdc_v2.1.5	22.06.2015	Extra coarse counter reset register for higher frequency.
tdc_v2.1.4	17.06.2015	Several bug fixes for the stretcher option.
tdc_v2.1.2	28.01.2015	In case of a missing reference time a header error bit is set and DAQ keeps running. Grass hits in ToT with calibration trigger is removed. The ToT mean value - 10ns gives the stretching offset of the channel. Channel invert bits are implemented. Trigger window bug-fix. Resource usages in Channel_200 and Channel entity are decreased. Hit detection is increased to 2 bits. Coarse counter number is increased to channel number for better timing. Instead of the internal oscillator 125MHz clock input is used for the calibration.
tdc_v2.1.1	28.01.2015	The dead time of the TDC is decreased to 20ns. Small bug with "Light Mode" is removed. "Data Limit" parameter is removed, as it is not needed due to the dynamic buffer size. Coarse/Epoch counter misalignment bug is fixed. Channel input is blocked until the falling edge information is written in the ring buffer to avoid data mismatch.

Continued on next page

Version	Release Date	Release Notes
		Ring buffer overwrite bit is implemented.
tdc_v2.1.0	15.12.2014	The ring buffer almost full threshold is made dynamic in order to "mimic" a adjustable ring buffer size.
tdc_v2.0.1	05.12.2014	Calibration-physic trigger switching problem is fixed. With the calibration trigger 50ns pulses are sent to the channels in order to calibrate the ToT measurements in the channels. There are some grass hits around the main peak.
tdc_v2.0	01.12.2014	Double edge detection in a single channel is implemented.
tdc_v1.7.3	15.08.2014	Hit scaler register size is increased to 31 bits.
tdc_v1.7.1	29.07.2014	Feature Bit support. Tidy up the entities.
tdc_v1.7	24.06.2014	Parallel working Readouts are implemented. Trigger time calculation is done in the trigger handler.
tdc_v1.6.3	24.06.2014	Bug fix in the hit rate counters (syncronisation problem).
tdc_v1.6.2	08.05.2014	Small bug fix in the wait time for data transfer to buffer.
tdc_v1.6.1	06.05.2014	Less EPOCH counter - unnecessary EPOCH words, which occur with enabled trigger window, are eliminated from the data stream. FSM initialisation problem by the Channel_200 entity is solved. Channel FSM debug words are written to bus 0xc200. Number of coarse counters is increased to 16 to ease the fanout. Bug fix for the missing data with the calibration trigger. Bug fix for the duplicate data when trigger window is enabled.
tdc_v1.6	20.01.2014	Epoch counter bug fix (data word - epoch word place swap). Trigger window bug fix (epoch counter more than 24 bit had integer conversion problem. Trigger window right side control is enabled). Readout algorithm change (the channel FIFOs are readout to intermediate buffer, so the later channels in the readout order are kept as the trigger arrival time). Trigger on TDC channel (the feature for triggering on TDC channel is implemented) Reference channel hit rate counter implemented. The channels (incl. ch0) can be calibrated with the internal oscillator with different frequencies (see manual slow control registers).

Continued on next page

Version	Release Date	Release Notes
		The coarse counter can be set to reset via slow control. The action will take place when the first valid trigger arrives.
tdc_v1.5.1	20.06.2013	Efficiency bug fix (epoch counter update - hit at the same time). Hit level bit bug fix for the web server. Reference Channel coarse counter alignment fix.
tdc_v1.5	03.05.2013	TDC calibration trigger is implemented in order to shoot every channel with sufficient # of hits for proper calibration. Also the TDC is adapted for short pulses.
tdc_v1.4	18.04.2013	Limiting data transfer functionality is added. Use 0xc804 register to define the # of word per channel to be read-out.
tdc_v1.3	05.03.2013	Encoder efficiency is increased to 100%. Extra bits are encoded in the data (low resolution and no successful binary conversion, see the manual). Channel block during the readout is removed. Only the relevant hits per trigger are readout. Control registers are moved to 0xc800.
tdc_v1.2	12.11.2012	First stecher prototype is successfully implemented. Some bugs are fixed.
tdc_v1.1.1	07.11.2012	The status registers are moved to the bus address 0xc100. Also debug registers (encoder start, FIFO write, lost hits) are included in the bus - 0xc200 0xc300 0xc400
tdc_v1.1	26.10.2012	Readout process is collected in an individual entity.
tdc_v1.0	25.10.2012	The time measurement interval is extended with a 28-bit epoch counter.
tdc_v0.5	22.10.2012	Hit counter registers and LVDS receiver output level can be reached via slow control.
		* Design under construction...

12. Additional Modules

12.1. DAC Programming

Programming the DAC for threshold generation is simple: A standard SPI interface takes 32 Bit of data, the device is chainable. A Perl software module cares about the data content, a simple VHDL core outputs the data and controls the CS signal.

Slow Control Interface

- 32 Bit Data Memory: 0xd400 - 0xd40f, Chain select mask 0xd410, Length register 0xd411
- Transfer is started when the length register (counting 32 Bit words) is written.
- While busy, the writing to the length register will be ignored and gives back a no-more-data flag.
- Doing a memory write with 18 words will do the job, should be faster than two individual accesses for data and length.
- All data is sent MSB first (Bit 31), Bits 31-24 are the don't-care-Bits of the DAC.
- Interface speed: e.g. 6.25 MHz -> max. 80us for 16 chips
- If a chain contains only one device, up to 16 commands can be sent to this device with one access. Bit 7 in length register 0xd411 has to be set to select this multi-write to single device mode.
- Register 0xd412 contains the read-back of data from SPI. Content depends on slave chip.

VHDL Configuration The number of bits per word can be set with a generic in the VHDL component instantiation. If a value below 32 is chosen, the upper bits in all registers are ignored. The number of wait-cycles between two edges on SCK can be selected as well. The default for LTC2600 and Padiwa is 7 wait cycles, I.e. 6.25 MHz.

Configuration File The software takes a text file as input and generates the correct SPI sequence to load and activate the DAC. The ASCII format is shown below, the commands can be found in table 16.

```
# Board    Chain     ChainLen   DAC     Channel       Command       Value
  f333     1         4          0       0             3             0x3450
  f333     1         4          0       1             3             0x1230
  f333     1         4          1       0             3             0x6780
!Reference 2500
  f333     1         4          2       0             3             1250
```

- Board: The TrbNet address of the board. Can be a broadcast address

0	Write Register N
1	Switch Output N on
2	Write Register N, switch on all
3	Write Register N, switch output N on
4	Switch Output N off
F	No Operation

Table 16: LTC2600 Commands

- Chain: A bitmask to select one or more individual SPI chains out of 16 possible ones
- ChainLen: The length of the selected chain, possible are 1 - 16 DACs in one chain. Valid values are 0x0001 to 0xffff
- DAC: The DAC number in the chain, counting from 0 to 15
- Channel: The Channel of the DAC (0..7)
- Value: The value to load. 16 Bit value. Note that we are using LTC2620 which are 12 Bit only, the lower 4 Bit are "don't care" in this case and should be 0
- The "!Reference" keyword is used to set a reference for all following values. E.g. one can set the reference voltage used on the DAC and give all subsequent values in plain voltages. The format can be (almost) any number: integer, float or hex. The use is optional, if no reference is given, the upper limit is assumed to be 65536.

Files

- Implementation: trbnet/special/spi_ltc2600.vhd
- Testbench: trbnet/testbenches/tb_ltc2600.vhd
- Software: daqtools/dac_program.pl
- Example Configuration: daqtools/config/DAC_config.db

Registers

0xd40N	Data	16 places for SPI commands
0xd410	Chip Select	CS output, one bit for each of the 15 outputs, positive logic
0xd411	Control	Control register. See next paragraph
0xd412	Readback	Data received on SPI
0xd413	Master	Block SPI for other use. See next paragraph
0xd414	Clear	Additional output to connect to a CLR-input on SPI devices
0xd418	Invert	Set lowest bit to invert all outputs
0xd419	Word Length	Number of bits of a SPI word, default: 32
0xd41a	Period	Half period of a SPI clock cycle, in system clocks. default: 7

Collisions during reading Writing to all registers is blocked while a transfer is in progress, i.e. writing can not be broken. Reading back a value can break, if to processes access the SPI port in a interleaved manner, because reading the read-back register is non-atomic.

There is a two-level locking mechanism:

If you intend to read back a value, set Bit 16 in the control register 0xd411. This blocks any subsequent SPI access until the read-back register has been read. This should be implemented in all software, but one has to take care that when killing a task, the reading of the register might be skipped and the locking therefore not be cleared.

Nevertheless, this can still be broken if a program does not make use of this feature. If you need secure access, first set Bit 17 in Register 0xd413. Now, only accesses are allowed for which the "su" bit (Bit 17 in the control register 0xd411) is set. Everything else is discarded and not executed. Make sure to clear this bit after finishing the secure register access. This should be used e.g. for reading the configuration Flash memory.

12.2. Forward inputs for trigger

The trigger module can be used to forward any input of a peripheral FPGA via the central FPGA to the CTS. E.g. any input to any TDC can be used to generate the trigger in the CTS. (resources: 400 slices in FPGA for 32 inputs, 2 outputs)

The VHDL code is available in `trb3/base/code/input_to_trigger_logic_record.vhd`

Setup The trigger module can feature up to 32 input signals and up to 16 independent outputs. An extension to 64 inputs (matching the maximal number of inputs to the TDC) is foreseen but not yet implemented. For each output, any of the connected inputs can be enabled or disabled as well as been inverted individually. The output is an 'or' of all enabled inputs. The peripheral FPGA can send four outputs to the central FPGA. Depending on the AddOn used, additional outputs can be routed to free I/O pins if available.

The central FPGA contains the same trigger logic to combine the four signals from each of the peripheral FPGAs to one common signal forwarded on the trigger output on the RJ-45 connector (middle pair). Additional signals can be forward to the AddOn connector of the central FPGA, e.g. to one of the RJ-45 sockets on the CTS-AddOn.

SlowControl Configuration of the module can be done in registers 0xcf00 to 0xcf3f. Each output has two configuration registers. The first one contains a bit mask to enable individual inputs, the second is unused. Additionally, inputs can be inverted and short signals can be stretched to at least 10 ns length.

Input Scalers An additional module is used to have counters for each of up to 32 input channels. These values can be stored in a Fifo at an adjustable rate. The Fifos for all channels are controlled by a common logic and store their data synchronously. Filling of the Fifos has

to be triggered and stops after 1024 samples have been acquired. To save resources, it is also possible to use only one monitoring Fifo combined with a multiplexer to select one of the inputs as source. (resources: 1300 slices in FPGA for 32 inputs)

12.3. Interfaces

SPI The module used for DAC programming is a generic SPI interface that can be used for any purpose. (resources: 300 slices in FPGA)

UART A generic UART master implemented in FPGA (resources: 300 slices in FPGA)

LCD Any numeric values can be shown on a graphic LCD (resources: 800 slices in FPGA)

Debug For debugging, a UART can be used to access the internal data bus (resources: 300 slices in FPGA)

The generic SPI interface which is used for many modules has the following structure:

Bit	Name	Content
31 – 24	8-Bit-Register	
23 – 20	Command	Command, 0: read, 8: write, other: no operation
19 – 16	Subregister	Additional 4 bits (only used for padiwa amps and v123)
15 – 0	Data	16 Bit data payload for write commands

The generic UART interface has the following structure:

Bit	Name	Content
39 – 32	8-Bit-Register	
31 – 0	Data	16 Bit data payload for write commands

12.4. Flash programming

Flash programming of the MACHX03 FPGAs (like the DiRich threshold FPGAs or the Logicbox, but also Padiwa-Amps2) is done via a generic flash controller which sits in the middle between the serial interface (SPI or UART) and the local logic (where the local registers are placed). Both, SPI and UART, have 16-bit adresses and a 16/32-bit data bus. The flash controller uses the adresses 0x40 - 0x5F and shades the local user logic for this address space.

The flash controller works in 2 modes: in mode 0, the flash programming is done in the same way as described in the Padiwa documentation, in order to stay consistent. In this case the register 0x50 (flash access) contains the 13-bit flash page and a 3-bit flash command. In mode 1 (16 bit mode), the access is done with 2 independend registers: first, one has to write first the 3-bit flash command in register 0x51 (this allows multiple uses), and subsequently the 16-bit page address in register 0x50.

The way the user flash and the config flash is selected is also different between the 2 modes. In mode 0, the user flash starts at 0x1C00, and EnableCfg has to be set to 1 only to enable

the config space. In mode 1, both address spaces are independent and start with 0x0, and EnableCfg is used to select the space (0 for user space, 1 for config space).

The flash page is mapped via the read/write command to 16 bytes RAM (0x40-0x4f). As each byte has its own register, 16 read commands are normally required to read the entire flash page. In order to speed up the readout for slow interfaces (like UART), the controller allows also multiple-read with a single command (bursts).

Register	Addr	Bit	Content	Description
FlashRAM (rw)	0x4X			Mapped flash page (16 bytes in total)
FlashAddress (w)	0x50	15-0		For mode 1
FlashCommand (w)	0x51			For mode 1
		3-1		(bit 0: don't care)
			0x0	Read a page from flash and store it in RAM
			0x4	Write a page from RAM to flash
			0x8	Enable flash (address: don't care)
			0xA	Disable flash (address: don't care)
			0xE	Erase user of config flash, depending on EnableCfg
FlashAccess (w)	0x50			For mode 0
Flash command		15-13		command like above
Flash address		12-0		
FlashCtrl	0x5C			
EnableCfg rw		0		Enable config flash
FlashErr r		1		Flash error
FlashBusy r		2		Flash busy (e.g. after erase)
MasterStart w		3		Starts the flash master by hand, which unpacks the user flash and writes the local registers
MasterRun r		4		is =1 while the master is running
FlashMode w		8		flash mode
			0x0	Mode 0 (13 bit address)
			0x1	Mode 1 (16 bit address)
FlashPageBurst	0x5D			Can be used to perform multiple reads on the flash page (0x4X)

Continued on next page

Table 17 – *Continued from previous page*

Register	Addr	Bit	Content	Description
MemWidth	w	1-0		Can be used to pack multiple flash bytes into one data word on SPI/UART (requires sufficient buswidth)
			00	8 bit
			01	16 bit
			10	32 bit
Endian	w	4		
			0	Little endian
			1	Big endian
NumWords	w	11-8		Number of data words (values 1 . . . 3), determines how many data words written to SPI/UART with one single read command. 0: burst disabled.
Debug	0x5D-0x5F			Reserved for debugging

Table 17: Status and Control registers of the flash controller

The flash controller adds in addition a master function which reads the user flash space, and writes the local registers after power up (or upon request). This allows to store default values for local registers (like thresholds) in a common way. The data which is unpacked starts at the first user page, and contains a version byte, an address byte, and 2-4 data bytes (depending on the data width). The data width is selected with the version byte (0x1: 16 bit, 0x2: 32 bit), and any other version byte means “end of file”. In 32 bit mode, 2 more padding bytes are added in order to align the data content with the flash page. This means in 16 bit mode 4 data words can be stored, and in 32 bit mode 2 data words. The data words are always stored in big-endian.

13. GbE Data Read-out

Communication with TRB3 is handled by the Gigabit Ethernet interface (SFP8 by default). It can be used for Slow Control connection (see next section) and for the readout of collected data. In order to act as standard network device, there are several protocols that share the same link. The basic ones for the network discovery are DHCP, ARP and ICMP. Protocols typical for data handling in standard TRB3 implementation are SCTRL and TrbNetData. Even though they all can run in parallel, processing data, they all share the same input and output link, distributing 125MBps bandwidth.

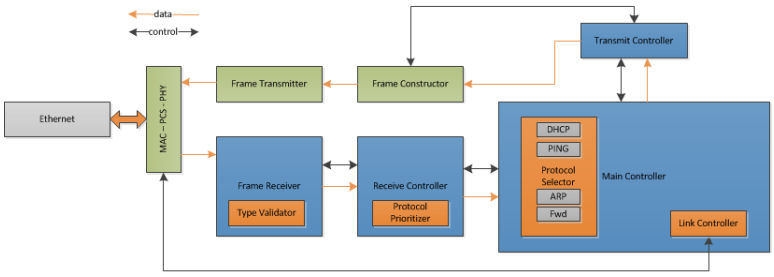


Figure 13: Block diagram of the GbE module.

Default buffer depths allow the following data sizes configuration:

Size	Description
1400 Bytes	Single reception only possible, no reassembly mechanism
4000 Bytes	Maximum MTU of outgoing Ethernet frames
64000 Bytes	Maximum size of a single UDP TrbNetData packet
64000 Bytes	Maximum size of a single UDP SlowControl packet

Table 18: Default maximum sizes of frames and packets

13.1. Data Readout

TrbNetData module constructs Hades data packets out of the fragments received from the TrbNet endpoints. In standard case, those are the edge FPGAs, but it's possible to use the TRB3 board as a HUB and collect also data from "slave" boards. A Hades packet is formed as an entity called HadesTransportUnitQueue and there are several ways of constructing it. In general event fragments from all the connected endpoints are buffered one after the other, encapsulated with proper headers on several levels: subevent headers, queue headers, UDP headers, IP headers and Ethernet as a final step.

13.2. Addressing

Each TRB3 board has a unique MAC address which is constructed in following way: 02:00:BE:UNIQUE_ID(31 downto 8), where UNIQUE_ID(31 downto 8) is a value read out from the temperature sensor and differs between boards, the first part is constant. This address is used only for the network and SlowControl packets. Data readout addressing is stored in a block memory under base address 0x8100 and has to be configured manually. As there is a way to distribute packets to several event building machines or processes, all of those addresses need to be written into this memory with the following structure:

Address	Description
0x81X0	Lower 32 bits of the destination MAC address
0x81X1	Bit 15..0: Higher 16 bit of the destination MAC, Bit 31..16: reserved
0x81X2	Destination IP
0x81X3	Bit 15..0: Destination UDP port, Bit 31..16: reserved
0x81X4	OBSOLETE (address automatically generated)
0x81X5	OBSOLETE (address automatically generated)
0x81X6	OBSOLETE (address aacquired from DHCP)
0x81X7	Bit 15..0: Source UDP port, Bit 31..16: reserved
0x81X8	OBSOLETE (switched to control register 0x8304)

Table 19: Addressing registers map

Each such block of addresses corresponds to one destination event builder. There can be up to 16 destinations configured, where each one has an offset in addressing of 0x10.

13.3. Configuration

Some header values as well as operation mechanics can be changed and adjusted, here's the table of control registers (R/W):

Address	Description
0x8300	The ID is written in each SubEventHeader to identify the source of data (default: x000000cf)
0x8301	Information sent in the SubEventHeader (default: x00020001)
0x8302	Information sent in the HadesTuQueue (default: x00030062)
0x8304	Maximum size of a Ethernet packet (default: 1400)
0x8305	Enable sending data over GbE (default: 0)
0x8307	Enable packing several events into one event queue (default: 0)
0x8308	The internal, 24bit trigger counter used for the SubEventHeader (default: 0)
0x8309	Enables/disables reception of frames (default: 1)
0x830B	Include Trigger Type in decoding field (default: 0)
0x830C	Max Subevent size, larger are discarded (default: 59800)
0x830E	Max number of Subevents in one Queue (default: 200)
0x830F	Max Subevent size, after which the Queue is closed immediately (default: 32000)
0x8310	Max size of a Queue (default: 60000)
0x83FF	When written to 0xFFFFFFFF: all values are reset to default

Table 20: Control registers map

13.4. Monitoring

The operation of the entire GbE module as well as individual protocols can be monitored using the following registers (R only):

Address	Description
0x83E0	Received bytes counter
0x83E1	Received frames counter
0x83E2	Transmitted bytes counter
0x83E3	Transmitted frames counter
0x83E4	Transmitted packets counter
0x83E5	Dropped RX frames counter

Table 21: Monitoring registers map

13.5. Building Blocks

13.6. Slow Control Registers

14. GbE Slow-Control

14.1. Getting Started

In order to control TRB3 or a larger system with TRB3 as slow control client via Ethernet link, one needs to properly install and compile the trbcmd server, load a correct FPGA design and configure DHCP daemon on the server PC. Follow the instructions described in the next points.

14.1.1. FPGA design

The optical link, activated for the Slow Control over GbE is the one labelled SFP8. After loading the design, TRB3 will automatically start to send control packets once per second. With any network monitoring tools (ex. Wireshark) one have to capture such packet and check the source MAC address. This address has to be added to the DHCP configuration on Slow Control server PC. MAC address is generated basing on the unique ID assigned to each FPGA and a constant part: 02:00:BE:UNIQUE_ID(31 downto 8).

Starting from 26/11/2015, the MAC address is generated as: DA:7A:nU:UU:UU:UU, were 'n' is the number of the interface, counting from 0 to 3 (Typical TRB3: 3, TRB3sc: 0, TRB3 second link: 2) and U are Bits 35..8 of the unique ID. I.e. a unique id 0x820000050dec0a28 corresponds to (TRB3sc) DA:7A:05:0D:EC:0A - that is all relevant bits of the unique id are contained in the MAC address.

To configure your PC, depending on your system configuration:

- Open /etc/dhcpd.conf and add an entry specifying hostname and/or IP and MAC address
- Open /etc/hosts and add an entry specifying hostanme and an selected IP address
- Restart DHCP daemon

Now you can reload the central FPGA and it should automatically acquire the IP address from the server. One can verify that by monitoring the network traffic or system log file.

14.1.2. Trbnetd

To access the TRB3 you can access it directly with trbcmd (the local version found in "trbsoft-/trbnettools/libtrbnet", but this is not recommended and only meant to be used by experts for debugging. To find out which version of trbcmd one uses you can type "trbcmd -V" and in the output you will either find "Local TRB3" (local version) or "RPC" for the RPC version. The correct way is to use a trbnetd running on any machine in the network, which has direct UDP access to the TRB3 to be controlled. This daemon then collects all requests from many different clients and takes care of the correct arbitration (doesn't work for non atomic accesses, like SPI-interface which needs to access several registers one after the other). The trbnetd needs to know the ip-address of the TRB3 and *can* additionally be identified via a 8 bit number. So, to start the trbnetd which connects to the TRB3 with the ip-name trb30 one has to start the trbnetd in the following way:

```
TRB3_SERVER=trb030 trbnetd
```

This opens a trbnetd with the RPC-id 0. It is recommended to write down explicitly the RPC-id-number when starting the daemon, e.g:

```
TRB3_SERVER=trb030 trbnetd -i 9
```

to start the trbnetd with the id 9.

For GbE designs older than August 2013 the correct port number for RPC communication has to be given, e.g:

```
TRB3_SERVER=trb030:25000 trbnetd -i 9
```

To access this trbnetd one uses trbcmd with the address of the daemon given in the environment variable DAQOPSERVER, e.g.:

```
export DAQOPSERVER=kp1pc105:9
trbcmd i 0xffff
```

Installation:

For security reasons normally RPC-calls are only accepted from the loop-back interface, so we have to tell the rpcbind daemon running on the machine where the trbnetd should run, that it should accept connections from anywhere. This is the option "-i".

OpenSuse

Edit the file `/etc/sysconfig/rpcbind` and make the following entry: `RPCBIND_OPTIONS="-i"` and then restart the daemon: `rcrpcbind restart`

Ubuntu

Using a recent ubuntu-flavoured Linux, the setting can be found in `/etc/init/portmap.conf`, change the line `OPTIONS="-w"` to `"-wi"`. Then restart portmap: `service portmap restart`.

If you have a running instance of trbnetd and want to know to which TRB3 it connects:

```
cat /proc/$(pgrep -f "trbnetd")/environ | strings | grep TRB3
```

14.1.3. Trbcmd server

- Download the latest release of the software from GIT repository
  `daqtools`
- Compile with appropriate flag:
  `make TRB3=1`
- Set up the environment variable DAQOPSERVER to your TRB3 hostname
- Make sure that the UDP port 5555 is open in your firewall

14.1.4. Usage

Now you are ready to use the trbcmd in the same way as it is for the normal HADES system.

14.1.5. Ping of Death

Currently (2017), TRBNet-Hubs have an inherent weakness, as the data which flows into the hubs are not checked for sanity (for example no CRC-check). Every bogus network packet, for example produced by TRBNet-Endpoint FPGAs which suffer from a voltage drop on the core supply, or from a SEU, can cause the TRBNet-HUB to crash. It is planned in the long-term to reduce these crashes by sanity checks of the data (actually TRBNet-headers) in the media interfaces. Therefore, it can happen that due to wrong data in the TRBNet, that the GbE-Slow-Control entity will hang and the user can not communicate with the TRB3/sc1 via the trbcmd. One way to solve this it to power-cycle the system. An other way is the following. As the TRB3 has a GbE-Interface, even though the TRBNet is down, the GbE-part is still active. When a special formatted "ping" packet arrives at the TRB3/sc1 it will initiate a reload of the FPGA which receives the data from Ethernet, so normally FPGA5 (central) on the TRB3 or the only FPGA on the TRB3sc. This allows for a much faster and less invasive action, compared to a power-cycle. The rule is the following: The GbE-entity will initiate the reload of the central FPGA if a ping-packet arrives which has the TRBNet-address as payload. So, for example, after a fresh start of the CTS-FPGA it's address is 0xf3c0.

```
ping -c3 -W2 -pc001 trb084
```

will reboot the central FPGA of trb084, if it's TRBNet-address is 0xc001. Don't be alarmed if the ping doesn't come back, as sometimes the interface is blocked to send back data, but it still receives the ping.

14.2. Building Blocks

14.3. Slow Control Registers

15. CTS[2]

15.1. Features

- **No additional hardware requirements** as design is embedded in the central FPGA.
- **100 MHz trigger logic frequency**. If faster logic is required, consider the dedicated CTS add-on.
- **Extensible and modular structure**. Both, the hardware description and the software, are designed in a modular fashion and allow for an easy implementation of new features. A fully automated enumeration process enables the software to determine the hardware's capabilities.
- **Master and slave mode operation**. As master, the CTS makes the trigger decision, as slave it listings it reacts to the trigger decision made by a foreign DAQ and distributes the information to a TrbNet subsystem. Currently, CBM-MBS is supported.
- **Up to 16 independent trigger modules** to implement complex behaviour.
- **8 general purpose trigger inputs** with independent spike rejection and delay lines.
- **4 channel TDC** to determine the trigger time with a resolution of 20 ps.
- **MBS master** Sends out a MBS trigger word for each trigger on a serial LVDS line parallel to a 50 MHz clock
- **Run-time configurable periodical and random pulsers modules**. The mCTS supports regular and (pseudo-)random pulsers to produce trigger decisions with an (average) interval of 10 ns to 40 s.
- **Run-time configurable Coincidence detection** based on the general purpose inputs. Criterion can be edge and/or level-sensitive.
- **Generic counters, scalers and debugging features** accessible via slow-control and embeddable into the data stream to the event builder.

15.2. Getting Started

15.2.1. The GUI


WARNING

It is experienced that the GUI is not displayed correctly with some browsers or some versions of the browsers. If you can not see the GUI correctly (no plot, no trigger registers etc.), but you are sure, that everything is set correctly in your set up and the output of the cts_GUI script doesn't show any error messages, then you should try to connect with a different browser. If available, a recent Firefox version is recommended.

[2] If not explicitly stated otherwise, in this chapter, CTS refers to the trigger system embeddable into the central FPGA of a TRB3.

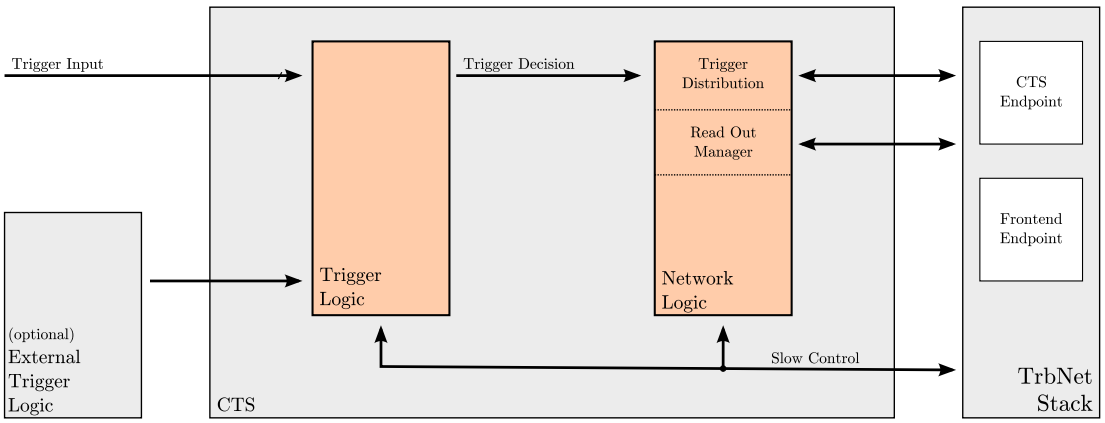


Figure 14: CTS building blocks. The design consists of two major building blocks – a generic network stack and an extensible trigger logic

15.3. Building Blocks

In order to increase the hardware independence of the design, the CTS consists of two major building blocks: The *Network Logic* handles the network interfaces (which are not part of the CTS design itself) and propagates event information gathered by the *Trigger Logic* (see figure 14).

15.4. CTS Network Logic

The CTS uses two dedicated network endpoints for communications: The *CTS Endpoint* is obviously needed due to its unique ability to send trigger packets and coordinate the readout process. However, it lacks the support to transmit arbitrary data to the storage servers, which is implemented using an ordinary *FEE endpoint*.

On the TRB3, the hub and two endpoints are directly linked without additional media-interface and encapsulated into the `trb_net16_hub_streaming_port_sctrl_cts` and are instantiated independently from the CTS in the top-entity of the design.

The logic's behaviour is modelled using two FSMs, one controlling the trigger distribution and one for the readout process. Both are connected by the so-called readout queue, a FIFO storing the identification of event that have been triggered, but not yet read out. In case of a malfunction during read out causing a time-out this queue is likely to overflow which inhibits the distribution of further trigger events.

If your CTS does not accept further events and has a full read-out queue, consider to check the correct configuration of the read-out. For a in-depth discussion of the FSM states see [Pen12].

Bit(s)	Description
15: 0	State of all Trigger Channels when trigger was accepted
19:16	Number of Input included (each input includes two words: the number of cycles asserted (lower address), number of rising edges (upper address))
24:20	Number of Trigger Channels used (two words per channel, same format as above)
25	Include *last idle*, *dead time* counters (two words)
26	Include Counters *Trigger asserted*, *Trigger Edges*, *Triggers Accepted* (three words)
27	Timestamp with resolution of 10 ns / tick (one word)
29:28	00: External Trigger Module (ETM) not present, 01: ETM sends 1 word, 10: ETM sends 4 words, 11: ETM sends data with header and arbitrary number of words

Table 22: CTS SubSubEvent Header. The upper two bytes describe the package's content. Its total length can be computed using the length denoted in the brackets behind each property. All flags are high-active. The number of inputs and ITCs must be specified as it depends on the configuration used during synthesis.

15.4.1. SubSubEvent Data Format

The CTS's FEE offers two independent readout-ports (and therefore the subsubevent data consists of two blocks): The first port is controlled by the CTS itself, while the second one may be connected to external trigger logic. On the top-entity the default values of the signal connected to optional external trigger logic are chosen to automatically disable the second port, if no external module is present.

The TrbNet does not automatically insert a header between the two sections, and as the amount of data sent by the CTS is configurable, the subsubevent includes a header in its first word (see table 22). It can be used to calculate the size of the CTS frame. All remaining words in the subsubevent originate from the external trigger module adapter. A commented example frame is shown in table 23.

15.4.2. Multiple Event Builders

The CTS supports multiple event builders using a simple *round-robing* scheme. If

Addr	Value	Description
-1	0x002cf3c0	SubSubEvent Header, indicating a length of 0x2c and a TrbAddress of 0xf3c0
0x00	0x1ee43c01	CTS Header. ITC status bitmask: 0x3c01 Number of Input Counters: 0x4 Number of ITC Counters: 0xe Idle/Dead counters: yes Trigger statistics: yes Timestamp: yes ETM present, sends 1 word
0x01	0xcb1a3130	Level Counter Input 0 (# cycles input was asserted)
0x02	0x00000000	Edge Counter Input 0 (# rising edges)
0x03	0x004118ba	Level Counter Input 1 (# cycles input was asserted)
0x04	0x02741321	Edge Counter Input 1 (# rising edges)
0x05	0xcb1a3130	Level Counter Input 2 (# cycles input was asserted)
0x06	0x00000000	Edge Counter Input 2 (# rising edges)
0x07	0x004118ba	Level Counter Input 3 (# cycles input was asserted)
0x08	0x02741321	Edge Counter Input 3 (# rising edges)
0x09	0x25e0fc0f	Level Counter ITC 0 (# cycles ITC was asserted)
0x0a	0x000a00cd	Edge Counter ITC 0 (# rising edges)
0x0b	0x0000000a	Level Counter ITC 1 (# cycles ITC was asserted)
0x0c	0x4af40000	Edge Counter ITC 1 (# rising edges)
...	...	...
0x21	0xe96d2bd1	Level Counter ITC 12 (# cycles ITC was asserted)
0x22	0x00000000	Edge Counter ITC 12 (# rising edges)
0x23	0xe96d2bd1	Level Counter ITC 13 (# cycles ITC was asserted)
0x24	0x00000000	Edge Counter ITC 13 (# rising edges)
0x25	0x00018c6a	16.2 ms Idle time (# cycles CTS was idle before trigger was accepted)
0x26	0x00000082	1.3 μs Dead time (# cycles CTS was busy in last event)
0x27	0x000ba1c8	Trigger Stats: Number of cycles trigger was asserted
0x28	0x000ba1c8	Trigger Stats: Number of rising edges asserted
0x29	0x00005de9	Trigger Stats: Number of events accepted
0x2a	0x35c3e3e1	Timestamp
End of CTS Data. Remaining words are from External Trigger Logic		
0x2b	0x10000000	External trigger module word

Table 23: Example of CTS Package. The data in the subsubevent appears in the same order as the properties in the header word

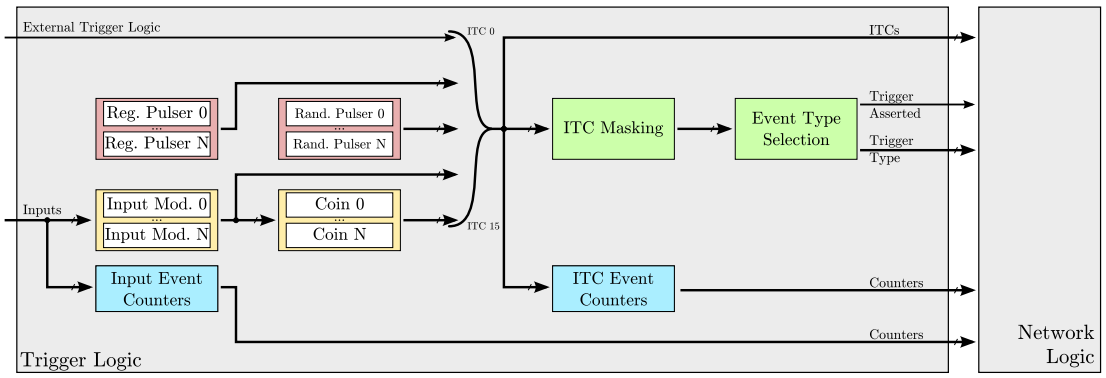


Figure 15: Structural overview of trigger logic. The trigger consists of a number of freely configurable trigger modules that are “plugged in" using the ITC channels. This gives the means for a generic selection of active modules and for generic statistics features.

15.5. Trigger Logic

The trigger logic internally offers 16 channels, subsequently commonly referred to as ITCs[3], to which modules are connected to. These channels in combination with the memory structure discussed in chapter 15.6 are the key concept to an extensible logic. Together both techniques allow to design universal modules with little code overhead. Furthermore, it is possible to include modules on a need-only basis, i.e. only the functions required by a specific setup are synthesised.

To prevent misfiring shortly after start-up, all ITC are disabled by default and have to be enabled via slow control. Each ITC can be configured to be sensitive to either rising edges or high levels. If an enabled channel is active, the trigger module propagates this information to the network logic. A TrbNet trigger type is assigned to each ITC. The type of a given event is then derived from the lowest ITC that fired.

15.5.1. Input module

Each input signal of the trigger logic is preprocessed by an independent *input module* to compensate typical issues of signals from off-board electronics, such as twisted differential pairs, improper relative signal runtime and electrical noise. Figure 16 illustrates the unit’s structure.

A spike rejection logic can be used to dismiss pulses up to 15 cycles in length. It is implemented using a 4 bit counter that is incremented in each cycle while the input is high and is reset if the signal becomes low. As soon as the value exceeds the configurable threshold T, the pulse is considered valid and is propagated. This design introduces a delay of T cycles.

While normally all inputs should have the same spike rejection factor, the logic shifts signals relative to each other when different values are used. This might lead to problems – e.g. for the

[3] **I**nternal **T**rigger **C**hannel

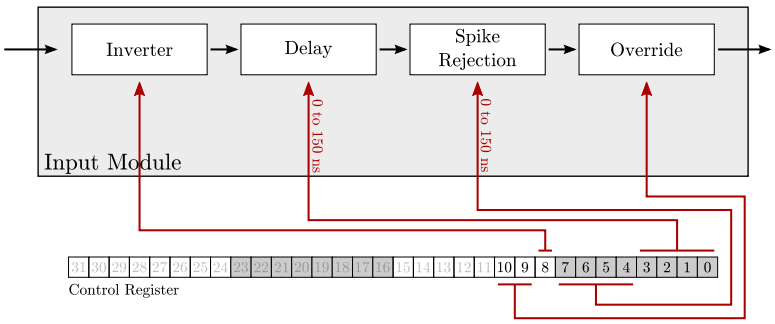


Figure 16: Block diagram of an input module. For each input signal an instance of an input module is generated. Its main task is to cancel out noise and equalise signals from different sources.

coincidence detection or in the later data analysis.

Other sources of signal runtime are the limited speed of particles and secondary charge carriers within detectors, external circuits, such as amplifiers, and the limited velocity of propagation of the signal through wires and optical fibres (typically 2 m per clock cycle).

Independent of the origin, signals that are out of phase can manually be synchronised by delay lines. In an input module, a delay line is built from a 15 bit shift register and a multiplexer used to select the required delay.

15.5.2. AddOn Input module

As the number of ITCs is limited to 16, it is not possible to map each input of the CTS AddOn directly to an dedicated trigger channel. Thus, a number of multiplexer modules can be used to select inputs from the CTS AddOn:

- `JECLIN`: Four differential ECL inputs, all connected via a RJ45 jack
- `JIN1`, `JIN2`: Two times four differential LVDS inputs connected via two RJ45 jacks
- `NIMIN1`, `NIMIN2`: Two NIM inputs

The output of each multiplexer is connected to an input module (see section 15.5.1), which offers statistics and filter capabilities. In fact, the AddOn inputs are processed by the CTS as ordinary input channels: If synthesised with $N =$ `TRIGGER_INPUT_COUNT` input channels and $M =$ `TRIGGER_ADDON_COUNT` AddOn inputs, the enumeration processes yields $N + M$ input modules. The same goes for the data sent to the event builders. The AddOn inputs are always mapped into the upper M slots.

The multiplexer introduces an deterministic delay of two system clock cycles, i.e. 20 ns.

15.5.3. Triggers from Peripheral FPGAs

Each peripheral FPGA can use its `FPGA5_COMM(10)` line to communicate a high-active trigger request to the CTS. The line should be asserted for at least 10 ns, however, can be safely hold up, until a trigger was received via TrbNet. In this case take care, that the line is pulled down before the busy-release is send over the network.

Using a bitmask in the CTS, each FPGA line can be selected individually. If any selected line fires, the trigger is propagated using to the ITC. Utmost one module can be synthesised.

15.5.4. Coincidence detection

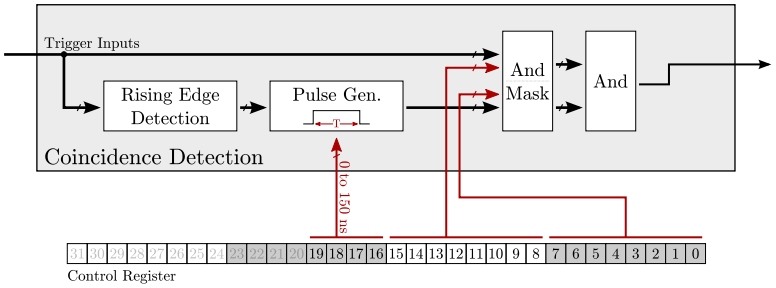


Figure 17: Block diagram of the coincidence detection

The coincidence detection logic is used to detect rising edges and high levels of multiple signals within an adjustable window of time. It is expected that the unit is most commonly employed to cancel out statistical effects, such as noise. However, in combination with a (possibly external) delay line, the module can also detect a sequence of pulses.

The implementation is encapsulated into an entity which can be instantiated multiple times during synthesis. The number is limited only by the amount of free ITCs. Each unit can be configured individually: One bitmask selects a set of trigger inputs that have to rise within a configurable time window. For each input the logic internally generates artificial pulses that start with a rising edge of the signal and last for the coincidence time. Hence, as soon and long as the artificial signals of all selected inputs are asserted, the first coincidence condition is fulfilled.

While the former logic monitors changes of the inputs, there is a second bitmask used for level-sensitive conditions. The mask defines inputs that have to be asserted in order to propagate a edge-coincidence detected by the previous stage. These signals are called inhibit inputs as they can be used to filter events based on the state of an external low-active circuitry. If only one of the masks is the selected, the unit can be used to monitor asserted or rising lines exclusively.

15.5.5. Pulsers

Any module which leads to trigger decisions that are based on no input but the system's clock is considered a pulser. There are two pulser types implemented: A *random pulser* and a *periodical pulser*. Both are useful to schedule events, such as debug, calibration and synchronisation triggers, and allow for (stress) tests of the whole DAQ system.

A periodical pulser repeatedly asserts its output, followed by a configurable pause. The interval can be specified with a resolution of 10 ns ranging from a continuously asserted signal to one event per 42.9 s. In the frequency domain, the discretisation error growths with shorter intervals. The relative error, however, is less $< 10^{-3}$ for rates below 100 KHz.

The *pseudo random pulser* available in the CTS employs a 32 bit CRC unit with a fixed data word at its input to generate pseudo random numbers (PRN). If multiple instance of the pulser are synthesised with different constants. As simulations suggest, the values generated are nearly uniform deviates. Furthermore, the distance between two successive numbers is also distributed almost uniformly and seems to be uncorrelated to their magnitude.

In each clock cycle a random number is compared with a configurable threshold. If it is smaller, an event is produced. Since the numbers are uniform deviates, the average duty cycle of the pulser is given by the threshold divided by the maximum value possible. In addition, the uniformly distributed distances prevent the clustering of events that can be observed with other pseudo random number generators, such as linear feedback shift registers.

15.5.6. External Trigger Logic

In contrast to common trigger modules, which are instantiated within the trigger logic's architecture and therefore inside the CTS's component hierarchy, *External Trigger Logic* lays outside – typically on the same level as the CTS's main entity. This concept is intended for modules, that required inputs different than the preprocessed general purpose trigger signals, such as bindings to foreign DAQ protocols. Currently only a CBM-MBS adapter is available. See chapter 15.7 for a short tutorial on how to implement a new module.

15.5.7. Free-Running with Spill-Dependent Frequency

If supported by the experimental set-up it might be beneficial to reduce the trigger frequency of a free-running system during offspill phase. Especially if there are many front-ends in the DAQ, substantial data reductions can be achieved.

As shown in Figure 18, you need two pulsers, two coincidence modules, four input modules and a spill-indicator, that is available to an input module (e.g. any input of the CTS-Addon or an appropriate pin of the onboard RJ45-jacks): In the diagram `Pulser3` is engaged during spill time: While its ITC is disabled, the signal is received from `InputModule2` and forwarded to `Coincidence1`. `InputModule2` is only necessary since coincidence modules are connected only to input modules and cannot receive the signal otherwise. The spill-signal is fed to the

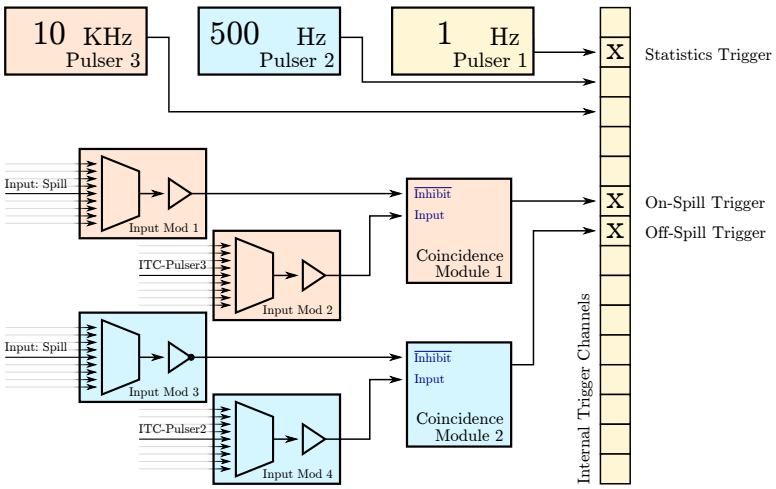


Figure 18: Free-Running mode with different onspill/offspill frequencies.

`Coincidence1`'s inhibit pin via a second input module: The inhibit signal has to be high, in order to select the pulser; so if the spill-signal is low-active, enable the input module's inverter. The frequency during offspill periods is derived similarly; just use the opposite setting for the spill-signal's inverter.

15.5.8. Latency and Jitter

Due to the CTS modular structure, the trigger latency is strongly influenced by the actual trigger modules used. Figure 19 includes the measurements for two different channels, both recorded on a TRB3:

- The setup of the first plot directly uses an external trigger input and. Thus, the signal takes the following path: In order to avoid meta-stabilities, the input is sampled using a flip-flop and then routed to an input module, which takes 3 cycles for the shortest delay and spike rejection settings. The propagation delay grows linearly, if either of those values is increased. The ITC handling further requires 2 cycle, followed by additional 2 cycles until the time reference signal is available. In total, this sums up to a delay of 80 ns. On a TRB3 board, the TrbNet stack requires another 450 ns to deliver the LVL1 packet. While this number significantly influences the system's dead time, front-ends with critical timing constraints typically use the time reference.
- The MBS input module is an external trigger module sampling a serial data stream of 50 Mbits^{-1}. It propagates the trigger information as soon as the specified start-pattern is received, resulting in an overall delay from the first falling edge of the start packet to the rising edge of the time reference signal of 6 clock cycles.

Both latencies measured have a positive jitter of 10 ns, i.e. the sampling period of the CTS

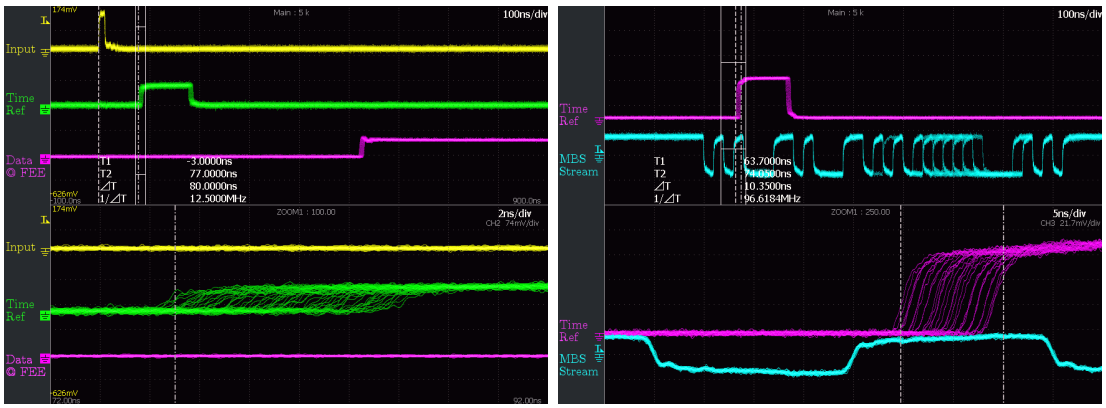


Figure 19: Latency and jitter measurement ... **(a) Left:** ... using trigger input. Yellow: Input, Green: Time Reference signal, Purple: Arrival at front-end (on board) **(b) Right:** ... using MBS module. Aqua (blue): MBS data stream (input), Purple: Time Reference. The first falling edge of the MBS data stream starts the trigger message.

running at 100 MHz. Thus, the design itself can be considered jitter free.

15.6. Slow Control Registers

The CTS's address range starts at `0xa000`, and hence overlays with the address area of the HADES-CTS. Just like the CTS's structure, the address space itself is separated into two major blocks, one controlled by the *network logic* and the other one by the *trigger logic*. As the former is not likely to change substantially in future developments, the first block has a fixed register layout shown in table 25.

The address block assigned to the *trigger logic* has to be more flexible as extensions to the trigger are very likely and encouraged by its design: Each module specifies its own – by definition – continuous address layout relative to a base address. During synthesis all individual blocks available are joint together, connected only by a header as described in table 24. The header identifies the following block by an 8 bit ID that includes the block's size and further informs the software which trigger channels are connected to the module.

Thus, when initially connecting to the CTS, the software has to read the first header from the address `0xa100`. Even if the block's ID is unknown, the next header's address can be derived using the length information included in the header word. The enumeration is completed when the last header – indicated by a 1 in its highest bit – is read. The order in which the modules appear depends on the specific hardware description, but it is not defined by any convention.

The only restriction is that every module id appears only once. If the trigger logic contains multiple instances of the same module, they have to share a block, which can be indicated by the header's length information. This decision reduces the amount of header words and thus speeds up the enumeration process. It further decreases the code complexity of the client

software.

Bit(s)	Description
	Block identification header
7:0	Block type
15: 8	Number of addresses in this block exclusively this header word
20:16	First internal trigger channel assigned to this block (0 if it does not apply)
25:21	Number of internal trigger channel assigned to this block (0 if it does not apply)
31	Last block indicator. Enumeration stops after reading this block

Table 24: Header used to identify an address block within the trigger logic's address range.


Info

The following list states block types currently used and roughly describes their structure. If you require detailed information about single bits, look into the source file of the corresponding driver module placed under `~/trbsoft/trb3/cts/CtsPlugins/CtsMod{ID}.pm`. These contain a (hopefully) self-explanatory register definition. This way you make sure, you get the most recent definitions. To obtain a full register list, use the `list` command of the cts CLI (see section **??**).

Currently the following IDs are used:

- **`0x00` Internal Channel Masking**. This block contains one control register holding two bitmasks. Each of the lower 16 bits enables one ITC (bit is 1), while the upper 2 bytes select whether the channel is edge- or level sensitive. After a reset all channels are disabled to ensure that no trigger is distributed before the whole network is initialised.
- **`0x01` Internal Channel Event Counter**. This block contains two 32 bit counters for each of the 16 ITCs. The first word of every pair represents the number of clocks in which the trigger channel was asserted, the second one holds the number of rising edges. All counters work independently and overflow without any notice. With the CTS's system clock of 100 MHz, they have to be polled at least every other 40 s to ensure that no register overflowed more than once.
- **`0x10` Input Module Configuration**. Each register holds the configuration of one input module as discussed in chapter 15.5.1.
- **`0x11` Input Event Counter**. This block has the same structure as `0x01`, however, its counters monitor the trigger inputs before they are processed by the input modules. Hence by comparing both counter types, one can infer the number of events filtered by the spike rejection. Please keep in mind that spike rejection and delay lines introduce a signal runtime and that there is a delay in the read access when accessing multiple registers. Thus, you should not compare two absolute figures obtained from a single memory

Address	Bit(s)	Description
0xa000		Statistics: Number of clock cycles with trigger asserted
0xa001		Statistics: Number of trigger rising edges
0xa002		Statistics: Number of triggers accepted
0xa003		Current trigger status
	15: 0	Trigger bitmask (before filtering)
	19:16	Current trigger type
	20	Trigger asserted
0xa004		Buffered trigger status
	15: 0	Trigger bitmask (before filtering)
	19:16	Trigger type
0xa005		TD FSM State (Trigger Distribution). One-Hot-Encoding:
	0	TD_FSM_IDLE
	1	TD_FSM_SEND_TRIGGER
	2	TD_FSM_WAIT_FEE_RECV_TRIGGER
	3	TD_FSM_FEE_ENQUEUE_INPUT_COUNTER
	...	
	12	TD_FSM_WAIT_TRIGGER_BECOME_IDLE
	13	TD_FSM_DEBUG_LIMIT_REACHED
0xa006		RO FSM State (Readout Handling). One-Hot-Encoding:
	0	RO_FSM_IDLE
	1	RO_FSM_SEND_REQUEST
	2	RO_FSM_WAIT_BECOME_BUSY
	3	RO_FSM_WAIT_BECOME_IDLE
	4	RO_FSM_DEBUG_LIMIT_REACHED
0xa007		Readout Queue
	15: 0	Words enqueued
	30	Empty
	31	Full
0xa008		Debug FSM limits
	15: 0	Number of Triggers (0xFFFF means no limit)
	31:16	Number of Read-Outs (0xFFFF means no limit)
0xa009		Trigger information to be send in read-out (default: 0x00000000)
	0	Input Counters
	1	Channel Counters
	2	Statistics: Idle- and Dead-Time counter
	3	Statistics: Trigger asserted, -edges, -accepted
	4	Timestamp
0xa00a		Statistics: Dead time of last trigger
0xa00b		Statistics: Time between last two accepted triggers
0xa00c		Event throttle
	9: 0	Maximal number of events accepted per millisecond
	10	Throttle enabled
	31	Stop Trigger
0xa00d		Event Builder selection
	15: 0	Event Builder mask (default: 0x1)
	23:16	Number of events before selecting next builder (useful to aggregate events to support large data packets)
	27:24	Event Builder number of calibration trigger
	28	Use special event builder for 0xe trigger, otherwise use ordinary round robin selection.

Table 25: Registers with fixed addresses, i.e. controlled by the network logic domain

dump. The later source of uncertainty does not occur when using the counters sent to the event builders.

- **`0x12` AddOn Input Multiplexer**. This block is only present, when the CTS was synthesised with at least one AddOn Input Multiplexer. It contains one word per module, which is interpreted as an unsigned index for the multiplexer. The mapping is defined in table 26.

- **`0x13` Peripheral FPGA Trigger Inputs**. This block is present, when CTS was synthesised with `PERIPH_TRIGGER_COUNT` = 1 and contains a single word payload of which the lower four bit are used as a bitmask to select the active FPGAs. The LSB corresponds to FPGA 1, the MSB to FPGA 4.

- **`0x20` Coincidence Configuration**. Each coincidence detection module (see 15.5.4) has one configuration register. Thus, the number of registers inside this block matches the number of `COIN`s.

- **`0x30` Periodical Pulser**. Each register in this block stores the low-period's length of a pulser in clock cycles. 0 results in a constant high channel.

- **`0x40` Event types**. This block contains exactly two registers that assign a *trigger type ID* (4 bit) to each ITC. Starting with the type of the first channel at the first register's lowest nibble, each word stores 8 types.

- **`0x50` Random Pulser**. A random pulser generates irregular event patterns. Each instance is configured with one control register, which holds its threshold T. For small T there is a linear dependency between the average trigger rate F and the threshold T given by $F(T) = \frac{100\,\mathrm{MHz}}{2^{32}-1} \cdot T$.

- **`0x60` External logic - CBM/MBS**. This module indicates the presence of the CBM adapter module. If set, the lowest bit of the control registers prevents the module from sending data to the event builder. The lower 24 bit of the status register contains the time-stamp of the last event seen. The MSB holds the error flag.

- **`0x61` External logic - Mainz A2**. This module indicates the presence of the Mainz A2 adapter module. If set, the lowest bit of the control registers prevents the module from sending data to the event builder. The lower 31 bit of the status register contains the time-stamp of the last event seen. The MSB holds the error flag.

Input Number	Input Description
0	jeclin[0]
1	jeclin[1]
2	jeclin[2]
3	jeclin[3]
4	jin1[0]
5	jin1[1]
6	jin1[2]
7	jin1[3]
8	jin2[0]
9	jin2[1]
10	jin2[2]
11	jin2[3]
12	nimin1
13	nimin2

Table 26: CTS AddOn Input Mapping

15.7. Trigger Generation Options

15.8. HowTo Implement an External Trigger Module

This short tutorial will guide you through the steps necessary to implement an External Trigger Module:

- VHDL: Encapsulate the Trigger Logic into an entity (only the interface is discussed here)
- VHDL: Instantiate the module and connected it to the system
- VHDL/Perl: Reserve a module id and implement the software support

It is also wise to start such developments in a new git branch to prevent that you break the existing master branch code. If you have successfully tested everything, the changes can easily be merged.

15.8.1. The module's interface

An External Trigger Module (ETM) gives you the means to implement a high-level trigger criterion that does not require the preprocessing of the general purpose trigger inputs. A typical example is a network bridge. A CTS build can include at most one ETM. The module is directly connected to the trigger channel with the highest priority, thus if the ETM asserts the trigger line while the CTS is idle, the TrbNet Event Type of the next event is defined by the type configured for the ETM.

Listing 1 shows a minimal ETM instantiation. For a better readability, it is highly recommended, that you keep the port's naming. Copy this into the top entity (`trb3_central.vhd`) and wrap it with an analogous `generate if` statement. Also add another config option in `config.vhd`. You probably need to add signals, e.g. to interface with off-board electronics,

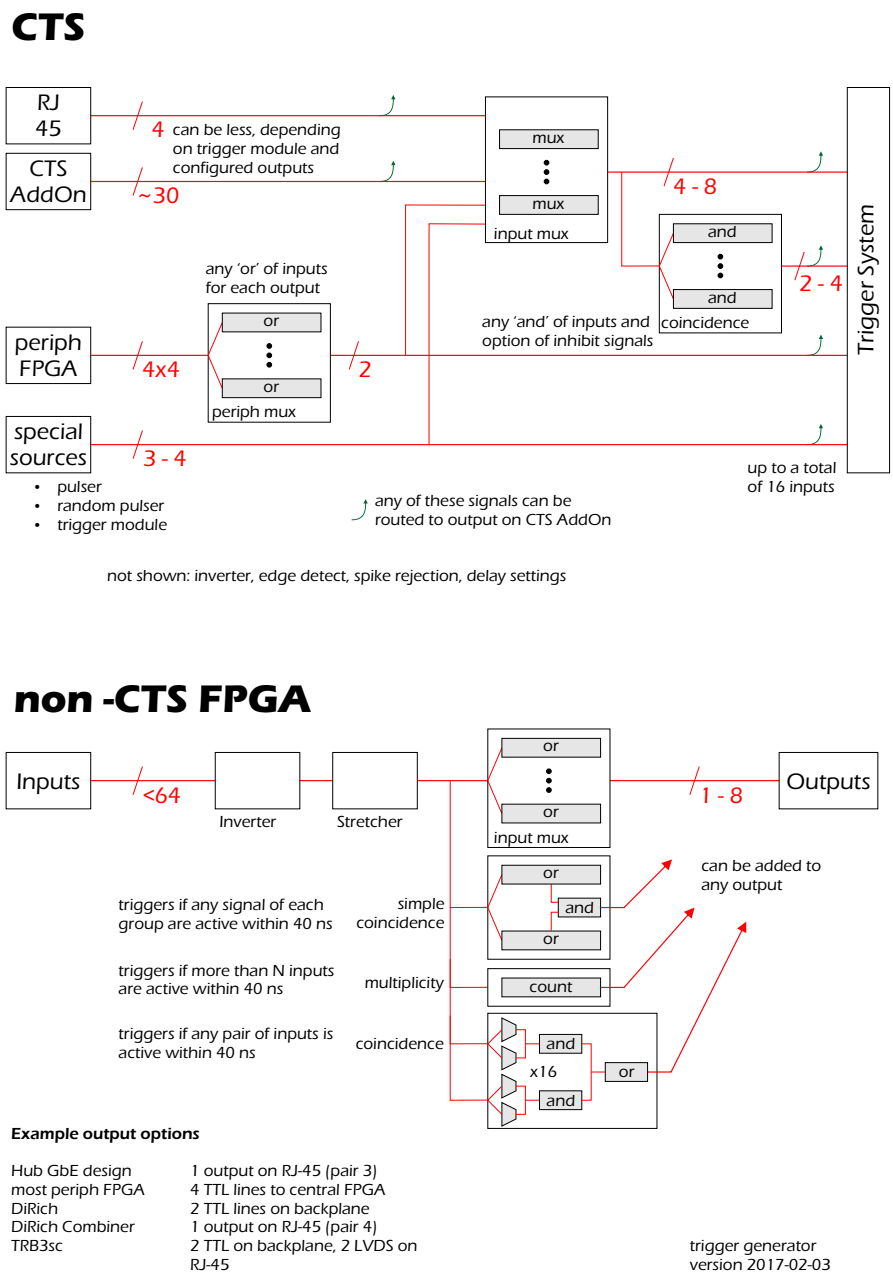


Figure 20: Summary of various trigger sources in CTS and all other FPGAs

Signal	FPGA Loc	Conn.	Wire	Usage
CLK_EXT(3)	U9 (P)	Clock	4 blue	TriggerIn0 / MbsIn / MbsOut / A2Data
	U8 (N)	Clock	5 wh/blue	
CLK_EXT(4)	Y34 (P)	Clock	7 wh/brown	TriggerIn1 / MbsClkOut / A2Clk
	Y33 (N)	Clock	8 brown	
TRIGGER_IN	–	Trigger	1 wh/orange	Global Reference Time (on non-CTS)
	–	Trigger	2 orange	
TRIGGER_EXT(2)	W2 (P)	Trigger	3 wh/green	TriggerIn2
	W1 (N)	Trigger	6 green	
TRIGGER_EXT(3)	W4 (P)	Trigger	7 wh/brown	TriggerIn3 / BusyOut
	W3 (N)	Trigger	8 brown	
TRIGGER_OUT2	W8 (P)	Trigger	4 blue	Trigger Output
	W9 (N)	Trigger	5 wh/blue	

Table 27: All signals are LVDS, hence P/N pins. The column signal refers to the name in `trb3/cts/trb3_central.vhd`. The colour coding standard of the RJ45 cable is T568B. The parenthesis in column denote the index of a std_logic_vector signal.

see table 27. you can get some inspiration from the two existing modules. The semantics of the CTS interface are very straight-forward:

- Synchronously assert the CTS_EXT_TRIGGER line to indicate an event. If the CTS is idle (as indicated by a non-asserted TRIGGER_BUSY_I), a pulse of one clock cycle suffices. If you want the CTS to distribute your event even if the system is busy, you have to keep the line asserted until the the busy line becomes low. However, take into account, that in this case the trigger distribution might be delayed depending on the system's dead time, thus introducing a jitter of several 100 ns.
- If your module has to send information to the event build, you need to implement the read-out channel. Otherwise just leave it unconnected - the default values instruct the front-end to send no data. The Read-Out Channel is directly connected to a FEE IPU Channel. See the HADES DAQ manual for more information.
- CTS_EXT_CONTROL offers you a 32 bit control register without any restrictions on the mapping. Bit 0 should be used to disable the trigger module, e.g. switch of writing of the data word for read-out. It is connected to a synchronous output of the CTS and controlled by the trigger logic. If you need more than 4 byte, just connect to the TrbNet regio bus handler (`THE_BUS_HANDLER`).
- CTS_EXT_STATUS offers you a status register accessible via slow control. If you need more than 4 byte, just connect to the TrbNet regio bus handler (`THE_BUS_HANDLER`).

Listing 1: `Example of a minimalistic ETM instantiation.`

```
THE_ETM: <your-entity-here>
```

```
  port map (
    CLK => clk_100_i,
    RESET_IN => reset_i,

    -- Trigger Interface
    TRG_SYNC_OUT => cts_ext_trigger,
    CTS_BUSY_IN => trigger_busy_i,

    -- READ-OUT Channel
    TRIGGER_IN => cts_rdo_trg_data_valid,
    DATA_OUT => cts_rdo_additional_data(31 downto 0),
    WRITE_OUT => cts_rdo_additional_write(0),
    STATUSBIT_OUT => cts_rdo_trg_status_bits_additional(31 downto 0),
    FINISHED_OUT => cts_rdo_additional_finished(0),

    -- Registers managed by the CTS
    CONTROL_REG_IN => cts_ext_control,
    STATUS_REG_OUT => cts_ext_status,

-- Additional IOs required for your module's logic
  );
```

15.8.2. Obtaining a module id and registering the module

As discussed in chapter 15.6, the software identifies the capabilities of any given CTS by enumerating over a sequence of module headers each containing a module id. The software then loads the drivers located in `daqtools/web/CtsPlugins/CtsMod<lower-case-hex-id>.pm`. This is also the central database of module ids. All ETMs are automatically assigned an ID starting from address 0x60. The order is defined by the `ETM_CHOICE_type` in `config.vhd`.


WARNING

The ETM's id has to be provided to the CTS via the generic constant EXTERNAL_TRIGGER_ID. The trigger logic then automatically reserves a memory block with two words payload (the status- and control register mentioned above). Even if your module, does not require slow control access, it is vital that you specify an id: The constant also functions as an indicator for the presence of an ETM. If the default value is not overridden, your trigger line is not routed to the trigger logic.

Listing 2: Each plug-in has to implement the moduleName method, which is used for introspection purposes, and the init method, that defines all registers and properties. Line 1 to 16 are identical (with expection to the header id and the module's name) in all plug-ins. Registers and properties are created as attributes of the plug-in instance. They are automatically linked to the central file by the parent class CtsBaseModule. This might, again, be useful for introspection purposes, as it allows to determine which registers are defined by a plug-in during run-time. It is, however, currently not used.

```
package CtsMod<ETM-ID>;
@ISA = (CtsBaseModule);
use warnings, strict, TrbRegister;

sub moduleName {"<Human Readable Name of ETM>"}

sub init {
    my $self = shift;
    my $address = shift;
    my $cts = $self->{'_cts'}
    my $regs = $self->{'_registers'};
    my $prop = $self->{'_properties'};
    my $header = $cts->{'_enum'}{<ETM-ID>}->read();
# END OF TEMPLATE

# registers
        $regs->{"<ETM-ABBR>_config"} = new TrbRegister($address + 0,
                                                       $cts->getTrb, {},
        {
           'label' => $self->moduleName . "Config", # feel free to change
           'accessmode' => "rw", # control register: read/write access
           'monitor' => 1, # regularly fetch in monitoring mode
           'export' => 1 # save value when exporting CTS config
        });

        $regs->{"<ETM-ABBR>_status"} = new TrbRegister($address + 1,
                                                       $cts->getTrb, {},
        {
           'label' => $self->moduleName . " Status", #feel free to change
           'accessmode' => "ro", # status register: read-only access
           'monitor' => 1 # regularly fetch in monitoring mode
        });

# human-readable itc assignment, e.g. displayed right to the ITC num in gui.
# feel free to change.
    $cts->getProperties->{'itc_assignments'}[$header->{'itc_base'}] = $self->moduleName;

# properties
    $prop->{"<ETM-ABBR>"} = True; # or some non-false value
} 1;
```

16. Nxyter Read-out[4]

16.1. Design Blocks

16.2. Data Format

16.3. Slow Control

[4]This space to be filled by Ludwig Maier

17. Billboard

The billboard block is designed to help embedding external data directly into the read-out data stream. The module consists of a 256x4b memory area that can be randomly accessed via slow-control and a trigger unit that controls how often data is sent.

17.1. Trigger Scheme

In order to reduce the band-width, the module can skip certain triggers; by default it is completely inactive. There are two methods to select a trigger, namely time-based and trigger-type-based. A combination of both concepts allows you, for instance, to send data at each status trigger, at each 1000th physics trigger, but at-least once every 10 ms. Of course, a time-based read-out only happens, if the CTS issues a trigger.

If either of the following two sub-modules fires, data is written into the stream (this is called an active event):

1. **Time-Based**: The `TimeThresholdReg` registers allows you to define a time interval between two active event. If a read-out happened, all events within this window are ignored for the time-based decision. The first event to arrive after the threshold is selected and triggers a read-out. The counter is reset by any active event. Set the threshold register to 0 (default) to prevent any time-based decision.

2. **Trigger-Type Thinning**: For each trigger type i there exists a so called skip register i. It specifies the number of triggers of this type to be ignored between two active read-out. Consider the physics trigger type 1 and let x be the value of the corresponding register: The first physics trigger received invokes an active read-out, while the next x physics trigger are ignored. The subsequent trigger then becomes active again (and so on). Time-based decisions from other trigger types do not interfere with this pattern. Set the a threshold to 0xffffffff to disable the corresponding trigger type (default).

17.2. Memory

The memory is directly mapped into the slow-control address space at addresses 0xb100 to 0xb1ff and is randomly accessible. The module employs a shadow-memory to prevent data corruption in case a read-out occurs while you update the data. This, however, happens fully transparent and is not resembled on the slow-control addresses.

Once you have written all values into the memory, you can *commit* the data by writing its length into `CommitReg`. In the next active event, the module will send the number of words specified whilst committing into the read-out stream, starting at the lowest memory address. Due to the shadow memory, you cannot reuse unchanged parts of the data, but have to write the whole block, before commit. **Do not assume any initial values!**

Note: In theory it can happen, that you commit while a read-out takes place. In this case, the new memory block is directly accessible (even if it is still used by the read-out). However, the module will stall writes to addresses, that are not sent yet. This may lead to a time-out error for the write-command via slow-control. In this case, simply write it again. The odds for this to happen are very small and it will never occur if you start writing at the lowest address.

17.3. Slow Control

Register		Addr	Bits	Description
CommitReg	(rw)	b000		When reading: Length of data block used for current read-out, When writing: Commit data in memory and set length of data
CommitLength			0–7	Length of data in 32-bit words
TimeThresholdReg	(rw)	b001		Time-Out used for time-based trigger decision (Once a read-out happened use first trigger after at least the time specified)
TimeThreshold			0–31	0: Disable timing-based decision, otherwise: Minimal time between two events in TrbNet clock cycles
FramesSentReg	(r)	b002	0–31	Statistics: Number of triggers with active read-out (frames sent)
WordsSentReg	(r)	b003	0–31	Statistics: Number of words sent (incl. header)
NumberCommitsReg	(r)	b004	0–31	Statistics: Number of commits issued
AgeLastCommitReg	(r)	b005	0–31	TrbNet clock cycles since last commit
SkipTriggerReg[0:15]	(rw)	b010 – b01f	0–31	Number of event of trigger type addr[3:0] to be skipped Default value: 0xffffffff (disable this trigger type)
Memory[0:255]	(rw)	b100 – b1ff	0–31	Memory mapped to slow-control

Table 28: Status and Control registers of billboard block

17.4. SubSubEvent Format

The read-out data-stream starts with a header word, that is present even if no data follows. If the event is active, memory words 0 to *n* follow, where *n* is the length specified when committing. In case of an inactive event, the length indicator still shows *n* but the ActiveEvent-bit is 0. This allows you to check, whether the slow-control update have been successful.

Register	Bits	Description
ReadoutHeader		First word of data-stream
CommitLength	0–11	Number of data (in 32-bit words) committed
NumberCommits	12–15	Lower bits of the NumberCommits statistics counter
CommitAge	16–30	Age of commit in 1.31 ms
ActiveEvent	31	0: No data follows
		1: Amount specified in CommitLength follows

Table 29: Readout Header

18. CBM-MBS Receiver

18.1. Data Format

The module has two distinct output formats that can be selected before synthesis and cannot be changed during runtime. Both share a common first word, containing the received running number as well as status- and the error-bits. If indicated in the header, two additional words storing information to reconstruct the arrival time of this MBS data follow:

In this case, the first additional word (2nd in stream) holds a time-stamp with a granularity of 5 ns. The time-stamp was reset by the arrival of the timing trigger belonging to the LVL1 trigger as indicated by the 3rd word in the data stream. Observe that this information is only valid, if a timing trigger was received not more than 20 s before the arrival of the data word.

Word	Addr	Bits	Description
CommitReg	0		Header
MBSNumber		0–23	MBS-Number received
DataFormat		24	0: Only this header, 1: Two words follow
MBSStatus		29–30	MBS-Status bits received
MBSError		31	MBS-Error bit (high-active)
Timestamp	1	0–31	Time (in 5 ns steps) since timing trigger of event indicated in next word
LVL1Info	2		LVL1-Trigger information of time-stamp
TriggerNumber		0–15	Trigger number sent by CTS
TriggerCode		16–23	Trigger code sent by CTS

Table 30: Read-out format of MBS receiver

19. CBMNet Bridge

The bridge offers two high-level features: it enables TRB3 installations to tunnel their complete "TrbNet data stream" via CBMNet and send it to the FLES.. Data of all CBMNet and TrbNet front-ends is therefore available from a common source and can be processed for online monitoring and analysis. The bridge additionally provides the means to synchronise both networks, i.e. to achieve a fixed delay between two networks that remains during operation and after a restart of the DAQ.

19.1. Synthesising the Bridge

The bridge is currently available for the CTS design only. In order to build it, you have to obtain a copy of the CBMNet source: `https://subversion.gsi.de/cbmsoft/firmware/trunk`

Place the `cbmnet` folder in the same directory the `trb3` and `trbnet` repositories reside.

To active the CBMNet module, make sure the CTS's `config.vhd` contains to following settings:

Listing 3: `Configuration required for building CBMNet bridge with CTS`

```
constant INCLUDE_CTS : integer range c_NO to c_YES := c_YES;
constant INCLUDE_CBMNET : integer range c_NO to c_YES := c_YES;
constant USE_4_SFP : integer range c_NO to c_YES := c_NO;
constant INCLUDE_ETM : integer range c_NO to c_YES := c_NO;

-- optional if, TDC messurements are required
constant INCLUDE_TDC : integer range c_NO to c_YES := c_YES;
constant TDC_CHANNEL_NUMBER : integer := 5;
```

19.2. Read-Out via CBMNet

By default the bridge listens on the Streaming API Bus between hub and Gbe and ignores all data. To active the uplink, set bit Listen of the ReadoutCtrl register (see XML-DB). The bridge will now try to uplink TrbNet events via CBMNet. If the FLES invokes back-pressure, the events are discarded and no pressure is put onto TrbNet.

By deselecting the EnableGBE bit in the ReadoutCtrl register, GBE does not receive data any more and the possible CBMNet uplink rate will increase. Even in this configuration, the bridge refrains from forwarding back-pressure into TrbNet. Thus, events still can get lost (around 1 to 10 ppm during for high data rates).

An unpacker for the CBMNet bridge is included into CbmRoot; this section is hence only included for completeness. The CBMNet bridge currently supports sub-events with up-to 64 KB of size and hence has to split the packet into multiple CBMNet frames. The transmission of a single sub-event is called a transaction. A transaction can only be started once a pending transaction is completed and cannot be gracefully stopped. Due to CBMNet's strong data integrity guarantees, no redundancy or recovery scheme was implemented; if a single error is detected, the whole transaction should be discarded. CBMNet v3 discards corrupt frames without retransmission. While this should happen very rarely (less than $\ll 1$ error in 10^7 frames), current tests show, much higher error rates.

A transaction consists of one or more CBMNet frames containing a single 16 bit header followed by up to 62 byte payload. No routing extensions are used (as inherently supported by TrbNet).

As shown in Figure 21, there are two types of frame headers `uint16_t hdr`. In both the MSB `(hdr»15)&1` indicates the last frame of a transaction, while `(hdr»14)&1` marks the start frame. Note, that both bits are set, if the transaction consists of a single frame.

- `(hdr » 14) & 3 != 0`: Start- or stop frame.
  The lower 12 bit store a sequential transaction number `hdrTransNo = hdr & 0xfff` intended to detect data loss: The bridge increments the number with each new transaction; if you detect a mismatch, it happened somewhere along the way.

- `(hdr » 14) & 3 == 0`: Inner rame.
  `frameTransNo = (hdr » 7) & 0x1f` stores the lower 5 bits of the transaction number and `frameNo = hdr & 0x7f` holds a 7 bits sequential frame number. The transaction number does not change within a pending transaction, i.e.
  `assert(frameTransNo == hdrTransNo & 0x1f)`. The start frame has the (omitted) frame number `frameNo = 0`, i.e. the first inner frame has `frameNo = 1`.

The bridge transmits all frames in order, so no sorting is necessary on the receiving side. Frame numbers only provide the means to detect transmission errors.

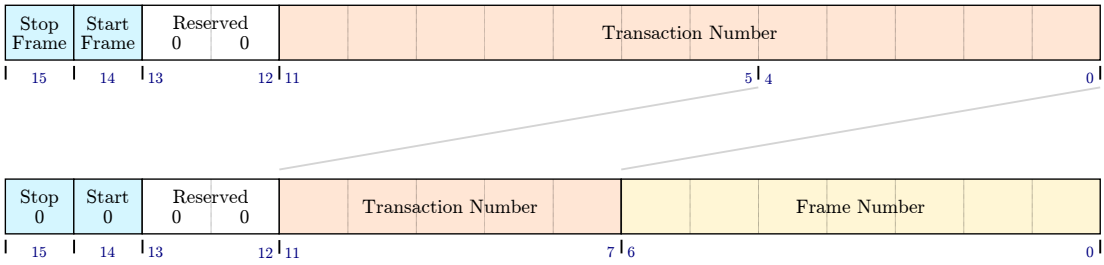


Figure 21: Two frame header types employed by the CBMNet Bridge. On top: A start- or stop-frame header. On bottom: An inner frame's header

For consistency reasons, the TrbNet's 32 bit words[5] are sent MSB first in two half-words via CBMNet, i.e. big-endian. The FLIB then represents the half-words in little-endian, leading to a middle-endian representation of the word that cannot be dealt with by the standard HLD unpackers. It is therefore advisable, to switch the half-words forming a TrbNet word and thus to obtain a complete little-endian representation.

All frames but the last one contain exactly 31 half-words payload; only the stop frame may be shorter. Note, that the last frame can contain as little as 1 half-word as payload (incl. header: 4 bytes). In order to satisfy CBMNet's minimal frame length of 8 byte, up to 2 half-words of padding `0xaaaa` may be appended by the bridge. Padding can be detected by considering the length information (in bytes, incl. all headers) stored in the first word of the sub-event header, i.e. the first word of payload in the transaction.

19.3. Synchronisation with CBMNet

In order to express the TrbNet reference time in CBMNet time using a TDC the following approach is suitable: For every physics trigger the CTS provides a reference time signal, that is distributed in the whole network using a dedicated physical line. The arrival time of this signal gives the means to synchronise the system and gives a T_0 for each event .

The reference time is also sampled in the 125 MHz CBMNet clock domain. Its leading edge triggers the storage of the CBMNet time-stamp (1 word, incremented with each clock cycle), yielding a granularity of 8 ns. To increase the accuracy, the moment of time-stamping is measured using TDC channel 3. Hence, the accurate time is given by

$$T_{\text{cbm}}^{\text{event}} = 8\ \text{ns} \cdot T_{\text{cbm}}^{\text{time-stamp}} - (T_{\text{cbm}}^{\text{ref}} - T_{\text{trb}}^{\text{ref}}),$$

where $T_{\text{cbm}}^{\text{time-stamp}}$ is given in the sync-module (2nd word), $T_{\text{cbm}}^{\text{ref}}$ is provided by TDC-channel 3 and $T_{\text{trb}}^{\text{ref}}$ by TDC-channel 0. The difference between the two TDC measurements should be in the order of 16 ns to 32 ns. In lab tests, the RMS jitter j of the $T_{\text{cbm}}^{\text{event}}$ measurement was found the be $40\ \text{ps} < j < 60\ \text{ps}$.

The Sync-Module can be configured to receive any DLM (but DLM0 which is reserved for CBMNet purposes). After enabling the i-th bit in the DLMSenseMask (see XML-DB), the sync-module begins listening to the DLMs of type i and store the arrival with with a CBMNet time-stamp (see read-out format). Additionally a TDC hit is registers for any active DLM; this, however, should not be required when using the aforementioned approach based on the TrbNet reference time.

19.3.1. Read-Out Format

The CTS-sub-event contains data from three sub-systems in the following order: CTS, CBM Synchronisation and TDC. Each component has its own data structure and must be dealt with

[5] In this document, a word is 4 bytes long. In this notation, CBMNet has a half-word granularity.

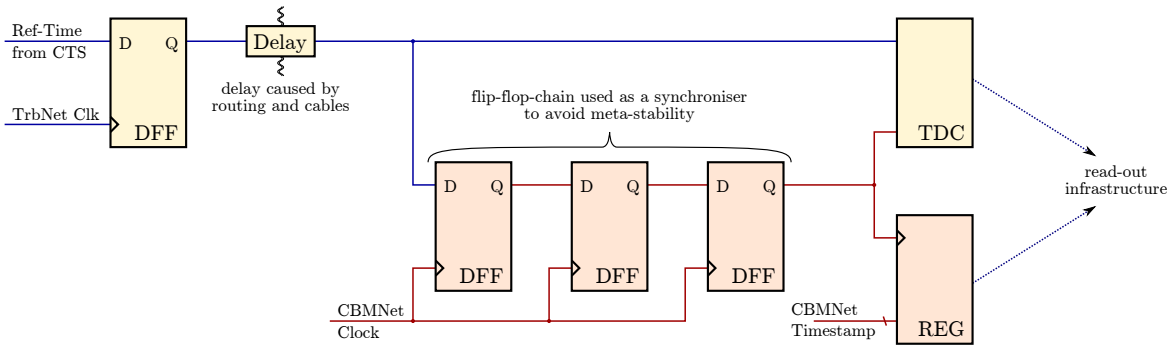


Figure 22: The TrbNet timing reference is sampled in the CBMNet clock domain using an ordinary register chain to avoid meta-stability. The arrival time is expressed in terms of the CBMNet-clock yielding an accuracy of 8 ns. To increase the accuracy, the original- and the sampled strobe signals are measured in the TrbNet domain using two TDC channels.

individually in order to find its length.

The CBM Synchronisation Module measures the relationship between the CBMNet- and TRBNet clock domains only with synchronous logic (in contrast to the TDCs). The first word of the data section contains the header `hdr` that indicates which data follows:

- `(hdr » 26) & 0x3 == 0` – NODATA: section's length (incl. header): 4 bytes
  The CBMNet link is not active, so no data follows.
- `(hdr » 26) & 0x3 == 3` – EXTENDEDDATA: section's length (incl. header): 44 bytes
  This type is transmitted for all timing-less triggers (i.e. status triggers). The header is followed by 10 words as shown in Table 31.
- `(hdr » 26) & 0x3 == 1` – SHORTDATA: section's length (incl. header): 20 bytes
  All physics trigger include only a shortened version, i.e. the first four data words of the EXTENDEDDATA format (Timestamps & Pulser Timestamp).

The TDC channels are mapped as follows:

channel 0: TrbNet Timing Trigger

channel 1: CBM Pulser generate by sync. module. (simple pulser operating at CBMNet clock with a configurable period.)

channel 2: CBM DLM received

channel 3: TrbNet Timing Trigger sampled in the CBMNet clock domain

channel 4: DLM timing reference provided by FLIB for debugging (n.c. in beam-time!)

Addr	Bits	Description
0		**Header**
	1–0	Epoch ID defined by: 0x0: Incremented epoch 0x1: Pre-set via TrbNet slow control 0x2: Pre-set via CBMNet DCM
	3	Low, if slow control update was chosen and no update between last two DLM
	7–4	Barrel shifter position of CBMNet PHY (always 0 in default config)
	23–8	Lower half-word of CBMNet pulser threshold (in clock cycles)
	26	CBMNet link active
	27	Extended Data
	31–28	Header version (= 0x1)
1	31–0	Timestamp (TrbNet clock)
2	31–0	Timestamp (CBMNet clock)
3	31–0	Timestamp of last pulse (TrbNet clock)
4	31–0	Timestamp of last pulse (CBMNet clock)
5	31–0	Current epoch
6	31–0	Timestamp of last DLM (TrbNet clock)
7	31–0	Timestamp of last DLM (CBMNet clock)
8	31–0	Number of DLM received
9	31–0	Number of Pulser pulses
10		**Reset counter**
	15–0	Number of CBMNet resets since FPGA start-up
	31–16	Number of TrbNet resets since FPGA start-up

Table 31: CBMNet synchronisation module read-out format.

Part V.
Experimental Setups and Configurations

20. Trigger Time vs Reference Time

For the experiments with trigger, where the coarse and epoch counters are reset after every event, the asynchronous trigger is used to start the data acquisition. This asynchronous trigger from the detector is processed and synchronised at the Central Trigger System (CTS) and a data readout trigger is sent to the End Points, e.g. TDCs. The arrival of this readout trigger is measured and stored at the Reference Channels (Channel 0) of the TDCs. However, as this data readout trigger is synchronised to the system clock at the CTS, it doesn't carry real time information of the trigger from the detector. The time information at the Reference Channel should be used to synchronise all of the TDCs in the system. In order to calculate the relative hit times to the asynchronous trigger from the detector, the asynchronous trigger must be physically applied to a channel input of a TDC and the time information must be gathered. In future designs the measurement of the asynchronous trigger signal will be done in the CTS by an embedded TDC but confirm the design version you are using by contacting one of the developers!

Part VI.
Software Quick Start

This section is supposed to give a quick overview of the steps to take to get a running TRB3 set-up. It includes installing software and configuring your PC with all necessary settings.

Note that most set-ups use a openSUSE 64 Bit installation - on other distributions things are likely to differ in more or less subtle ways...

21. Distribution Related Notes

21.1. How to set up SUSE Tumbleweed (64bit) on a PC

This is a short overview how you set up SUSE Tumbleweed (64bit). 32-bit will not work with our software.

Due to the many dependencies we recommend to use the proposed system for your setup. If you are not afraid of installing many packets with different names and tweak many different configuration files at different places than described here then it is no problem to use any modern linux distribution.

Listing 4: `Checklist: What do you need`

```
PC (with registered MAC adress)
2 network cards (one for TRB3 communication and one for your institute-
    network)
2 HDD with the same size for RAID-1 (mirroring)
SUSE bootable USB stick for network installation
```

Now you can start with the installation of SUSE. Connect the registered network card with the Institute-network and put in the USB stick. When you have a DELL PC press **F12** and select **USB** in the bootmenu. Now you can go through some self-explanatory steps (choose i.e. language, keyboard language, date/time).

When you come to the point **Partitioning**, choose **Custom Partitioning** and use the following settings.

Listing 5: `Custom Partitioning`

```
1. HDD:
2GB Swap
0xFD Linux Raid (rest of your HDD space)

2. HDD:
```

```
2GB Swap
0xFD Linux Raid (rest of your HDD space)
```

Now click on **Add RAID** and choose **RAID1 (Mirroring)**. As file system chooses **Btrfs**. There are some self-explanatory steps again (choose i.e. computer name, login name, password). You have to **disable** the Firewall, because it only causes many problems. Be sure what you are doing, as a computer directly visible from the internet has its own risks, so an external firewall is recommended. And also make sure to **enable** SSH service.

In the next step you can install some additional software. This would be helpful.

Listing 6: Additional software

```
Remote Desktop
Console Tools
Network Administartion
Base Development
C/C++ Development
Linux Kernel Development
Perl Development
QT 4 Development
Tel/Tk Development
```

Now you can press **INSTALL**. After some time there a some self-explanatory steps again. After that you have successfully installed an SUSE on a machine.

You can also install all the above patterns later with the following command, to avoid waiting times at the initial installation.

Listing 7: Additional software after initial installation

```
zypper in -t pattern xfce console devel_C_C++ devel_perl devel_kernel
    kde mail_server multimedia network_admin office non_oss techn x11
    x86 file_server fonts
```

Some additional packages you need:

Listing 8: Additional Packages

```
sudo zypper ar http://download.opensuse.org/repositories/devel:/
    languages:/perl/openSUSE_Tumbleweed// "devel_perl"
sudo zypper ref

sudo zypper install perl-Parallel-ForkManager perl-IPC-ShareLite perl-
    Log-Log4perl perl-Log-Dispatch perl-Data-TreeDumper perl-File-chdir
    perl-Text-TabularDisplay perl-Text-TabularDisplay perl-CGI perl-JSON
```

```
    perl-qt4-devel perl-Gtk2 gnuplot perl-Config-Auto automake autoconf
    gcc gcc-c++ make dhcp-server rcs git tcl-devel libqt4-devel xorg-
   x11-devel rpcbind emacs subversion cmake xorg-x11-Xvnc
```

"Optional" good stuff (actually it will be needed over time wotking with the system):

Listing 9: Additional Optional Packages

```
sudo zypper install git bash cmake gcc-c++ gcc binutils xorg-x11-
    libX11-devel xorg-x11-libXpm-devel xorg-x11-devel xorg-x11-proto-
    devel xorg-x11-libXext-devel # for root
sudo zypper install gcc-fortran libopenssl-devel pcre-devel Mesa glew-
    devel pkg-config libmysqlclient-devel fftw3-devel libcfitsio-devel
     graphviz-devel libdns_sd avahi-compat-mDNSResponder-devel
    openldap2-devel python-devel libxml2-devel krb5-devel gsl-devel
    libqt4-devel # optional for root
sudo zypper install MozillaFirefox tmux x11vnc openbox lxpanel
    parcellite rxvt-unicode wireshark htop iptraf nmap iftop gkrellm
    xkill ncdu dnsmasq socat
sudo zypper install -t pattern xfce
```

Currently (2017-11), CERN root 6 is again compatible with the gcc-compilers gcc6 and gcc7, so the following is only for a reference if in the future again incompatibilites will happen again.

If needed, you can change the default gcc-compiler like this:

Listing 10: gcc5

```
zypper ar http://download.opensuse.org/repositories/devel:/gcc/
   openSUSE_Factory/ gcc_devel
zypper ref
zypper install gcc5 gcc5-c++ gcc5-fortran

update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5 20
update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-5 20
update-alternatives --install /usr/bin/gfortran gfortran /usr/bin/
   gfortran-5 20
```

Now you can open a shell and make a full system update. Use the package manager `zypper`.

Listing 11: Update SUSE

```
$ zypper ref; zypper -vn dup -l
# Start a sytem update
```

21.2. How to prepare a Debian distribution (and others)

If you installed a minimal[6] Debian/squeeze distribution (or Debian/wheezy), you should install the following additional packages with the following command:

Listing 12: Additional Packages

```
apt-get install build-essential rcs autoconf \
tcl-dev libjson-xs-perl libjson-perl \
libipc-sharelite-perl liblog-log4perl-perl \
liblog-dispatch-perl libtext-tabulardisplay-perl \
libdata-treedumper-perl libfile-chdir-perl \
libhash-fieldhash-perl gnuplot-nox rpcbind \
libconfig-auto-perl automake gcc git wireshark \
htop iptraf nmap iftop
```

This list has been generated from log files, so it might be incomplete. Maybe you want to replace gnuplot-nox by the X-enabled version gnuplot. The best approach is to start the compilation, then look what file seems to be missing, then search the contents of all packages for that file, and install the corresponding package. The recommended way of installing the packages is to use the package manager (synaptic for Ubuntu).

22. Software installation

For convenience I start each code line with prompt ($). If the command (in brown) is too long for one line, then its continuity starts with indentation in the next line. Lines starting with # are comments (in green).

The software is kept in a git repository with the master in Frankfurt. This repository is open and you don't need a password. The mandatory repositories containing software are listed in section 1.2.

Listing 13: Preparation of TRB3 tools

```
cd ~/trbsoft/trbnettools
cat HOWTO_COMPILE.TXT # now you can see all posible ways to compile
    software
 # we compile for TRB3
make distclean
make TRB3=1
sudo make TRB3=1 install # you need root to install the perl library
    part
```

[6] Nothing but base packages and sshd selected

```
 # now export your binaries to PATH
export PATH=${HOME}/trbnettools/bin:${PATH}
 # to make life easier add this line above to your profile to to env
    script
```

To prepare Event Builder you must fetch via git the "daqdata" repository, run the build script. At the end do not forget to export your new locations to `$PATH`. The full procedure is shown in following snippet.

This is not required if you use DABC as data taking platform.

Listing 14: `Preparation of Event Builder`

```
cd ~/
git clone git://jspc29.x-matter.uni-frankfurt.de/projects/daqdata
./make_script.pl
export PATH=${PATH}:${HOME}/daqdata/bin
# this exports also can be added to profile or init script
```

If you get errors like
`evtbuild.c:46:38: fatal error: rawapin.h: No such file or directory`
then edit file `evtbuild.c`, comment out `#define RFIO` and repeat last command (the `for` loop).

22.1. User scripts

On the end it is recommended to create init script for preparing environment. User can create his own or use already prepared user scripts. To get them he must run following

Listing 15: `User scripts preparation`

```
cd ~
mkdir userscripts
```

Listing 16: `/userscripts/startup.sh - Variables to be adjusted`

```
export TRB3_SERVER=trb046
pkill -f "trbnetd -i 11"
$HOME/trbnettools/trbnetd/server/trbnetd -i 11
```

For GbE designs older than August 2013 the correct port number for RPC communication has to be given, e.g:


WARNING

```
export TRB3_SERVER=trb046:25000
pkill -f "trbnetd -i 11"
$HOME/trbnettools/trbnetd/server/trbnetd -i 11
```

This script only has to be started if the server you are using to communicate with the TRB3 has been rebooted.

Usually no changes are required if all other software were installed in user's `${$HOME}` directory (recommended way).

The following (as an example if your trbnetd runs with the id 11) should be added to your shells .rc-script, so for example .basrc or .zshrc:

Listing 17: `.rc-file for your shell`

```
export DAQOPSERVER=localhost:11
export PATH=$PATH:~/trbnettools/bin
export PATH=$PATH:~/daqdata/bin
export PERL5LIB=~/daqtools/perllibs
```

23. Configuration

The TRB software allows you to run several TRBnets for different TRB3 at the same machine at the same time. Each TRB has its own identifier. Your system should have working DHPC and DNS servers.

23.1. Preparing DHCP

The DHCP is used to assign IP address to each TRB3. You must know MAC address of TRB3 in order to configure DHPC server.


WARNING

Running DHPC server in your institution network may run you into troubles with your network administrator. If you are not sure, contact him first. It is safer to run you TRB3 system on separated local network. Ask you computer administrator (AYA) for details.

Commands in this part you must execute as a superuser.

In order to get MAC address of the TRB3 you can ask someone or check by yourself. When TRB3 is starting up it sends request to the network to DHCP servers. Check your system logs to see this requests or use Wireshark program. Here is example

Listing 18: Sample of DHCP request

```
 # you must run following command as a root
watch tail /var/log/messages # it may be different for your system, AYA
 # now restart your TRB3 and watch incoming messages, you should see
     something like this
Nov 29 10:55:18 localhost dhcpd: DHCPDISCOVER from 02:00:be:d9:21:90 via
     eth0
 # if there is another DHCP in the network then you will see its answer
Nov 29 10:55:18 localhost dhcpd: DHCPOFFER on 10.155.59.130 to 02:00:be:
    d9:21:90 via eth0
Nov 29 10:55:18 localhost dhcpd: DHCPREQUEST for 10.155.59.130
    (10.155.59.47) from 02:00:be:d9:21:90 via eth0
Nov 29 10:55:18 localhost dhcpd: DHCPACK on 10.155.59.130 to 02:00:be:d9
    :21:90 via eth0
```

The MAC address of TRB3 is `02:00:be:d9:21:90` and DHCP assigned the IP address `10.155.59.130` (this IP will be different for your network).

We need to know the network address and network mask. For example we take local separated network with address `10.0.0.0` and mask `255.255.255.0`. Your computer (server) is `10.0.00.1`. We assign address `10.0.0.33` to TRB3 with serial number (S/N) 033. Your TRB will have different number, it is good practice to keep IP address the same like serial number, it is easier to find them later.

Now we can configure the DHCP. Open file `/etc/dhcp/dhcpd.conf` with your favourite editor and put

Listing 19: /etc/dhcp/dhcpd.conf

```
non_authoritative;

shared-network YourNetworkName {
       default-lease-time 43200;
       max-lease-time 86400;
       allow unknown-clients;
       allow bootp;

       option subnet-mask 255.255.255.0;

       option domain-name "mylovelytrb3.hades.net";
       option domain-name-servers 10.0.0.1;
```

```
        option ip-forwarding false;

        use-host-decl-names on;

subnet 10.0.0.0 netmask 255.255.255.0 {
# Optionally here you can set default gateway for routing.
# option routers 10.0.0.0;

        group {
                host trb033 {
                        hardware ethernet 02:00:be:d9:21:90;
                        fixed-address trb033;
                }
        }
}
```

This assumes that trb033 is defined in /etc/hosts.

Now we can start your DHCP server. How to do this ask AYA (it is different for different LINUX distributions). You should also configure your RPCBIND (it has nothing to do with HADES RPC) to run with option `-i` → AYA.

Now you can restart your TRB3 and watch again log of your system, you should get following output

Listing 20: `Sample of DHCP request`

```
Nov 29 10:55:18 localhost dhcpd: DHCPDISCOVER from 02:00:be:d9:21:90 via
     eth0
Nov 29 10:55:18 localhost dhcpd: DHCPOFFER on 10.0.0.33 to 02:00:be:d9
    :21:90 via eth0
Nov 29 10:55:18 localhost dhcpd: DHCPREQUEST for 10.0.0.33 (10.0.0.1)
    from 02:00:be:d9:21:90 via eth0
Nov 29 10:55:18 localhost dhcpd: DHCPACK on 10.0.0.33 to 02:00:be:d9
    :21:90 via eth0
```

This means that DHCP works fine. If you have more TRB3s then you need to add more `host` sections inside subnet's group.

23.2. Preparing DNS

If you are using only one computer then you don't need to run DNS server, your local one is enough. As a superuser edit file `/etc/hosts` and add this line

Listing 21: `/etc/hosts`

```
10.0.0.33 trb033
```

If you have more TRB3s add them there also.

Now you can test your DNS with ping. You should get something like this.

Listing 22: `Example of ping results on working TRB3`

```
\$ ping trb033 -c 3
PING trb033 (10.0.0.33) 56(84) bytes of data.
64 bytes from trb033 (10.0.0.33): icmp_req=1 ttl=255 time=0.077 ms
64 bytes from trb033 (10.0.0.33): icmp_req=2 ttl=255 time=0.058 ms
64 bytes from trb033 (10.0.0.33): icmp_req=3 ttl=255 time=0.053 ms

--- trb033 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.053/0.062/0.077/0.013 ms
```

23.3. dnsmasq as an alternative to ISC dhcpd and DNS over /etc/hosts

This section briefly outlines an alternative (and maybe easier and more comfortable) way to configure the TRB boards. It uses dnsmasq to provide the IP addresses over DHCP and also auto-magic DNS resolution. That means that you need to add only one line for every new TRB board. We assume that your TRB boards are connected to a network reachable over `eth1`, and that your PC is the only DHCP server on this network. This section has also some slightly different IP configurations, but this is easy to adapt.

So, starting from the default configuration file `/etc/dnsmasq.conf` on Debian/stable, add the following lines (or uncomment/change the appropriate existing ones):

Listing 23: `Changes to /etc/dnsmasq.conf`

```
interface=eth1
dhcp-range=192.168.0.0
dhcp-host=02:00:be:f9:df:37,trb019,192.168.0.19,infinite
dhcp-authoritative
```

You can add more TRB boards with appropriate `dhcp-host=` lines. Now restart dnsmasq to make your changes take effect. Monitor the successful DHCPACK in `/var/log/syslog`. If you have an dhclient on eth0 (maybe your usual, non-TRB network), you can add your localhost (i.e. dnsmasq) as the first DNS server by adding (or uncomment existing):

Listing 24: `Changes to /etc/dhcp/dhclient.conf`

```
prepend domain-name-servers 127.0.0.1;
```

Now something like `ping trb019` should work nicely.

23.4. Starting TRBnet

[Please also read section GbE Slow-Control (14.1) for additional information.]

Each TRB3 can be controlled with `trbcmd` command over then TRBnet. Each TRBnet is assigned to TRB3 and when communicating to TRB3 you must inform `trbcmd` which TRBnet you want to use. The TRBnet id is the number from range 0...255 (0 is default). In following we will run the TRB3 with serial number (S/N) = 33 on TRBnet with id = 1;

There are several commands and variables which you must know:

TRB3_SERVER This variable keeps the DNS name of the TRB for which we run the TRBnet. Must be set before starting TRBnet.

trbnetd -i $< id >$ This command starts the TRBnet for the TRB3 defined in `TRB3_SERVER`. The `id` is the id for TRBnet.

DAQOPSERVER This variable keeps the server and TRBnet id for running trb commands (see `trbnetd` command).

trbcmd The basic command to manage all TRBnet.

Here is example how to start TRBnet and get IDs of all FPGAs on the TRB3

Listing 25: `Testing TRBnet for multi TRB3 system`

```
TRB3_SERVER=trb033 trbnetd -i 1
DAQOPSERVER=localhost:1 trbcmd i 0xffff
0xf3c0 0x7c0000039021d928 0x05
0xf306 0x9e00000390195f28 0x00
0xf306 0x5100000390195528 0x01
0xf306 0xfc00000390225628 0x02
0xf306 0xb600000390225b28 0x03
```


WARNING

For GbE designs older than August 2013 the correct port number for RPC communication has to be given, e.g:

```
TRB3_SERVER=trb033:25000 trbnetd -i 1
```

If we have several TRB3 and we don't know which TRBnets are assigned to them, we can easily find them with following commands

Listing 26: Identifying of TRBnet daemons

```
ps ax | grep trbnetd
 2556 ? S 1:15 trbnetd -i 33
21818 pts/21 S+ 0:00 /bin/grep trbnetd
cat /proc/\$(pgrep -f "trbnetd -i 33")/environ | strings | grep TRB3
TRB3_SERVER=trb033
```

If you are using only one TRB3 then you can also export variables globally

Listing 27: Recommended way to run TRBnet for single TRB3 system

```
export TRB3_SERVER=trb033
trbnetd -i 1
export DAQOPSERVER=localhost:1
trbcmd i 0xffff
0xf3c0 0x7c0000039021d928 0x05
0xf306 0x9e00000390195f28 0x00
0xf306 0x5100000390195528 0x01
0xf306 0xfc00000390225628 0x02
0xf306 0xb600000390225b28 0x03
```

In future we assume that we have only one TRB3 and these variables are exported.

To start TRBnet user can use script `~/rtb3user/start_trbnet.sh` (see Listing 15). On need to adjust only three variables with the name of DAQ host (DNS name), TRB number (DNS name) and ID of TRBnet daemon to run.

Listing 28: `/trb3user/setenv.sh` - Adjusting variables for TRBnet

```
export DAQHOSTNAME=localhost
export TRBID=trb033 # this is the number labelled on the trb3
export TRBNETID=1
```

This script should be called once with command `source ~/rtb3user/start_trbnet.sh` after setting the proper TRB3 environment.

23.5. Configuring TRB3

As you can see in Listing 27 the FPGA addresses (first column) are not configured yet in your system. The middle columns is the unique ID of the FPGA (temp sensor), the last column is the FPGA number (0x05 is the central FPGA). To set proper addresses execute following command

Listing 29: Updating addresses and serial numbers

```
~/trbsoft/trb3/merge_serial_address.pl ~/trb3/base/serials_trb3.db ~/
    trb3/base/addresses_trb3.db > /dev/null
```

It will load proper configuration to your TRB3. After this we can check whether our changes made expected result.

Listing 30: `Results of addresses changes`

```
trbcmd i 0xffff
0x8000 0x7c0000039021d928 0x05
0x1000 0x9e00000390195f28 0x00
0x1001 0x5100000390195528 0x01
0x1002 0xfc00000390225628 0x02
0x1003 0xb600000390225b28 0x03
```

The files `serials_trb3.db` and `addresses_trb3.db` are the global databases of the all TRB3 produced ever for HADES. If you must assign new addresses to your TRB3 setup, you should open file `~/trb3/base/addresses_trb3.db`. In column `S/N` you should find your TRB3 number and then change TRBnet addresses to proper one (if you don't know → AMT).

After this your changes should be saved, for this commit changes to the repository (requires password).

Listing 31: `Commiting changes to repository`

```
cvs commit -m "Description of your changes" addresses_trb3.db
```

WARNING

Be sure that everything is OK, your changes may break system of all other users of the TRB3.

As the last you should rerun merging of the addresses and serials.

23.6. DAQ configuration

You must configure your TRB3 which is the IP address of the Event Builder. This you can set in proper registers of the central FPGA of the TRB3. For this in the file `~/trb3user/configure_trb3.sh` (see Listing 15) find line starting with `# Event Builder host configuration`, in register `0x8100` write lower four bytes of the MAC, in `0x8101` two upper bytes, in register `0x8102` put IP address of the EB and in `0x8103` the port number.

Listing 32: `Settings Event Builder address for TRB3`

```
# Event Builder host configuration
# registers 0x8100 and 0x8101 - MAC address
# host MAC: 00:11:22:33:44:55
```

```
trbcmd w 0x8000 0x8100 0x22334455
trbcmd w 0x8000 0x8101 0x0011 #upper byte
# registers 0x8102 and 0x8103 - ip address and port
# 192.168.1.1 in hex is c0.a8.01.01
trbcmd w 0x8000 0x8102 0xc0a80101
# port, default if 5000 -> c350 in hex
trbcmd w 0x8000 0x8103 0xc350
```

As there are many steps are necessary to get a correctly configured DAQ it is recommended to use a well supported setup-script as an example and change it to suit it to your needs. A good example is: `~/trbsoft/daqtools/users/gsi_dirc/startup.sh`

After an update of the daqtools repository, you have to update the xml-db register files: `cd trbsoft/daqtools/xml-db;./xml-db.pl`

23.7. CTS monitor configuration

In order to inform CTS server which endpoint hosts the CTS, you must or edit config file (is recommended if you do not change it to often), or give it as a command line parameter for the server. For the first one edit the file `~/trbsoft/trb3/cts/CtsConfig.pm` and change the return value from the getDefaultEndpoint function.

Listing 33: `${HOME}/trbsoft/trb3/cts/CtsConfig.pm`

```
package CtsConfig;

#default cts endpoint. can be overriden by a command line parameter
sub getDefaultEndpoint {
   return 0x8000;
}

1;
```

24. DAQ startup

24.1. Starting TRB3

The next task is to setup all necessary registers in the TRB3, which include TRBNet addresses and Ethernet-MAC-destination addresses. As examples you can find in `~trbsoft/daqtools/users/gsi_ee_lab_kp1pc105/start_test_system_internal_cts.sh`. It contains all vital settings and can be extended as needed. If you want to share your own script, put it to ./daqtools/users/yourproject/.

First the script updates all TRB3 with proper addresses, then the basic configuration for GbE is set. The last part sets registers for TDC and CTS configuration. For further details, read the comments in the example script.

24.2. CTS monitor

Control and monitoring of the CTS is done using a web-server and browser interface. To start it, run `./cts_gui`, to be found in `daqtools/web` (must be the working directory!). There are few option to set the network port the server runs on (default: 1234), configure the output on the command line or to override the default CTS endpoint. Use the `--help` parameter for a short overview. Before executing the script, make sure the DAQOPSERVER environment variable is set correctly.

24.3. Event builder

To start taking a file we need to run event builder. There are two programs which must be run on the same machine. It is important to run both in the same directory as they use shared memory via files. What more, to allow them to communicate each other you need to set the same "system" name via the -S option. The eventbuilders need large receive buffers, which they only can allocate if the system allows them to do this. The command (as root) to set the buffers is:

```
$ sysctl -w net.core.rmem_max=10485760
```

And permanently (persistent after reboot) one can change this file:

```
/etc/sysctl.conf
```

So, to run the eventbuilders, we have to run in one terminal

Listing 34: Running Event Builder, part 1

```
cd ~
daq_evtbuild -m 1 -o /hldfiles -x te -I 1 --ebnum 1 -q 32 -S test -d
    file
# To generate small files with event builder you need to add these
    options to command above:
# --resdownscale 20 --resnumevents 2000 --respath /shldfiles --
    ressizelimit 80
```

where most important options are:

- -o *path* — location for hld files
- -x *prefix* — prefix for hld file, typically te == test, be == beam

- --ebnum *num* — number of event builders
- -S *key* — unique key for SHM
- -d file — write to file
- --respath *path* — location for small hld files
- --resdownscale *num* — scale down for small files
- --resnumevents *num* — number of events in small file

The `/hldfiles` and `/shldfiles` are locations for big and small hld files and should be adjusted to your system.

When daq_evtbuilder is working we need to run daq_netmem to start capture data to file. For this run

Listing 35: `Running Event Builder, part 2`

```
daq_netmem -m 1 -i UDP:0.0.0.0:50000 -q 32 -d 1 -S test
```

where value for parameter `-S` should be the same like for `daq_evtbuild`.

As and quick example you can use

`~/trbsoft/daqtools/users/gsi_ee_lab_kp1pc105/start_readout.pl`.

25. Analysis Software

Now you can control your DAQ system and take data. To analyse the data different groups prepared software packages. All tools need root as a prerequisite the root package from CERN. To install do the following:

Listing 36: `Install ROOT`

```
sudo mkdir /opt/root/
sudo chown hadaq.users /opt/root/
cd /opt/root
wget ftp://root.cern.ch/root/root_v5.34.08.source.tar.gz
tar -xvf root_v5.34.08.source.tar.gz
cd root-v5-34-00-patches
./configure linuxx8664gcc
make -j8
```

Needs quite some time to compile. All the following tools need the root environment, so you have to set the environment variables with:

```
cd /opt/root/root-v5-34-00-patches/
. bin/thisroot.sh
cd -
```

25.1. Mainz Unpacker

Listing 37: Mainz Unpacker

```
cd ~/trbsoft
git clone https://github.com/neiser/mz-unpacker.git
cd mz-unpacker
make
```

The most important point for using this software is the macro to start the analysis.

Listing 38: Mainz unpacker: Example Macro

```
void analyze_macro () {
   gROOT->ProcessLine(".x BuildTrbUnpacker.cpp");
   TTrbUnpacker a("/tmp/xx13120123415.hld" , 0x8000, 0x8000, "",
                  "TDC_Addresses_test.txt", 0, kFALSE);
   a.Decode(0);

   gROOT->ProcessLine(".x BuildTrbCalibration.cpp");
   TTrbCalibration b("/tmp/xx13120123415.hld.root",1,0,kFALSE);
   b.DoTdcCalibration();

   gROOT->ProcessLine(".x BuildTrbAnalysis.cpp");
   TTrbAnalysis c("/tmp/xx13120123415.hld.root_calibrated.root",
                  "TDC_Addresses_test.txt",kFALSE);
   c.Analyse("/tmp/xx13120123415.hld.anal.root",0x200,0);
   TBrowser* t = new TBrowser;
}
```

25.2. DABC and go4

This tool allows a live view to the data. This is *very* useful if you want to find effects while tampering with your setup and see the time precision online.

A manual how to use the software can be obtained here:

```
wget https://subversion.gsi.de/dabc/trb3/Readme.txt
```

In the end the whole building process can be reduced to:

```
cd ~/trbsoft/
svn co https://subversion.gsi.de/dabc/trb3 trb3
cd trb3
make -j4
cd ..
```

However make sure that you use the correct Root version (indicated in the requirements). For DABC (data readout) one need a configuration file:

```
cd ~/trbsoft/trb3/dabc
wget https://subversion.gsi.de/dabc/trunk/plugins/hadaq/app/EventBuilder
    .xml
```

This config file has to be edited.

Then the specific analysis part has to be installed.

```
cd ~/trbsoft/trb3

wget https://subversion.gsi.de/go4/app/stream/include/hadaq/
wget https://subversion.gsi.de/go4/app/stream/framework/hadaq/
wget https://subversion.gsi.de/go4/app/stream/applications/trb3tdc/first
    .C
```

Then continue as described in the Readme.txt.

You will not see any event delivered by DABC, if your trigger rate is set too low, e.g. 100 Hz. First set your trigger rate to a higher value, 1kHz, start the DABC and go to lower rates, if you still need to.

25.3. DABC documentation

A TRB3-related web-page for the DABC-software is here:

```
http://dabc.gsi.de/doc/dabc2/hadaq_trb3_package.html
```

26. Web interface

Now in your browser open the page `ctshost:1234`, where `ctshost` is the machine when CTS server is started. If the CTS server is running we can also have access to configuration pages for different components, described in following parts.

/ Main page with links to all individual screens

cts.htm The main control screen for all CTS functions

tdc/tdc.htm The TDC status - input status, hit counter, status registers and control function to enable individual inputs

network/gbe.htm The Gigabit Ethernet status information

thresh/threshold.htm Manual setting of thresholds for CBM-Rich and Padiwa boards

padiwa/padiwa.htm Temperature, IDs and threshold information about Padiwa front-ends

Note: All pages despite the CTS monitor fetch data directly from the DAQ network. Make sure that you don't read too often or from too many open browser windows at the same time.

27. Data File Format

The data in the hld file is binary data, organized in 32 Bit words. For historic reasons, some parts are big endian, some are little endian - check the existing data reading code for their detection method. Each event has an event header, followed by an arbitrary number of subevents. The subevents contain a header and a data block, consisting of subsubevents, which are data blocks from individual FPGAs. The structure is shown in figures 24 and 25.

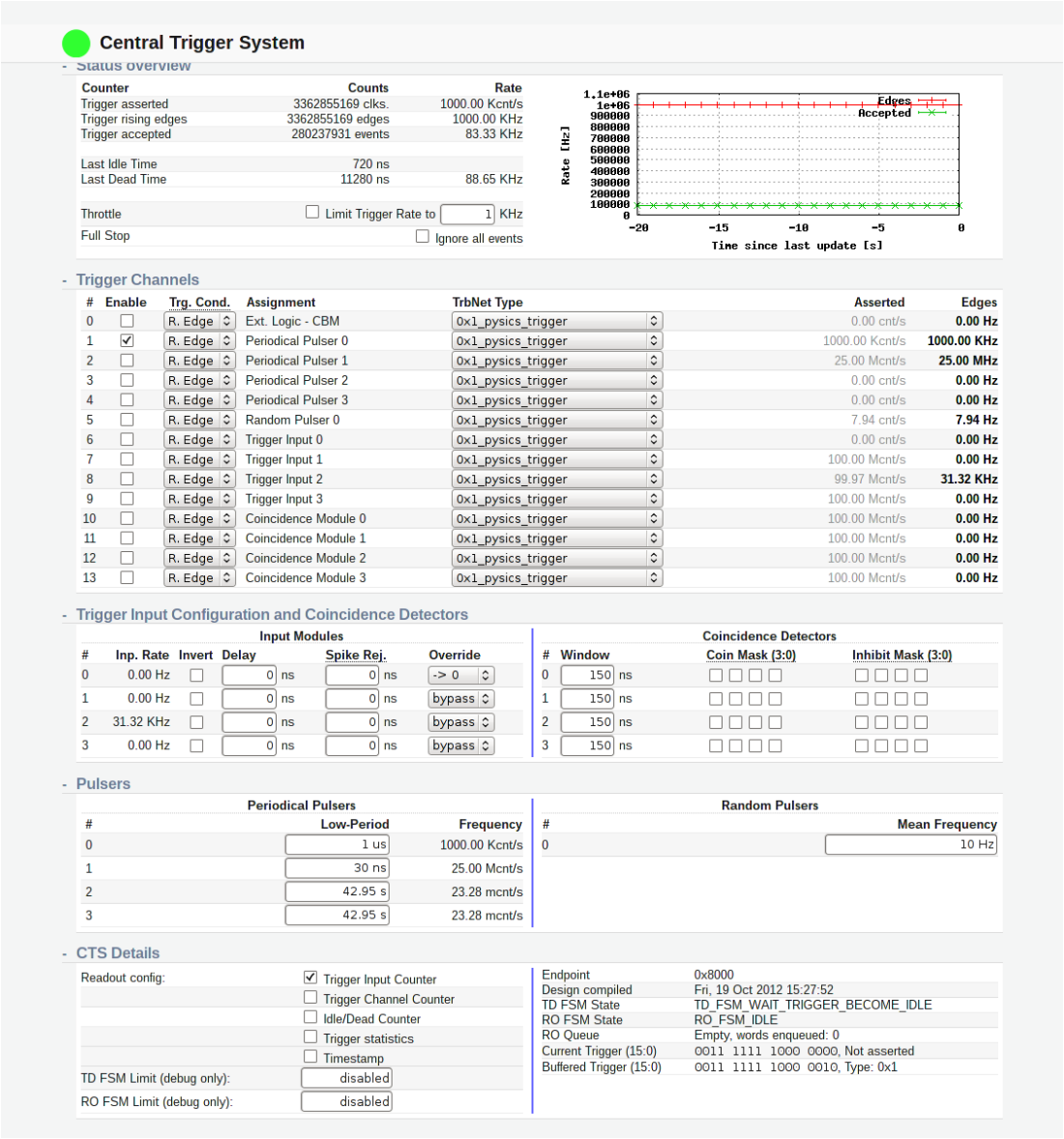


Figure 23: Screenshot of the CTS window. In this example the Pulser #0 with frequency 1 MHz is used as a trigger source.

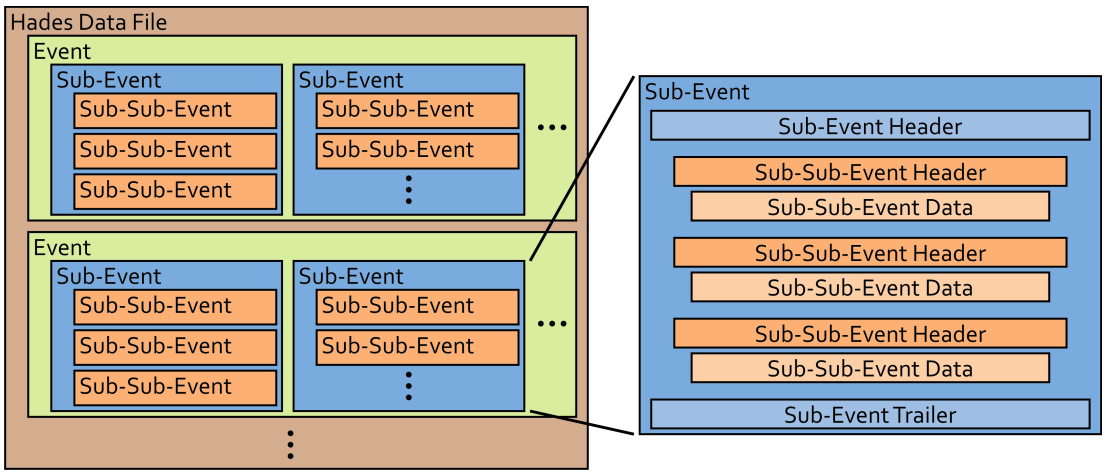


Figure 24: The structure of the hld file.

HADES Event Header

	31 … 24	23 … 16	15 … 8	7 … 0
Size	event size (byte)			
Decoding	0x0	word size	reserved	1
ID	reserved		version (0x20)	trigger type
Sequence	event number per file			
Date	reserved	year	month	day
Time	reserved	hour	minute	second
Run ID	Run Number			
Padding	reserved			

Sub Event

	31 … 24	23 … 16	15 … 8	7 … 0
Size	size (byte)			
Decoding	0x0	word size	trg.type	reserved 1
ID	reserved		sub-event ID	
Number	trigger number		trg. code	
SSE-Hdr	length		sub-sub-event ID	
SSE-Data	sub-sub-event data (arbitrary size)			
SSE-Hdr	length		sub-sub-event ID	
SSE-Data	sub-sub-event data (arbitrary size)			
Trailer	length (0x0001)		SEB-ID (0x5555)	
Status	error & status information			

Figure 25: The structure of the event header and sub-event inside the hld file.

Part VII.
Synchronous TrbNet

28. Media Interfaces

28.1. Central Aspects

- All FPGA run on a central clock, each network node recovers the clock from the optical link and uses it as the main clock source for all design components.
- Low-level messages with deterministic latency can be inserted in the data stream and forwarded in the direction toward the front-ends
- Links toward the front-ends run in bit-synchronous mode, i.e. the word alignment of the Serdes is always locked on Bit 0.
- Links toward the read-out run in clock-synchronous mode only, i.e. the round-trip time for a packet can be measured with one word length (currently 5 ns) precision only.

28.2. Clock Measurements

To check the limits of the clock recovery circuitry of the FPGAs a system consisting of 5 TRB3 was setup. One master TRB3 with the TRBHub design in one peripheral FPGA, 4 slaves, where 3 of the slaves were in one chain, and the 4th slave was connected to the master as a single board. So, each slave received the clock of the level above in the central FPGA (FPGA5) where the clock was recovered. This clock is then used to communicate to the peripheral FPGA, where again a HUB-design was running. This peripheral FPGA again recovered the clock. And so on.

This means the clock-recover and transmit was done for each FPGA, so in total 6 times on the chain with the 3 TRB3.

The measurement shows the following:

- the scope *shows* the clock itself with a jitter of 10ps. In reality it is known that the jitter is several hundred femto seconds, so this is the offset of the measurement. The analysis was done with the installed “Advanced Jitter Software” on a Tektronix-Scope.
- after the first clock recovery of the SERDES clock we measure a clock jitter of 30ps sigma
- after 6 recoveries in a chain we measure 40ps sigma.
- The recovered clocks of the two branches, one with one TRB3 as a leaf and the other with a chain of 3 TRB3s show the are phase locked.
- The transmission of data didn’t show any transmission errors for a full day.

All done without jitter cleaner.

The time-trend analysis of the measured deviations of the recovered clock to the ideal clock shows already after the first recovery that there is very small random jitter, but a dominant synodical time deviation with a frequency of 4MHz. This suggests that a jitter cleaner (if the time constants are too short) will not help a lot and that the jitter comes from the power-supply of the FPGAs-PLL-circuitry. The DC/DC-converters on the TRB3 run with 4MHz, which can be a coincidence, but directly points to them as the source of the deterministic "jitter" (period length oscillation). This has to be investigated further to be understood in detail.

28.3. Media Interfaces

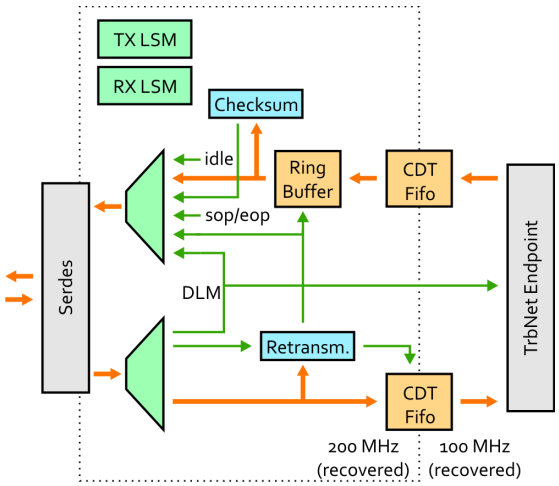


Figure 26: New Media Interface Block Diagram

The link start-up procedure

Each link has a defined master and slave side, given by the hard-wired configuration of each design. The reset of the TX and RX reset state machines follows the Lattice reference design.

1. The master switches on its transceiver
2. The slave recovers the clock, resets itself as often as needed to lock on Bit 0. Lattice ECP3 reset recommendation takes 4 ms per try, but actual locking time seems to be about 1 ms. During this time, the RX clock (which is the main system clock) is not stable so that the slave logic needs to be in reset state.
3. The slave activates its transceiver (or: transceiver is already on, and slave switches to another idle comma character to signal readiness - to be investigated).
4. The master locks on the received stream / detects the new comma character. To make sure that the link is stable (e.g. when plugging in a cable by hand), a delay of about 500 ms is started

5. The master declares the connection active and changes its own idle comma character.

Using a recovered clock to drive its own transceivers has the implication that if one link is lost and needs to be restarted, the full tree of boards behind the erroneous one needs to be resynchronized. It needs to be investigated if this is possible in a clean way or all these FPGA need to be reset automatically because of loss of the RX clock. This needs in detail check of the transceiver PLLs in case of an unlock of the clock recovery PLL.

The bi-directional hand-shake by changing comma characters is necessary when front-ends are able to actively send data to the read-out - it can be omitted if there is only a central master as in triggered systems.

Comma Characters

Each control word consists of two parts, a comma character and a data character. Assignment of individual character is not fixed and can be changed by constants in a VHDL package. CBMnet uses a set of 64 16 bit characters with big Hamming distance for error correction - this will not be done in TrbNet due to additional overhead and low error rates on transmission via SFP.

Name	Full Name	K Character	D Character
IDLE	Idle	BC	C5 (idle0) or 50 (idle1)
SOP	Start of Packet	DC	Channel / Packet Type (not in first version)
EOP	End of Packet	FD	CRC
BGN	Begin of Transmission	1C	Buffer Position
REQ	Retransmit Request	7C	Buffer Position
DLM	DLM	FB	Deterministic Latency Message
		3C	
		5C	
		9C	
		FC	
		F7	
Reset	Link reset	FE	none

Table 32: Comma Characters

Appendix A TDC Calibration

This will be the appendix for the TDC calibration.

References

[Kal04] J. Kalisz. Review of methods for time interval measurements with picosecond resolution. *Metrologia*, 41(1):17–32, 2004.

[Lat09] Lattice Semiconductor Corporation. *LatticeECP2/M family handbook*, March 2009. HB1003 Version 04.3.

[Pen12] Manuel Penschuck. Development and implementation of a central trigger system for trbnet-based systems. Bachelor Thesis, Uni Frankfurt, 2012.

[SAL06] J. Song, Q. An, and S. Liu. A high-resolution time-to-digital converter implemented in field-programmable-gate-array. *IEEE Transactions on Nuclear Science*, 53(1):236–241, February 2006.

[SKP97] R. Szplet, J. Kalisz, and R. Pelka. Nonlinearity correction of the integrated time-to-digital converter with direct coding. *IEEE Transactions on Instrumentation and Measurement*, 46:449–453, April 1997.

[WS08] J. Wu and Z. Shi. The 10-ps wave union tdc: Improving fpga tdc resolution beyond its cell delay. *Nuclear Science Symposium Conference Record, 2008 IEEE*, pages 3440–3446, 19-25 October 2008.