Documentation of FCS Software, Version 3

T. Fuljahn (thomas.fuljahn@desy.de) January 5, 1999

Contents

1 Introduction

This document describes the control software for the HERA-B Fast Control System (FCS), described in [1]. The Fast Control software is part of the HERA-B online software. It is devided up into three major parts.

First there is a server process called `FCS_server` which works as an interface between the HERA-B online software and the Fast Control System. The server process will be the only process accessing the Fast Control System. It receives messages from other processes and commits requested settings to the Fast Control System.

The second part of the Fast Control software is a library of client functions called `libfcsc.a` which can be used by any online program to send messages to the server process.

Finally three client programs are provided. One is `FCS_send` which transforms command line arguments to messages sent to the server. It can be used for manual access to the FCS or in shell scripts. The second one is `FCS_set_mother` which works the same way, but only for setting mther parameters. The third program is a graphical user interface consisting of a shell `fcssh` and a script `fcspanel`.

2 Overview

The FCS software source code can be devided like the following:

Hardware Drivers provide a way of accessing the internal (hardware) resources of the FCS without providing much "userfriendlyness".

Logical Drivers provide a way to configure, control and monitor parts of the FCS on a slightly more complex level than the hardware drivers. Both hardware drivers and logical drivers are used by the server.

Server To ensure proper interlocking and enable resource management only one process should run locally on the CPU in the FCS VME crate and access the FCS hardware directly. This process is called the FCS_server. The server process initializes the FCS during startup. Then it waits for messages from clients and every time it receives a message the server interpretes the message and executes a code sequence depending on the type of the message.

Utilities Can be written using logical and hardware drivers to perform specific commonly needed maintenance or test procedures.

Client functions The functions in the client library libfcsc.a are available from any remote node in the HERA-B computing system. The client functions use the HERA-B message system RPM ("Really Powerful Messaging") [4] to send requests and commands to the server.

Clients Programs using functions from the client library.

3 Dependencies

The hardware of the Fast Control system is specified in [1]. There should be always an updated version of [1] available on the HERA-B DAQ Web-pages [3].

A VME driver is required. Therefore the MIZZI VME Library is used with LynxOS on the Cetia computers.

Data Base Service is required to store the configuration. See section 5.

Message system communication is needed for the FCS Application. The RPM package [4] fulfils this requirement.

4 Code Organization

The FCS_server is using the hardware driver to access the FCS hardware and some higher level interfaces for genaral setup of mother or daughter modules and for initialization.

The following files are used for the FCS server software:

`FCS_hardware.h`	contains VME addresses and prototypes for the hardware drivers
`FCS_hardware.c`	contains the hardware driver source code
`FCS_hardware.o`	contains the hardware driver object code
`FCS_logical.h`	contains typedefs and the logical driver prototypes
`FCS_logical.c`	contains logical driver source code
`FCS_logical.o`	contains logical driver object code
`FCS_server.h`	contains definitions of all FCS specific RPM messages
`FCS_srv_subs.h`	contains global variables and prototypes of the server parts
`FCS_srv_subs.c`	contains the server parts executed on each type of message
`FCS_srv_subs.o`	contains the server parts object code
`FCS_server.c`	contains the FCS_server source code
`FCS_server`	is the FCS_server executable

And the following files are used for the FCS_client software:

`FCS_server.h`	contains definitions of all FCS specific RPM messages
`FCS_client.h`	contains typedefs, macros and client function prototypes has to be included by any client program
`FCS_client_int.h`	contains internal data for the client interface to rpm has to be included in library functions, not in clients
`FCS_*.c`	Every client function source code is in its own file
`libfcsc.a`	contains the FCS Client object code to the logical drivers

To compile an application with an FCS client interface the header files `FCS_server.h` and `FCS_client.h` are needed. The object library `libfcsc.a` has to be linked.

The hardware driver object file `FCS_hardware.o` is compiled using the files `FCS_hardware.h` and `FCS_hardware.c`.

The logical driver object file `FCS_logical.o` is compiled using the files `FCS_hardware.h`, FCS_logical.h, FCS_logical.c and FCS_hardware.o.

The server application FCS_server needs the header files FCS_hardware.h, FCS_logical.h, FCS_srv_subs.h, FCS_server.h, the source code file FCS_server.c and the object files FCS_hardware.o, FCS_logical.o and FCS_srv_subs.o. The server waits for RPM messages and on every received message executes a switch(header->type). In each case a function from FCS_srv_subs.ois called with the received message as parameter. All these functions are called FCSSRV_$*$ where $*$ is related to the name of the message type as defined in FCS_server.h.

5 To be clarified...

Process management How will it be assured that only one server is running.

Adaption of FCS software to the DAQ configuration database. Instead of using the special configuration files FCS_mother_defaults.dat and FCS_daughter_defaults.dat the configuration data base could be used to store the default parameters. Additionally one could think of, for example, the assignment of FCS daughter module $\Longleftrightarrow$ FEDs. If this change will be done, it would have to be clarified how these data could be written to and read from the database. Ref. [2] contains function prototypes for reading and writing daughter and mother parameters from and to the database:

FCS_read_dau_par(unsigned char cluster, unsigned char daughter,
    FCS_daughter *par)
FCS_store_dau_par(unsigned char cluster, unsigned char daughter,
    FCS_daughter *par)
FCS_read_mot_par(FCS_mother *par)
FCS_store_mot_par(FCS_mother *par)

6 Hardware Driver

The following set of functions allows for an easy access to the hardware registers of the Fast Control System . This is the lowest level interface between any software and the FCS hardware. Data transmitted as function parameters are written into and read from registers using VME accesses.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **usiesta**(int val);
/* Smaller siesta than usleep provides */

Parameters

val calibrated for the old CETIA, 1 unit $\approx 2\ \mu$s

Return values

Discussion

Usleep is implemented in LynxOS in a way that it takes at least tens of ms.
F. Sanchez provided this usiesta which works in steps of $\approx 2\ \mu$s.

FCS_get_version

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_get_version(unsigned char *logic, unsigned char *stat);**
/* returns the FCS master FPGA version numbers */

Parameters

logic	version number of LOGIC FPGA
stat	version number of STAT FPGA

Return values

FCS_ok	normal return
FCS_bus_error	VME bus error

Discussion

The possibility to read back the XILINX version is a new feature of FCS master Rev. 3. It started with version number 1 in both XILINX's. The actual **LOGIC** version is **2**. (With regard to version 1 the VME trigger logic was corrected). The actual **STAT** version is **1**.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_set_control**(**unsigned char control**);
/* writes the FCS control register */
int **FCS_get_control**(**unsigned char *control**);
/* returns the FCS control register content */

Parameters

control	is the control register as defined in [1]

Return values

FCS_ok	normal return
FCS_bus_error	VME bus error

Discussion

the control register has the following bits:
0 BX enable
1 FLT enable
2 Overflow scheme
3 BX stop

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_set_time**(**int time_register, int time**);
/* writes a FCS time register */
int **FCS_get_time**(**int time_register, int *time**);
/* reads a FCS time register */

Parameters

time_register	is one of the time registers as defined in [1]
time	is the value the register

Return values

FCS_ok	normal return
FCS_out_of_range	time is out of allowed range for register or register number is not valid
FCS_bus_error	VME bus error

Discussion

the following time registers are available:
FCS_latest_accept
FCS_fifo_depth
FCS_readout_duration
FCS_buffer_delay
FCS_trigger_offset
FCS_random_factor

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_reset_sharc**(**void**);
/* clears the SHARC-link FIFO */

Parameters

none

Return values

FCS_ok	normal return
FCS_bus_error	VME bus error

Discussion

The SHARC-link FIFO is cleared.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_clear_counters**(**void**);
/* clears FLT and statistics counters */

Parameters

none

Return values

FCS_ok	normal return
FCS_bus_error	VME bus error

Discussion

The FLT counter, the FLT overflow counter and all FLT inhibit counters are cleared.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_set_counters**(**long *values**);
/* sets the FLT and inhibit counters */
int **FCS_get_counters**(**long *values**);
/* reads the FLT and inhibit counters */

Parameters

values is the counter condition pattern of size 6*32bit as defined in [1]

Return values

FCS_ok	normal return
FCS_is_fishy	unclassified error (to be confirmed)

Discussion

the following logical counters are available:
0 FCS_flt_inhibit
1 FCS_fp_inhibit
2 FCS_vme_inhibit
3 FCS_bufdel_inhibit
4 FCS_bufof_inhibit
5 FCS_flt_of

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_set_bx_count**(**unsigned long *counter**);
/* sets the BX counter */
int **FCS_get_bx_count**(**unsigned long *counter**);
/* reads the BX counter */

Parameters

counter	is an array of size 2 for the BX-counter (64bit)

Return values

FCS_ok	normal return
FCS_bus_error	VME bus error

Discussion

The counter overflows after 64 bit. With a rate of 10.417 MHz this is sufficient for 56,000 years which is definitely more than the HERA–*B* lifetime. With this functions the counter can be set to any value and the status of the counter can be read.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_set_histo**(**long *data**);
/* writes the FLT time histogram into the FCS */
int **FCS_get_histo**(**long *data**);
/* reads the FLT time histogram from the FCS */

Parameters

data	is a buffer of size 256 (times 32bit)

Return values

FCS_ok	normal return
FCS_is_fishy	unclassified error (to be confirmed)
FCS_bus_error	VME bus error

Discussion

See [1] for the definition of the histogram contents.

FCS_trigger FCS_trigger

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_trigger(unsigned char physBx);**
/* issue a VME trigger on physical BX number physBx */

Parameters

physBx	number of HERA-bucket to trigger on

Return values

FCS_ok	normal return
FCS_out_of_range	physBx > 219
FCS_bus_error	VME bus error

Discussion

A VME trigger is sent. All daughters with the corresponding enable bit set send this puls to their backplanes.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_handshake**(**void**);
/* give a VME handshake */

Parameters

none

Return values

FCS_ok	normal return
FCS_bus_error	VME bus error

Discussion

A VME handshake is issued. This decreases the buffer count in the FCS mother by one.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_valid_address(unsigned char cluster, unsigned char daughter);**
/* check whether the addressed daughter exists */

Parameters

cluster	cluster number
daughter	daughter number

Return values

1	address is valid
0	address is not valid

Discussion

This function checks whether the addressed daughter exists.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_test_data(unsigned char cluster, unsigned char daughter, int frontpanel, int backplane, unsigned char delay);**
/* set the test pulse data */

Parameters

cluster	is a cluster address (0 for all clusters)
daughter	is a daughter address (0 for all of cluster)
frontpanel	if =1, test pulse will be sent to frontpanel
backplane	if =1, test pulse will be sent to backplane
delay	is the test pulse trigger delay in units of BX cycles

Return values

FCS_ok	normal return
FCS_out_of_range	delay out of range
FCS_unknown_daughter	unknown daughter
FCS_bus_error	VME bus error

Discussion

This function is used to prepare a test pulse. It is defined whether the pulse is sent to the daughter's frontpanel or backplane and the test pulse trigger delay is set.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_testpulse(unsigned char cluster, unsigned char daughter, unsigned char time);**
/* issue a test pulse */

Parameters

cluster	is a cluster address (0 for all clusters)
daughter	is a daughter address (0 for all of cluster)
time	the time between the testpulse and the following BX in units of 0.5 ns

Return values

FCS_ok	normal return
FCS_out_of_range	time out of range
FCS_unknown_daughter	unknown daughter
FCS_bus_error	VME bus error

Discussion

A test pulse is issued time * 0.5 ns before a following BX clock to the selected daughters. A trigger accept for this BX is issued by the mother afterwards. The timing for this accept was set before with FCS_test_data. The FLT should be masked accordingly at the target daughters. The server calls FCS_test_data and enables the testpulse trigger before the testpulse is issued.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_set_strobes**(**unsigned char cluster, unsigned char daughter, unsigned char pattern);**
/* set strobe pattern */

Parameters

cluster	is a cluster address (0 for all clusters)
daughter	is a daughter address (0 for all of cluster)
pattern	is a strobe pattern

Return values

FCS_ok	normal return
FCS_bus_error	VME bus error

Discussion

The strobes at the selected daughters are set to the specified pattern.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_toggle_strobes(unsigned char cluster, unsigned char daughter, unsigned char pattern);**
/* toggle selected strobes */

Parameters

cluster	is a cluster address (0 for all clusters)
daughter	is a daughter address (0 for all of cluster)
pattern	is a strobe pattern

Return values

FCS_ok	normal return
FCS_bus_error	VME bus error

Discussion

All strobes with a set bit in the pattern get toggled once.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_set_delay(unsigned char cluster, unsigned char daughter, unsigned char delay);**
/* set daughter delay */

Parameters

cluster	is a cluster address (0 for all clusters)
daughter	is a daughter address (0 for all of cluster)
delay	the delay of the daughter in units of ns/2

Return values

FCS_ok	normal return
FCS_out_of_range	delay out of range
FCS_unknown_daughter	unknown daughter
FCS_bus_error	VME bus error

Discussion

Sets the BX clock delay for the specified daughter

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_set_offset(unsigned char cluster, unsigned char daughter, unsigned char offset);**
/* set daughter BX offset */

Parameters

cluster	is a cluster address (0 for all clusters)
daughter	is a daughter address (0 for all of cluster)
offset	BX offset of daughter

Return values

FCS_ok	normal return
FCS_out_of_range	offset out of range (¿7)
FCS_unknown_daughter	unknown daughter
FCS_bus_error	VME bus error

Discussion

The function sets the BX number offset for the specified daughter. The hardware offers only 3 bit. Therefore the offset might not be larger than 7.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_mask_flt(unsigned char cluster, unsigned char daughter, unsigned char mask);**
/* set daughter FLT mask */

Parameters

cluster	is a cluster address (0 for all clusters)
daughter	is a daughter address (0 for all of cluster)
mask	FLT mask

Return values

FCS_ok	normal return
FCS_unknown_daughter	unknown daughter
FCS_out_of_range	bit 7 of unsigned char mask must not be set
FCS_bus_error	VME bus error

Discussion

Every set bit enables the corresponding FLT kind for the specified daughter.

sources: `FCS_hardware.h`, `FCS_hardware.c`
object: `FCS_hardware.o`

int **FCS_latch_flt(unsigned char cluster, unsigned char daughter,);**
/* latch daughter's local FLT counter */

Parameters

cluster	is a cluster address (0 for all clusters)
daughter	is a daughter address (0 for all of cluster)

Return values

FCS_ok	normal return
FCS_is_fishy	unclassified error (to be confirmed)
FCS_unknown_daughter	unknown daughter
FCS_out_of_range	bit 7 of unsigned char mask must not be set
FCS_bus_error	VME bus error

Discussion

The local FLT counter on the specified daughter(s) is latched to be read out via the CAN-bus monitoring interface.

7 FCS addressing

The daughter addressing works with a two stage hierarchy. On each daughter board there are coded a daughter number and a number of a logical cluster the daughter belongs to. In general, all daughters supplying one subdetector will have the same cluster number. The pretrigger- and trigger-link-boards of different subdetectors are treated as clusters of their own. The outer tracker is special in two respects: the trigger-link-boards are not supported by separate daughters because they are included in the FED crates and the number of daughters is so large that a division of the detector is necessary. In the header files FCS_logical.h (for the server) and FCS_client.h (for the clients) abbreviations for the subdetectors are defined as constants to address the cluster. The definitions look like:

```
#define SVD 1
#define ITR 2
#define ITR_TL 4
#define OTR_MC 8
#define OTR_PC1 16
#define OTR_PC2 32
#define OTR_PC3 64
#define OTR_PC4 128
#define OTR_TC1 256
#define OTR_TC2 512
#define HIGH_PT 1024
#define RICH 2048
#define TRD 4096
#define ECAL 8192
#define E_PRE 16384
#define MUON 32768
```

The server interpretes the cluster number it receives from a client as a bit mask. A group of clusters can be addressed by adding the values of the corresponding bits. For example, the inner tracker FEDs and trigger links can be addressed together writing either `(ITR & ITR_TL)` or `(ITR + ITR_TL)` or directly `(6)`.

Daughter number 0 is used for a broadcast to all daughters belonging to one cluster. Cluster number 0 is used for a broadcast to all daughters:

```
#define ALL_CLUSTERS 0
#define ALL_DAUGHTERS 0
```

sources: `FCS_logical.h`, `FCS_logical.c`
object: `FCS_logical.o`

int **FCS_cluster_ids(unsigned short cluster_mask, int* number_of_clusters, unsigned char* cluster_list);**
/* interprete cluster addressing bit mask */

Parameters

cluster_mask	is a cluster mask
number_of_clusters	number of entries in cluster_list
cluster_list	list of hardware addresses of specified clusters

Return values

FCS_ok	normal return

Discussion

This function interpretes the cluster_mask. For each set bit in the 16-bit-number the corresponding cluster address is put into a list. If the cluster_mask is “0”, the list contains one entry, which is “0”, the code for a broadcast.

8 FCS configuration (server subroutines)

The parameters of a daughter are saved in a structure of type FCS_daughter, the ones for the mother in FCS_mother.

```
typedef struct{
 int ovfl_scheme;          /* readout buffer overflow scheme */
 int flt_offset;           /* offset of test & random FLTs to current BX */
 int latest_accept;        /* effective length of FLT pipeline */
 int fifo_depth;           /* depth of readout buffer fifo */
 int readout_duration;     /* time in BX cycles to readout one event */
 int buffer_delay;         /* deadtime after one flt accept (in 96 ns) */
 int random_factor;        /* time between two random triggers (in 96 ns) */
}FCS_mother
```

```
typedef struct{
 int strobe_pattern;     /* default pattern in run mode */
 int delay;              /* delay in 0.5 ns */
 int bx_offset;          /* BX offset */
 int flt_mask;           /* mask for FLT enables */
}FCS_daughter
```

The parameters of the daughters cannot be read back. To be able to inform clients about the state of a daughter module the server keeps all daughter's states in a list of daughter structures during runtime. To be sure that this works every hardware driver call, which changes a daughter status, has to be followed by a change of the status in this list. This concept requires discipline and accuracy if the server is changed.

These structures are contained in the header file `FCS_logical.h`.

sources: `FCS_hardware.h`, `FCS_logical.h`, `FCS_hardware.o` **and** `FCS_logical.c`
object: `FCS_logical.o`

int **FCS_get_mot_par(FCS_mother *par);**
/* read mother parameters from hardware mother */
int **FCS_set_mot_par(FCS_mother *par);**
/* write mother parameters to hardware mother */

Parameters

par is the parameter struct for the mother

Return values

FCS_ok normal return

Discussion

FCS_set_mot_pargives a possibility to initialize the mother. A modification of a parameter can be done by first issuing FCS_get_mot_par - modifying the parameter - and then FCS_set_mot_par.

sources: `FCS_hardware.h`, `FCS_logical.h`, `FCS_hardware.o` **and** `FCS_logical.c`
object: `FCS_logical.o`

int **FCS_set_dau_par(unsigned char cluster, unsigned char daughter, FCS_daughter *par);**
/* write daughter parameters to daughter(s) and current values in server */

Parameters

cluster	specifies a cluster
daughter	specifies a daughter
par	is the parameter struct for this daughter

Return values

FCS_ok	normal return
FCS_is_fishy	unclassified error (to be confirmed)
FCS_unknown_daughter	invalid daughter

Discussion

This is a way to initialize the daughters.

9 Control and Monitoring (Client Functions)

The counters of the FCS mother are stored in the following structure which is defined in the header file `FCS_client.h`:

```
typedef struct{
 int FCS_fp_inhibit;          /* front penal inhibit */
 int FCS_vme_inhibit;         /* VME inhibit */
 int FCS_flt_of;              /* FLT time was overflowed */
 int FCS_bufof_inhibit;       /* readout buffer overflow */
 int FCS_bufdel_inhibit;      /* deadtime for buffer latency */
 int FCS_flt_inhibit;         /* sum of FLT inhibits */
 int FCS_BX_high;             /* upper 32 bit of bx counter */
 int FCS_BX_low;              /* lower 32 bit of bx counter */
}FCS_counters
```

The RPM error codes are defined in the header grr.h by J. Zweizig, described in [4].

sources: `FCS_server.h`, `FCS_client.h` **and FCS__start.c**
object: `libfcsc.a`

int **FCS__start**(**void**);
/* startup the client */

Parameters

none

Return values

any RPM error code

Discussion

This function locates the server and defines all FCS messages. It has to be called once at the beginning of any client's execution. The name of the server process is read from the environment variable $FCSSRVNAM.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_check_server.c**
object: `libfcsc.a`

int **FCS_check_server**(**void**);
/* check whether the server is running */

Parameters

none

Return values

any RPM error code

Discussion

Returns 0 if RPM messages can be sent to the server, some error code if not.

sources: `FCS_server.h`, `FCS_client.h`, **FCS_get_dau_par.c** and **FCS_set_dau_par.c**
object: `libfcsc.a`

int **FCS_get_dau_par(unsigned char cluster, unsigned char daughter, FCS_daughter *par);**
/* request current daughter settings stored in server */
int **FCS_set_dau_par(unsigned char cluster, unsigned char daughter, FCS_daughter *par);**
/* write daughter parameters to daughter(s) */

Parameters

cluster	bit mask specifying target cluster(s)
daughter	target daughter
par	is the parameter structure for this daughter

Return values

any RPM error code

Discussion

This is a way to initialize the daughters. And a client can modify a certain parameter by first issuing FCS_get_dau_par - then modifying the parameter - and then FCS_set_dau_par.

sources: `FCS_server.h`, `FCS_server.h`, **FCS_get_mot_par.c** **and** **FCS_set_mot_par.c**
object: `libfcsc.a`

int **FCS_get_mot_par(FCS_mother *par);**
/* read mother parameters from hardware mother */
int **FCS_set_mot_par(FCS_mother *par);**
/* write mother parameters to hardware mother */

Parameters

par is the parameter struct for the mother

Return values

any RPM error code

Discussion

FCS_set_mot_par provides a possibility to initialize the mother. A modification of a parameter can be done by first issuing FCS_get_mot_par modifying the parameter, and then FCS_set_mot_par.

FCS_get_counters FCS_set_counters

sources: `FCS_server.h`, `FCS_client.h`, **FCS_get_counters.c**, **FCS_set_counters.c**, **FCS_reset_counters**
object: `libfcsc.a`

int **FCS_get_counters**(**FCS_counters *counters**);
/* read mother counters from hardware mother */
int **FCS_set_counters**(**FCS_counters *counters**);
/* write mother counters to hardware mother */
int **FCS_reset_counters**();
/* clear all hardware mother counters with a pulse */

Parameters

counters are the logical counters of the mother

Return values

any RPM error code

Discussion

The statistics counters are read from or written to the mother module.

sources: `FCS_server.h`, `FCS_client.h`, **FCS_get_bx_count.c**, **FCS_set_bx_count.c**
object: `libfcsc.a`

int **FCS_get_bx_count(unsigned long *bx_high, unsigned long *bx_low);**
/* read BX counter from hardware mother */
int **FCS_set_bx_count(unsigned long *bx_high, unsigned long *bx_low);**
/* write BX counter to hardware mother */

Parameters

bx_high	the upper 32 bit of the 64 bit BX counter
bx_low	the lower 32 bit of the 64 bit BX counter

Return values

any RPM error code

Discussion

The BX counter is read from or written to the mother module. The value of the 64 bit counter is stored in two long (32 bit) integers.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_get_histogram.c**
object: `libfcsc.a`

int **FCS_get_histogram(unsigned long *histogram);**
/* get the FLT time histogram */

Parameters

histogram	is a pointer to the histogram (256*32 bit)

Return values

any RPM error code

Discussion

The time histogram shows the distribution of times the FLT needs to find its decision. Each bin corresponds to one BX cycle. The histogram reaches from 0 to 24.6 μs.

FCS_flt_enable FCS_flt_enable

sources: `FCS_server.h`, `FCS_client.h` **and FCS_flt_enable.c**
object: `libfcsc.a`

int **FCS_flt_enable**(**int enable**);
/* enable or disable FLT */

Parameters

enable if 0 the FLT gets disabled - else enabled.

Return values

any RPM error code

Discussion

The FLT enable bit in the mother control register is set according to the enable parameter.

FCS_rnd_enable FCS_rnd_enable

sources: `FCS_server.h`, `FCS_client.h` **and FCS_rnd_enable.c**
object: `libfcsc.a`

int **FCS_rnd_enable**(**int enable**);
/* enable or disable FLT */

Parameters

enable if 0 the random trigger gets disabled - else enabled.

Return values

any RPM error code

Discussion

The random trigger enable bit in the mother control register is set according to the enable parameter.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_vme_trigger.c**
object: `libfcsc.a`

int **FCS_vme_trigger(unsigned char physBx);**
/* send a trigger accept with specified physBx number */

Parameters

physBx number of HERA-bucket for the trigger

Return values

any RPM error code

Discussion

One trigger pulse is issued. It is given to the backplane by all daughters with the corresponding bit of their enable pattern set. The physBx-counter in the FCS hardware counts the HERA-buckets and is resetted once per HERA-round-trip-time (2,1 μs). This function allows to measure detector occupancies as a function of the bucket number to check synchronism of the detector components.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_do_handshake.c**
object: `libfcsc.a`

int **FCS_do_handshake**(**unsigned char number**);
/* do a readout-buffer handshake */

Parameters

number number of handshake pulses to be issued

Return values

any RPM error code

Discussion

Readout buffer handshakes are issued. Each handshake decreases the buffer count in the FCS mother by one. This function is used if a strict handshaking between FLTs and readout is used (overflow mechanism 2). This function is only available as a client function. The server uses FCS_handshake() instead.

sources: `FCS_server.h`, `FCS_client.h`and **FCS_do_reset.c**
object: `libfcsc.a`

int **FCS_do_reset(unsigned short cluster, unsigned char daughter);**
/* make specified daughter(s) issue a reset pulse */

Parameters

cluster	bit mask specifying target cluster(s)
daughter	the target daughter number

Return values

any RPM error code

Discussion

The reset at the daughters backplane is toggled for the specified daughters.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_test.c**
object: `libfcsc.a`

int **FCS_test(unsigned short cluster, unsigned char daughter, unsigned char pulseDelay, unsigned char triggerDelay, int frontpanel, int backplane);**
/* issue a test pulse on the specified daughters */

Parameters

cluster	bit mask specifying the target cluster(s)
daughter	the target daughter(s)
pulseDelay	time in 0.5 ns between a BX clock and the test pulse
triggerDelay	time in 96 ns between a test pulse and the belonging trigger
frontpanel	if 1 the testpulse goes to the daughter frontpanel
backplane	if 1 the testpulse goes to the daughter backplane

Return values

any RPM error message

Discussion

First the triggerDelay and the frontpanel/backplane flags are set to the selected daughter(s), then the FLT mask is set to enable the test pulse FLT. A test pulse command is issued at the selected daughters. The daughter hardware is issuing a FLT accept after this. Finally the test FLT is disabled again.

sources: `FCS_server.h`, `FCS_client.h`, **FCS_single.c** and **FCS_single_request.c**
object: `libfcsc.a`

int **FCS_single**(**unsgined short cluster, unsigned char daughter, unsigned char physBx);**
/* enable vme trigger - send one pulse - disable again */

int **FCS_single_request**(**unsgined short cluster, unsigned char daughter, unsigned char physBx);**
/* enable vme trigger - send one pulse - disable again
server sends an acknowledge message */

Parameters

cluster	bit mask specifying target cluster(s)
daughter	the target daughter(s)
physBx	bucket number for the VME triggers

Return values

any RPM error code

Discussion

The FLT mask of the selected daughter(s) is set to enable the VME trigger. One trigger pulse is issued. Finally the VME trigger is disabled again. For FCS_single_request the server sends back an acknowledge message.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_toggle.c**
object: `libfcsc.a`

int **FCS_toggle(unsgined short cluster, unsigned char daughter, unsigned char pattern);**
/* toggle selected strobes at selected daughters */

Parameters

cluster	bit mask specifying target cluster(s)
daughter	the target daughter(s)
pattern	strobe pattern to be toggled

Return values

any RPM error code

Discussion

Each bit corresponds to a strobe. Each strobe with a set bit gets toggled at the selected daughters. Bit 0 corresponds to reset. Called from a Client this call contacts the server. The function only exists as a client function. The server uses FCS_toggle_strobes.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_strobes.c**
object: `libfcsc.a`

int **FCS_strobes(unsgined short cluster, unsigned char daughter, unsigned char pattern);**
/* set strobe pattern at selected daughters */

Parameters

cluster	bit mask specifying target cluster(s)
daughter	the target daughter(s)
pattern	strobe pattern to be set

Return values

any RPM error code

Discussion

Each bit corresponds to a strobe. Each strobe with a set bit gets set — all others are cleared — at the selected daughters. Bit 0 corresponds to reset. The function only exists as a client function. The server uses FCS_set_strobes.

sources: `FCS_server.h`, `FCS_client.h` and **FCS_dau_mask.c**
object: `libfcsc.a`

int **FCS_dau_mask(unsgined short cluster, unsigned char daughter, unsigned char pattern);**
/* set enable pattern at selected daughters */

Parameters

cluster bit mask specifying target cluster(s)
daughter the target daughter(s)
pattern strobe pattern to be set

Return values

any RPM error code

Discussion

Each bit corresponds to a trigger type. Each trigger with a set bit gets enabled at the selected daughter(s). All others are disabled. The corresponding bits and trigger types are:

Bit	Trigger type
0x01	FLT
0x02	Random trigger
0x04	VME trigger
0x08	Testpulse trigger
0x10..0x80	LEMO trigger 1..4

The function only exists as a client function. The server uses FCS_mask_flt.

FCS_delay FCS_delay

sources: `FCS_server.h`, `FCS_client.h` **and FCS_delay.c**
object: `libfcsc.a`

int **FCS_delay(unsgined short cluster, unsigned char daughter, unsigned char offset, unsigned char delay);**
/* set offset and delay at selected daughters */

Parameters

cluster	bit mask specifying target cluster(s)
daughter	the target daughter(s)
offset	offset value to be set
delay	delay to be set

Return values

any RPM error code

Discussion

The function only exists as a client function contacting the server.
The server uses hardware driver functions directly.

10 Initialization (client functions and server subroutines)

In this section few calls are listed to set up the whole FCS or parts of it.

sources: `FCS_hardware.h`, `FCS_hardware.o`, `FCS_logical.h`, **and** `FCS_logical.c`
object: `FCS_logical.o`

int **FCS_init_mother**(void);
/* set the mother paramters to default */

Parameters

none

Return values

FCS_ok	normal return
FCS_is_fishy	unclassified error

Discussion

This function is a server subroutine. The mother is set to its default state. The default paramters are yet defined in `FCS_logical.h`. All counters and the time histogram are cleared. The SHARC link sender FIFO is cleared. All triggers are disabled at the mother.

sources: `FCS_hardware.h`, `FCS_hardware.o`, `FCS_logical.h`, **and** `FCS_logical.c`
object: `FCS_logical.o`

int **FCS_init_daughter(unsigned char cluster, unsigned char daughter, FCS_daughter state[NUM_CLUS][NUM_DAU]);**
/* set the specified daughter(s) to their default state */

Parameters

cluster	cluster number
daughter	daughter number
NUM_CLUS	number of clusters in the system
NUM_DAU	number of daughters in the system
state[NUM_CLUS][NUM_DAU]	In this array the server stores the states of all daughters. In this function the array is updated for the specified daughter(s).

Return values

FCS_ok	normal return
FCS_is_fishy	unclassified error
FCS_unknown_daughter	specified daughter does not exist

Discussion

This function is a server subroutine. The specified daughter(s) are set to their default states and the server's state-array is updated. Yet the defaults are read from the file FCS_daughter_defaults.dat. Later this is to be adapted to the DAQ configuration data base.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_reload_daughter.c**
object: `libfcsc.a`

int **FCS_reload_daughter(unsigned int cluster, unsigned char daughter);**
/* reload the settings stored in server to hardware daughter(s) */

Parameters

cluster	the target cluster mask
daughter	the target daughter(s)

Return values

any RPM error code

Discussion

The server keeps track of all the settings it commits to daughters. After a power failure of a FED crate the status of the daughter can be recovered in the way that the server commits the settings it assumes the daughter to have.

sources: `FCS_server.h`, `FCS_client.h` **and FCS_init.c**
object: `libfcsc.a`

int **FCS_init(unsigned short cluster, unsigned char daughter, int daughter_flag, int mother_flag);**
/* initialize the mother or selected daughter(s) */

Parameters

cluster	the target cluster mask
daughter	the target daughter(s)
	both parameters have to be given always,
	but are ignored if daughter_flag=0
daughter_flag	if 1, the daughter(s) get(s) initialized
mother_flag	if 1 the mother gets initialized

Return values

any RPM error code

Discussion

The mother will be initialized if the corresponding flag is set. The selected daughter(s) will be initialized if that flag is set. Initialization means that the server executes one or both of the functions FCS_init_mother or FCS_init_daughter described above.

FCS_exit FCS_exit

sources: `FCS_client.h`, `FCS_client.h` **and FCS_exit.c**
object: `libfcsc.a`

int **FCS_exit**(void);
/* exit FCS processing */

Parameters

none

Return values

any RPM error code

Discussion

This is a client function which shuts down the server. When this function is executed the server releases all occupied resources. BX distribution and all Triggers are disabled on the FCS hardware and all allocated memory on the CETIA is freed before the server stops running.

11 The server

To ensure proper interlocking and enable resource management only one process should run locally on the CPU in the FCS master VME crate and access the FCS hardware directly. This process is called the FCS_server. The server process initializes the FCS during startup. Then it waits for messages from clients and every time it receives a message the server interpretes the message and executes a code sequence depending on the type of the message (see section 2).

Changes of the system configuration are logged to a file. The name of the logfile consists of the RPM_PROCESS_NAME, year, month and day. A new file is opened automatically each day. In case of a server restart an existing file is continued and the restart is logged. The log files are written in the directory where the executable is located.

For initialization the server needs a file containing default settings for the FCS daughter modules. This can either be read from a central place which is currently `/home/herab/dev/etc/` in the old Cetia file system. Or the file can be specified locally for test setups.

A print-flag which enables and disables printf-commands to stdout during runtime can be set using the `parlist` program. And a single-trigger-message counter is readable through `parlist`.

Some command line parameters are available when starting the server:

```
**********************************************************************

Available options for the FCS_server:

 -help                Print this screen

 -p                   Enable printfs for debugging

 -850                 Reset FLT counter after every 850 events

 -Defaults:{filename} Specify daughter default parameter file.
**********************************************************************
```

12 A simple client example

This program is dealing with all TDCs of outer tracker TC chambers. To run it on a arbitray CPU the program has to be linked with `libfcsc.a`. In this case a running `FCS_server` is required for execution.

```
#include "FCS_client.h"             /* constants and prototypes */
#include "FCS_server.h"             /* RPM message definitions */
#define TDC_TEST 0x4                /* an example strobe pattern */

main()
{
int err;
FCS_daughter par;                   /* create parameter structure; */

err = FCS_init(OTR_TC1 & OTR_TC2, ALL_DAUGHTERS, 0, 1);
                                    /* initialize daughters */
err = FCS_strobes (OTR_TC1 & OTR_TC2, ALL_DAUGHTERS, TDC_TEST);
                                    /* set strobe pattern */
err = FCS_dau_mask (OTR_TC1 & OTR_TC2, ALL_DAUGHTERS, TP_TRG);
                                    /* enable test pulse trigger */
err = FCS_test(OTR_TC1 & OTR_TC2, ALL_DAUGHTERS, 50, 100, 0, 1);
                                    /* issue a test pulse 25ns after a BX
                                    to the backplane and send a trigger 100
                                    cycles afterwards */
err = FCS_init(OTR_TC1 & OTR_TC2, ALL_DAUGHTERS, 0, 1);
                                    /* re-initialize these modules */
if (!err) printf("everything ok");
return err;
}
```

13 FCS_send

This is a command line interpreting client program which offers possibilities to send several commands to the server by hand. FCS_send can be used on any machine. The executable is in the bin-directory of the online library. The source code can also be regarded as an example for writing a client.

FCS_send provides the following commands:

option	meaning
-i -m	initialize FCS mother
-i -d *clustermask daughter*	initialize *daughter*(s) in *cluster*(s),
-r *clustermask daughter*	reset *daughter*(s) in *cluster*(s),
-e 0	disable FLT on mother
-e 1	enable FLT on mother
-e 2 *clustermask daughter pattern*	set enable *pattern* on *cluster(s)*, *daughter*(s)
-rnd 0	disable random trigger
-rnd 1	enable random trigger
-p 0 *physBx*	send VME trigger
-p 1 *[number]*	send *number* VME handshake pulses
-s 0 *clustermask daughter pattern*	toggle strobe *pattern* on *cluster*(s), *daughter*(s)
-s 1 *clustermask daughter pattern*	set strobe *pattern* on *cluster*(s), *daughter*(s)
-d *clustermask daughter offset delay*	set *offset* and *delay* for *cluster*(s), *daughter*(s)
-t [-f] [-b] *clustermask daughter* *pulse-delay, trigger-delay*	send test pulse with *pulse-delay* to front or back *cluster*(s), *daughter*(s) followed by a trigger *trigger-delay* bx cycles later
-o *clustermask daughter physBx*	enable VME trigger on *cluster*(s), *daughter*(s); send one trigger; disable again
-w *clustermask daughter*	the same, but wait for a reply from the server
-b -g	get current value of the BX counter
-b -s *high32bit low32bit*	set the BX counter
-c	get current values of the statistics counters
-cr	reset FLT and statistics counters
-x	shutdown FCS, stop signals to all subsystems

The clustermask is a bit mask specifying one or more clusters. It can be given as decimal or hexadecimal number.

This style also provides the possibility of writing script files which do a certain sequence of commands. One example is given here:

Command	action
`FCS_send -e 2 2 3 4`	enables the Fake trigger on daughter 3 in cluster specified with mask 2
`FCS_send -d 2 3 5 70`	sets offset=5 and delay=40 to daughter 3 in cluster specified with mask 2
`FCS_send -p 0`	initiates a trigger
`FCS_send -s 0 2 3 1`	reset pulse to daughter 3 in cluster specified with mask 2
`FCS_send -p 0`	initiates another trigger

14 FCS_setMother

This program allows to change settings of the FCS mother parameters from a command line. For each parameter one wants to change the program has to be called once. The program does a FCS_get_mot_par, changes one parameter and a FCS_set_mot_par afterwards. Hence all other parameters remain unchanged. Possible calls are:

`FCS_setMother` -offset *offset*;
`FCS_setMother` -overflow *overflow-scheme*;
`FCS_setMother` -latest *latest-accept*;
`FCS_setMother` -depth *fifo-depth*;
`FCS_setMother` -duration *readout-duration*;
`FCS_setMother` -delay *buffer-delay*;
`FCS_setMother` -random *random-factor*;

15 Graphical User Interface

The graphical user interface (GUI) provides simple, more or less self-explaining, possibilities to configure the FCS, monitor settings, send test data and so on. It provides access to nearly all FCS features and both experts and non-experts can use it.

The GUI uses functions from the client library libfcsc.a to contact the server via the HERA–*B* message system RPM [4].

The GUI consists of a shell called fcssh and a Tcl/Tk script fcspanel wich has to be executed in the fcssh. The fcssh has to be linked with the object file hfcs_tcl.o which contains interface functions from Tcl to client functions from the library libfcsc.a. The interface functions will not be discussed in detail because these are just a lot of simple functions interfacing command line arguments from the Tcl script to parameters given to client functions. Additionally the following libraries have to be linked: grr, msg, util, tk, tcl, X11, bsd and m. A more detailed description which might be interesting for future developers can be found in [5].

15.1 Main window

The main window (fig. 1) povides a menuboard and some buttons dealing with the whole system. The menuboard provides four menus:

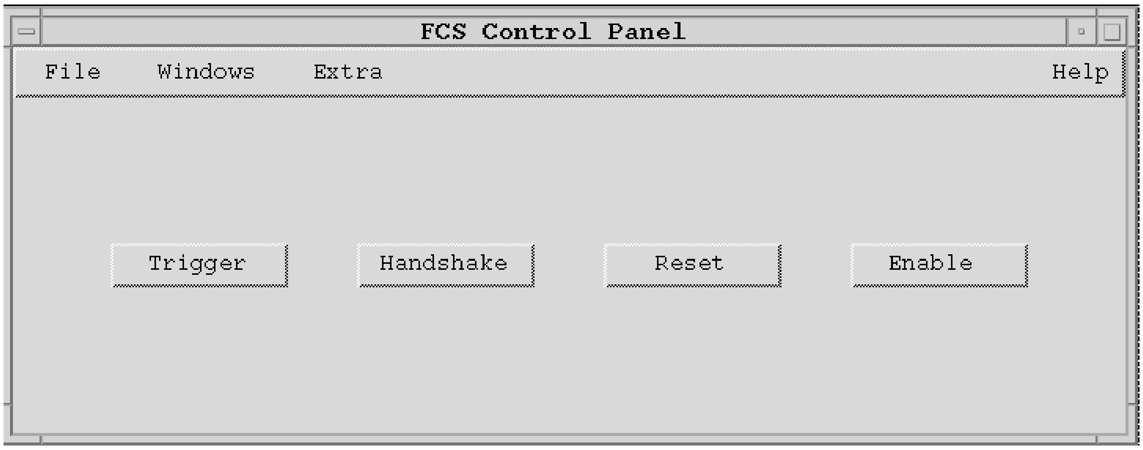

Figure 1: The main panel of the GUI.

File for the time being just provides an **exit** item.

Windows provides access to several subwindows. Each of them is good for dealing with certain parts of the Fast control system. These windows will be discussed in the following sections.

Special hides the possibility to send a kill-message to the server. Beware that there should be just one server running which could support more than one users.

Help gives information how to get context-sensitive help. The help-function for each button on the panels is quite clumsy due to technical reasons. One has to place the mouse on the button one is interested in, press the right mouse button and hold it while dragging the pointer away from the button.

The buttons directly on the main panel are:

Trigger sends a VME trigger to the FCS. If the mother accepts it, it is given to the FEDs by all daughters with the corresponding enable bit set.

Handshake sends a VME handshake which decreases the internal FIFO-load counter on the mother by one. For further details see Readout buffer overflow mechanism 2, described in [1].

Reset sends a reset to all FEDs.

enable/disable enables/disables trigger processing in the mother.

15.2 Mother parameter window

This window provides a way to initialize, set and monitor the overflow scheme and all time registers on the mother.

15.3 Counter window

The counter window is designed to provide a way to reset and read the FLT counters and the different FLT inhibit counters as well as to read the BX counter.

The window also shows 'set'-buttons which are meant to set the counters to arbitrary values. But this is not supported by the hardware. This feature would be useful for the BX counter at least and might be implemented in a hardware redesign.

15.4 Subdetector window

This window (fig. 2) is the most complex one and the most interesting for non-DAQ-users who want to provide signals to their FEDs. The GUI provides the possibility to send different signals to different FCS daughter modules.

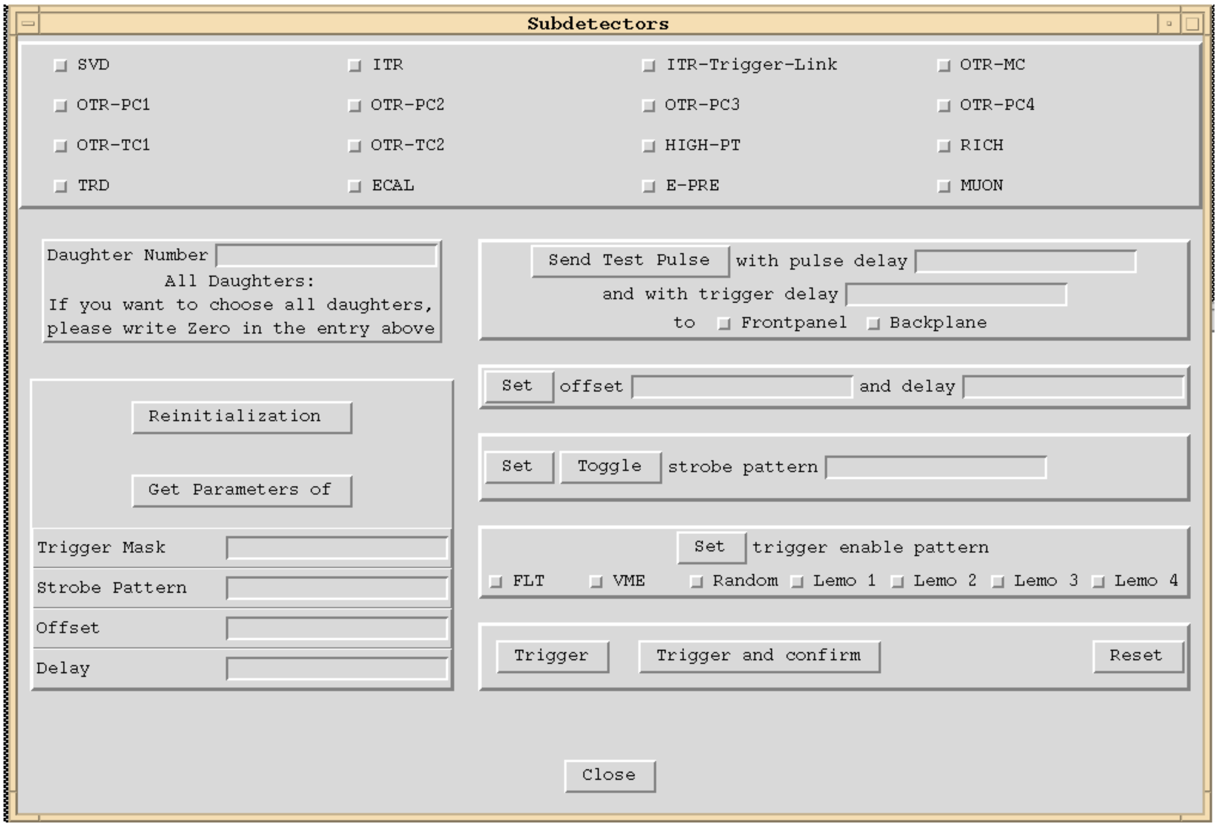

Figure 2: The panel for dealing with the daughter boards.

Addressing

At the top of the window the clusters to be addressed can be selected.

A bit mask is used for specifying the clusters. This bit mask is decoded in the FCS_server within the function FCS_cluster_ids (section 7). The mapping between cluster numbers and detectors is defined in the header FCS_logical.h.

On the left-hand side of the window there is a field to specify a daughter number inside a cluster. If more than one cluster is selected, this field is ignored and all daughters in the selected clusters are addressed. If exactly one cluster is specified, this field is read and interpreted. "0" specifies all daughters in this cluster, positive integers are used to select exactly one daughter.

Commands

Pressing the "reinitialization" button causes a reinitialization of all selected daughters. Default settings are read from the configuration data base (currently the ASCII-file FCS_daughter_defaults.dat).

If exactly one daughter is selected, the field on the bottom of the right hand side can be used to read the status of this daughter. The status is not read from the hardware, but the server keeps all the settings it commits to the hardware and sends the status the daughter is assumed to have.

On the right hand side of the window there are five fields which are dedicated to committing settings and sending pulses to the daughter modules. They all work independent of the number of selected daughters.

Test pulse: Test pulses with editable timing settings can be issued to the front panel or the back plane of the daughter modules.

Timing: The BX number offset (in integers) and BX clock delay (in units of 0.5 ns) can be set in this field.

Strobes: Strobe patterns can be set and toggled.

Trigger enable pattern: The trigger types to be enabled can be selected and the setting can be submitted to the modules.

Pulses: Three buttons are located in this field. A reset pulse can be sent to the selected daughter(s). The trigger buttons issue a series of commands: First the VME trigger is enabled for the selected daughter(s), then a trigger is sent and finally the VME trigger is disabled again. With the "trigger and confirm"-button the server sends a confirmation after the commands have been issued (see descriptions of the client functions FCS_single and FCS_single_request).

15.5 FLT timing histogram

When this window is selected via the menu a request is sent to the server. The server reads the current FLT timing histogram from the FCS mother board. This histogram is displayed in a window opened afterwards. The histogram shows on a semi-logarithmic scale the distribution of the time the FLT needed to accept an event in units of 96 ns. Actually the display shows $\log_{10}|entry{+}1|$ which does nearly not differ for positive integers $\gg 1$, but prevents the code from giving invalid parameters to the logarithm function.

Acknowledgements

The Software described here and so this document are based on the FCS software specification by D. Ressing [1].

V. Bartsch worked on implementing the graphical user interface during her summer study at DESY.

References

[1] D. Ressing: "FCS specifications", **HERA–*B*** note, not registered

[2] D. Ressing: "FCS software", **HERA–*B*** note, not registered

[3] http://www.desy.de/ftp/pub/herab/daq/projects/FCS.html,
http://www.desy.de/ftp/pub/herab/daq/projects/FCSSRV.html

[4] J. Zweizig: Proposal for a HERA–*B* Message Handler Interface,
HERA–*B* note, not registered

[5] V. Bartsch: Documentation of the script fcspanel