

# DCT upgrade test-stand and DAQ.

Gérald Grenier\*      Xuedong Chai  
The University of Iowa

4th April 2003

## Abstract

This document summarizes the various aspects of the DCT upgrade online software including test-stand diagnostic tools and Feature EXtraction (FEX).

## Contents

<b>I</b>	<b>Basic tests and algorithm tests</b>	<b>2</b>
1	The problem	2
2	Test-stand software overview	2
3	User interactions	4
3.1	Interactive session . . . . .	4
3.2	GUI . . . . .	4
3.3	Script . . . . .	5
3.4	Running a Doctor . . . . .	6
4	More on doctors	6
4.1	Playing simulated data . . . . .	6
4.2	Parallel testing . . . . .	6
4.3	DCT to DCT interface board tests . . . . .	7
<b>II</b>	<b>DAQ</b>	<b>7</b>
5	The BaBar DAQ system	7
6	The new TC	9
7	Parallel commissioning	10
8	Dealing with DAQ data format changes	11

---

\*Contact person

<b>III Calibration-based test</b>	<b>12</b>
9 Calibration with doctors : testing DAQ	12
10 DCH to TSF interface tests	12

## Part I

# Basic tests and algorithm tests

## 1 The problem

All the tests will be done within the BaBar Dataflow system. This system allows to interact with the Front-End Electronics, here the trigger boards. The communication with the boards is done through two optical fibers named C-link and D-link. The C-link brings commands to the boards which answer through the D-link. The links connect the FEE with a Read-Out Module (ROM). The ROM is a board equipped with a power PC processor running under a VxWorks operating system. The communication protocol used on the C- and D-links is called the Fast Control. The test-stand software has been designed to use the Fast Control protocol to perform all the board tests. Almost all tests can be described by the following steps:

1. Write something in some board memory.
2. Run the test, usually play the written data through the board.
3. Read the result in an other board memory and compare with expectation.

To speed up the test process and to reduce the amount of manual operations it needs, the ROM CPU is used to perform the comparison with the expectation.

## 2 Test-stand software overview

The figure 1 shows the various components of the test-stand software. The software is organized in layers with two fundamental layers. The first one, the Fast Control Interface implements the Fast Control protocol. The Fast Control protocol uses two links<sup>1</sup> called A and B. Each link addresses up to 16 trigger boards (A trigger crate can contain up to 16 boards and a ROM can communicate with up to 2 crates). It is possible to set which boards are addressed by providing a 32-bits mask, the lower 16-bits on that mask correspond to the crate connected to link A while the upper bits are associated with the crate connected to link B. To include a board in the Fast Control command emission, the mask bit corresponding to the board's slot number should be set to 1. All the fast control commands implemented in the software allow to specify that mask value and can communicate with up to 32 boards. For the upper layers of the software, the Fast Control interface provides an interface to read and write trigger board memories. The interface is independent of the actual Fast Control command used to read or write (by block, by word, ...). Hence, the read/write Fast Control commands can be changed dynamically.

---

<sup>1</sup>here a link is a pair C-link, D-link.

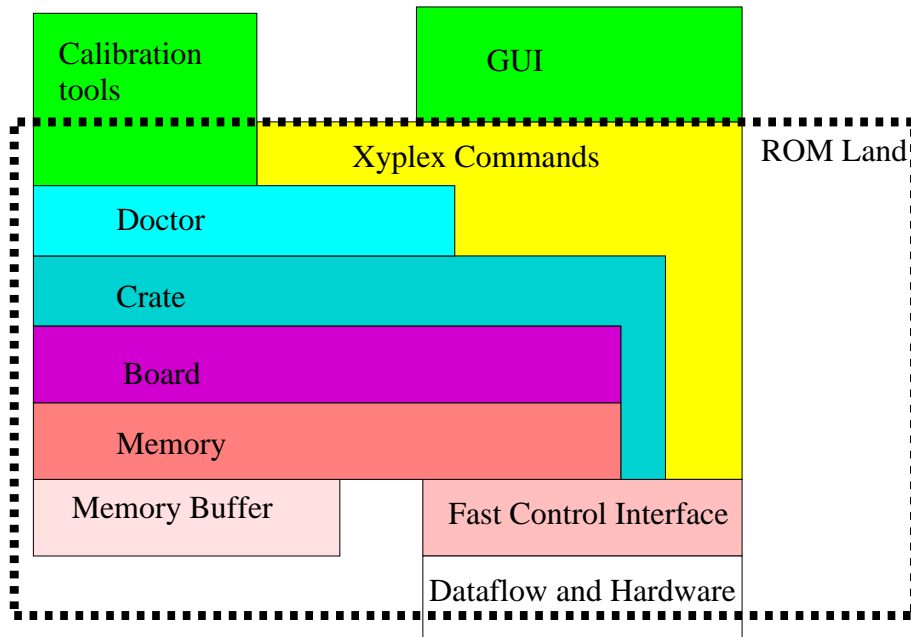


Figure 1: The test-stand software is organized in a superposition of layers. Each layer represents the functionalities associated with the correspondent hardware level. Everything which is inside the dotted region runs in the ROM.

The second fundamental layer is the Memory Buffer. A Memory Buffer is an image of a real trigger board memory. It is used to store the expected output needed for the comparison step of a test. The Memory Buffer functionality has been extended to serve as an intermediate storage for all transfers between the trigger board's memories and the outer world<sup>2</sup>. The Memory Buffer layer contains (and needs) all the information concerning the trigger board memories (block address, address ranges, masked bits, ...).

The lowest non-fundamental layer is the Memory layer. It connects the Fast Control Interface with the Memory Buffer. This layer provides code to transfer the content of a Memory Buffer into the corresponding trigger board memory, to transfer the content of a trigger board memory into the corresponding Memory Buffer and to compare the content of the Memory Buffer with this of the trigger board memory. All transfers have been optimized so as to minimize the number of Fast Control commands sent down through the C-link.

The level just above is the Board level. A Board describes a trigger board which is seen as a set of memories with a set of available read and write Fast Control commands to exchange data with those memories. The board provides names for each of its components, the memories are called "input", "output", ..., the read Fast Control commands are "one word", "incremental one word", "block". All references to block addresses, addresses, Fast Control Opcode are hidden by

<sup>2</sup>Here outer world means everything outside the ROM and the trigger boards

this level. It provides also names for the boards : “tsf”, “zpd”, ...

The next upper level is the Crate level. This level provides the interface to drive up to 32 boards. The higher level of the system is the Doctor level. A doctor is a class that performs a test using the Crate interface. A test is divided in 3 operations: init, test and check. Each of these 3 operations accept a 32-bit word parameter for configuration. The Doctor level provides also a mechanism for choosing a particular doctor by mapping doctors with a 32-bit integers.

## 3 User interactions

### 3.1 Interactive session

In order to perform a test, a user needs to be able to communicate with the software. This is provided with a set of Xyplex commands. Xyplex commands are a set of global C++ functions. It is possible to open a “telnet” session on the ROM CPU by using the Xyplex protocol. Once connected, the VxWorks environment allows you to load a library and interactively call any global function contained in the library. The commands available in an interactive session are split into 3 categories: a first set of commands runs directly the Fast Control Interface without using The Memory Buffer layer and above layers of the software. Those commands are mainly for debugging purposes and usually start with “dct...”. A second set of commands allows to interact with the Crate layer and all the underlying layers. Those are the main commands used in an interactive session. Those commands start with “crate...”. A final set of commands deals with doctors. It allows to select a doctor and run the associated test. Those commands start with “doctor...”.

### 3.2 GUI

A Graphic User Interface, TestandGUI implementing the test-stand software has been developed. TestandGUI is a Tcl/Tk application, which lets trigger people run trigger lab-test on many flavors of Unix. By using pipe technology in UNIX, it opens a telnet session, sends Xyplex commands to the ROM, then reads back the information and filters it, displaying the output inside the GUI windows. With the GUI’s help, the tester can do simple, or routine test quickly and easily by button clicking, without having to remember Xyplex command syntax, and new people can begin test quickly without knowing any test-stand software.

The GUI functionalities:

1. Supplying interfaces for different set of Xyplex command: ”dct...” command, ”crate...” command and ”doctor...” command.
2. Supplying interfaces for different trigger board: GLT, ZPD,... Figure 2 2 illustrates the window interface to the GLT board.
3. CSR status updating: Every time the CSR status is changed using the GUI, this is shown on the GUI.
4. Dynamic help function: when mouse moving on some widgets, the operation you can do or the parameters you need to input will be shown in a message widget.
5. Button addition on the fly: For the doctor window, new doctor buttons can be created without editing the Tcl/Tk source files.

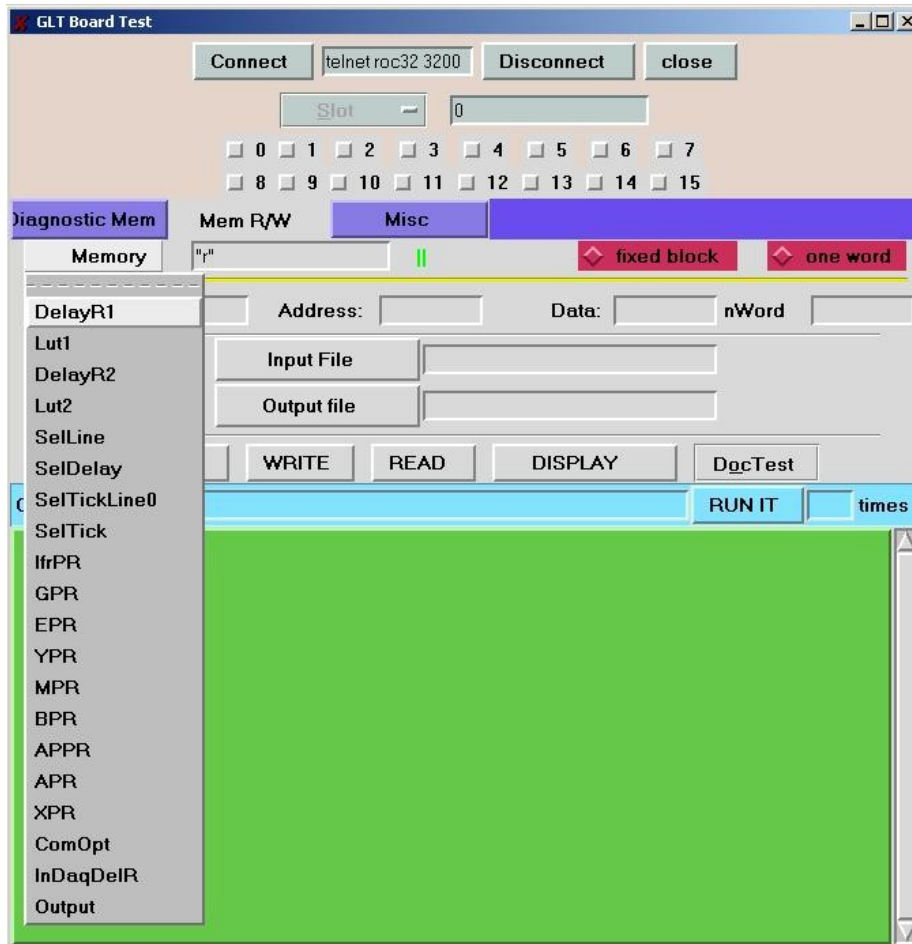


Figure 2: The GUI window for the GLT board.

6. Bookkeeping: commands sent and the resulting output can be saved in a file. The user can activate the bookkeeping at any time.

### 3.3 Script

A script file in a VxWorks environment is a file containing Xyplex commands on each line. Below is an example of such VxWorks script using the Crate level Xyplex commands that loads a file in the TSF input memory, reads it back and checks that what is read is what has been written:

- cratetalkTo "tsf"
- cratesetNoRun
- cratesetMemory "input"
- cratesetAsciiFileLoader "filename.txt"
- cratesetMemoryWriter "input","block"

- crategoMemory 0,0
- crateloadMemoryBuffers
- cratewriteOnBoard
- cratesetMemoryReader “input”,”block”
- cratecompare

The VxWorks script interpreter doesn't allow to have loops in the scripts. However, the GUI provides a mechanism to repeatedly run a VxWorks script.

### 3.4 Running a Doctor

A doctor is a piece of code that performs a test. A set of Xyplex commands allows to select a doctor and run each of the steps of the test independently. Another command allows to repeatedly run the same doctor with the same parameters. Finally, a last command allows to use a flat file to run doctors: each line of the file contains the 32-bit word that selects the doctor followed by the three 32-bit parameters used by the doctor.

## 4 More on doctors

### 4.1 Playing simulated data

Doctors can be dedicated to a particular test. For example, there is a ZPD megabus transmission doctor which writes random numbers in the ZPD input memory, runs a playback of the ZPD with the input memory playing and the megabus memory recording and checks at the end that what is in the ZPD megabus memory is what has been written in the input memory.

Other doctors are not board specific. There is 2 such doctors. One tests that a memory can be written and read back without error. The second one is able to play simulated data through the board. Simulated data are provided in a file which also contains informations on which memory of which boards those data are for. It also describes which data are input data to write and which one are output data to compare. The doctor parses the file and performs the requested tests. The file containing the simulated data is produced by another program. This program reads simulated data, put them in a Memory Buffer and then dumps the Memory Buffer in the file. As the Memory Buffer layer is used both for the production of the file and for its parsing in the ROM, the coherence is ensured.

The algorithm tests request that the boards are correctly configured. To achieve that, one can either create a Xyplex command that configures the board and run it before starting the test or the configuration can be put in the input flat file of simulated data as a portion of the input data of the first event.

### 4.2 Parallel testing

All doctors can performs their test on any number of identical boards. A typical test which for one board is sequenced as:

1. load once input data in Memory Buffer.
2. write once Memory Buffer to the board memory.

3. run a board playback.
4. load once output data in Memory Buffer.
5. compare once output data with board memory.

becomes for n boards:

1. load once input data in Memory Buffer.
2. write once Memory Buffer to the board memory.
3. run a board playback.
4. load once output data in Memory Buffer.
5. compare n times output data with board memory.

Only the comparison step needs to loop on the boards to be done. All other steps use resources (time and memory) which are independent of the number of boards tested.

### **4.3 DCT to DCT interface board tests**

This kind of tests is identical to an algorithm test: write an input in some memory, do a playback, read back and compare. The Crate layer is able to handle different kinds of boards at the same time. A DCT to DCT (or DCT to GLT) interface boards tests can be done either with a dedicated doctor or by using the doctor which play simulated data through the boards.

## **Part II**

# **DAQ**

## **5 The BaBar DAQ system**

The DAQ system is composed of a run controller to drive the system, all the ROMs and their FEE and an event level which collects the data of all ROMs. To drive such an heterogeneous system, a Finite State Machine (FSM) is used. All processors in the system share a common Finite State Machine. The run controller initiates transitions to go from one state to another. Upon reception of a transition each ROM execute the necessary action needed to change its FSM state. The figure 3 shows the BaBar DAQ Finite State Machine. Among all available transitions, two have an importance for the DAQ itself: the configure transition and the L1Accept transition. A transition can transport data between different CPUs. Those data have to be in a Tagged Container (TC). Among all TCs, the most common types are eXtended Tagged Container (XTC) in which data can be added as they are traveling from the run controller to the event level. The configure transition comes with a parameter called a key which allows to retrieve TCs containing the configuration data. The L1Accept transition should process the FEE L1Accept data and store them in an XTC attached to the transition. Each ROM so produces a ModuleTC which contains a sectionTC for each occupied slots of its connected crate. The old DCT, is producing at each L1Accept, two L1DTsfModuleTC (TSFX and TSFY crate) and one L1DBltPtdModuleTC.

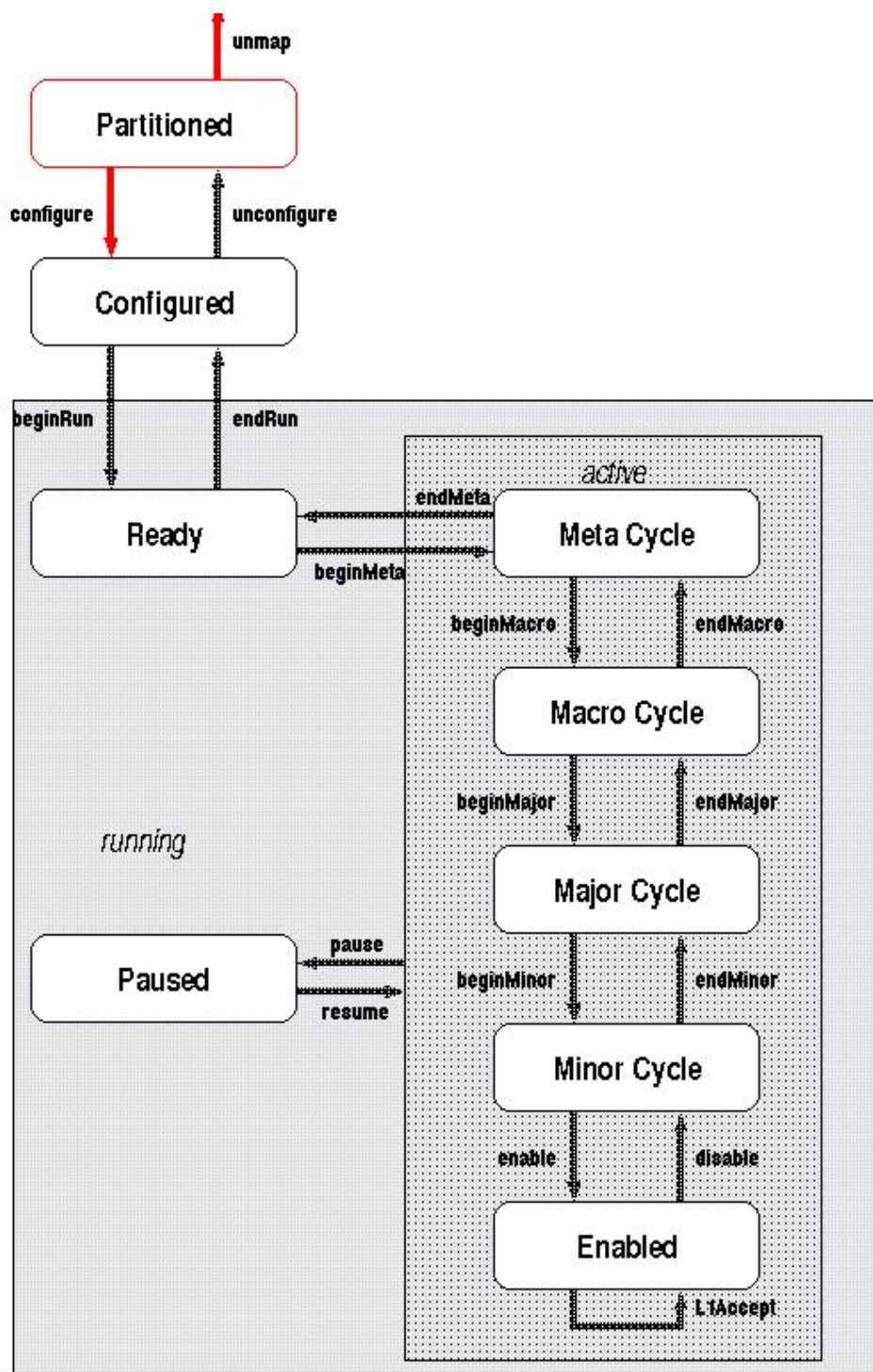


Figure 3: The BaBar DAQ Finite State Machine.

LSB										→										MSB									
1	1	trigger tag[0:4]								R	trigger time counter[0:4]								bf[1:0]	R									
CSR1 word[15:0]																													
input TSF segment mask(T=0)[0:15]																													
input TSF segment mask(T=0)[16:31]																													
...																													
input TSF segment mask(T=0)[144:152]										R	R	R	R	R	R	R	R												
input TSF segment mask(T=1)[0:15]																													
...																													
input TSF segment mask(T=7)[144:152]										R	R	R	R	R	R	R	R												
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R										
z <sub>0</sub> (T=0,F=0)[0:7]										Δz <sub>0</sub> (T=0,F=0)[0:3]					R	R	R	R											
track curvature(T=0,F=0)[0:7]										tan λ(T=0,F=0)[0:7]																			
z <sub>0</sub> (T=0,F=1)[0:7]										Δz <sub>0</sub> (T=0,F=1)[0:3]					R	R	R	R											
track curvature(T=0,F=1)[0:7]										tan λ(T=0,F=1)[0:7]																			
...																													
z <sub>0</sub> (T=0,F=11)[0:7]										Δz <sub>0</sub> (T=0,F=11)[0:3]					R	R	R	R											
track curvature(T=0,F=11)[0:7]										tan λ(T=0,F=11)[0:7]																			
output to GLT(T=0)[0:7]										R	R	R	R	R	R	R	R												
z <sub>0</sub> (T=1,F=0)[0:7]										Δz <sub>0</sub> (T=1,F=0)[0:3]					R	R	R	R											
track curvature(T=1,F=0)[0:7]										tan λ(T=1,F=0)[0:7]																			
...																													
z <sub>0</sub> (T=7,F=11)[0:7]										Δz <sub>0</sub> (T=7,F=11)[0:3]					R	R	R	R											
track curvature(T=7,F=11)[0:7]										tan λ(T=7,F=11)[0:7]																			
output to GLT(7)[0:7]										R	R	R	R	R	R	R	R												

Table 1: ZPD raw DAQ format. bf stands for buffer number and like CSR has its MSB first. T is the Clock4 tick and F the fitted track number.

## 6 The new TC

For the new TSF, the configuration parameters and the DAQ data are expected to be identical to the old TSF ones. Hence no new TCs are expected to be necessary. The only difference will be in the way the configuration data are written to the boards.

For the ZPDs which will run with the BLT, a L1DBltZpdModuleTC is needed. If the L1DBLTSectionTC can be kept, a new L1DZpdSectionTC has to be written.

The data sent by the ZPD at each L1Accept is shown in table 1. The input mask will be send only during the commissioning phase. Once the system is proved to work, there is no need to send this information which is redundant with the TSF data.

Although, this is not yet fixed, the data in the L1DZpdSectionTC will be close to what is shown in table 2.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f			
LSB							→									MSB		
1	1	trigger tag[0:4]					R	trigger time counter[0:4]				bf[1:0]		R				
DAQ mode		DCH sector		L1DZpdSectionTC version number														
input TSF segment mask(T=0)[0:15]																		
input TSF segment mask(T=0)[16:31]																		
...																		
input TSF segment mask(T=0)[144:152]									R	R	R	R	R	R	R			
input TSF segment mask(T=1)[0:15]																		
...																		
input TSF segment mask(T=7)[144:152]									R	R	R	R	R	R	R			
clock tick $T_1$ [0:2]		n = # of fitted tracks[0:3]			R	output to GLT( $T_1$ )[0:7]												
$z_0(T_1, F_1)$ [0:7]						$\Delta z_0(T_1, F_1)$ [0:3]			track number $F_1$ [0:3]									
track curvature( $T_1, F_1$ )[0:7]						$\tan \lambda(T_1, F_1)$ [0:7]												
...																		
$z_0(T_1, F_n)$ [0:7]						$\Delta z_0(T_1, F_n)$ [0:3]			track number $F_n$ [0:3]									
track curvature( $T_1, F_n$ )[0:7]						$\tan \lambda(T_1, F_n)$ [0:7]												
...																		
clock tick $T_m$ [0:2]		n = # of fitted tracks[0:3]			R	output to GLT( $T_m$ )[0:7]												
$z_0(T_m, F_1)$ [0:7]						$\Delta z_0(T_m, F_1)$ [0:3]			track number $F_1$ [0:3]									
track curvature( $T_m, F_1$ )[0:7]						$\tan \lambda(T_m, F_1)$ [0:7]												
...																		
$z_0(T_m, F_n)$ [0:7]						$\Delta z_0(T_m, F_n)$ [0:3]			track number $F_n$ [0:3]									
track curvature( $T_m, F_n$ )[0:7]						$\tan \lambda(T_m, F_n)$ [0:7]												

Table 2: ZPD FEXed data: Empty Fitted tracks and decision module data for clock tick with no fitted tracks and a zero output to GLT are discarded.

## 7 Parallel commissioning

The new DCT will be commissioned by running in parallel with the old DCT. For the TSF, it implies that there will be two TC in the DAQ data containing TSFX segments. To avoid the run-time duplication of the ModuleTC, each ModuleTC class will exist in two versions: a production version which is looked for and a parasite version which is ignored. During the commissioning, the new trigger will produce parasite version of its TCs and once commissioned, it will produce production version.

An other aspect of the commissioning phase is that the upgrade DCT will be progressively deployed as boards are produced. It is therefore necessary not to code the hardware position of each slots. To solve this problem, an automatic board detector and identifier has been implemented. The parasite trigger can then run on what has been detected.

The figure 4 shows the generic set of FSM actions that will be installed for the new DCT. A Map object contains the information on which boards should be installed to which slots and to which DCH sector each of them corresponds. This map is filled by a BaseMapFiller object which uses both configuration data and

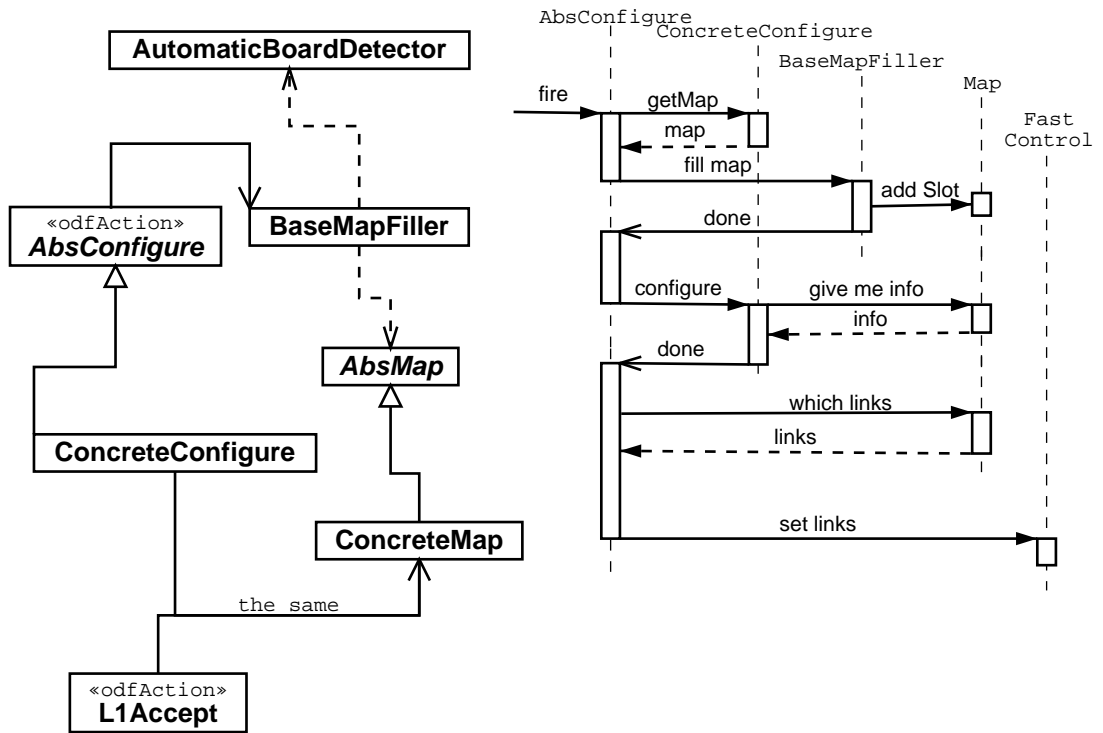


Figure 4: Left: UML class relationships for the DCT upgrade FSM actions. Right: Collaborations between classes during the configure transition to internally configure the trigger boards and to set the C- and D- links to only the present boards.

the automatic board detector to fill the map correctly. Board specific information are installed in a subclass of the Map class. A generic Abstract configure action is responsible for calling a BaseMapFiller to fill the Map. Board specific configurations are handled by a specific subclass of the abstract configure action. The L1Accept action in turn uses the Map the configure action has filled.

## 8 Dealing with DAQ data format changes

A first evolution in the DAQ data format is already planned: the input mask will disappear. Also, experience gained during the commissioning might lead to some changes in the information sent by the ZPD. Any change in the ZPD DAQ data will be reflected in the corresponding L1DZpdSectionTC. However, any data taken should remain readable. When changing the L1DZpdSectionTC format, one has to make sure that the old format is still readable. To ease the change of the DAQ format, the operations that read the L1DZpdSectionTC will be implemented in a separate class hierarchy. All L1DZpdSectionTC readers will be derived from a common base class and the L1DZpdSectionTC contains a version number and will be responsible to provide the L1DZpdSectionTC reader corresponding to its version

number. All the offline code will be developed with the L1DZpdSectionTC reader base class and hence all changes in the DAQ format will be completely transparent in the offline world.

## **Part III**

# **Calibration-based test**

The calibration mechanism provided by the Dataflow system runs in the DAQ Finite State Machine but allows to have 3 Fast Control commands sent to the boards during the L1Accept transition. The set of Fast Control commands that can be sent during the calibration L1Accept is very limited but includes startPlayback and L1Accept.

## **9 Calibration with doctors : testing DAQ**

A calibration Finite State Machine has been implemented. This FSM drives both a doctor and a normal DAQ code. With this FSM, one can use all the test-stand software functionalities for loading trigger board memories and use the normal DAQ code to both configure the boards during the configure transition and perform the Feature EXtraction during the L1Accept transition. Hence, this authorizes to take simulated data with the trigger boards. The special configuration TC needed for the calibration can be constructed from the flat file described in section 3.4.

The calibration FSM selects the doctor at the beginMacro transition. It runs the doctor's init function at beginMajor, the doctor's test function at endMinor and the doctor's check function at endMajor. The FSM also accumulates statistics on the performed tests and shipped out these statistics in a dedicated TC attached to the endMacro transition. This FSM installed on many ROMs can also be used to perform a test that involves many crates. For example a full system tests from the TSF input memory to the GLT input memory is runnable with this FSM.

## **10 DCH to TSF interface tests**

A joint DCH-DCT test-stand has been installed to study problems in the DCH to TSF communications. This test-stand uses a DCH calibration procedure to check the correct transmission of the DCH data. However, for the moment, the comparison can only be done on the DAQ data: DCH hits and TSF segments. To have a more complete test, a new DCT FSM will be deployed. This FSM will be a normal DAQ FSM with its configure and L1Accept transition but it will add 2 components: at beginMinor, it will enable in record mode the TSF input memory and at end Minor, it will read the TSF input memory and put the read content in an XTC attached to the transition. An offline application will be used to compare the TSF input memory content with the DCH data and eventually the TSF DAQ data. This mechanism will also be used with both an old TSF and a new one in the crate. By comparing the input memory of the two TSF, one can test the new TSFi even if the new TSF DAQ vhdl engine is not ready. Although different from the

calibration FSM described in section 9, this FSM will use the Crate layer of the test-stand software for reading the TSF input memory.