Study of the Performance of the Data Acquisition Chain for BCM1F Software Upgrade

Bachelor Thesis in Physics

at

Brandenburg University of Technology

Supervisor: Prof. Dr. Wolfgang Lohmann

Maria Hempel

September 2010

Abstract

BCM1F, the Fast Beam Conditions Monitor, is a sub-detector of the CMS experiment at LHC. It monitors the beam halo and the collision product rates inside the CMS experiment.
The data acquisition of BCM1F is independent from CMS. Major components of the BCM1F back-end are discriminators, ADCs, TDCs, look-up tables and a Veto module.
In the thesis the performance of several components is investigated.
For the TDC two different readout modes are compared, and the impact of a Ring Buffer in the readout software was investigated.
For one discriminator the thresholds of all channels are investigated and offsets of about 10 mV are found.
Data taken in the LHC runs with the TDC are presented and discussed.
Also the of BCM1F as a luminosity monitor is studied.

Contents

Nomenclature

ADC	Analog-to-Digital Converter
BCM1F	Fast Beam Conditions Monitor
BLT	Block Transfer
CERN	Conseil Européen pur la Recherche Nucléare
CMS	Compact Muon Solenoid
DAQ	Data Acquisition
ECL	Emitter-Coupled Logic (electronic signal standard)
FIFO	First-in, First-out
HF	Forward Hadron Calorimeter
HPTDC	High Performance Time-to-Digital Converter
IP	Interaction Point
LHC	Large Hadron Collider
LSB	Least Significant Bit
LUT	Look-Up Table
NIM	Nuclear Instrument Modules (Electronic Signal Standard)
TDC	Time-to-Digital Converter
VME	Versa Module Eurocard

1 Introduction

1.1 Large Hadron Collider (LHC)

The LHC is a particle accelerator at CERN near Geneva. It is a storage ring of 27 km circumference located about 100 metres underground where two proton or heavy ion beams are accelerated in opposite directions and brought into collisions in the experiments.
The energy of each proton bunch can reach up to 7 TeV with a luminosity of 10^{34} $cm^{-2}s^{-1}$. Not only one particle will be accelerated by the LHC but a packet of particles which is called bunch. The minimal time between two bunches is 24.95 ns. The bunch needs 89 μs for one orbit at the LHC ring. Several bunches form bunch trains covering a few μs. Between bunch trains time intervals without bunches are foreseen to activate magnets for a beam dump, the so called abort gap. The gap is needed to allow the dump steering magnet to switch from off to on. The bunch-structure is illustrated in Figure 1. After the collision of the two bunches at very high energy, the particle detectors measure the collision products of proton-proton interaction. New fundamental particles or interactions are expected to be discovered, which may have governed the universe in a very early stage, just after the Big Bang [1] [2].

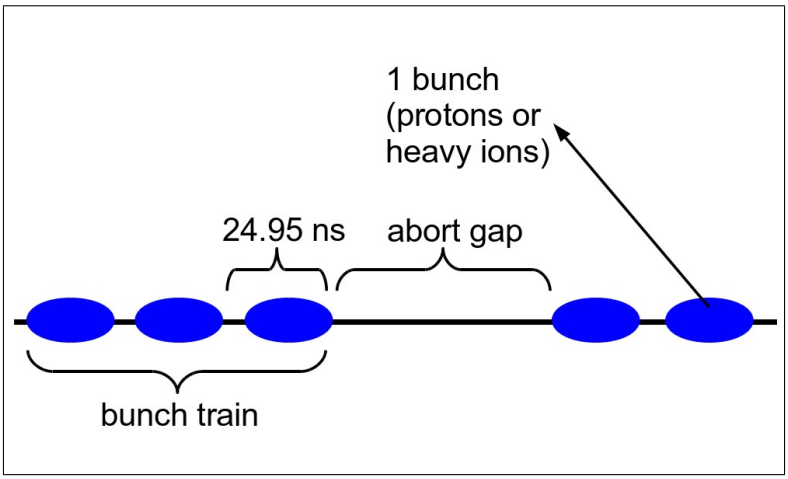


Figure 1: Scheme of the bunch structure

A major motivation of the LHC is the explanation of the electroweak symmetry breaking, which is caused by the Higgs mechanism. It will be a great success to find the Higgs boson with one of the experiments at the LHC.
Six experiments at the LHC were designed for different goals. The two largest experiments are ATLAS (**A** **T**oroidal **L**HC **A**pparatu**s**) and CMS (**C**ompact **M**uon **S**olenoid). ATLAS and CMS are searching for the Higgs boson to study the electroweak symmetry breaking mechanism. If they do not find the Higgs boson, they will look for alternative models. ATLAS and CMS have different detector concepts for mutual confirmation if one of them has made a new discovery.
ALICE (**A** **L**arge **I**on **C**ollider **E**xperiment) and LHCb (**LHC** **b**eauty) are two medium size detectors. With ALICE one can recreate the conditions as they existed in early stages of the universe by colliding ions. LHCb is studying the beauty quark and investigating the matter-antimatter asymmetry.
The smallest detectors are TOTEM (**TOT**al **E**lastic and diffractive cross section **M**easurements) and LHCf (**LHC** **f**orward). TOTEM is monitoring the luminosity and LHCf uses forward particles as a source to study interactions important to understand cosmic ray showers.

1.2 Compact Muon Solenoid (CMS)

The detector is permeated by a magnetic field of 4 T, which is generate by a superconducting solenoid. This magnetic field is needed to measure the momenta of charged particles.
Different sub-detectors are arranged inside the coil to detect collision products, such as photons, protons, electrons, and others [3]. The layout of the CMS detector is shown in Figure 2.

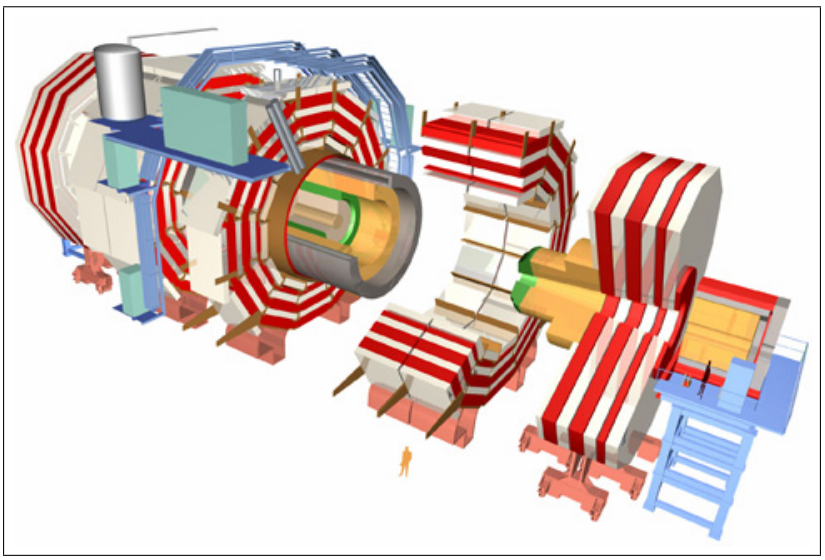


Figure 2: 3-dimensional view of the CMS detector [`http://public.web.cern.ch/public/en/lhc/CMS-en.html`]

Charged particles crossing the detectors ionise the material. In an electric field the charge carriers drift and generate electric signals to be measured. A pixel detector, that measures space coordinates along the track of charged particles, is located close to the IP. The pixel detector allows also the reconstruction of secondary vertices, created by the decay of instable particles.

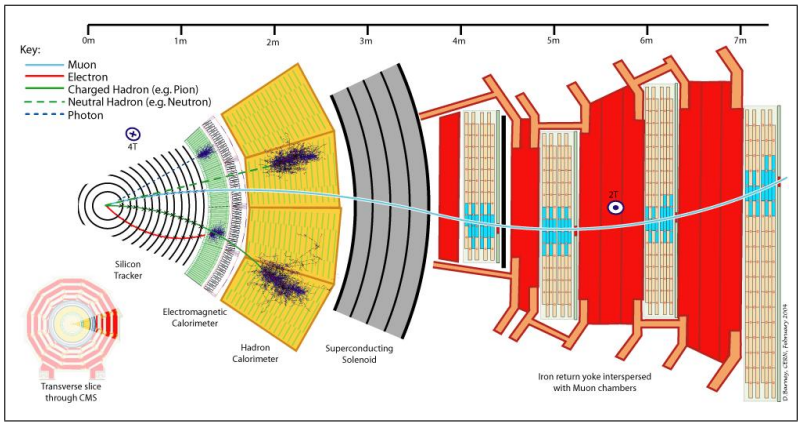


Figure 3: Signature of different particles in the detectors [`http://cms.web.cern.ch/cms/Detector/FullDetector/index.html`]

At large radius from the IP follows the silicon tracker built of silicon strip sensors that reconstruct together with the pixel detector the path of charged particles. Neutral particles, such as photons or neutrons, cross the silicon tracker usually without any signal.

The silicon tracker is encompassed by the electromagnetic calorimeter (ECal). The hadron calorimeter (HCal) is around the electromagnetic calorimeter. The electromagnetic calorimeter detects electrons and photons because they produce a shower of electrons and positrons due to bremsstrahlung and pair production. All hadrons are detected in the hadron calorimeter, where also neutrons and neutral kaons produce a shower. Both calorimeters are able to measure the energy of the particles that create a shower in the calorimeters. Two hadronic forward calorimeters (HF) are positioned at either end of CMS and close to the beam pipe. HF detects particles at low polar angles.

The outermost detector is the muon detector. Muons are charged particles like electrons, but they are 200 times heavier and penetrate several meters of iron. While most particles, such as electrons, protons, or neutrons, are stopped by the calorimeters, muons can pass through all calorimeters.

To detect muons, CMS has a detector in the iron yoke where muons are the only particles likely to deposit a signal. Because of the different orientation of the magnetic field the muon follows an S-shape curve. From the bending radii the momentum of a muon can be measured both in the inner tracker and in the muon detector. It is very important to detect muons because it can be a signature of the Higgs boson. When the Higgs boson is heavy enough it decays in two Z-bosons with subsequent decay into four muons. In Figure 3, one can see the interaction of different particles in each sub-detector.
Table 1 summarizes the signatures of each particle in the detectors.

Particle	Silicon Tracker	ECal	HCal	Muon Detector
Muon	track	track	track	track
Electron	track	shower	-	-
Charged Hadron(e.g. Pion)	track	track	shower	-
Neutral Hadron (e.g. Neutron)	-	-	shower	-
Photon	-	shower	-	-

Table 1: Signatures of different particles in the sub-detectors of the CMS experiment

1.3 Beam Conditions and Radiation Monitoring System (BRM)

CMS consists not only of detectors that measure and identify particles, it also has sub-detectors for protection and monitoring. The Beam Conditions and Radiation Monitoring system (BRM) performs both of these tasks [2].

sub-system	Sampling time	Function
Passives	Long Term	Monitoring
RADMON	1s	Monitoring
BCM2	40 μs	Protection
BCM1L	sub orbit 5 μs	Protection
BSC	bunch-by-bunch counter with ns time resolution	Monitoring
BCM1F	bunch-by-bunch counter with ns time resolution	Monitoring
BPTX	200 ps	Monitoring

Table 2: Sub-systems of BRM [3]

All BRM sub-systems are listed in Table 2. Each sub-system of BRM is working at a different sampling time. Two of them can initiate a beam abort to protect the CMS in case of adverse beam conditions. Others monitor the beam conditions and signal e.g. the need of improvements in the beam optics and collimators.
The protection systems are BCM1L and BCM2 being fast enough to match beam abort scenarios. BCM1L and BCM2 are positioned on either side of the IP but BCM2 is farther away than BCM1L. These detectors are used in the current measurement mode as particle flux monitors. In case the currents exceed certain thresholds a beam abort is initiated.
The monitoring system contain slow and fast detectors. The passive detectors measure the radiation dose at several positions in and around the CMS detectors over several weeks or month. The RADMON, based on FET transistorsconsists, is readout each second, BCM1F (Fast Beam Conditions Monitor), BSC (Beam Scintillation Counters) and BPTX (Beam Pickup tp Trigger Experiment) are fast detectors. BCM1F counts particles crossing its sensors. The precision of the time measurement, about 1.3 ns, is small in comparison to the time between two bunch-crossings. Hence, halo particles or collision products originating from bunch crossings can be resolved. For precise information of the bunch arrival time, BRM has BPTX. The BSC detector is a set of scintillators (32) mounted in front of HF to provide raw timing and beam halo information.
In case of a beam abort, the BRM system will provide data to allow a post-mortem analysis. All BRM detectors are working independently of other CMS sub-detectors.

2 Fast Beam Conditions Monitoring (BCM1F)

BCM1F is vital for monitoring beam conditions in CMS [4]. It is sensitive to fast changes of beam conditions and provides diagnostics with a time resolution much better than the time between bunch crossings. In case of a beam abort it will provide data for a post-mortem analysis. BCM1F is working independently of other CMS sub-detectors to monitor the beam halo continuously.

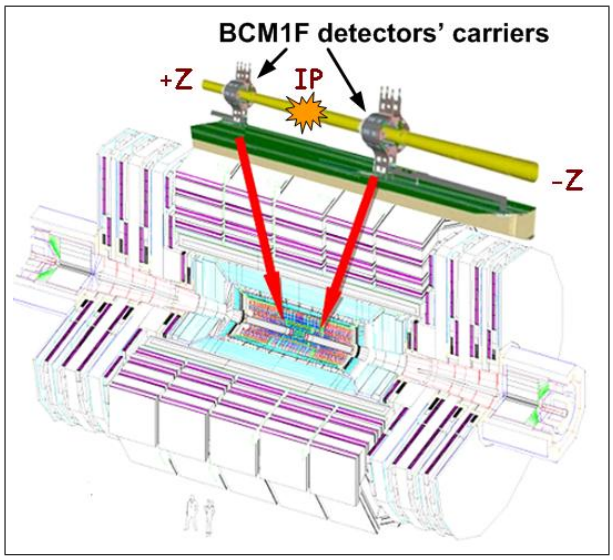


Figure 4: Position of BCM1F inside the CMS detector [4]

Figure 4 shows the location of BCM1F inside the CMS detector. BCM1F has 2 planes with 4 single-crystal chemical vapor deposition (sCVD) diamond sensors. Each sensor has a volume of $5\times5\times0.465$ mm^3. The planes are positioned at each side of the IP and the distance between plane and IP is 1.8 m. Relativistic particles need 6 ns to move that distance. Figure 5 shows the arrangement of the eight sensors with respect to the beam pipe and the labeling scheme. The labeling "near, far, top and bottom" refers to the location of the diamonds with respect to the center of the LHC ring. The distance from the IP is optimized to separate the beam halo coming from +Z and -Z directions and the collision products using the arrival time information. Details are described in Chapter 6.

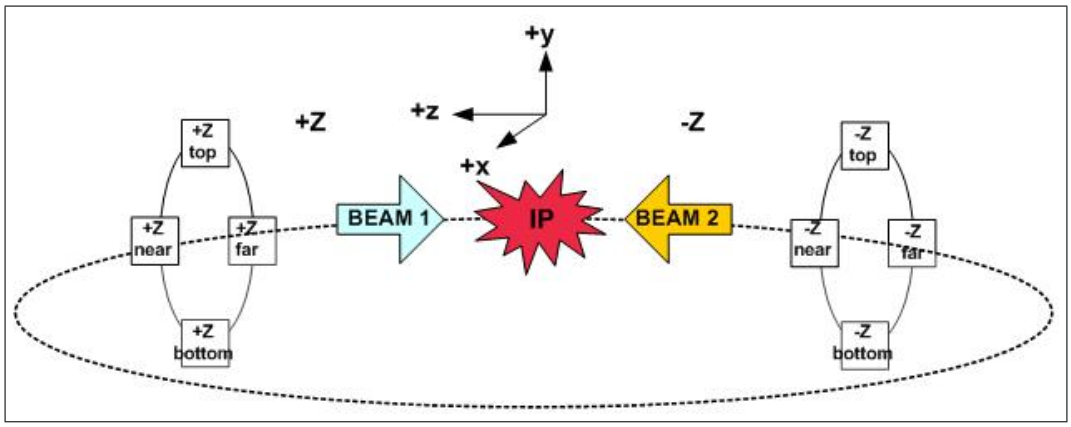
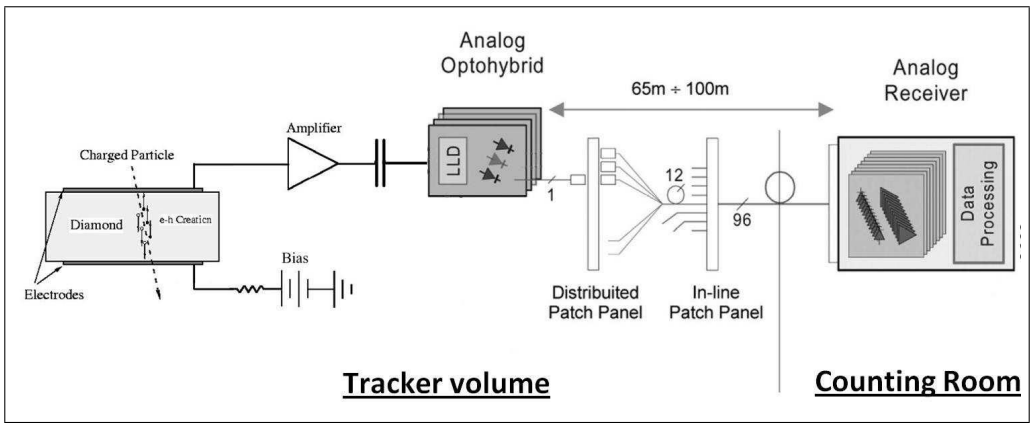
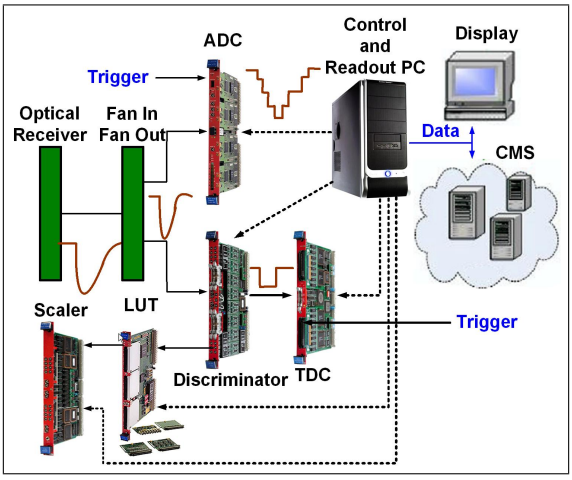


Figure 5: Layout of BCM1F Setup

Each sensor is connected to a radiation-hard preamplifier. If a charged particle passes through a sensor, it generates an electrical signal (Hit) that is amplified. After the amplifier the signal is converted into an optical signal. The optical signal is transmitted to the counting room. The whole chain is shown in Figure 6(a). In the counting room, the signals are digitized and analyzed by a power PC. The readout of the back-end is illustrated in Figure 6(b). For more details see Section 4.1.



(a) Scheme of the analog part of the BCM1F readout system



(b) Scheme of the back-end readout of BCM1F

Figure 6: Readout System [4]

3 Luminosity Measurements at LHC

The luminosity measurement is used to monitor the LHC performance and to provide an overall normalization for physics analysis. It is needed for the calculation of the cross-section of the recorded collision processes.
In case of two particle beams a and b are colliding in a storage ring the instantaneous luminosity is given by the following equation [5]:

$$\mathcal{L} = \frac{N_a \cdot N_b \cdot j \cdot v}{A \cdot U} , \tag{1}$$

where j is the number of particle bunches, each of them have N_a or N_b particles. U is the circumference of the collider. Due to a magnetic field, beams a and b circulate in opposite directions with the velocity v. The two beams will collide at an interaction point $j \cdot \frac{v}{U}$ times per unit time. A is the beam cross-section at the collision point. For a Gaussian distribution of the beam particles around the beam center with horizontal and vertical standard deviation σ_x and σ_y , A is given by:

$$A = 4\pi\sigma_x\sigma_y . \tag{2}$$

The dimension of the luminosity is $\frac{1}{\mathrm{t \cdot s}} = \frac{1}{\mathrm{cm^{-2}s}}$. Instead of cm^2, very often another unit of area, the barn, is used:

$$\begin{aligned} 1 \text{ barn} &= 1 \text{ b} \\ &= 10^{-24} \text{ cm}^2 . \end{aligned}$$

To get a high luminosity, the beam must be focused at the interaction point into the smallest possible cross-sectional area.
The cross-section is a yardstick of the probability of a reaction between the two colliding particles. For a certain process it is given by:

$$\sigma = \frac{\dot{N}}{\mathcal{L}} , \tag{3}$$

where $\dot{N}$ is the number of events of the process per time unit.
An often used quantity in storage ring experiments and at LHC is the integrated luminosity which is defined as:

$$\mathrm{L} = \int \mathcal{L} dt . \tag{4}$$

With the integrated luminosity, the cross-section can be written as following:

$$\sigma = \frac{\dot{N}}{\mathcal{L}} = \frac{dN}{dt\mathcal{L}} = \frac{N}{\mathrm{L}} , \tag{5}$$

where N is the total number of events selected for a certain process.

The luminosity measurement is made by HF. HF is close to the beam pipe and covers a pseudorapidity range $3 < |\eta| < 5$ [2]. The pseudorapidity is given by the following equation:

$$\eta = -log\left\{tan\left(\frac{\Theta}{2}\right)\right\} , \tag{6}$$

where Θ is the angle between the particle being considered and the beam axis. A method to measure the luminosity with HF is based on the measurement of the transverse energy, E_T, in the pseudorapidity of HF. It exploits the linear relationship between E_T deposited in the HF and the number of interactions. The number of interactions is proportional to the instantaneous luminosity. However, a gauge is needed to translate the measurements in HF into the absolute pp (proton-proton) luminosity. To obtain it, van-der-Meer scans [6] are done by LHC once per running period. In these scans beams with known σ_x and σ_y are moved systematically in x and y to maximize the collision rate, and the absolute luminosity is obtained using Equation (1).
It is planned to use BCM1F as a luminosity monitor to get an additional fast luminosity monitoring.

4 Performance of the Data Acquisition of BCM1F

The BRM group at DESY-Zeuthen was in charge of designing, building and testing the back-end part of BCM1F. The structure of the back-end is illustrated in Figure 7. Two identical setups are installed at CERN and in the laboratory at DESY-Zeuthen. Before the installation of the DAQ modules and the readout software at CERN, the whole system was tested in the laboratory in DESY-Zeuthen. One of the requirements on the system is continuous running without crashes. Hence, also long term test have been done in the laboratory. In addition, the system needs to be fast in terms of double hit resolution and data delivering to the control rooms. The limitations of the system with respect to readout speed and dead time are investigated. This is important at higher luminosity, since the system must be able to cope with a larger amount of data.

4.1 Experimental Setup of the Back-End

The readout at back-end consists of different VME based readout modules produced by CAEN. The modules are listed in Table 3.

Module	CAEN model	Task	Features
Discriminator	v258B	Discrimination of signals produced by particles	Threshold voltage [0,-510 mV]
TDC	v767	Time information	0.8 ns/bin, 10 ns double hit resolution,
ECL↔NIM converter	v538	Signal conversion for adaption of output/ input of different modules	I/O delay < 5 ns
LUT	v1495	coincidences in pairs of detectors, veto signal during FIFO readout	programmable general purpose board
Scaler	v560E	Hit rates	32 bit counter 5 ns double hit resolution
ADC	v1721	signal sampling	4.5 mV/bin 500 Ms/s (Megasamples/s)
Veto-Module	-	blocking incoming signals	needs activation signal

Table 3: Modules for the readout

A discriminator receives the signals from the analog chain and generates a standard output signal only if the input signal is above a threshold set via software. The TDCs provides the time information of the Hits. An ECL↔NIM converter is needed for the conversion of an ECL electronic signal to a NIM electronic signal or NIM to ECL since NIM and ECL logics are different. NIM logic is a current based logic with negative true at -0.8 V with an input impedance of 50Ω and false at 0 V. The ECL based logic varies from -0.8 V(true) to -1.6 V(false).
A LUT is a programmable board that builds coincidences of two signal inputs and it can be used as a register. For more details see Section 4.1.3. The Scaler allows to measure the count rates (# Hits/time).
A 8-bit fast sampling ADC digitizes the signal shape and can be used for the measurements of signal spectra.
The Veto module blocks new incoming Hits in case of an high Hit rate. It is described in Section 4.1.5.
To check the functionality of the system in the laboratory the scheme of the back-end that is illustrated in Figure 7 was used. Instead of incoming Hits produced by particles test-pulses from a pulse generator were used. The Hits simulate the passage of charged particles trough the diamond detectors.

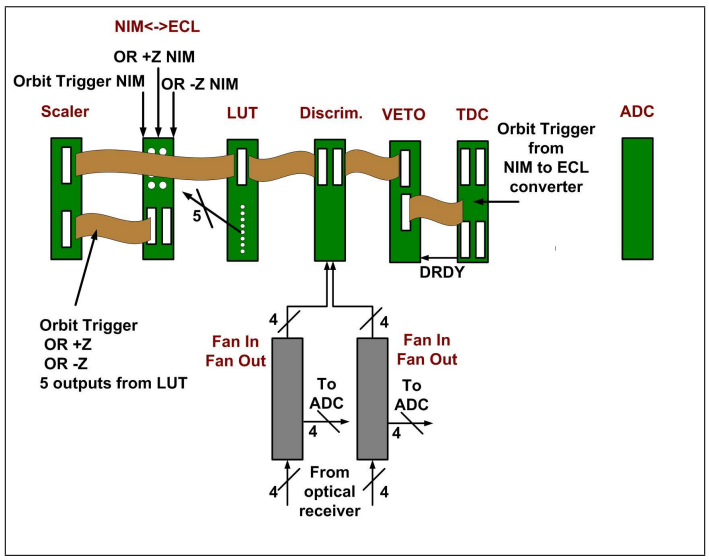


Figure 7: Scheme of the data processing at the BCM1F back-end

First the signals from the analog chain go to a Fan-In-Fan-Out unit and then to the discriminator. In the discriminator for each channel a threshold voltage is set and only pulses larger than the threshold voltage produce an output signal.
After the discriminator, the Hits pass through the so-called "Veto module" and then to the TDC. The measurements with the TDCs are made with and without the Veto module to identify possible performance differences.
The Hits go also from the discriminator to the Scaler to measure the count rates. An additional board, the Look-up Table (LUT), is included between Scaler and discriminator.

The setup at DESY-Zeuthen is identical to the one installed at CERN. Also the labeling scheme of the inputs/outputs of each module is the same as the labeling scheme at CERN. Table 4 lists the inputs/outputs of the modules and the specification of the sensors.

Location of sensor with respect to the center of the LHC ring	Fan-in,Fan-out channel	Discriminator input/output	TDC channel input/output	Scaler input/output
-Z, Top	1/1	08	0	08
+Z, Top	2/1	09	1	09
-Z, Far	1/2	10	32	10
+Z, Near	2/2	11	33	11
-Z, Bottom	1/3	12	64	12
+Z, Bottom	2/3	13	65	13
-Z, Near	1/4	14	96	14
+Z, Far	2/4	15	97	15

Table 4: Nomenclature of the Channels

4.1.1 Discriminator

The discriminator v258B [7] has 16 channels and each channel has its own threshold. The threshold for each channel can be set individually via a program in a range from 0 mV (0x00) to −510 mV (0xFF) in steps of −2 mV. The module accepts only negative input voltages and produces ECL outputs. However, the width of the output signal is independent of the input signal time over threshold. The pulse width can be set in two ranges: 5-40 ns and 40-300 ns, using jumpers and it is adjustable with a front panel trimmer ("WDT"). For our setup, we are using a range of 5-40 ns because the time between the LHC bunches is 25 ns and the range 40-300 ns would hide consecutive bunches. The time between the bunches is 25 ns that means during 100 ns, we may see 4 Hits in

our system. If we would use a higher width of the output pulse, for example 100 ns, we may lose after a signal potentially signals from the following 3 bunches.

4.1.2 Time to Digital Converter (TDC)

The TDC module v767 houses 128 independent TDC channels [8] which are included in 4 TDC chips [9]. Hence, each TDC chip houses 32 channels. Two important specifications of TDC are the Double Hit Resolution (the minimal time between two consecutive Hits), which is 10 ns, and the LSB (time resolution) which is 800 ps.
A TDC works as an electronic clock. It needs a reference signal that is called Start and measures the time of the Hits with respect to the Start. The Start signal is needed for the calculation of the time of the Hits. In the laboratory setup at Zeuthen we generate the Starts and Hits with a pulse generator. The output of the generator is a NIM signal, but the TDC accepts only ECL signals. That means the Start goes from the generator to an ECL←NIM converter and then to the START input of the TDC board in the front panel. The period of the Starts, T_{Start}, are for each measurement 100 μs. The real LHC Start period is 89 μs which correspond to the time needed for a bunch to make one turn. Nevertheless, a test period of 100 μs allows us to check the performance of the DAQ in an easy way since we can easily generate Hits as a multiple of the Start period. The period of the Hits is adjustable. The Hits and Starts will be processed in the TDC chips and will be stored in the FIFO (output buffer) that will be filled with up to 32k words of 32 bits. The FIFO is described in Chapter 4.3.

4.1.3 Look-Up Table (LUT)

The LUT is included between the Scaler and discriminator. The LUT v1495 [10] is user-programmable I/O and logic blocks. It can be directly customized by the user and the management is handled by two FPGAs. The FPGA is provided with a basic firmware which allows to define coincidences matrices, I/O register, and asynchronous timer functions.
For our setup it will be used as a veto register for the Veto module and for building coincidences. The Veto is active when a specific register of the LUT is set to 1, and a 0 resets it.
The coincidences are built by looking at the respective two inputs. Figure 8 shows how to get the coincidences. Two Hits, which are coming from the discriminator, arrive in the LUT. A logical AND of both signals is building a coincidence.

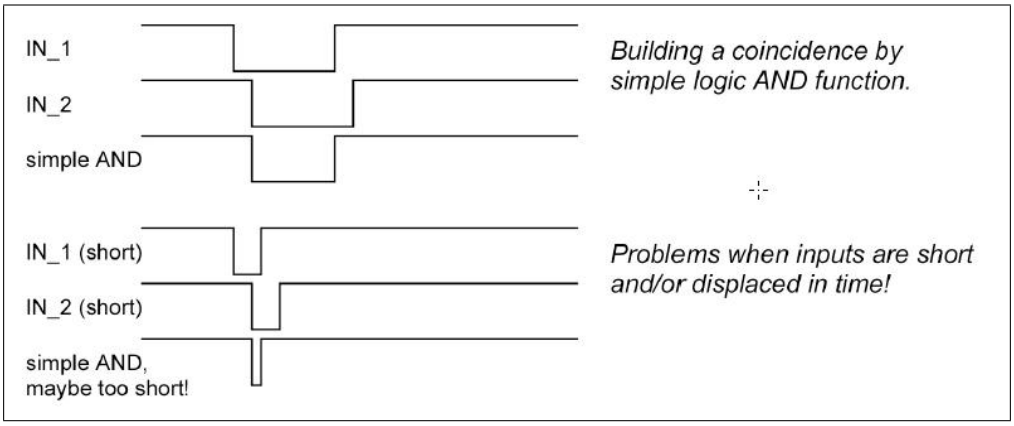


Figure 8: Scheme of a coincidence of two signals

In the future, it is planned to use the LUT for changing the operation mode to post-mortem analysis after beam abort.

4.1.4 Scaler

The Scaler module v560 has 16 independent 32-bit counting channels [11] which accept only ECL signals. Each channel count the number of Signals per time. The minimum time interval between two signals is 5 ns.
In eight channels we are counting the Hit rate of the eight diamond sensors in BCM1F. Two other

channels count the OR of the four sensors in +Z (OR +Z) and in -Z (OR -Z). The Scaler channel OR +Z counts every time when the OR is 1. The OR becomes 1 when in at least one of the four sensors is a Hit.
The coincidence, that are built in the LUT, are count in four other channels of the Scaler. The four coincidences are important for the luminosity measurements with BCM1F. For more detail see Section 7.1.
The last two channels count the number of Starts (orbit trigger) that is provided by LHC and the number of active veto signals. The veto signal is always active during the block transfer (BLT). It is described in Section 4.1.5.

4.1.5 Veto Module

The Veto module was developed in DESY-Zeuthen. The need of the Veto module came up after the first test in the laboratory with the TDC module.
We observed that the data transfer from the TDC to the computer via the VME bus was to slow to prevent overwriting of the data in the FIFO by the new incoming data. It was decided to veto the storage of new data in the FIFO during the transfer of data. With the Veto board only Start signals may reach the TDC FIFO during the transfer of data. The Veto blocks new Hits if the FIFO of the TDC is half full. The half full condition is used because the size of one BLT of data is a half of the TDC FIFO (16k).
An external signal is needed to activate the Veto module. To get an active veto signal during the whole BLT of 16k, we are using the LUT. A register in the LUT is set to 1 when the data is ready. The data is ready when the FIFO is half full. The register is 1 during the whole BLT of 16k. After the BLT the register is set to 0 and the TDC is filled with new Hits. Such a register signal is illustrated in Figure 9.

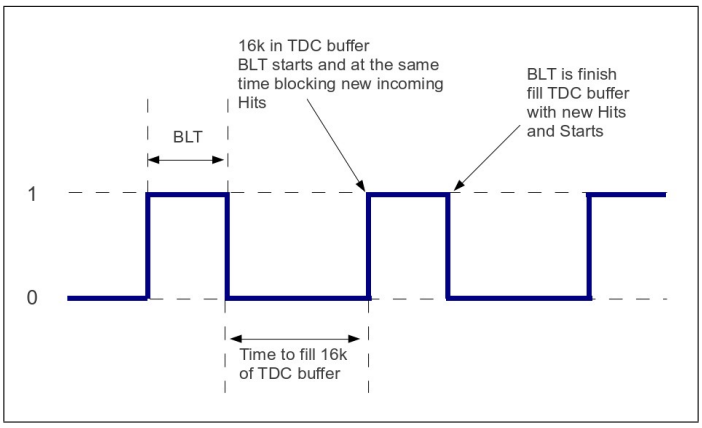


Figure 9: The output of the LUT register

For the TDC measurements with the Veto module we are using the LUT, which is also included in the final DAQ setup of the TDC.

4.2 Principles of Operation

The data acquisition in the TDC can be programmed in different modes. We are using the *continuous storage* mode [8]. In this acquisition mode, all Hits that reach the TDC are stored in the FIFO. Optionally, Starts can be stored in the FIFO.
The time measurement can be done in different ways. One way is the absolute time measurement. For this, the zero time, given by the reset of the TDC, is a reference for time calculation. An other way is the relative time measurement. For this, each Hit will be referred to a Start as shown in Figure 10. The START period, T_{Start}, is well-known. The Hits which arrive at time t_1, t_2 and t_3 are referred to the START time T_1. The following Hits t_4 and t_5, which arrive at the second START period, are referred to the START time T_2. The arrival time of STARTS and Hits is written in sequential order in the FIFO and the order reflects the time evolution (see Table 5).
It is possible to disable the writing of the Start times. If the total rates of Starts and Hits are higher than the transfer rate of data from the TDC to the FIFO, an overflow will be the result

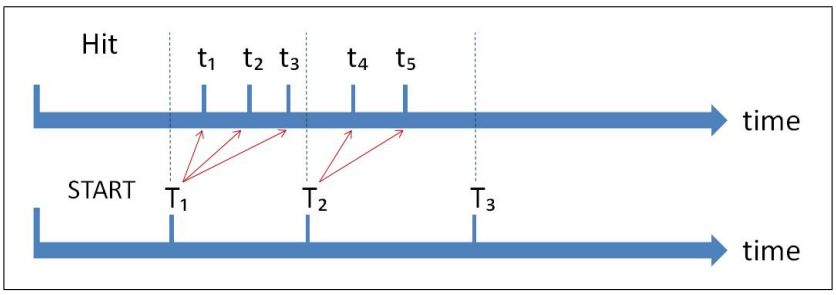


Figure 10: Continuous storage mode [8]

Datum	Signal
Datum n. 1	Start time T1
Datum n. 2	Hit time t1
Datum n. 3	Hit time t2
Datum n. 4	Hit time t3
Datum n. 5	Start time T2
Datum n. 6	Hit time t4
Datum n. 7	Hit time t5

Table 5: Storage of data in the FIFO

and data will be lost.

4.3 TDC FIFO and Data Structure

All the data (Hits and Starts) that come from the TDC chips are stored in the FIFO. The FIFO contains 32k words and each word is 32 bit long. In case of continuous storage mode, one word contains 20 bit time information and one bit gives information if it is a Start signal (1) or a Hit signal (0). The last bits contain information about the channel number. The structure of one word is given in detail in Figure 11. Each Hit and Start is one word and they will be stored in sequential order in the FIFO.

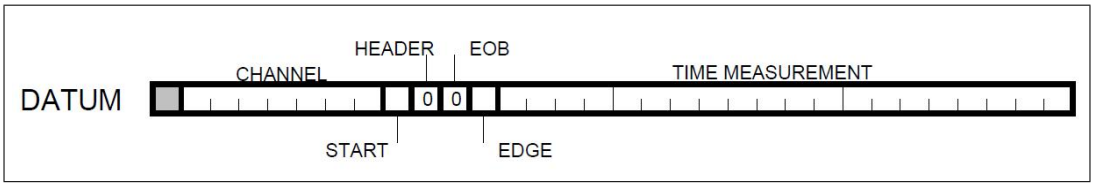


Figure 11: Structure of one word (DATUM) in the FIFO

To start the readout of the FIFO we use the "Data Ready" signal of the TDC. This happens when the "Buffer Almost Full" level is reached. This level goes from 0x0002 to 0x3FFF[1] corresponding to 16k words = half of the TDC FIFO. For the readout we can choose between two methods: single memory access and block transfer via the VME bus.
With the single memory access, we are reading out the data from the FIFO word by word. This readout requires one VME cycle per 32-bit word transfer. The BLT reads out a whole block of memory - 16k of 32 bit words - in one single VME access.
If the FIFO is full and the readout is slow, new data will overwrite the old data in the FIFO. In this case the data will be corrupted. When the DAQ collapses, we can determine the time we need to fill the FIFO. The measurements in this section characterize the FIFO of the TDC v767.

[1] hexadecimal numbers

4.4 Readout Code

For the control and the readout of the TDC module CAEN v767 via the VME bus we use a code that is written in C++ where the TDC module is defined as a C++ class. The code comprises four files. With the file "tdc_v767_caen.hh" we define the map of registers and functions that are needed for the control of the TDC. In the file "tdc_v767_caen.cc" we implement the functions that were declared in the previews file. The "Makefile" is needed for the C++ compilation. At the end we have the main file that is called "DAQ_code.cc". This code we use for the DAQ. In the following sections different versions of "DAQ_code.cc" are described and the results are reported. The two different versions are "TDC_BufferAlmostFull.cc" using single memory access (with and without Veto module) and "BLT_tree_vmartin" using the block transfer (with and without Veto module). Both codes are given in the Appendix.

4.4.1 Measurements using Single Memory Access without Veto module

To read out the FIFO with single memory access we use the program "TDC_BufferAlmostFull.cc". A flow diagram is given in Figure 12.

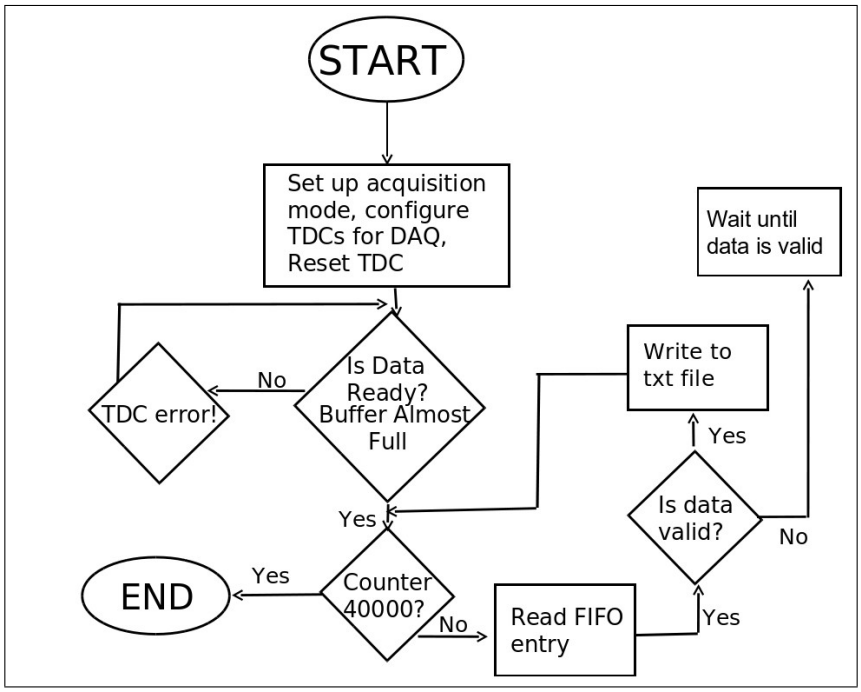
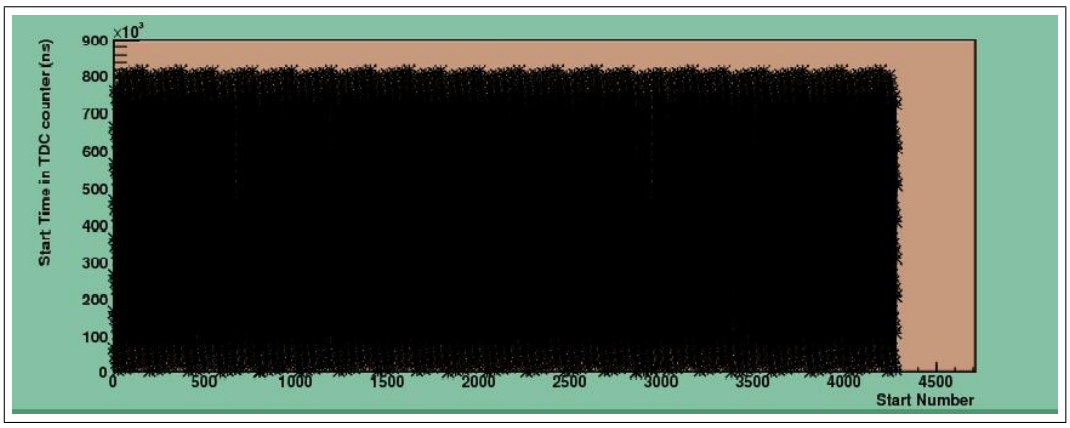


Figure 12: DAQ code flow diagram for single memory access

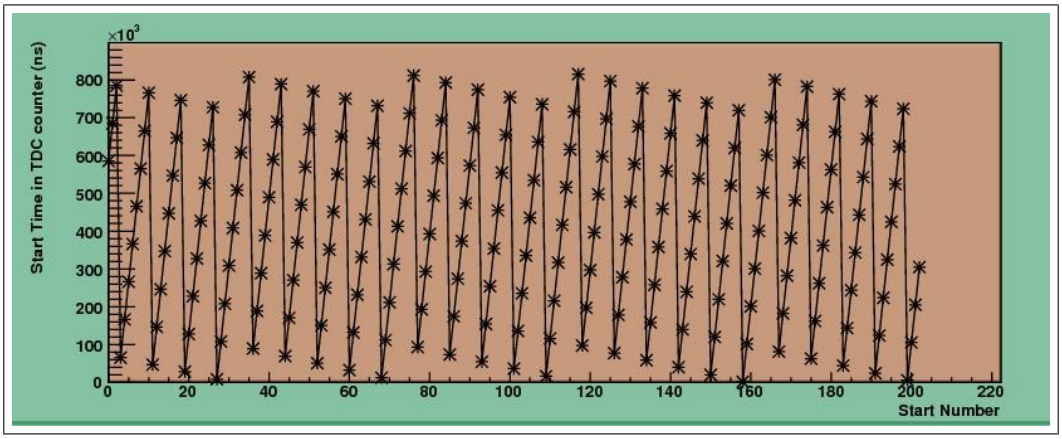
The memory of the TDC v767 FIFO has ≈32k words and to read out all of them, the program can loop over a readout of 40000 words. With this loop, we cover the whole TDC memory.
The program runs with the following parameters:

- 40000 acquisitions
- 1 channel enabled (number of channel is 32)
- $T_{\text{Start}} = 100\ \mu$s
- $t_{\text{Hit}} = 10\ \mu$s ,

where t_{Hit} is the time between two consecutive Hits.
The results of the tests are shown in Figure 13 and 14.



(a) Dynamic range as a function of the number of T_{Start}



(b) Zoom of upper plot

Figure 13: Dynamic Range

Figure 13(a) shows in the y-axis the dynamic range of the TDC and in the x-axis the number of T_{Start} which are counted in the FIFO. In Figure 13(b) is given the dynamic range from $0\times T_{\mathrm{Start}}$ till $200\times T_{\mathrm{Start}}$. The dynamic range of the TDC is 840000 ns. If the period of the T_{Start} is 100 μs and the Start time of the TDC begins to count at 0 μs, the Start time increases 100 μs after each Start till reaching 840000 ns. After eight Starts it achieves 8×100 μs=800000 ns=800 μs and goes back to 0 μs. This behavior is illustrated in Figure **??**fig:ZOOM.

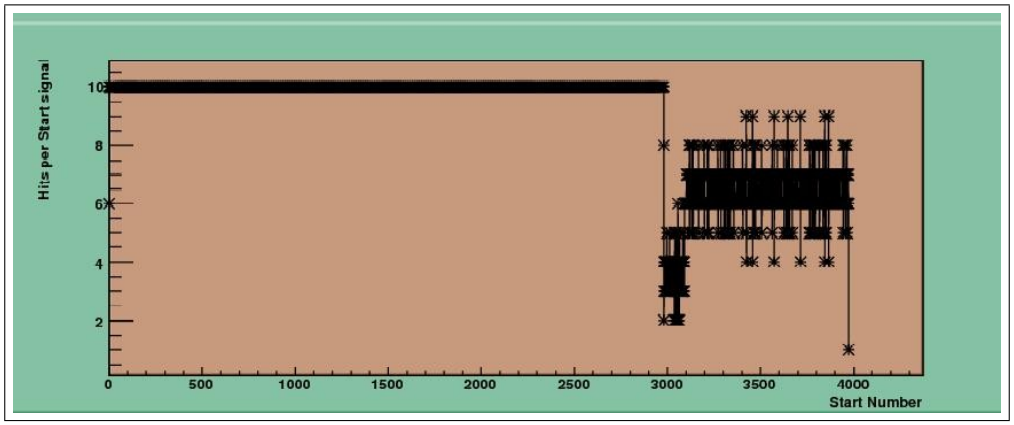


Figure 14: Hits per T_{Start} as a function of T_{Start}

Figure 14 shows the Hits registered per Start. In this case we have a Hit each 10 μs, or 10 Hits per Start because:

$$\begin{aligned} \text{Hits per Start} &= \frac{T_{\text{Start}}}{t_{\text{Hit}}} \\ &= \frac{100\ \mu s}{10\ \mu\text{s}} \\ &= 10\ . \end{aligned} \tag{7}$$

Since the readout rate is slower than the data input rate old data may be overwritten when the FIFO is full. In Figure 14 we see above a Start number of 2980 the obtained number of Hits becomes wrong, since the FIFO is full. With the Start number of 2980 the time to fill the FIFO can be calculated:

$$\begin{aligned} \text{time to fill FIFO} &= \text{Start number} \cdot T_{\text{Start}} &(8)\\ &= 2980 \cdot 100\ \mu\text{s} &(9)\\ &= 289\ \text{ms}\ . \end{aligned}$$

From this point on the DAQ collapses. Furthermore, we can see how many words are in the FIFO without collapse of the DAQ:

$$\begin{aligned} \text{size of full readout} &= \text{Hits per Orbit} \times \text{Start Numbers} + \text{Start Numbers} \\ &= 10\frac{\text{Hits}}{\text{Start}} \times 2980\ \text{Starts} + 2980\text{Starts} \\ &\approx 32k\ \text{words}\ . \end{aligned} \tag{10}$$

In this way, one can measure the time to fill the FIFO for different Hit rates. In this experiment, we increased the Hit rate and measured the time to fill the FIFO. Table 6 shows the relation between Hit rate and the time to fill the FIFO under the following conditions:

- 40 000 acquisitions
- 1 channel enabled (channel number 32)
- $T_{\text{Start}} = 100\ \mu\text{s}$

t_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	Hit rate	time to fill the TDC FIFO (32k)
10 μs	10	100 Hits/ms	298 ms
2.5 μs	40	400 Hits/ms	80 ms
1 μs	100	1000 Hits/ms	32.4 ms
500 ns	200	2000 Hits/ms	16.5 ms
200 ns	500	5000 Hits/ms	6.5 ms

Table 6: Time to fill TDC FIFO using single memory access, 1 ch enabled

We can see in Table 6 how the time to fill the FIFO decreases when the Hit rate increases.
The same measurements can be done with 7 channels enabled (channels 1, 32, 33, 64, 65, 96, 97). The results are shown in Table 7. For 7 channels enabled, data loss happens after 46.2 ms for a Hit rate of 100 Hits/ms. This 46.2 ms is $\approx \frac{1}{7}$ of 298 ms which is the time to fill the FIFO for 1 channel enabled.

t_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	Hit rate	time to fill the TDC FIFO (32k)
10 μs	10	100 Hits/ms	46.2 ms
2.5 μs	40	400 Hits/ms	11.6 ms
1 μs	100	1000 Hits/ms	4.6 ms
500 ns	200	2000 Hits/ms	0.65 ms
200 ns	500	5000 Hits/ms	-

Table 7: Time to fill TDC FIFO using single memory access, 7 channels enabled

To summarize, for a Hit rate of 10 per 100 μs using single memory access we get a relation between the time to fill the FIFO and the number of channels enabled.

$$\text{time to fill FIFO} = \frac{298\ ms}{\text{number of channels enabled}} \tag{11}$$

4.4.2 Measurements using Single Memory Access with Veto module

To characterize the Veto module, the same measurements as in the previous case were done. As illustrated in Figure 7, the Veto module is included between the discriminator and the TDC and it is controlled by the LUT. Using the Veto module, we wait until the data is ready, 16k words are in the FIFO. Then we enable the Veto module by the LUT providing a signal to stop sending data to the TDC. Then we loop over the whole FIFO and read out 40000 words.
The conditions of measurement were:

- 40 000 acquisitions
- 1 channel enabled (channel number 32)
- $T_{\text{Start}} = 100\ \mu$s
- $t_{\text{Hit}} = 10\ \mu$s

Figure 15 shows the time when the DAQ collapse for a Hit rate of 100 Hits/ms (10 Hits per Start).

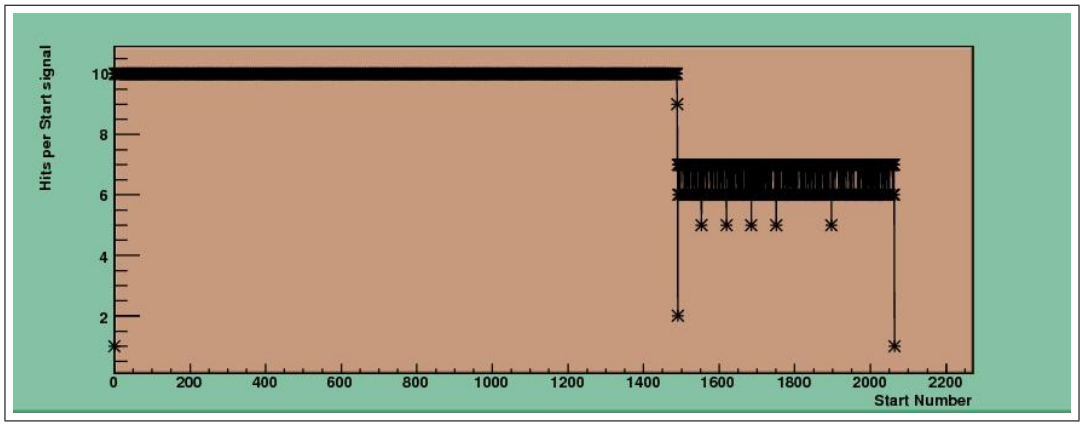


Figure 15: Hits per T_{Start} as a function of T_{Start}

In Figure 15 we see above the Start number of 1488 the obtained number of Hits becomes wrong. The size of the data readout can be calculated with the Start number of 1488:

$$\begin{aligned} \text{size of full readout} &= \text{Hits per Orbit} \times \text{Start Numbers} + \text{Start Numbers} \\ &= 10\frac{\text{Hits}}{\text{Start}} \times 1488\ \text{Starts} + 1488\text{Starts} \\ &\approx 16k\ \text{words}\ . \end{aligned}$$

The size of the readout is only 16k instead 32k as in the case before even though we loop over the whole FIFO of 32k. In Table 6 is given the time to fill the FIFO for different Hit rates. Figure 16 illustrates the time to fill the FIFO without losing data over the Hit rate with and without Veto. We can see that the time to fill the FIFO in case of using the Veto module is a half of the time needed without Veto module.

T_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	Hit rate	time to fill the TDC FIFO (16k)
10 μs	10	100 Hits/ms	148.8 ms
2.5 μs	40	400 Hits/ms	39.9 ms
1 μs	100	1000 Hits/ms	16.1 ms
500 ns	200	2000 Hits/ms	8.1 ms
200 ns	500	5000 Hits/ms	3.2 ms

Table 8: Time to fill the TDC FIFO using single memory access, 1 ch enabled and with Veto module

As explained in Section 4.1.5 the Veto module is inactive while the level of the FIFO filled is less than 16k (half of the FIFO). If the FIFO is half full the Veto module will block new Hits. Meanwhile the program writes the information of 16k from the FIFO to a txt file and only Starts are written to the FIFO. Only the half of the FIFO is filled with Hits. That is why we get a half of the time to readout the FIFO.
The advantage of the Veto module is that we can control when we want to read out the data from the FIFO and we prevent that a high rate of incoming Hits corrupt the FIFO during readout via the VME bus.

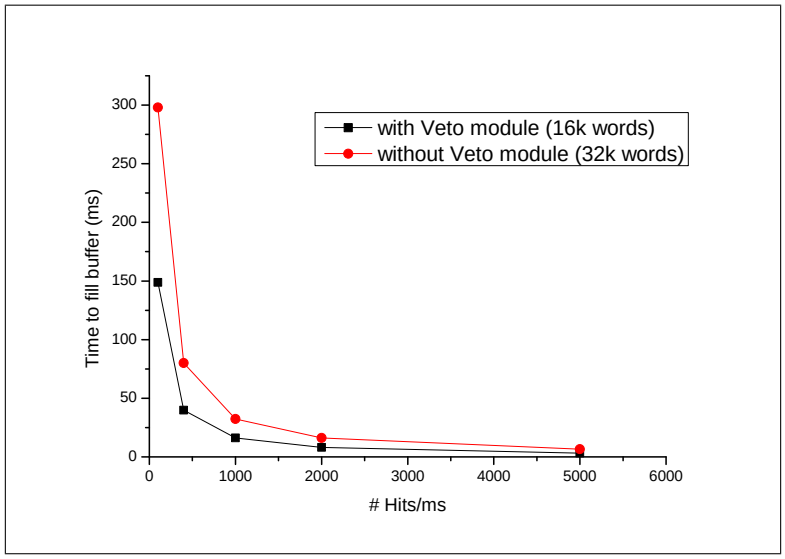


Figure 16: Time to fill the FIFO as a function of the Hit rate with and without Veto module for 1 channel enabled

4.4.3 Time of Readout for Single Memory Access

To compare the single memory access with the block transfer, it is important to know the time for the readout. For the single memory access, we measured the time for a readout of 16k and 32k. The measurements for 16k were done via software and with the help of an oscilloscope. For the time measurement with the software we included in the DAQ code C++ time functions. We read the time before starting the loop of 16k readouts and after the loop. The time difference is the time of a data transfer of 16k words. For the time measurement with the scope, we observed the output of the Veto signal provided by the LUT. The Veto is set to 1 during readout of the TDC, then it is set to 0 to allow writing data into the TDC FIFO and then it is again set to 1 during the readout. After the readout the Veto is reset to 0. With the scope we can measure the time of the readout. This is illustrated in Figure 17. The time of the readout that was measured with the scope is 218 ms.
In Table 9 is shown the time for the readout via the software. The measurements show that the time of the readout is independent of the Hit rate. The readout time via software for 16k is obtained to 229 ms, roughly the same time measured with the scope. The time measurements for 32k words are only done via software. For a readout of 32k, about 470 ms are needed.

t_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	time for readout of 32k for 1 ch enabled [ms] No Veto	time for readout of 16k for 1 ch enabled [ms] Veto
10 μs	10	462.4	239.7
2.5 μs	40	466	225.8
1 μs	100	476	228
500 ns	200	478.6	226.2
200 ns	500	466.4	225.1

Table 9: Time for readout using single memory access, measured via DAQ software

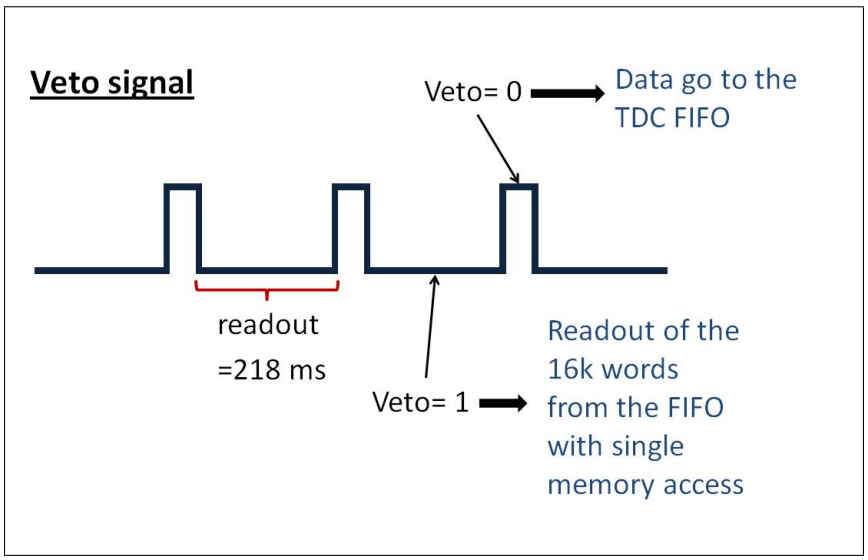


Figure 17: Readout signal using single memory access observed on scope

4.4.4 Measurements using Block Transfer and without Veto Module

In the previous measurements, the program with single memory access reads out word by word. Each readout cycle needs a VME access. To speed up the DAQ we used the program "BLT_tree_vmartin.cc" (Block Transfer) transferring the whole memory as one block. Here only one VME access is needed. The DAQ scheme is given in Figure 18.

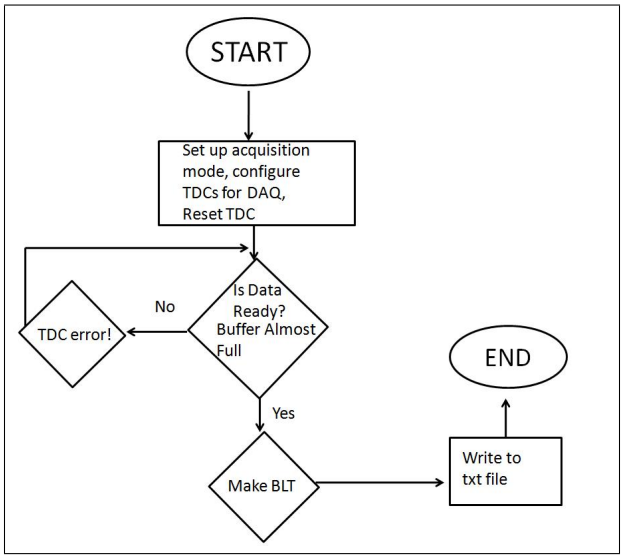


Figure 18: DAQ code flow diagram for reading data from the FIFO as a block (BLT)

The advantage is that the time of the readout is expected to be smaller than the single memory access but this will be checked in the this section.
The test conditions were:

- channel 32 enabled
- $T_{\text{Start}} = 100\ \mu\text{s}$
- $t_{\text{Hit}} = 10\ \mu\text{s}$

Figure 19 shows the time when the DAQ fails for 10 Hits/Start. On the x-axis is the number of T_{Starts} and on the y-axis is the number of Hits per T_{Start}. For BLT we read out the FIFO before it is full. Hence, data will not be overwritten and we cannot observe wrong values for the number of Hits per T_{Start}. The size of the readout block is given by the last Start number because the

program reads out 16k:

$$\begin{aligned} \text{size of readout} &= 10\frac{\text{Hits}}{\text{Start}} \times 1488 \text{ Starts} + 1488 \text{ Starts} \\ &\approx 16k \text{ words} \end{aligned}$$

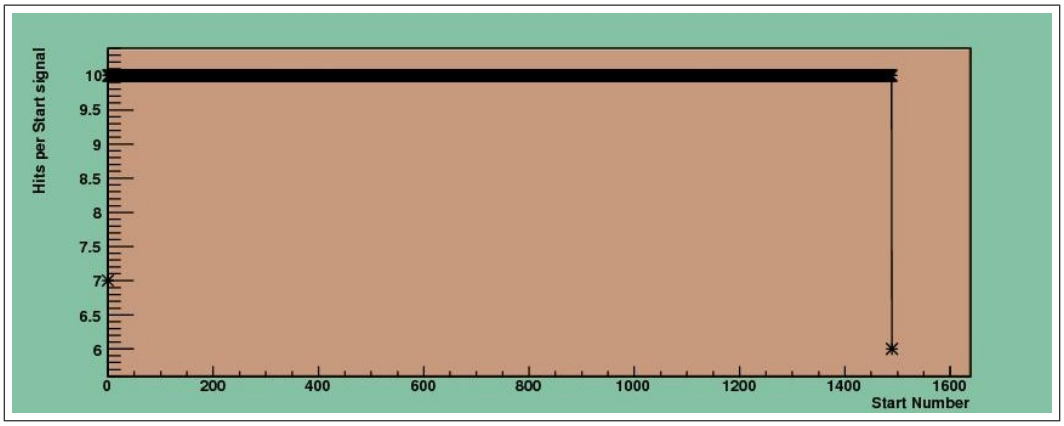


Figure 19: Hits per T_{Start} as a function of T_{Start}

Table 10 gives the time to fill the FIFO without Veto module for different Hit rates. For single memory access and without Veto module, we lost data for a Hit rate of 100 Hits/ms after 298 ms and for BLT after 148.8 ms, which is about half of 298 ms. This can be explained, since the readout size of the FIFO for single memory access is 32k words and for BLT it is 16k words. The values are, as expected, the same as given in Table 8 because the readout size is 16k.

t_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	Hit rate	time to fill the TDC FIFO (16k)
10 μs	10	100 Hits/ms	148.8 ms
2.5 μs	40	400 Hits/ms	39.9 ms
1 μs	100	1000 Hits/ms	16.1 ms
500 ns	200	2000 Hits/ms	8 ms
200 ns	500	5000 Hits/ms	3.1 ms

Table 10: Time to fill the TDC FIFO using BLT, 1 ch enabled and without Veto module

Table 11 confirms the observation when 7 channels are enabled. In addition, the time to fill FIFO for 7 channels enabled is $\frac{1}{7}$ of the time to fill the FIFO for 1 channel enabled.

t_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	Hit rate	time to fill the TDC FIFO (16k)
10 μs	10	100 Hits/ms	20.1 ms
2.5 μs	40	400 Hits/ms	5 ms
1 μs	100	1000 Hits/ms	1.9 ms
500 ns	200	2000 Hits/ms	0.5 ms
200 ns	500	5000 Hits/ms	-

Table 11: Time to fill the TDC FIFO using BLT, 7 ch enabled and without Veto module

4.4.5 Measurements using Block Transfer and with Veto Module

Going back to the measurements with single memory access, we found out that with Veto module we lose data after half the time of what we got without the Veto module. According to Table 12, however, the time to fill the FIFO using BLT is the same with and without Veto module.

t_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	Hit rate	time to fill the TDC FIFO (16k)
10 μs	10	100 Hits/ms	148.8 ms
2.5 μs	40	400 Hits/ms	39.9 ms
1 μs	100	1000 Hits/ms	16.1 ms
500 ns	200	2000 Hits/ms	8 ms
200 ns	500	5000 Hits/ms	3.1 ms

Table 12: Time to fill the TDC FIFO using BLT, 1 ch enabled and with Veto module

Figure 20 shows the comparison of the measurements without and with Veto module. The measured data are exactly the same for both conditions.

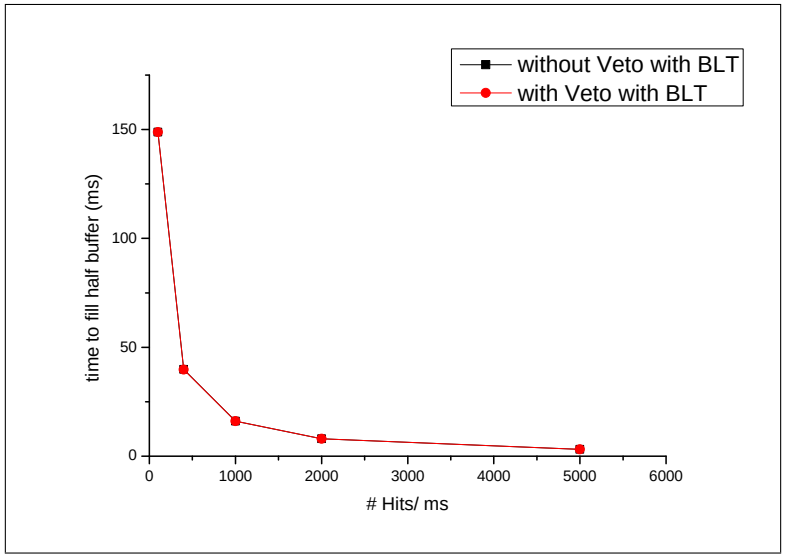


Figure 20: Time to fill the FIFO with and without Veto module for 1 channel enabled as a function of Hit rate

For BLT we can use the Veto module without any changes.

4.4.6 Time of Readout for Block Transfer

The readout time measurements with the BLT were done in the same way as for single memory access. The observation of the veto signal on the scope is illustrated in Figure 21. With the scope, the readout time for BLT is ≈ 3.76 ms.

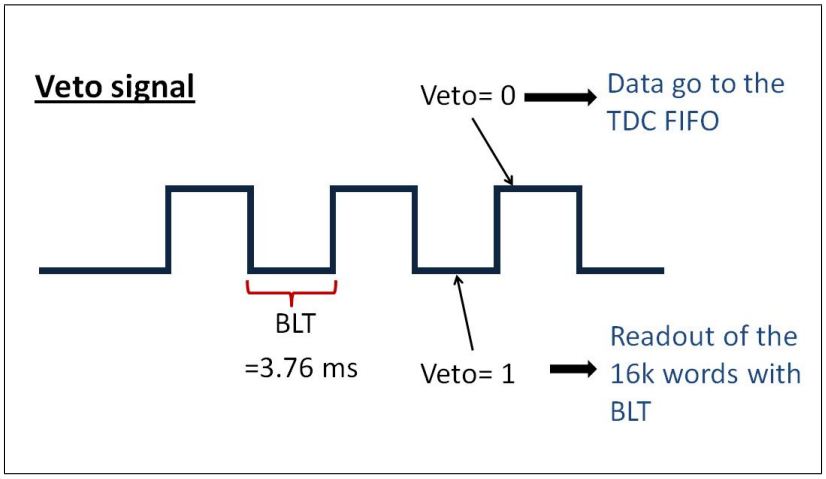


Figure 21: Readout signal using BLT observed on scope

The second time measurement of the BLT, which was done with the DAQ software, is given in Table 13. The readout with the program needs 3.65 ms and as observed before, the time of the readout is independent of the Hit rate.

t_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	time for readout of 16k for 1 ch enabled [ms]
10 μs	10	3.69
2.5 μs	40	3.55
1 μs	100	3.69
500 ns	200	3.65
200 ns	500	3.63

Table 13: Time for readout of BLT, measured with the program

To summarize, 16k can be read out in about 3.7 ms.

4.4.7 Conclusion

After the measurements with single memory access and BLT, we can conclude that the BLT is more performant than the single memory access. The disadvantage of the single memory access is the larger latency in the readout of data.
We recall the results of the readout time. The single memory access needs 229 ms and the BLT needs 3.7 ms for a transfer of 16k words. The time for readout is independent of the Hit rate.
For most of the measurements, we concentrate on the BLT because of the higher speed of the readout.
In addition, the Veto module can be used for the BLT DAQ scheme without any complications and it is working as expected.

4.5 High Performance Time-to-Digital Converter (HPTDC)

In order to compare the performance of the TDC chip integrated on the TDC v767 CAEN board, we tested the board v1290 from CAEN with an updated version of the chips: the HPTDC [12]. Table 14 shows a comparison of the old TDC and the new HPTDC [13].

Module	LSB	double hit resolution	FIFO size	BLT
v1290 (HPTDC)	800 ps	5 ns	32k	1k
	200 ps			
	100 ps			
	20 ps			
v767 (TDC)	800 ps	10 ns	32k	16k

Table 14: Comparison of LSB, double hit resolution, FIFO and BLT size between TDC and HPTDC

The new HPTDC board should improve the time resolution of the measurements. The BLT, however, can only be done in a block size of 1k words.
For the new HPTDC, we can change the time resolution (LSB) using different dynamic ranges. However, the dynamic range of the TDC should be larger than 1 orbit (89 μs). Otherwise we have to change the programs and we need other references to calculate the number of Hits. The dynamic range of the TDC for different LSBs is calculated in Table 15. It can be calculated with the LSB and the number of bits that are used for the time measurement.

time resolution (LSB)	# bits for time measurements	dynamic range
800 ps	17 bit	104 μs
200 ps	19 bit	104 μs
100 ps	19 bit	52 μs

Table 15: Time resolution and dynamic range of the HPTDC

For example, if we take a resolution mode of 200 ps we have 19 bits for the time measurement.

$$200 \text{ ps} \cdot 2^{19} = 104 \ \mu\text{s}$$

We can see in Table 15 that we can only use the 800 ps and 200 ps modes.
For the old TDC and the new HPTDC we need the same data acquisition conditions in order to compare the performance. That means for the new HPTDC we have to use the 800 ps mode because for the TDC only the 800 ps mode is available. For the DAQ of the HPTDC, we use the continuous storage. The HPTDC needs a trigger signal that is the same as the Start signal in the TDC. The trigger is applied through a normal HPTDC channel.
For the HPTDC we have a limitation, the BLT is only 1k words long. To compare the HPTDC with the TDC, we will work with a BLT of 1k words also in the TDC v767 CAEN module. To check the functionality of the HPTDC, we compare the readout using single memory access and BLT for the HPTDC and the TDC.

4.5.1 Measurements with Single Memory Access

First we checked the single memory access and the time to fill the FIFO.
Test conditions:

- 40k acquisitons
- $T_{\text{Start}} = 100 \ \mu$s
- 1 channel enabled
- LSB: 800 ps

Table 16 shows that the time to fill the FIFO is more or less the same for HPTDC and TDC. The time to fill the FIFO as a function of the Hit rate is shown in Figure 22. It is the same for both TDCs.

t_{Hit}	Number of Hits / Start	Hit rate	HPTDC time to fill FIFO [ms]	TDC time to fill the FIFO [ms]
10 μs	10	100 Hits/ms	290.8	298
2.5 μs	40	400 Hits/ms	79.8	80
1 μs	100	1000 Hits/ms	32.3	32.4
500 ns	200	2000 Hits/ms	16.2	16.2
200 ns	500	5000 Hits/ms	6.4	6.5

Table 16: time to fill FIFO (32k) for single memory access and 1 ch enabled for different Hit rates

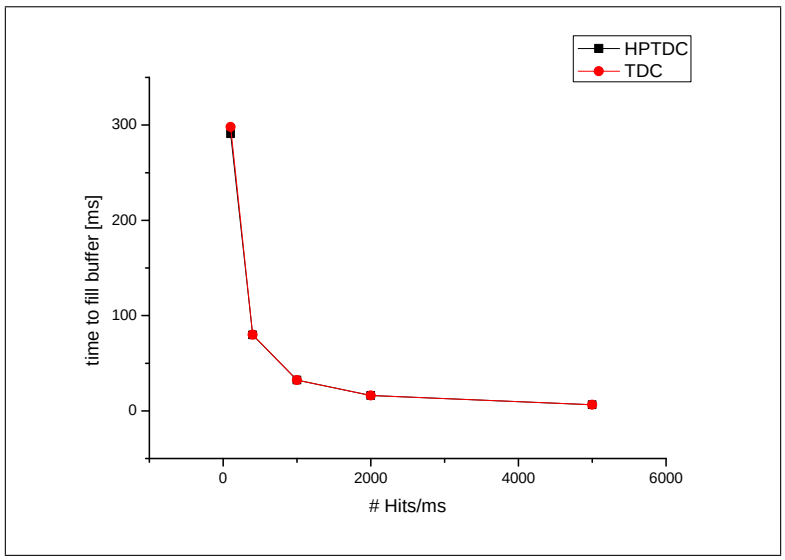


Figure 22: Comparison of time to fill FIFO (32k) with single memory access for 1 ch enabled as a function of Hit rate

For all channels enabled, we come to the conclusion that the results are the same for the HPTDC and the TDC. The results are given in Table 17 and Figure 23[2].

t_{Hit}	Number of Hits / Start	Hit rate	HPTDC time to fill FIFO [ms]	TDC time to fill the FIFO [ms]
10 μs	10	100 Hits/ms	40.3	46.2
2.5 μs	40	400 Hits/ms	10.1	11.6
1 μs	100	1000 Hits/ms	4.0	4.6
500 ns	200	2000 Hits/ms	1.9	0.6
200 ns	500	5000 Hits/ms	-	-

Table 17: Comparison of time to fill FIFO (32k) with single memory access for 8 ch enabled as a function of Hit rate

We conclude that the new HPTDC is working like the old TDC for single memory access.

4.5.2 Measurements with Block Transfer

The conditions for both TDCs should be the same. That means the BLT of the old TDC and the new HPTDC has to be set to 1k words. These measurements were only made with one channel enabled.
Test conditions:

- Block Transfer: 1024 words=1k
- $T_{\text{Start}} = 100\ \mu s$
- LSB: 800 ps

[2]The graphs are a little bit different because one channel of the TDC v767 was not working

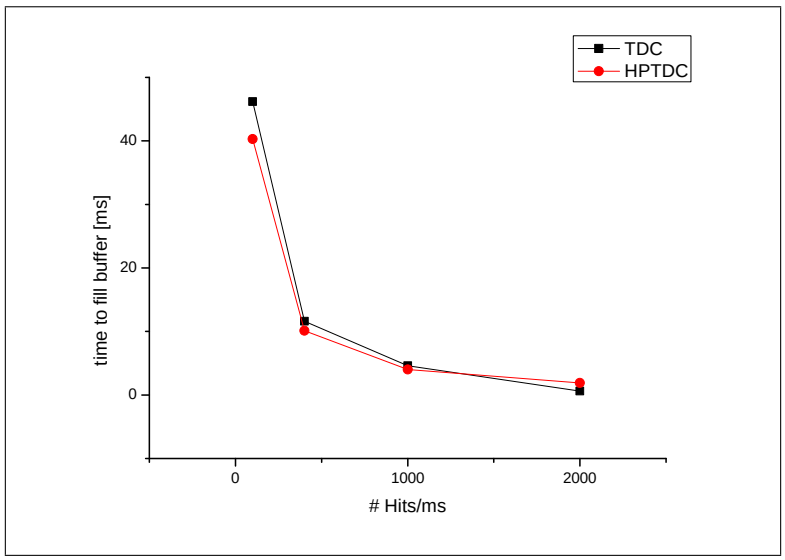


Figure 23: Comparison of time to fill FIFO with single memory access for 8 ch enabled as a function of Hit rate

T_{Hit}	Number of Hits / Start	Hit rate	HPTDC time to fill FIFO [ms]	TDC time to fill the FIFO [ms]
10 μs	10	100 Hits/ms	9.3	9.2
2.5 μs	40	400 Hits/ms	2.4	2.4
1 μs	100	1000 Hits/ms	1	0.9
500 ns	200	2000 Hits/ms	0.4	0.4
200 ns	500	5000 Hits/ms	0.1	0.1

Table 18: Time to fill the FIFO (1k) using BLT and 1 ch enabled

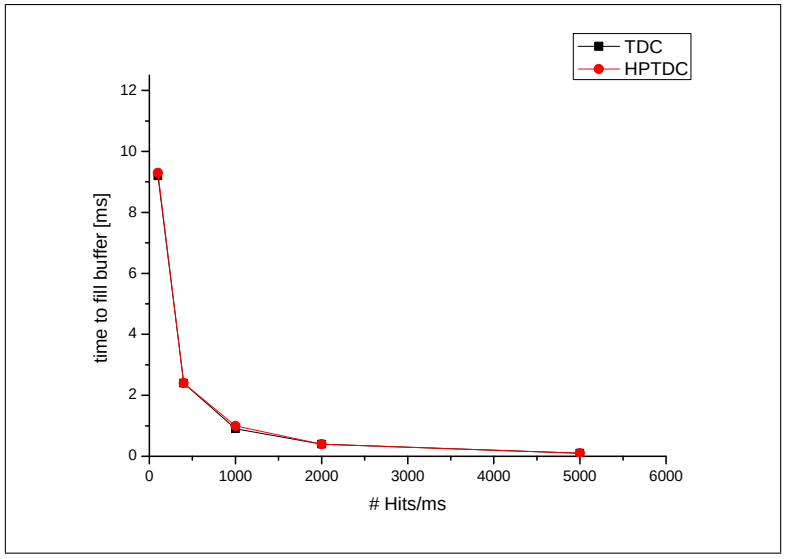


Figure 24: Comparison of time to fill FIFO (1k) with BLT for 1 ch enabled as a function of Hit rate

The time to fill the FIFO buffer is the same for both TDCs, as shown in Table 18. The comparison in Figure 24 illustrates that there are no differences between TDC and HPTDC. Using BLT of 1k the time to fill the HPTDC buffer is the same as the TDC.

4.5.3 Time of Readout

The first time measurement contains a BLT of 1k and was made with the software. For each Hit rate the time for readout was measured and the results are given in Table 19.

T_{Hit}	HPTDC time for BLT (1k)[μs]	TDC time for BLT (1k)[μs]
10 μs	264	301
2.5 μs	259	293
1 μs	262	306
500 ns	257	297
200 ns	258	313

Table 19: Time for BLT of 1k

We see that the time for a BLT of 1k is above 260 μs for the HPTDC and above 300 μs for the TDC. The HPTDC is 40 μs faster than the TDC.
In previous measurements we came to the conclusion that the BLT is independent of the Hit rate. These results can also be observed for the new HPTDC.
However, the BLT of 1k is too small because for the analysis we need the T_{Start} as a reference for the calculation of the Hits. If the Hit rate is sufficiently high, we fill 1k words with Hits and may not get one Start signal (T_{Start}) so the analysis program fails. Since we cannot transfer 16k in one BLT for the HPTDC, we loop over 16 block transfers and each block transfer is 1k words. In this way we can compare the HPTDC with a single BLT in the TDC. The time for the full readout of 16k is given in Table 20.

T_{Hit}	HPTDC time for BLT (16k)[ms]	TDC time for BLT (16k)[ms]
10 μs	4.4	3.69
2.5 μs	4.28	3.55
1 μ s	4.23	3.69
500 ns	4.28	3.65
200 ns	4.3	3.63

Table 20: Time for BLT of 16k

For a readout of 16k we need 4.3 ms for the HPTDC and 3.6 ms for TDC. In this case the HPTDC is 0.7 ms slower than the TDC.

4.5.4 Comparison of TDC and HPTDC

After all these measurements, we can find some advantages and some disadvantages for the HPTDC. An advantage is that with the HPTDC we could get higher precision of the time measurement for the Hits, up to 100 ps. Also we could get a better double hit resolution.
We are using the continuous storage with Block Transfer for our readout. With the v767 module we transfer 16k per block. However, for the v1290 module, we can read at maximum 1k which is a major disadvantage. It would be a limitation in case of high luminosity since it might happen that we get only Hits with no Start signal in a block of 1k. That means we could lose data. However, this disadvantages come from the CAEN module v1290, not from the HPTDC chip.
Table 21 shows the time for the BLT of 1k and 16k for TDC and HPTDC. The HPTDC is 40 μs faster for a readout of 1k but 0.7 ms slower for a readout of 16k because the readout needs more time for 16×1k than for 1×16k.

Size of BLT	TDC	HPTDC
1k	0,3 ms	0,26 ms
16k	3,6 ms	4,3 ms

Table 21: Time for the Block Transfer

For our purpose we prefer a BLT of 16k and in this case the HPTDC is too slow. Finally, the

disadvantages of the HPTDC outweigh the advantages, such as the double hit resolution. The better double hit resolution is not important for the application in BCM1F, since the shaping time of the preamplifier is longer than 10 ns. At the moment, it is not planned to replace the TDC with the HPTDC in the back-end part of BCM1F.

4.6 Ring Buffer

The Ring Buffer was developed by the software engineer Konstantin Boyanov (DESY-Zeuthen). A Ring Buffer is a memory area used to handle a continuous data stream and allowing at a given moment to save data of a previous time interval. Incoming data are filled successively into the memory, starting to overwrite previous content after complementation of a full cycle. The filling of new data in the buffer is starting at the beginning of the buffer. Circular buffers are typically used to hold data written by one process and read by another process. In such cases, we can pick a block of data and make the analysis.
The data structure of the Ring Buffer was developed in order to decouple the processes of data taking from the TDC and the analysis and publishing of data via the Data Interchange Protocol (DIP). In addition, the Ring Buffer will make it easier to rescale the amount of data used for data analysis and provide the history of data in case of a beam dump. The Ring Buffer class is a template C++ class that can be configured to store different numbers of objects or other classes.

4.6.1 Ring Buffer Scheme

The Ring Buffer scheme, which is illustrated in Figure 25, can be separated in two main parts that are needed for the DAQ.

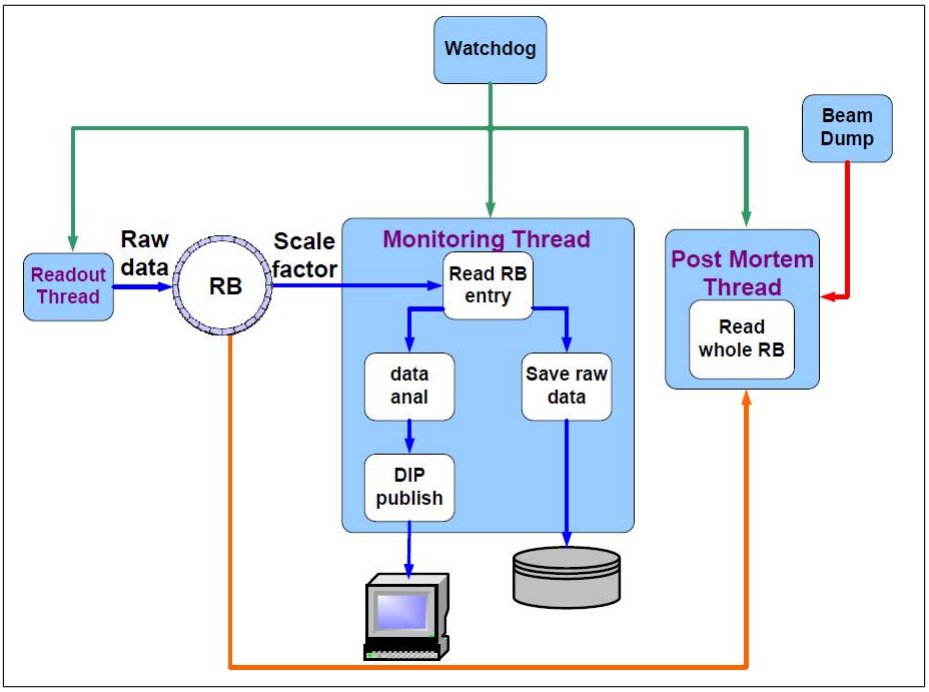


Figure 25: Scheme of Ring Buffer

The first part is the readout of the TDC. The program ”Readout.h” reads out the TDC and writes the data to the Ring Buffer. The second part is the analysis. The program ”Analysis.h” picks up data from the Ring Buffer and writes them into Root files. When the Ring Buffer is full, old data will be overwritten. A third process was developed for the case of a beam abort- the post mortem analysis. The post mortem analysis will be invoked by the beam dump condition. In case of beam abort, the TDC has to stop and the data in the Ring Buffer will not be overwritten. All data from the Ring Buffer can than be used in the post mortem analysis ”PostMortem.h”. With the post mortem analysis, the last 10 sec before beam abort can be inspected and the reason of beam abort can be analyzed.

4.6.2 Measurements with Block Transfer

The Ring Buffer was implemented in the TDC with the continuous storage mode and the BLT. In this case, the BLT goes to the Ring Buffer and then the data are taken from the Ring Buffer, processed and stored in Root files. This was tested by feeding the Ring Buffer with Hits and

observing at which time the DAQ fails.
The tests of the Ring Buffer prove the functionality of the version running at DESY-Zeuthen and if there are limitations in the use. The condition of the measurements are the same as before:

- channel 32 enabled
- $T_{\text{Start}} = 100\ \mu\text{s}$
- without Veto module
- readout block: 16k

T_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	Hit rate	time to fill the TDC FIFO (16k)
10 μs	10	100 Hits/ms	148.8 ms
2.5 μs	40	400 Hits/ms	39.9 ms
1 μs	100	1000 Hits/ms	16.2 ms
500 ns	200	2000 Hits/ms	8.2 ms
200 ns	500	5000 Hits/ms	3.4 ms

Table 22: Time to fill Ring Buffer, 1 ch enabled and without Veto module

Table 22 gives the time to fill the TDC FIFO but using Ring Buffer. The results are the same as without Ring Buffer as illustrated in Figure 26. We do not have any differences between usage with and without the Ring Buffer.

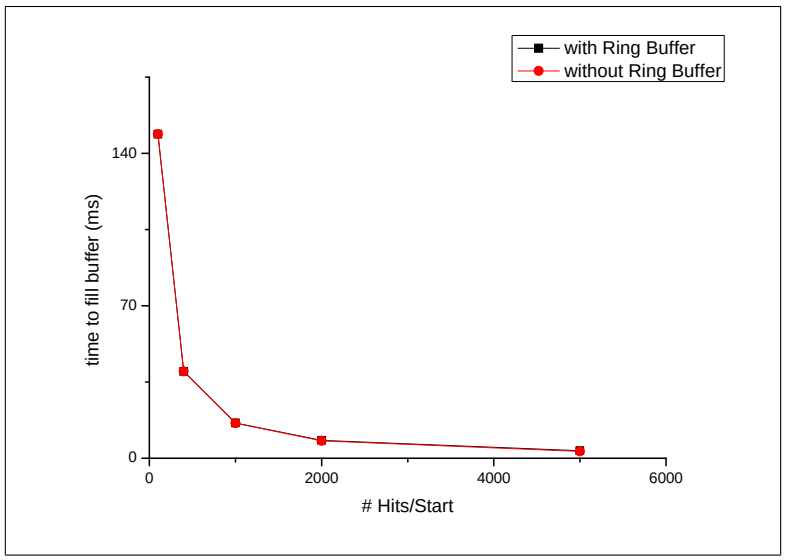


Figure 26: Comparison of time to fill the FIFO as a function of Hit rate with the usage of the Ring Buffer and without Veto module

Table 23 shows the measurements with the Veto module. Figure 27 shows that the time to fill the TDC FIFO with Ring Buffer is exactly the same as without the Veto module.

T_{Hit}	$\frac{\text{Number of Hits}}{\text{Start}}$	Hit rate	time to fill the TDC FIFO (16k)
10 μs	10	100 Hits/ms	148.9 ms
2.5 μs	40	400 Hits/ms	39.9 ms
1 μs	100	1000 Hits/ms	16.1 ms
500 ns	200	2000 Hits/ms	8.1 ms
200 ns	500	5000 Hits/ms	3.2 ms

Table 23: Time to fill Ring Buffer, 1 ch enabled and with Veto module

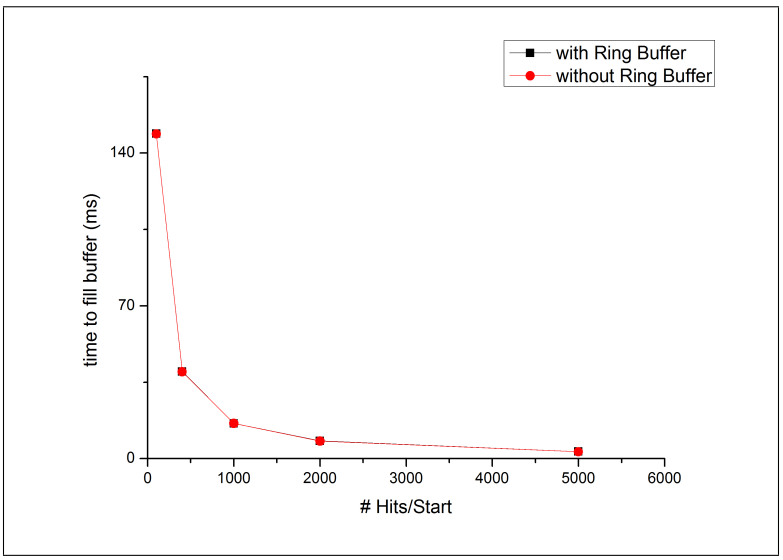


Figure 27: Comparison of time to fill the buffer as a function of Hit rate with the usage of the Ring Buffer and with Veto module

As a conclusion, the BLT with the Ring Buffer is working as the BLT without the Ring Buffer. The Ring Buffer can be used as an intermediate storage without an additional limitation of the DAQ.

5 Performance of the Discriminator

The input signals from the detector are fed into a discriminator. The discriminator applies thresholds and supplies ECL logic signals to the Scaler and the TDC. These thresholds should allow each signal that comes from a particle, e.g. collision products or beam halo, to be counted and at the same time suppress smaller signals from noise. For this task, the discriminator has to be characterized and thresholds to be set properly.

5.1 Experimental Setup

For the characterization of the discriminator, we used pulses from a pulse generator. The pulses go through the discriminator. The output of the discriminator is an ECL signal that will be converted to a NIM signal in the ECL $\leftrightarrow$ NIM converter. The NIM signal can be observed on the scope. In our measurements, we compare the discriminator output to the well-known input. We should see an output signal when the input signal is higher than the threshold.

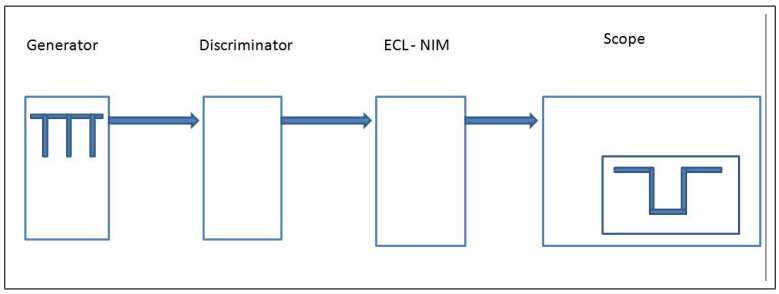


Figure 28: Experimental setup of the offset measurements

5.2 Offset in the Threshold of the Discriminator

In some measurements, we found out that we do not get an output signal from the ECL $\leftrightarrow$ NIM converter even though the amplitude of the signal from the pulse generator exceeded the threshold set. To investigate this behavior in detail, we set the threshold (V_{thr}) in the program of the discriminator to a constant value and increased the amplitude of the input pulse in the generator until we got an output signal from the discriminator. However, the output signal was flickering for amplitudes of the input pulse that were slightly higher than the threshold. To be sure that we have a stable signal, $V_{\text{in\_min}}$ is the amplitude of the input signal at which the output signal was not flickering anymore. With this method, we can find the real threshold and determine the offset of the discriminator threshold.
We measured the real threshold for each channel.
For example for channel 1/1 we calculated the offset for a threshold ranging from -10 mV till -100 mV. In Table 24 is shown the V_{thr} set in the code and the $V_{\text{in\_min}}$ for which we saw an output in the discriminator. For example: we set $V_{\text{thr}} = -10$ mV and our output is visible when $V_{\text{in\_min}} = -23$ mV. For channel 1/1 ΔV is about -13 mV.

V_{thr}	$V_{\text{in\_min}}$ for channel 1/1
-10 mV	-23 mV
-20 mV	-33 mV
-30 mV	-43 mV
-40 mV	-53 mV
-50 mV	-64 mV
-60 mV	-75 mV
-70 mV	-85 mV
-80 mV	-95 mV
-90 mV	-105 mV
-100 mV	-115 mV

Table 24: Offset of the discriminator

For a plot of the relation between $V_{\text{in\_min}}$ and V_{thr} a linear dependence can be observed. After fitting it and extrapolating the linear fit until it crosses the y-axes as shown in Figure 29, one can read the offset, V_{offset} and ΔV.

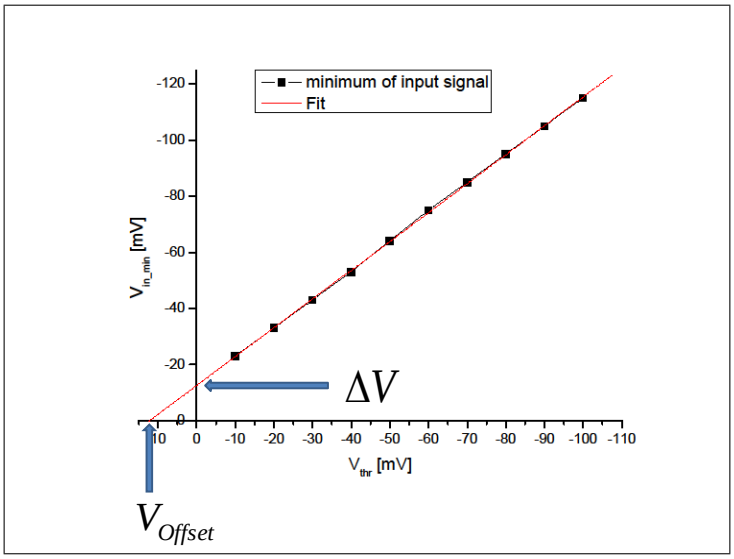


Figure 29: The voltage setting at the discriminator compared to the input voltage of the generates signal

ΔV is the value that has to be added to the threshold set to get the real threshold. The real threshold is:

$$V_{\text{realthreshold}} = V_{\text{threshold}} + \Delta V \ . \tag{12}$$

The calculation of the real threshold was done with the fit. A linear dependency is described by:

$$V_{\text{in\_min}} = a \cdot V_{\text{thr}} + \Delta V \ . \tag{13}$$

with the variable a as the slope.
ΔV is given by the following equations:

$$\Delta V = V_{\text{in\_min}}(V_{\text{thr}} = 0) \ . \tag{14}$$

For channel 1/1, the linear fit is:

$$\begin{aligned} V_{\text{in\_min}} &= 1.0327 \cdot V_{\text{thr}} - 11.8\ mV \\ \rightarrow \Delta V &= -11.8\ mV \ . \end{aligned} \tag{15}$$

At the end, you have the real threshold for channel 1/1 is:

$$V_{\text{realthreshold}} = V_{\text{threshold}} - 11.8\ mV \ . \tag{16}$$

The offset can be calculated when $V_{\text{in\_min}} = 0$:

$$\rightarrow V_{\text{Offset}} = -\frac{\Delta V}{a} \ . \tag{17}$$

If $a \approx 1$ it follows that:

$$V_{\text{Offset}} = -\Delta V \ . \tag{18}$$

In Table 25 all offsets and ΔV are listed.

channel	slope a	ΔV [mV]	V_{Offset} [mV]
1/1	1.0297	−12.466	11.43
1/2	1.0333	−12.867	12.74
1/3	1.0345	−13	12.48
1/4	1.0327	−11.8	12.57
2/1	1.0333	−11.67	12.45
2/2	1.0352	−12.067	12.1
2/3	1.0152	−12.666	11.29
2/4	1.0097	−12.866	11.66

Table 25: Offset of all channels at the discriminator

It must be pointed out that the measured offsets are only valid for the Discriminator v258B that we are using in Zeuthen. They could be different for another model. We come to the conclusion that we have to calibrate the thresholds of a discriminator.

6 TDC Measurements at LHC

At the moment, LHC is running with proton beams of 3.5 TeV energy each and intensities of up to $1.4 \cdot 10^{13}$ protons/beam. At CMS up to 140 bunches collide. The TDCs are storing time information of the Hits that are registered in the diamonds due to beam halo or collision products. The reference or Start signal, T_{Start}, is delivered by LHC and it is called "orbit-main".
The DAQ of the TDC at BCM1F is running with continuous storage, BLT and with the Veto module.

Figure 30 shows one of the plots that are being delivered to the control room of CMS: the bunch identification plot.

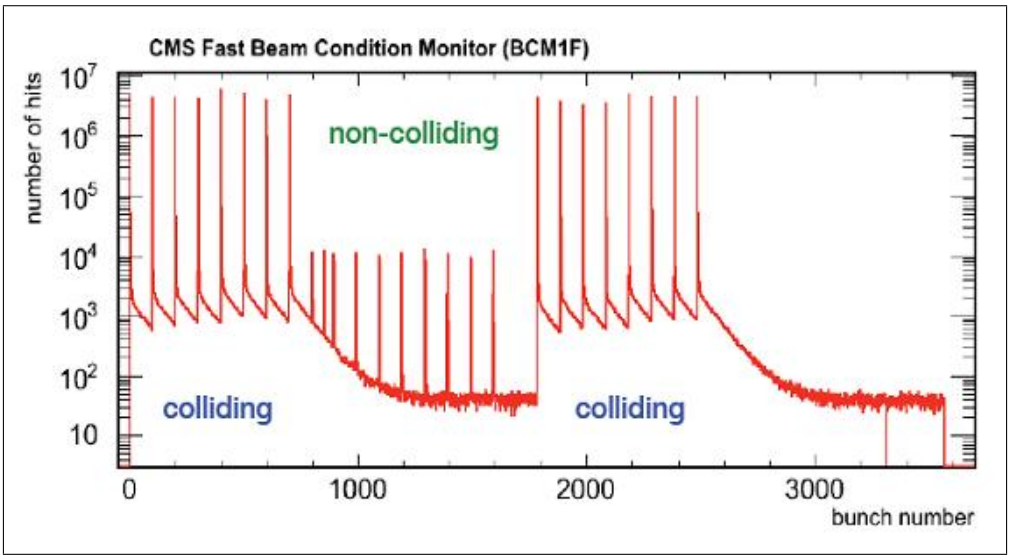
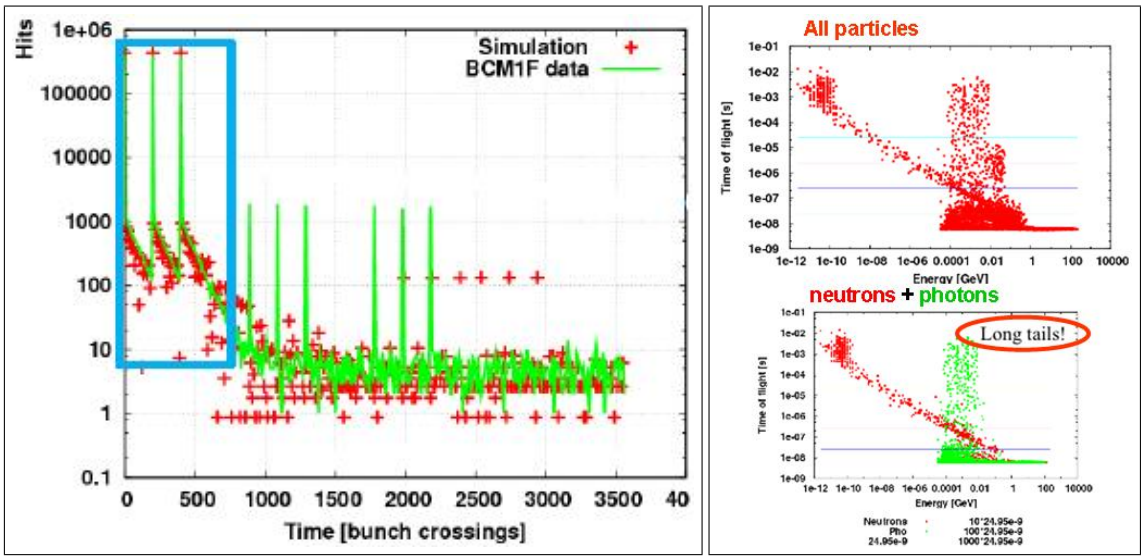


Figure 30: The count rates as a function the bunch number. The maximum bunch number correspond to the time of an orbit, 89 μs

The TDC time axis covers an entire LHC orbit. The formula used for the bunch identification is:

$$\text{bunch number} \quad = \quad \frac{t_{\text{TDC}} - 6290 \text{ ns}}{24.95 \text{ ns}} + 1 \ , \tag{19}$$

where t_{TDC} is the time of the Hit given by the TDC, 6290 ns is the time between "orbit-main" and the first bunch registered by BCM1F, and 24.95 ns is the time between two consecutive bunches. The scale of the Hits is logarithmic and some peaks are much higher than others. The first eight peaks, containing above 10^6 Hits, represent bunches that collide in CMS. The rates are high because the Hits are mainly caused by collision products. Then ten peaks follow with a low Hit rate. These are Hits that are generated by the beam halo of non-colliding bunches from the 2 beams.
BCM1F is able to separate the beam halo and the collision products as explained in Section 2.
A secondary effect can be observed in Figure 30. A long tail is after each colliding bunch. The long tails come from collision products that excite the material of the detector and subsequently decay. This effect is called after glow. An exponential fit of the tails gives a lifetime of 2.21 μs or 89 bunches. With Monte Carlo simulations done by Steffen Müller (KIT) at 7 TeV, we can identify the particles. There are mostly caused by neutrons and photons. The Monte Carlo simulations are shown in Figure 31.



(a) Monte Carlo simulation of the bunches

(b) Identification of the tails with Monte Carlo simulations

Figure 31: Monte Carlo simulations[3] of the expected Hit rate as a function of the bunch number

In addition, BCM1F was optimized to separate the beam halo coming towards the CMS interaction point from +Z and -Z direction. With the beam halo of non-colliding bunches, we can resolve the time of flight of the particles. Figure 32 shows the arrival time of beam halo particles at counters in the +Z and -Z planes.

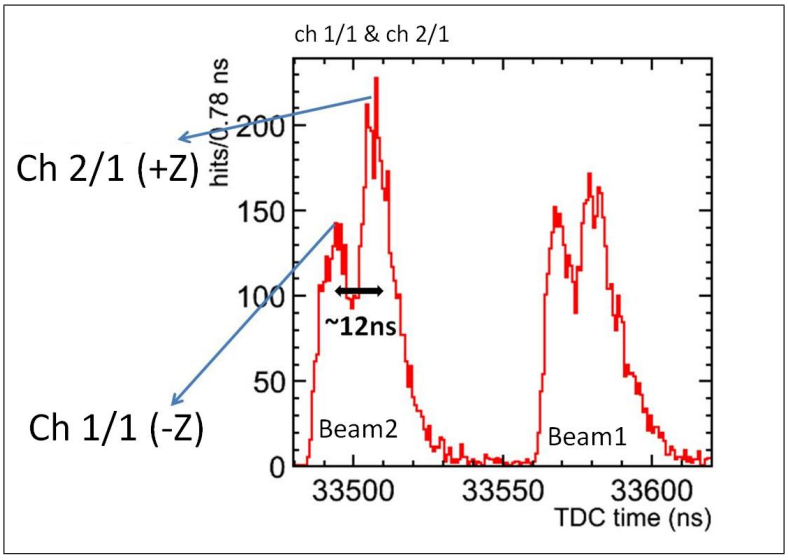


Figure 32: Time of flight measurements with ch1/1 and ch2/1

The particles hit channels in the same azimuthal angle (ch 1/1 and ch 2/1). First ch 1/1 registers Hits and then ch2/1 because a bunch of beam 2 comes from -Z side (ch 1/1) and goes to the +Z side (ch 2/1) as shown in Figure 32. The time between the two peaks of ch 1/1 and ch 2/1 is ≈12 ns. The bunch needs 12 ns to arrive from -Z to +Z because the distance of the diamonds in the channels is 3.6 m. If we consider that the protons travel with the speed of light, the proton

[3]The Monte Carlo simulations were done by Steffen Müller

needs:

$$\begin{aligned} t &= \frac{s}{v} \\ t &= \frac{3.6\ m}{3 \cdot 10^8 \frac{m}{s}} \\ t &= 12\ ns\ . \end{aligned} \tag{20}$$

To summarize, BCM1F has an excellent time resolution to separate beam halo from the luminosity of collision products and beam halo from each direction.

7 BCM1F as a Luminosity Monitor

During the last year of operation of LHC, BCM1F diamonds were detecting beam halo and collision products. The high sensitivity of the detectors and the reliability of the data acquisition lead us wonder if we could obtain further information such as luminosity. So far HF is used to measure the instantaneous and integrated luminosity of the LHC beams and provides the information to CMS. As an additional luminosity monitor, the BCM1F Scaler measurements may be used since BCM1F counts collision products at low angle where a large fraction of elastic pp (proton-proton) scattering is expected. In addition, HF is coupled to the main DAQ, while BCM1F is independent. A fail in the main DAQ stops the luminosity measurements. BCM1F is independent of the main DAQ and may give in addition a fast on-line luminosity monitoring.

7.1 Detection of Collision Products with the Look-Up Table

In order to detect collision products, we made 4 back-to-back coincidences of Hits using the scheme that is displayed in Figure 33. These coincidences should not contain the beam halo, since beam halo particles hit the counters at the same azimuthal angle.

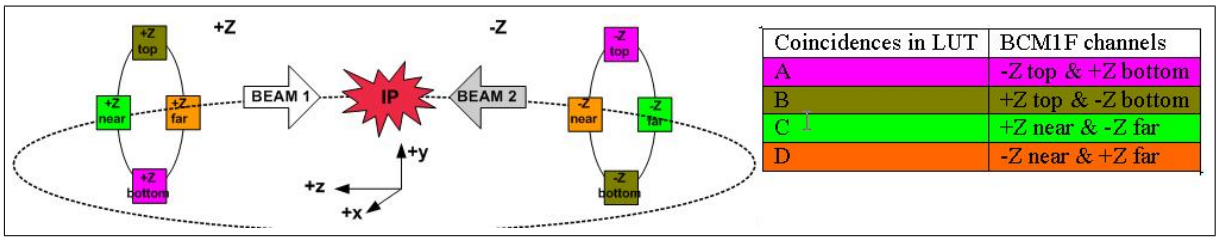


Figure 33: Coincidences in the LUT

The coincidences are built in the LUT, for example coincidences A is an AND function of Hits in -Z top and +Z bottom. The four coincidences are fed in four channels of the Scaler CAEN v560. The rate of the coincidences in the data file of the Scaler can be used for the comparison with the HF rate data files.

7.2 Comparison of BCM1F Rates with HF Rate

To use BCM1F as a luminosity monitor we assume the count rates of BCM1F (4 coincidences and 8 raw count rates) as being proportional to the HF rate.

$$\mathcal{L}_{HF} \quad = \quad a \cdot C_{BCM1F} \ , \tag{21}$$

where $\mathcal{L}_{HF}$ is the instantaneous luminosity that is given by HF, C_{BCM1F} is the sum of all coincidences that are measured with BCM1F, and a is a scale factor. For the comparison data files of HF are used. Each data file can be identify by the run number. A list of run numbers with validated runs can be found in the link: https://twiki.cern.ch/twiki/bin/viewauth/CMS/LumiWiki2010Data. We get the HF data files with instantaneous luminosity in .csv format. The files contains UTC (coordinated universal time) time, a unix time stamp and the luminosity value in units of $\frac{1}{\text{nb}\cdot\text{s}}$.
HF calculate[4] the luminosity every 23 s. Whilst, BCM1F delivers the coincidences and count rates every 1 s. For the comparison with one lumi_section, we sum up the 4 back-to-back coincidences of BCM1F over a period that is a multiple of 23 s.
In addition, we sum up the count rates of the four channels in +Z plane and the four channels in -Z plane and compare also these with the luminosity given by HF.

7.3 First Results

The first comparison was done with the run on August 6th 2010 and it is shown in Figure 34.
A fit is included for the coincidences for better comparison with HF.

[4]HF calculate the luminosity/rate with the Van-der-Meer scans

The HF rates are scaled by a factor of 200, which is why the scale of the luminosity is $\frac{200}{nb \cdot s}$. The count rates of +Z and -Z were divided by a scale factor of 165 to bring the two curves close to each other.

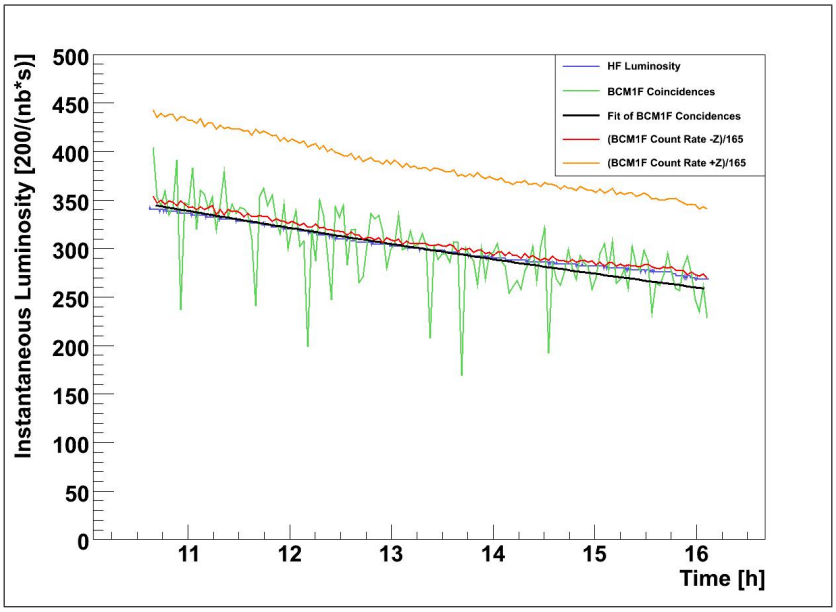


Figure 34: The luminosity values given by HF and the BCM1F count rates as a function of time from August 6th

It has to be pointed out that the same scale factors are taken for all plots.

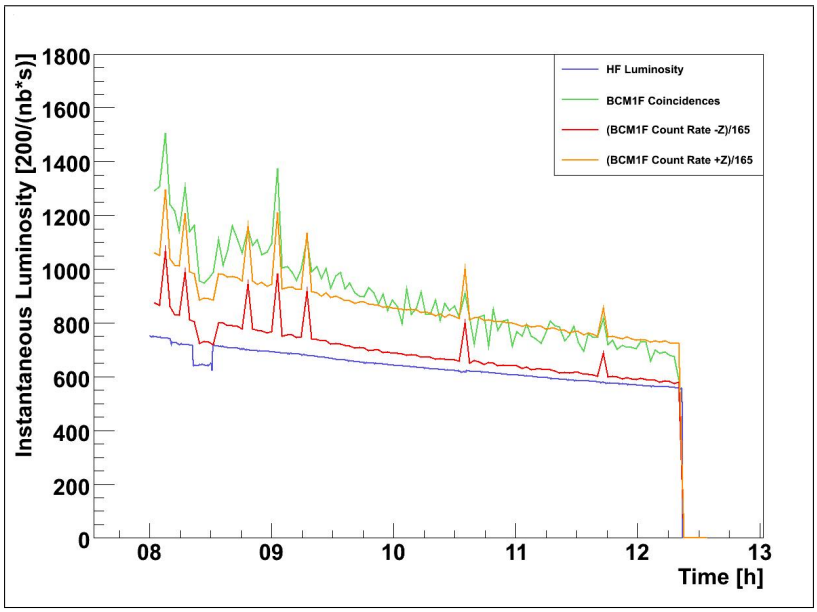


Figure 35: The luminosity and the BCM1F count rates as a function of time from August 10th 2010. The large peaks in BCM1F data are caused by a fail in the readout of the Scalers.

Figure 35 displays the run on August 10th 2010. In this case the coincidence rate and the count rates at the beginning of the run are much higher than the HF data. Towards the end of the fill the rates approach to each other. For this run, a recalibration of the scale factor is needed.
The run on August 26th 2010 that is given in Figure 36 shows a different behavior than the previous ones. The coincidences and count rates are higher than the HF data at the beginning of the fill. After a time of 6 hours, the BCM1F count rates are similar to the HF rate. The BCM1F

coincidences have a different slope than the count rates or the HF rate. This observation needs to be investigated.

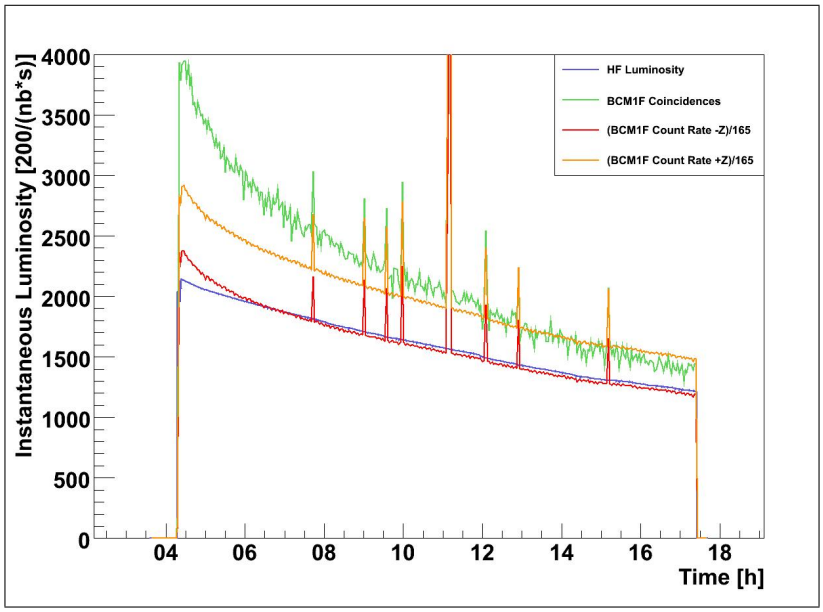


Figure 36: The luminosity and the BCM1F count rates as a function of time from August 26th 2010. The large peaks in BCM1F data are caused by a fail in the readout of the Scalers.

The last run, Figure 37 shows a run on August 29th 2010. In this case, all slopes are similar. Only the height of the coincidences is different to the HF rate, which is not explainable because the calibrations for all the runs are the same. Nevertheless, the count rates and the HF rate have similar time dependences.

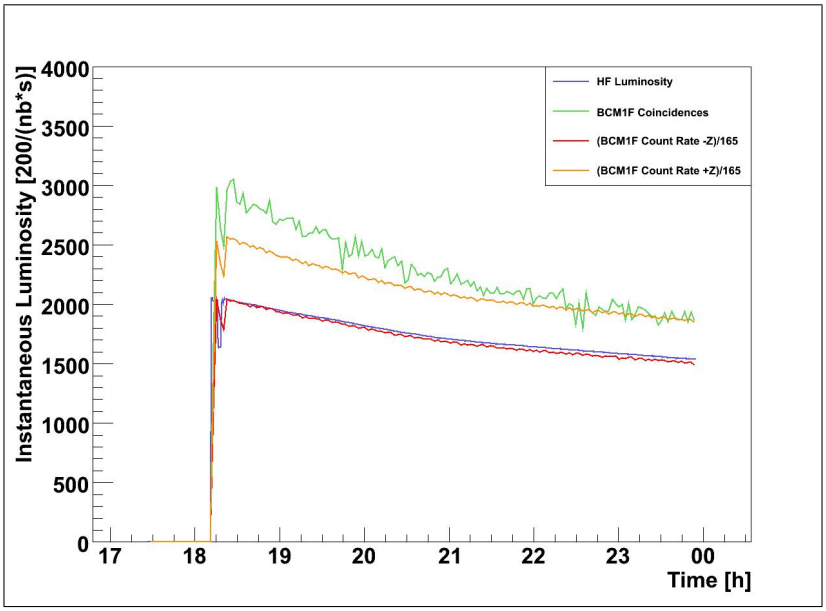


Figure 37: The luminosity and the BCM1F count rates as a function of time from August 29th 2010

It needs to be investigated why the slope of the BCM1F coincidences and the count rates are different in some runs and why the BCM1F coincidences and count rates are higher than the HF rate in some runs. The use BCM1F as a luminosity monitor is still under investigation.

8 Conclusion

CMS consists of different sub-detectors. One sub-detector is BCM1F that is vital for monitoring the beam conditions. The readout at back-end of BCM1F contains several modules to handle the raw data - a TDC, a discriminator and a Veto module. These modules were tested and there performance measured.
Two different readout modes are available for the TDC: single memory access and BLT. Tests showed that the BLT is much faster than the single memory access and the time for the readout is independent of the Hit rate. However, the time to fill the FIFO for the same Hit rate is independent of the readout mode. The TDC will be read out with BLT since the BLT is much faster than the single memory access.
Additionally, a Veto module can be used to avoid the overwriting of data not yet readout. The Veto module needs an external signal that activates Veto signal during the whole BLT. The LUT provides the Veto signal during the BLT. Due to the signal of the LUT the veto module works as expected and without any limitations.
For the TDC readout a Ring Buffer is necessary in case of high data rates. The software of the Ring Buffer was tested with the readout mode BLT. The performance is exactly the same as without the Ring Buffer. Additional limitations cannot be observed.
A second module of the BCM1F setup was tested- the discriminator. Each discriminator channel has his own offset voltage. Tests showed an offset of 12 mV but the offset is only available for the discriminator at DESY-Zeuthen. Hence, before using a discriminator, the thresholds of all channels must be calibrated.
The TDC with the current settings is used in BCM1F in CMS at LHC and it measures the arrival time of Hits. The count rate over an entire LHC orbit can be measured as a function of time.
In this way BCM1F can identify colliding bunch and non-colliding bunches. Since in the case of colliding bunches the count rates is by orders of magnitude larger due to the collision products.
Due to the high time resolution BCM1F can separate the beam halo moving towards the CMS interaction point from each direction.
In the last chapter are the first results to use BCM1F as a luminosity monitor. In this study we compare the HF data with BCM1F count rates and back-to-back coincidences.
First results show a proportionality between these quanties. However, the possibility to use BCM1F as luminosity monitor is still under investigation.

List of Figures

List of Tables

References

[1] found on 13th August
http://public.web.cern.ch/public/en/LHC/LHC-en.html

[2] The CMS Collaboration, S. Chatrchyan, et al., The CMS experiment at the CERN LHC
JINST 3S08004, 2008

[3] CMS Physics, Technical Design Report
found on 17th August
http://cdsweb.cern.ch/record/922757/files/lhcc-2006-001.pdf

[4] A. Bell et al., Fast beam conditions monitor BCM1F for the CMS experiment
Nuclear Instruments and Methods in Physics Research A 614 (2010) 433-438

[5] B. Povh et al., Particles and Nuclei: An Introduction to the Physical Concepts
4th Edition, 2004

[6] S. van der Meer, Calibration of the Effective Beam Height in the ISR
CERN-ISR-PO/68-31, 1968

[7] Technical Information Manual
CAEN Mod. V258B (DISCRIMINATOR)

[8] Technical Information Manual; Revision n.4
CAEN Mod. v767 (128 CH. GENERAL PURPOSE MULTHIT TDC)

[9] J.Christiansen CERN/ECP, 32 Channel general purpose Time to Digital Converter
Version 1.0, January 1997

[10] Technical Information Manual; Revision n.8
CAEN Mod. V1495(GENERAL PURPOSE VME BOARD)

[11] Technical Information Manual; Revision n.1
CAEN Mod. V560 (16 CHANNEL SCALERS)

[12] J. Christiansen CERN/EP - MIC, HPTDC - High Performance Time to Digital Converter
Version 2.2, March 2004

[13] Technical Information Manual; Revision n.11
CAEN Mod. V1290 A/N (32/16 CHANNEL MULTIHIT TDC's)

A TDC_BufferAlmostFull.cc

```
/*************************************************/
/*Description: program to read out data from buffer*/
/*in each vme access we read 1 datum, we write it  */
/*to CPU mem and later to a txt file               */
/*************************************************/

#include <iostream>
#include <iomanip>
#include <fstream>
#include "tdc_v767_caen.hh"
#include <sys/time.h>
#include <time.h>
#include <string>

#define lred "\033[1;49;31m"
#define norm "\033[0m"


#define coutDefault      "\033[0m"
#define coutRed          "\033[31m"
#define coutGreen        "\033[32m"
#define coutBlue         "\033[34m"
#define coutPurple       "\033[35m"
#define coutUnderLine    "\033[4;30m"

using namespace std;


TDC_V767_CAEN *b;

unsigned int buffer_level=0x3FFF;      //range [0x2:0x3FFF]

unsigned int Test_StartCounter, Test_EventCounter = 0;

//-------------------------------------------------------------
// Setup TDC parameters for DAQ
//-------------------------------------------------------------
int set_tdc_board()
{
 //SET UP TDC MODE
  b->Reset();
  sleep(2);
  b->OPCODE_Enable_AutoLoad();
  sleep(1);
  b->OPCODE_Load_Default_Config();
  sleep(1);
  b->OPCODE_Disable_allChannel();
  sleep(1);
  b->OPCODE_CONT_STO();
  sleep(1);
  //b->OPCODE_Set_BufferNotEmpty();
  //Data Ready asserted if there is at least 1 Datum in memory
  //sleep(1);
  b->OPCODE_Set_AlmostFullLevel(buffer_level);
  //Set memory level
  sleep(1);
  b->OPCODE_Set_BufferAlmostFull();
   //Data Ready asserted if there is at least n Datum in memory
  sleep(1);

  b->OPCODE_Enable_ReadoutOfStarttime();
  sleep(1);
  //b->OPCODE_Disable_ReadoutOfStarttime();
  //Start data is not stored in FIFO
  //sleep(1);
  b->OPCODE_Enable_SubtractionOfStarttime();
  sleep(1);
  b->OPCODE_Set_FallingEdgeOnlyOnAllChannels();
  sleep(1);
  b->OPCODE_Set_RisingEdgeOnlyStart();
```

```
  sleep(1);

  //b->OPCODE_Enable_allChannel();
  //sleep(1);


  b->OPCODE_Enable_Channel(32);
  sleep(1);


  b->Clear_Register();
  sleep(2);

return 0;

}


//------------------------------------------------------------
// Readout of data
//------------------------------------------------------------
int readout_data()
{
  //OUTPUT BUFFER
  unsigned long out_data=0x09;
  unsigned long *p_out_data;
  p_out_data=&out_data;


  //OUTPUT FILE
  FILE *output;
  output=fopen("blabla.txt", "w");



/***********Check DATA READY before reading out FIFO***********/
  while(!(b->Is_Data_Ready())){
  //DREADY =1 when Buffer Almost Full level is reached in FIFO
//while(!(b->Is_Busy())){   //DREADY when FIFO is Full
//usleep(500000);

if(b->Is_Global_TDC_Error()){
for(unsigned int j=0; j<4; j++)
if(b->Is_TDC_Error(j)){
cerr << j << "     " << b->OPCODE_Get_ErrorCode_TDC(j) << endl;
                                                    }
break;
}
 }
 //cerr << "\tData is fine " << endl;
/*****************************************************/


for(unsigned int i=0; i<1000; i++)
//while((b->Is_Data_Ready()))
//loop while there is at least 1 Datum in memory
{
//READ OUTPUT BUFFER
 b->Read_Output_Buffer(p_out_data);  //1 datum per vme access

if(!(out_data & 0x600000))  //Check if VALID DATUM
fprintf(output, "%lx\n", out_data);  //Save to file

//cout << "Buffer output data =0x " << hex << *p_out_data << endl;

//if(!(out_data & 0x600000))  //Check if VALID DATUM
//{fprintf(output, "%lx\n", *p_out_data);  //Save to file
//}

}
```

```
            fclose(output);

 return 0;
}


/*-------------------------------------------------------------------*/
int main(int argc, char * argv[]){


  b = new TDC_V767_CAEN(0x99990000,"TDC_V767_CAEN",1,0);

  set_tdc_board();

  readout_data();

  delete b;

  return 0;
}
```

B BLT_tree_vmartin.cc

```
/*************************************************/
/*Description: program to read out data from buffer*/
/*and write data in ascii mode to file           */
/*************************************************/
/*
  -side  ADCch  TDCch  |  +side  ADCch  TDCch
  1/1     0       0    |   2/1    4       2
  1/2     1       32   |   2/2    5       33
  1/3     2       66   |   2/3    6       65
  1/4     3       96   |   2/4    7       97
 */
/*************************************************/

#include "TROOT.h"
#include "TFile.h"
#include "TTree.h"
#include "TH1S.h"


#include <iostream>
#include <iomanip>
#include <fstream>
#include <sys/time.h>
#include <time.h>
#include <string>

#include <cmath>
#include <sstream>

#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/stat.h>teached me
#include <sys/types.h>
#include <unistd.h>

#include "tdc_v767_caen.hh"

#define lred "\033[1;49;31m"
#define norm "\033[0m"


#define coutDefault     "\033[0m"
#define coutRed         "\033[31m"
#define coutGreen       "\033[32m"
#define coutBlue        "\033[34m"
#define coutPurple      "\033[35m"
#define coutUnderLine   "\033[4;30m"


//Maximum
#define MAXSAVESIZE 2000  //Bytes

//-Z sideteached me
#define CH11 0   //ch0 TDC
#define CH12 32  //ch32 TDC
#define CH13 64  //ch64 TDC
#define CH14 96  //ch96 TDC


//+Z side
#define CH21 1   //ch1 TDC
#define CH22 33  //ch33 TDC
#define CH23 65  //ch65 TDC
#define CH24 97  //ch97 TDC




//BLTMAX is the size of the BLT
```

```
#define BLTMAX 16384
//#define BLTMAX 2048

#define CH_ANAL 32



using namespace std;

//TDC board parameters
TDC_V767_CAEN *b;// ADC handle
unsigned long baseaddress=0x99990000;// 1 : baseaddress
short vme_link=1;                    // 2 : vme link
short board_no=0;                   // 3 : PCI board
string DO_Path="./root_files";   // 4 : data output path
string DO_Path2="./txt_file";   // 4 : data output path


unsigned long BLTData[32500];     //array to store the data in each BLT
unsigned long * pBLTData;     //array to store the data in each BLT

timeval tot_time, run_time;
unsigned long header[2];         // long statt int !!   // General Event header
unsigned int  not_valid_datum=0;

// copied to in case of writing/treatment
unsigned long TotalData[0x1000010];  //storage of the whole data of BLT

//Parameters related with creation of ROOT Trees and Files
unsigned int NHit=0;  //counter for +Z and -Z hits
unsigned int NHitm=0;  //counter for -Z hits
unsigned int NHitp=0;  //counter for +Z hits

TTree *t;
TFile *f ;

//Hit info
unsigned int tdc_ch[BLTMAX]={0};
double hit[BLTMAX]={0};
double minus_hits_orbit[BLTMAX]={0};  //distribution of Hits in an orbit for -Z
double plus_hits_orbit[BLTMAX]={0};   //distribution of Hits in an orbit for +Z

unsigned int Channel=0;

//Start info
double start=0;
double prev_start=0;
double delta_start=0;
double hits_start[BLTMAX]={0};

unsigned int NStart=0;
double corr_start=0;
unsigned int TDC_counter_restarts=0;
double time_start=0;
double total_time_start=0;


//Nwords is the nr of 32 bit words transferred per BLT access
unsigned long Nwords=16383; //Maximum nr of 32bit words per BLT = 16383
//unsigned long Nwords=2047; //Maximum nr of 32bit words per BLT = 16383

int ret=0;

unsigned long buffer_level=0x3FFF;
//Set Almost Full level (range [0x2:0x3FFF])
//unsigned long buffer_level=0x20;
//Set Almost Full level (range [0x2:0x3FFF])

string txt_filename;

//-----------------------------------------------------------------------------
// Setup TDC parameters for DAQ
```

```
//-----------------------------------------------------------------------------
int set_tdc_board()
{
 //SET UP TDC MODE
  b->Reset();
  sleep(2);
  b->OPCODE_Enable_AutoLoad();
  //Loads automatically the User Config at Power ON or General RESET
  sleep(1);
  b->OPCODE_Load_Default_Config();//Loads User Config previously saved
  sleep(1);

  b->OPCODE_CONT_STO();  //Set Continuous Storage Mode
  sleep(1);
  b->OPCODE_Set_AlmostFullLevel(buffer_level);
  //Mem level that fires Buffer Almost Full
  sleep(1);
  b->OPCODE_Set_BufferAlmostFull();
  //Data Ready Mode. Asserted when Almost Full Level is reached
  sleep(1);
  b->OPCODE_Enable_ReadoutOfStarttime();
  sleep(1);
  b->OPCODE_Enable_SubtractionOfStarttime();
  sleep(1);
  b->OPCODE_Set_FallingEdgeOnlyOnAllChannels();
  sleep(1);
  b->OPCODE_Set_RisingEdgeOnlyStart();
  sleep(1);


  //Disable all Channels before enabling desired ones
  b->OPCODE_Disable_allChannel();
  sleep(1);

  //Enable desired Channels
  //TDC A
  b->OPCODE_Enable_Channel(CH11);
    sleep(1);
    b->OPCODE_Enable_Channel(CH21);
    sleep(1);

  //TDC B
  b->OPCODE_Enable_Channel(CH12);
  sleep(1);
  b->OPCODE_Enable_Channel(CH22);
  sleep(1);

  //TDC C
  b->OPCODE_Enable_Channel(CH13);
  sleep(1);
  b->OPCODE_Enable_Channel(CH23);
  sleep(1);

  //TDC D
  b->OPCODE_Enable_Channel(CH14);
  sleep(1);
  b->OPCODE_Enable_Channel(CH24);
  sleep(1);


  b->OPCODE_Save_User_Config();
  //Saves Acq Mode, Enabled Channels, Data Ready mode
  sleep(1);


  b->Clear_Register();
  sleep(2);

return 0;

}

//-----------------------------------------------------------------------------
```

```
// provides the unique data_outputfile name
//  {DO_Path}/data_(datrt date/time)_(time in usec).root
//-------------------------------------------------------------------------------
string create_file_name(){
  timeval writetime;
  gettimeofday(&writetime,0);


//create name of ROOT file
ostringstream os1;
os1 << DO_Path << "/tdc_data_"
<< string(ctime((time_t *)&writetime.tv_sec))
.substr(0,string(ctime((time_t *)&writetime.tv_sec)).length()-1)
      << "_" << writetime.tv_usec << ".root";

  string filename = os1.str();


while(filename.find_first_of('␣',0)<filename.length())
filename.replace(filename.find_first_of('␣',0),1,1,'_');
while(filename.find_first_of(':',0)<filename.length())
filename.replace(filename.find_first_of(':',0),1,1,'.');


//create name of TXT file
ostringstream os2;
os2 << DO_Path2 << "/tdc_data_"
<< string(ctime((time_t *)&writetime.tv_sec))
.substr(0,string(ctime((time_t *)&writetime.tv_sec)).length()-1)
<< "_" << writetime.tv_usec << ".txt";

txt_filename = os2.str();  //Note: txt_filename declared as global variable

while(txt_filename.find_first_of('␣',0)<txt_filename.length())
txt_filename.replace(txt_filename.find_first_of('␣',0),1,1,'_');
while(txt_filename.find_first_of(':',0)<txt_filename.length())
txt_filename.replace(txt_filename.find_first_of(':',0),1,1,'.');


  return filename;
  }



//-------------------------------------------------
// set Branch of tree
// Open a new File (must be closed before)
//-------------------------------------------------
void ChangeFile(){


//f = new TFile("TDCblt.root","RECREATE");//create the Tree file
f = new TFile(create_file_name().c_str(),"RECREATE");
t = new TTree("T","Tree");

//Header of file info
t->Branch("BLT_time_sec_nsScale",&header[0],"time_sec/i");
t->Branch("BLT_time_usec_nsScale",&header[1],"time_usec/i");

//Start info
t->Branch("nbrOrbits"    ,&NStart    ,"NStart/i");  //Total number of Starts in the BLT
t->Branch("DeltaOrbit",&delta_start,"delta_start/d");//create Branches of Tree
t->Branch("TimeOrbitInTDCcounter",&time_start,"time_start/d");
t->Branch("TotalTimeDAQ",&total_time_start,"total_time_start/d");

//Hit info
t->Branch("HitsperOrbit"    ,&NHit    ,"NHit/i");  //Total number of Hits in the BLT
t->Branch("PlusZhitsPerOrbit"    ,&NHitp    ,"NHitp/i");  //Total number of Hits in the BLT
t->Branch("MinusZhitsPerOrbit"    ,&NHitm    ,"NHitm/i");  //Total number of Hits in the BLT
t->Branch("TdcCh",tdc_ch,"tdc_ch[NHit]/i");  //Distribution of Hits in TDC channels
//t->Branch("HitRawData",hit,"hit[NHit]/i");  //Raw Hit info (channel+time)
t->Branch("HitTimeInOrbit",hit,"hit[NHit]/d"); //Only time Hit time info
```

```
//Start + Hit info together
t->Branch("DistributionHitsInDAQ",hits_start,"hits_start[NHit]/d");
t->Branch("PlusZhitsInOrbit",plus_hits_orbit,"plus_hits_orbit[NHitp]/d");
//Distribution of Hits in orbit for +Z
t->Branch("MinusZhitsInOrbit",minus_hits_orbit,"minus_hits_orbit[NHitm]/d");
//Distribution of Hits in orbit for -Z


  }

//-----------------------------------------------
// Function for storage and processing of data
//-----------------------------------------------
void process_Data(){
NHit=0;
NHitm=0;
NHitp=0;
NStart=0;
TDC_counter_restarts=0;

//------------WRITE DATA TO txt FILE
 ofstream of(txt_filename.c_str());


 bool first=true;
 for(int i=0; i<BLTMAX; i++){//loop over all Block of Data

if(i==BLTMAX-1)
{
t->Fill();  //store last data in tree
t->Write("",TObject::kOverwrite);
// RSS: this ensures that a complete BLT is saved in same file
// and is not splitted in several files
f->Write();
break;
}

if(!(TotalData[i] & 0x600000)){  //check for valid datum
 //cerr<< "TotalData=0x " << hex <<TotalData[i]<< endl;
     of<< hex << TotalData[i] <<dec<<endl; //write data to file

/***********************************/
 if(TotalData[i] & 0x800000){ //check for Starts, not Hits
  if(first==false){
   t->Fill();  //Get all parameters of 1st event before filling Tree

    // debug info only
    //t->Write("",TObject::kOverwrite);             // RSS
    //cout<< "sizeFile="<< f->GetSize() << endl;    // RSS

    }else{
    first=false;
    }

    //Read raw start data
   start=((double)(TotalData[i] & 0xFFFFF))*25/32;
   //convert into ns, zero time with respect to DAQ restart
        //cout << " START=" << dec << start  << " ns " << endl;

 //Time difference between consecutive Start signals
if(NStart==0)
{prev_start=start;//Initialize prev_star for calculating Delta Start

corr_start=start;  //correction needed to normalize time to TDC time 0 point
time_start=start-corr_start;  //plot time of Start over the TDC counter.

total_time_start=start;  //plot time of Start with respect to TDC reset
//cout << " START=" << dec << total_time_start  << " us " << endl;
}
else{
if(start<prev_start){   //TDC counter restarts
//cout<< "COUNTER RESTARTS"<<endl;
delta_start=(start-prev_start)+(0xFFFFF)*25/32;
```

```
corr_start=start;

TDC_counter_restarts++;
}
else{
delta_start=(start-prev_start);
}
prev_start=start;
//}
time_start=start-corr_start;  //plot time of Start over the TDC counter.
total_time_start=(start+TDC_counter_restarts*(0xFFFFF)*25/32);
//plot time of Start with respect to TDC reset
//cout << " START=" << dec << total_time_start  << " us " << endl;
//total_time_start=start;  //plot time of Start with respect to TDC reset

}


//cout<<"number of hits in Minus side per orbit="<<dec<< NHitm <<endl;

NHit=0;
NHitm=0;
NHitp=0;
//cout << " NBR STARTS=" << dec << NStart  << endl;

NStart++;  //1 event starts with a new START signal

}  //end of checking if Start
 /**********************************/

else if(!(TotalData[i] & 0x800000)){  //check for Hits, not Starts
//cout<< "HIT"<<endl;

//Extract Hit channel
Channel=(TotalData[i]>>24) & 0x7F;//create variable for branch in Tree

//if(Channel==CH_ANAL){

//hit[NHit]=TotalData[i];  //raw info of hit
hit[NHit]=(double(TotalData[i]&0xfffff)*25./32.);
 //calculate hit time with respect to START

tdc_ch[NHit]=Channel;  //tdc channel of hit
//Hit_raw_time[NHit]=(double(TotalData[i]&0xfffff)*25./32.);
//calculate hit time with respect to START

hits_start[NHit]=total_time_start+hit[NHit];
//cout << " DISTRIBUTION OF HITS IN START=" << dec << hits_start[NHit+1]  << " us " << endl;

 NHit++;

//}  //end check of channel


//-Z side
if(Channel==CH11 | Channel==CH12 | Channel==CH13 | Channel==CH14){
minus_hits_orbit[NHitm]=(double(TotalData[i]&0xfffff)*25./32.);
 //cout << "-Z time in orbit=" << dec << minus_hits_orbit[NHitm]  << " us " << endl;

NHitm++;
}
//+Z side
if(Channel==CH21 | Channel==CH22 | Channel==CH23 | Channel==CH24){
plus_hits_orbit[NHitp]=(double(TotalData[i]&0xfffff)*25./32.);

NHitp++;
}

}  //end of checking if Hit

} //end of checking if NOT VALID DATUM
else{
```

```
cout << "␣NOT␣VALID␣DATUM" << endl;
not_valid_datum++;
   }

/**********************************/

 } //end of for loop in whole BLT data


 of.close(); //close txt file
 cerr << "␣Found␣" << NStart << "␣Starts␣in␣the␣BLT␣and␣wrote␣it␣to␣tree" << endl;
 cerr << "␣TDC␣counter␣restarts=␣␣" << dec << TDC_counter_restarts << endl;

}



//-------------------------------------------------------------
// Perform readout of data using BLT
//-------------------------------------------------------------
int readout_data()
{



unsigned int bytes=0;
bool last=false;
unsigned long NwordsLeft=Nwords;
//unsigned long Nwords=16383;
//Maximum nr of 32bit words per BLT = 16383
 unsigned long TotalNwords=0;
unsigned int index=0;


/***********Check DATA READY before reading out FIFO***********/
/*Waits for DREADY set to 1*/
while(!(b->Is_Data_Ready())){
//loop while the Buffer Almost Full level is not reached
if(b->Is_Global_TDC_Error()){
for(unsigned int j=0; j<4; j++)
if(b->Is_TDC_Error(j)){
cerr << j << "␣␣␣␣␣" << b->OPCODE_Get_ErrorCode_TDC(j) << endl;
                                                    }
//break;
  }
   }
//cerr << "\tData is fine " << endl;
/*********************************************************/



gettimeofday(&run_time, 0);                    // Eventtime
header[0]=run_time.tv_sec;
header[1]=run_time.tv_usec;

while(true){
 //READ OUTPUT BUFFER

ret=b->BLTRead_Output_Buffer(BLTData, NwordsLeft);
//BLT of 32bit Nwords
//cerr<< "Nr transferred bytes= " <<dec<< ret <<"\t Nr transferred words="<<
dec<<ret/4<< endl;

if(last || (ret<=0)) //check if previous BLT was the last one
{memcpy(&TotalData[index],BLTData,ret);    //copy last BLT to memory
 bytes+=ret;
 TotalNwords+=NwordsLeft;
 break;}

// Copy Data
//memcpy(&TotalData[bytes],BLTData,ret);
```

```
 memcpy(&TotalData[index],BLTData,ret);

//cout<< "index=0x" << hex << index <<"\t TotalData[index]=0x " <<
hex <<TotalData[index]<< endl;
 //cout<< "BLTData=0x " << hex <<BLTData[index]<< endl;

index+=NwordsLeft;

bytes+=ret;

TotalNwords+=NwordsLeft;


if((BLTMAX*4-bytes)<=4*NwordsLeft){
//Note: 16384 is the maximum nr of 32bit words in a BLT
NwordsLeft=BLTMAX-bytes/4;
if(NwordsLeft==0) break;
//To avoid going for a last VME access for transfer of "0" words
last=true;}


}//end while(true)
cerr<<"Nr␣transferred␣words="<<TotalNwords<< endl;

 return 0;
}


//-----------------------------------------------------
// Perform readout of data using BLT
//-----------------------------------------------------
int readout_data_new()
{

   unsigned long TotalNwords=0;
   unsigned long Nwords=16384;

/***********Check DATA READY before reading out FIFO***********/
  /*Waits for DREADY set to 1*/
while(!(b->Is_Data_Ready()))
//loop while the Buffer Almost Full level is not reached
{
if (b->Is_Global_TDC_Error())
{
for (unsigned int j=0; j<4; j++)
 {
 if (b->Is_TDC_Error(j))
       {
 cerr << j << "␣␣␣␣␣" << b->OPCODE_Get_ErrorCode_TDC(j) << endl;
 }
   }
 //break;
 }
}
   /*************************************************/


   //READ OUTPUT BUFFER

 pBLTData = &TotalData[0];


// Start time stamp
gettimeofday(&tot_time, 0);            // Starttime
unsigned long start_blt_time=tot_time.tv_usec;

ret = b->BLTRead_Output_Buffer(pBLTData, Nwords);
//BLT of 32bit Nwords


// End time stamp
gettimeofday(&tot_time, 0);           // Endtime
unsigned long end_blt_time=tot_time.tv_usec;
```

```
cout<<"Time diff= "<<dec<<end_blt_time-start_blt_time <<"us"<<endl;


TotalNwords = ret/4;

cerr<<"Nr transferred words=" << TotalNwords << endl;

return 0;
}



//----------------------------------------------------
// Main
//----------------------------------------------------
int main(int argc, char * argv[]){


// Start time stamp
  gettimeofday(&tot_time, 0);             // Starttime
  cerr << " Start time          : " << ctime((const time_t *)&tot_time.tv_sec)<<"us= "
  <<tot_time.tv_usec<<endl;
  //cerr << "       since 1.1.1970 : " << tot_time.tv_sec << " sec, " <<
  tot_time.tv_usec << " usec\n";

  b = new TDC_V767_CAEN(baseaddress,"TDC_V767_CAEN",vme_link,board_no);

  set_tdc_board();

// File and Tree initialise (1st time the DAQ runs)
  ChangeFile();
// RSS remark: This is the initial file, opened before any loop is entered.

  for(unsigned int i=0;i<1;i++){
  //while(1){
  //Perform a single & complete BLT
    //readout_data();
    readout_data_new();
    process_Data();


// Check for file size and close/create new file
/*if(f->GetSize() >= MAXSAVESIZE){    // RSS version for debugging
cout<< "sizeFile="<< dec << f->GetSize() << ",    allowed are: " <<
(int)MAXSAVESIZE <<"\tClose current file and open new\n"<< endl;

f->Close();
ChangeFile();
        }*/


      }  //end for loop


  delete b;

  return 0;
}
```

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Cottbus, May 2, 2011

Maria Hempel

Acknowledgment

I would like to express all my gratitude to all those people who made the thesis possible.
This thesis would not have been possible unless my professor at the university and supervisor of the thesis Prof. Dr. Wolfgang Lohmann. He provide me an insight into particle physics and gave me the chance to work in his study group. Thank you for all your support.
A special thanks goes to all the members of Mr. Lohmanns group. I want to thank Elena Castro and Ringo Schmidt who answered all my questions during my internship. Elena Castro explained me all the basics for a better understanding of the subject.
I would like to show my gratitude to Wolfram Zeuner who took the time to read the thesis and gave very helpful hints.
Thank you to all my professors at the University for all the interesting lectures and my fellow students for each discussion and coffee break.
Special thanks to all my friends in Berlin and Cottbus who supported me and spend free time with me.
A sincere thanks goes to my parents who made my studies at the university possible. They encouraged me to study physics and stand by me in each situation.

Thank you to all.