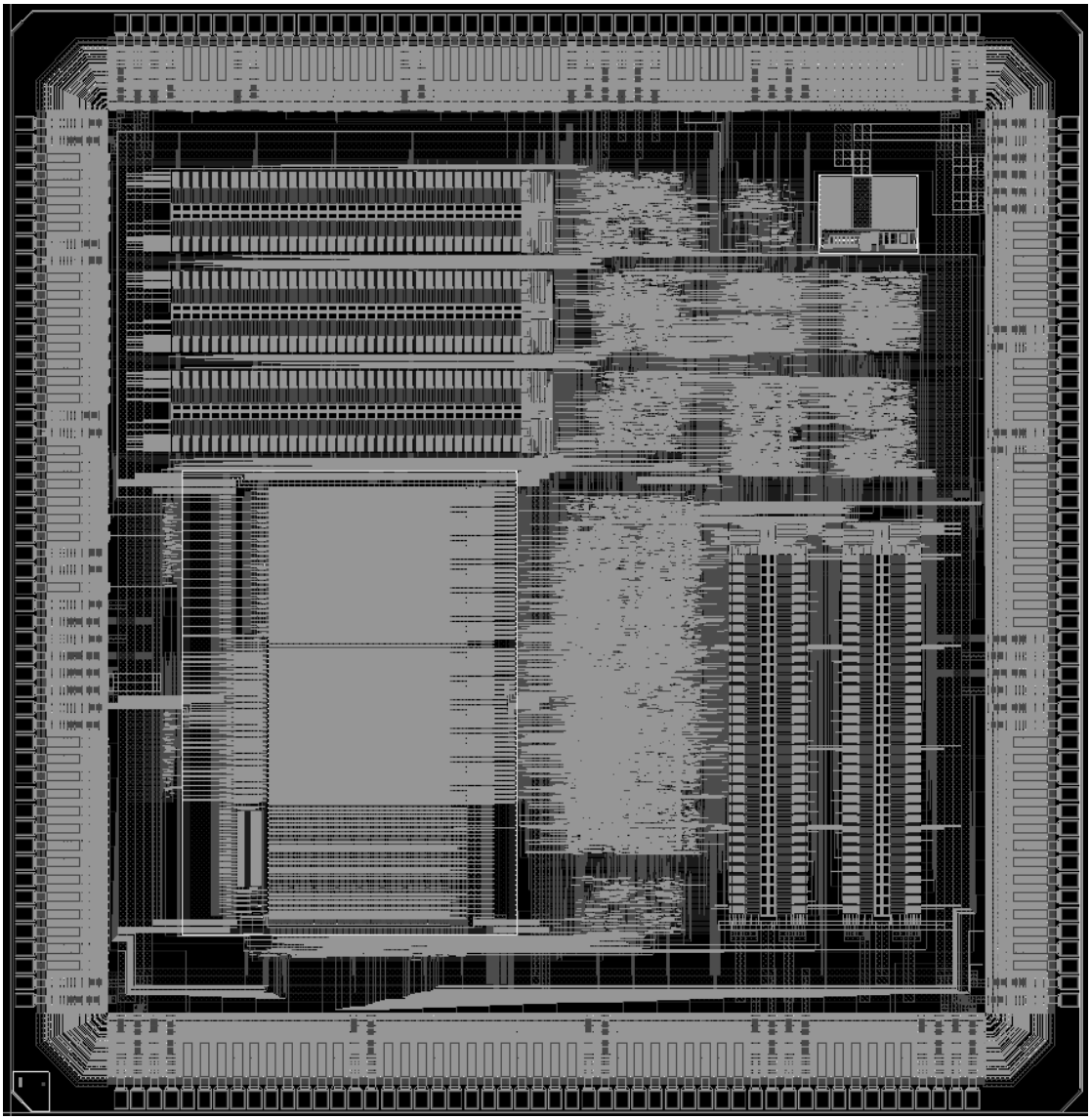
HPTDC

High Performance Time to Digital Converter

Version 2.2, March 2004

for HPTDC version 1.3



J. Christiansen
CERN/EP - MIC

Email: jorgen.christiansen@cern.ch

Table of contents

Document status

Version 0.5: **June 2000**
HPTDC version 1.0
First version of manual released to users
No 3.3v IO.

Version 1.0: **September 2001**
Corrections made for HPTDC version 1.0
PLL lock status bit removed as not implemented
Channel dead time changed to 10 ns guaranteed
Clarify that hits gets rejected when L1 buffer full
Clarification of trigger and control signals in LVDS or LV TTL.
Clarification of token passing when no trigger
Clarification of debugging data in readout
Clarify that serial readout LVDS or LV TTL
Clarify that read of setup scan path destructive.
Clarify update of parity error checks of configuration data.
JTAG state diagram added
Clarification of individual setup bits
Default DLL and PLL current settings.
Clarify that reloading setup data will introduce loss of DLL and PLL lock
Inclusion of measured power consumption

Version 2.0: **September 2001**
Corrections made for HPTDC version 1.1 in standard BGA 225 package.
Full 3.3 v IO version
Updated PLL current level recommendation
Chapter on power-down mode
Update of power supply naming and pin mapping
Separate power supply for clock input
Separate power supply for hit inputs
Pins R02, R05, L08, N13, K13, F14, E13 left unconnected to be pin-compatible with previous versions.

Version 2.1: **July 2002**
Corrections made for HPTDC version 1.2 in customized BGA package
Correction of RC bin mapping
RC delay line adjustment procedure modified
Time resolution characterization measurements included
Use of correction table to obtain maximum time resolution described
Recommendations on power supply sequence
Redefined restrictions on 2.5v power supply for correct function.
Comment on max rate from groups into readout FIFO
Describe group/chip trailer word count bug
JTAG instruction register can be read
Correction of JTAG readout bit sequence
DLL tap adjust mapping
Include time offset measurements
All 2.5v power now connected to internal power plane
Updated pin mapping (backwards compatible)
Pins R02, R05, L08, N13, K13, F14, E13, C14, F10, B15, C13, K01, K02 left unconnected

Update of known bugs

Version 2.2: February 2004
Corrections made for HPTDC version 1.3
Minor clarification of use of bunch count reset and Roll_over
JTAG ID updated
Update of RC bins
Bugs corrected in version 1.3 removed from bug list

1. Introduction

The High Performance general purpose TDC (HPTDC) is based on a series of TDC's developed in the Micro electronics group at CERN. Previous versions of general purpose TDC's have shown that the concept of a programmable TDC is highly appreciated in the high energy physics community. The high flexibility has enabled the use of this kind of TDC's in many different experiments with very different system requirements. The large programmability is also extremely useful within the individual experiments as the operation of the TDC can be optimized to the actual running conditions of the experiment which are to a certain extent unknown before the experiment is "turned on".

Previous versions of programmable TDC's (NA48 TDC ref[1], TDC32 ref[2], AMT0 ref[3]) were sufficiently flexible that they could have been used in many coming experiments. Unfortunately the technologies used for these implementations have been phased out. To ensure that the new HPTDC is available for production over a time period of ~5 years, after the development has been finalized, it must be implemented in one of the most modern technologies available today. The top of the line technology available today (1999) on normal commercial terms is 0.25 um CMOS. This kind of technology will enable the implementation of a high performance TDC with 32 channels per chip.

The general architecture developed for the previous versions of TDC's is reused in the new TDC implementation. This architecture has proven itself to be extremely flexible and work well in many different kinds of experiments. The use of a data driven architecture enables the TDC to work well in both triggered and un-triggered applications. A trigger matching function based on time tags allows the trigger latency to be programmable over a large dynamic range and also ensures the capability of supporting overlapping triggers, where individual hits may be assigned to multiple events. The use of a more modern technology will enable a significant increase in the performance of the TDC when introducing minor changes to the architecture.

The use of a 40 MHz clock as a time reference is a requirement for all LHC experiments as this clock is directly available in the front-end electronics to synchronize the acquisition of detector signals to the bunch crossings of the LHC machine. In addition it is required to correctly identify the bunch crossing number within the LHC machine cycle consisting of 3564 bunch crossing periods.

Most users of such a new TDC need a time resolution of the order of 250ps RMS to measure drift time in drift based tracking detectors. This level of resolution can be obtained directly from the 40 MHz clock using a Delay Locked Loop (DLL) with 32 delay elements, as done in previous versions of TDC's. Other potential users like Time-Of-Flight detectors may though require improved resolution. In a 0.25 um CMOS technology the time resolution can be improved to the level of 30 - 50 ps RMS using a Phase Locked Loop (PLL) to generate a 320 MHZ clock to drive the DLL. In the low resolution mode, the use of a PLL is not strictly required. The experience with previous TDC's have though shown that the generation (distribution) of a stable (low jitter) 40 MHZ clock to the front-end can pose significant problems. In this case the use of a PLL in the TDC itself can reduce jitter on the incoming clock signal.

The micro electronics group has shown that improved time resolution from a DLL based TDC can be obtained using a R-C delay chain to control the sampling of the timing signals from the DLL ref[4]. This scheme requires the DLL signals to be sampled several times (4) controlled from a precisely calibrated R-C delay line. The implementation of the R-C delay line itself only occupies very limited chip area and relying on an external software driven calibration procedure results in a very small hardware overhead. The multiple sampling of the DLL signals can be performed using the sampling registers from several "normal" TDC channels. Using the sampling

registers from 4 channels results in a TDC with 8 very high resolution channels with an estimated RMS resolution of the order of 10 - 20 ps.

The signal interface to the latest version of TDC's (AMT0) has by several different potential users been found attractive. All inputs related to the time measurements (hits, clock) and the interface to the trigger system can be programmed to use either differential LVDS signalling levels or single ended TTL levels. A 0.25 um CMOS technology can not be made to interface directly to 5 V TTL signals but can be made to accept 3.3 v TTL input levels (HPTDC version 1.0 only accepts 2.5 volt IO).

The high integration level of 32 channels per TDC chip has in previous versions been obtained by merging data from 32 channels into one common data path, using small channel buffers as derandomizers before this "bottleneck". This level of channel multiplexing was required in old technologies to "squeeze" 32 channels into one chip. The cost of this is that the hit rate which can be sustained per channel is lowered by the same factor. In a modern sub-micron technology the integration level is much higher and a multiplexing factor of 8 (instead of 32) seems to be a good choice. This will enable the new TDC to sustain hit rates per channel of the order of 2.5 MHz. This results in a TDC architecture with four independent groups of 8 channels each, which are finally merged into one common readout buffer after trigger matching has reduced the required bandwidth several orders of magnitude. An additional performance improvement (higher hit rates) can be obtained by running the logic of each channel group at a higher clock frequency (from on-chip PLL).

The readout interface required for different applications depends strongly on the physical location of the TDC (inside detector, on VME board, etc.) and on the data bandwidth required. A 32 bit parallel readout interface will be available for high bandwidth readout. The voltage levels used on the parallel readout port will be 3.3 v levels dictated by the IC technology used. A serial readout at 80 MHz and below using LVDS signalling levels will be available for low bandwidth readout sending data off detector. Readout data will be formatted in 32 bit packages (words) with a 4 bit type identifier. A byte wise readout is also available to enable the TDC to drive directly commercial serializing IC's (Hot link, LVDS links, etc.). A token passing mechanism will enable up to 16 TDC chips to share a common readout bus or serial link.

The use of a JTAG port for programming and performing tests has been found useful in previous versions and will be maintained in the TDC.

The TDC will not be implemented in a technology guaranteed to be radiation hard. It will most likely work correctly up to levels above 30Krad total dose with a slight increase in power consumption. It will neither be guaranteed to be insensitive to Single Event Upsets (SEU) but all internal memories and state machines will be self checking, enabling the TDC itself to identify a malfunction.

The correct function of a data driven architecture in high energy physics experiments, with varying channel occupancies (hit rates) and different types of correlations between channels, can in some cases be difficult to ensure based on simple estimates. The HPTDC has been designed to be capable of handling high instantaneous rates which may overflow internal buffers. When internal buffers overflow and hits or triggers are lost, the TDC will identify and mark events which have been influenced by such overflows. Loss of triggers are identified internally and regenerated as empty events to prevent the DAQ system to loose event synchronization. The best way to prove the correct function of the HPTDC for a given application is to simulate its behavior with a given set of hit and trigger rates (with sufficient safety factors). A Verilog behavioral or a register level model is available for such simulations and can be requested by Email (jorgen.christiansen@cern.ch). In certain cases the micro electronics group can perform the simulations at the given

rates and verify its correct function and generate histograms of buffer occupancies.

2. Architecture

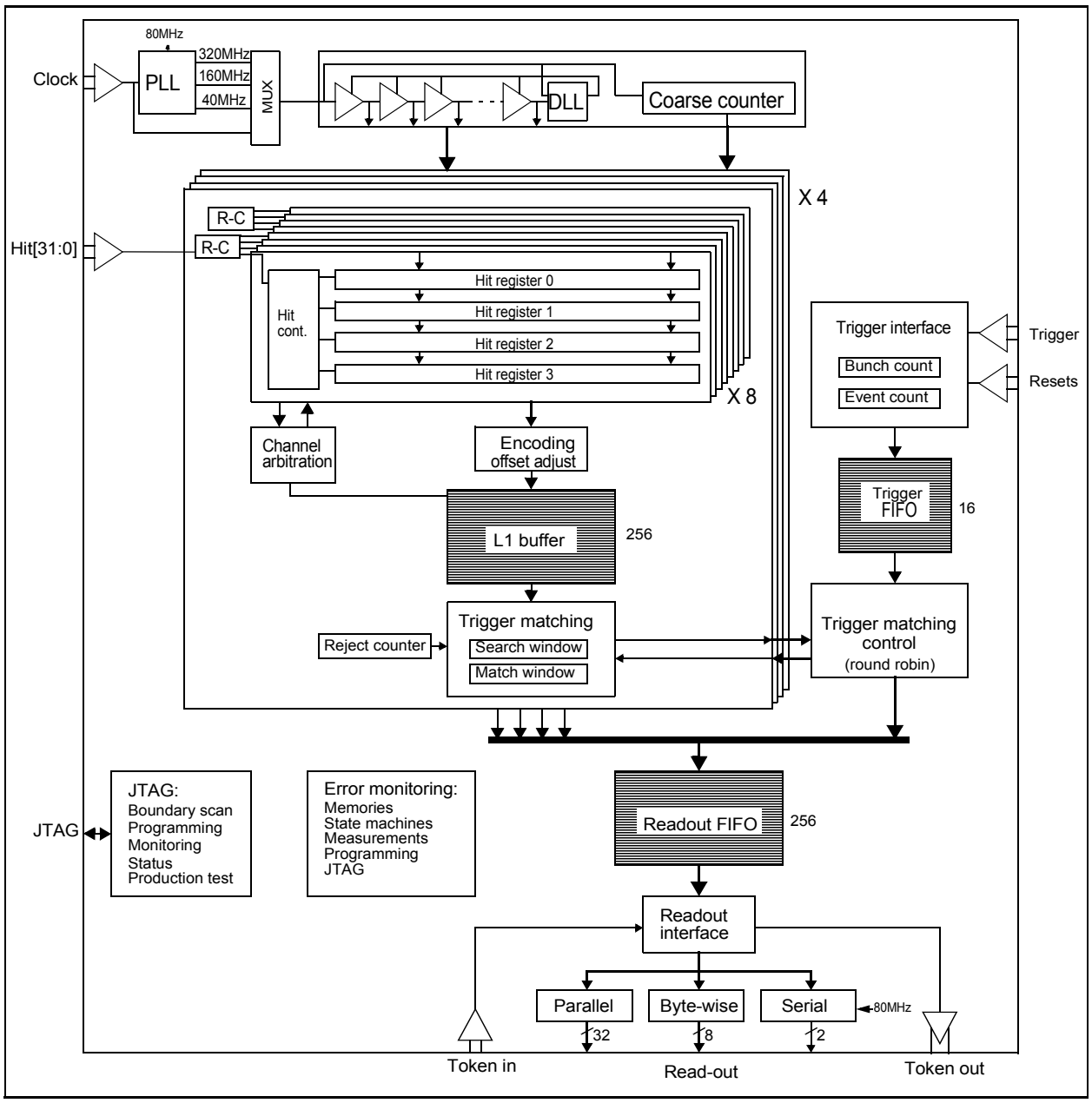


Fig. 1 Architecture of HPTDC.

The time base for the TDC measurements is a Delay Locked Loop (DLL) with 32 delay elements and a clock synchronous counter, both driven from the same clock reference. The clock reference can be taken directly from the clock input of the TDC chip or can come from an on-chip PLL (40/160/320 MHz). The PLL can perform clock multiplication to increase time resolution or can be used as a filter to remove jitter on the input clock. A hit measurement is performed by storing the state of the DLL and the coarse counter when a hit (leading and/or trailing edge) is detected.

Each channel can buffer 4 measurements until they are written into a 256 words deep level 1 buffer shared by a group of 8 channels. The individual channel buffers work as small derandomizer buffers before the merging of hit measurements into the L1 buffer. Measurements stored in the

level 1 buffer can be passed directly to a common 256 words deep read-out FIFO, or a trigger matching function can select events related to a trigger. The trigger information, consisting of a trigger time tag and an event id, can be stored temporarily in a 16 words deep trigger FIFO. A time window of programmable size is available for the trigger matching to accommodate the time spread of hits related to the same event. The trigger time tag can optionally be subtracted from the measurements so only time measurements relative to the trigger need to be read-out. Accepted data can be read out in a direct parallel format or be serialized at a programmable frequency.

3. Phase Locked Loop.

The PLL takes care of performing clock multiplication from a 40 MHz input clock to 160/ 320 MHz, when high resolution time measurements are required. It can alternatively be used to generate a clock with the same frequency when its capability of filtering jitter on the input clock is required. A PLL is a complicated and sensitive second order control loop consisting of the components shown in the figure below.

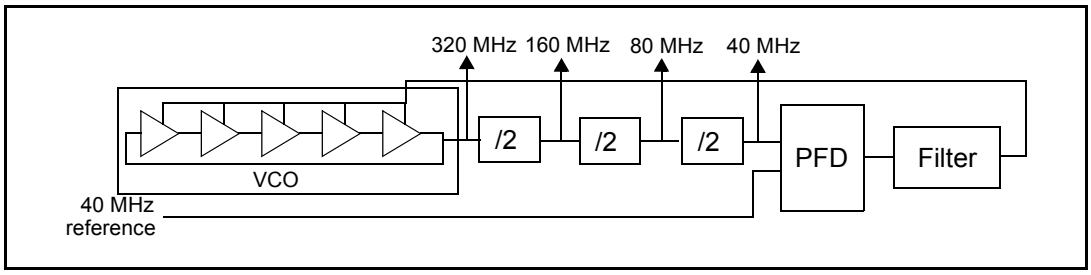


Fig. 2 PLL block diagram

The Voltage Controlled Oscillator (VCO) generates a symmetrical clock signal who's frequency and phase is compared with the reference signal. If a frequency or phase discrepancy is detected by the Phase - Frequency Detector (PFD), the control voltage to the VCO is adjusted via a charge pump and filter circuit. By dividing the frequency generated by the VCO, before comparing it with the input reference, the VCO frequency becomes multiplied by the given factor. The dynamics of the PLL control loop can be changed from the programming by setting the currents used in the charge pump

The PLL must be initialised after a stable reference clock has been supplied to the TDC chip. The correct frequency and phase lock will be obtained after ~10 ms.

4. Delay Locked Loop.

The DLL, which generates the basic timing signals to obtain the required time resolution, consists of three major components: A/ Chain of 32 delay elements which delay can be adjusted by a control voltage. B/ Phase detector measuring the phase error between the clock and the delayed clock from the delay chain. C/ Charge pump and filter capacitor generating the control voltage to the delay elements based on the measured phase error from the phase detector.

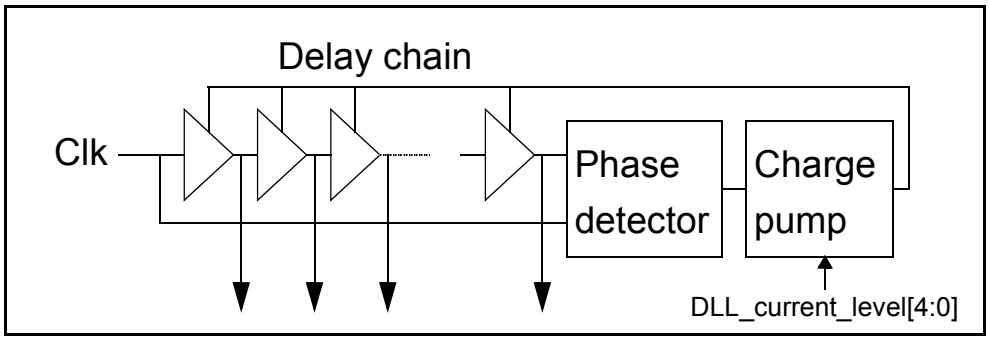


Fig. 3 DLL with its main components.

After a reset of the DLL a certain time is required for it to reach lock. When correct locking has been obtained the DLL lock status is set and the TDC is ready to perform time measurements. During normal operation the DLL is continuously monitored. If locking is lost (may happen if the clock has been removed for some time or a large clock frequency change has occurred) and a time measurement is performed a vernier error status is set. If this happens the DLL may need to be reinitialized to guarantee correct function.

A DLL is sensitive to jitter on its input clock. Any jitter on the clock will directly deteriorate the precision of the time measurements. If the jitter is larger than two time bins in the DLL it may result in the vernier error status bit to be set when a measurement is performed.

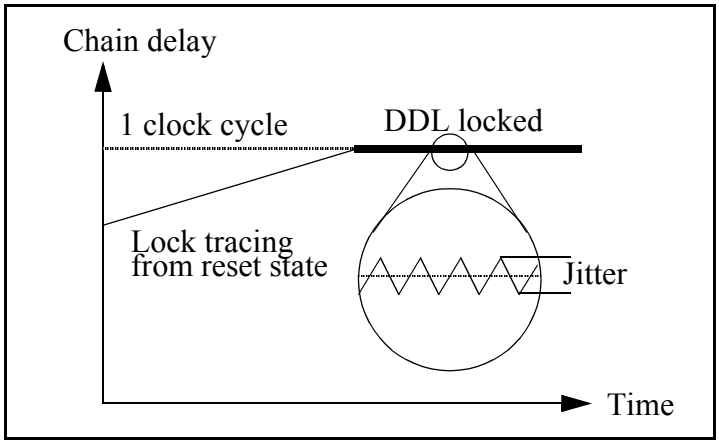


Fig. 4 Closed loop behaviour of DLL from out of lock condition.

The dynamics of the control loop in the DLL can be set by controlling the current levels used in the charge pump circuit. Low current levels results in slow lock tracing but reduced jitter when locking has been obtained. Large current levels gives fast locking but increased jitter.

If the PLL is used before the DLL to do clock multiplication or jitter filtering the PLL must have obtained stable locking before the DLL can be initialised. Otherwise the DLL may arrive in a false locking state. The DLL has different operating modes depending on the operating frequency (40/160/320 MHz)

5. Very high resolution mode

By performing multiple sampling of the DLL signals, controlled from a precisely calibrated R-C delay line, the time resolution can be improved significantly. An interpolation within a DLL cell is obtained by sampling the DLL signals four times, equally spaced over the interval of a delay cell. The sampling signals must be generated with very small delays (25 ps) and high precision and stability. This is done using an R-C delay line having very small dependencies on temperature and

supply voltage. Such a R-C delay line though has large dependencies on processing parameters and each chip needs to be calibrated. Afterwards the calibration parameters can be considered to be constant within reasonable variations of temperature and voltage (+/- 25 ^{0}C, +/- 10% Vdd).

The R-C delay line itself occupies a very limited chip area and the large number of sampling registers needed is obtained using 4 normal TDC channels. The time measurements from the four normal channels can still be considered as individual time measurements from which a high resolution interpolation can be performed. These four measurements can be read out individually (for debugging and calibration) or an on-chip interpolation can be performed to compress them into one single very high resolution time measurement.

Because of the process variations of the R-C delay line a calibration of its parameters are required. Such a calibration can be performed using a statistical code density test based on a source of random hits ref[4] (see chapter: Calibration of very high resolution mode). A calibration of this kind can in principle be completely integrated on the chip but will increase the complexity of the design. It has been proven with prototype chips that when the correct calibration parameters have been determined they do not need to be changed under normal working conditions. It is therefore acceptable that the calibration procedure is software driven from the outside and only is performed infrequently. The calibration constants per chip are limited to 4 x 3 bits used to control the four sampling taps from the R-C delay lines.

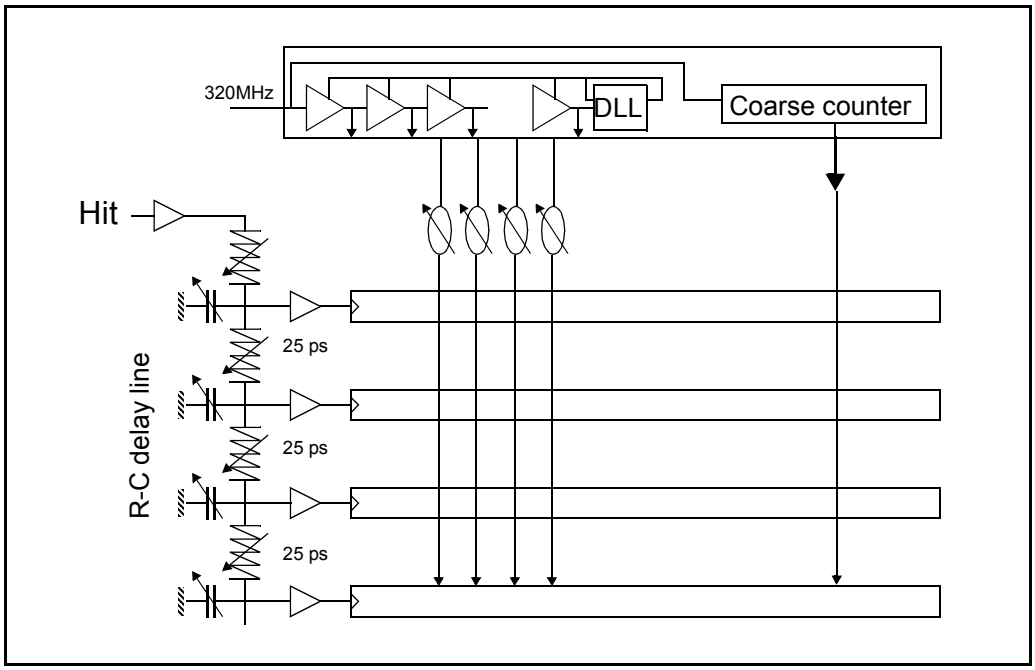


Fig. 5 Principal diagram of using R-C delay line to improve TDC resolution

Integral non-linearities in the DLL can be compensated for by small adjustable delays on each signal tap from the DLL. For normal applications these can be left with an initial value of zero. For applications where a very precise integral linearity is required these can be used to compensate for mismatches in the delay line of the DLL. The calculation of the optimal adjustment parameters can also in this case be found from a simple code density test ref[4] (see chapter: Adjustment of DLL taps).

6. Coarse Time Count.

The dynamic range of the fine time measurement, extracted from the state of the DLL, is expanded by storing the state of a clock synchronous counter. The hit signal is asynchronous to the

clocking and the coarse counter may be in the middle of changing its value when the hit arrives. To circumvent this problem two count values, 1/2 a clock cycle out of phase, are stored when the hit arrives. Based on the fine time measurement from the DLL one of the two count values will be selected, such that a correct coarse time value always is obtained.

At reset the coarse time counter is loaded with a programmable coarse time offset. The 15 bit coarse time counter can be clocked directly by the 40 MHz input clock or alternatively by the clock multiplied by 4 or 8. In the 40 MHz case the 12 least significant bits are used as the bunch identification (coarse time) and the 3 most significant bits are ignored. In case of a PLL multiplication factor of four, bit 13 - 2 are used as the bunch identification (coarse time at the level of 40 MHz) and the 2 least significant bits are appended to the fine time from the DLL (in this case 2 + 5= 7 bits fine time). For a PLL multiplication factor of eight, bit 14 - 3 are used as the bunch identification (coarse time at the level of 40 MHz) and the 3 least significant bits are appended to the fine time from the DLL (3 + 5= 8 bits fine time). The programmable 12 bit coarse count offset, loaded into the coarse counter with the bunch count reset, always represent bunch counts at the level of 25 ns cycles (40MHz).

The bunch structure of LHC is not compatible with the natural binary roll over of a counter. The coarse counter can therefore be reset separately by the bunch count reset signal and/or the counter can be programmed to roll-over to zero at a programmed value. The programmed value of this roll-over is also used in the trigger matching to match triggers and hits across LHC machine cycles.

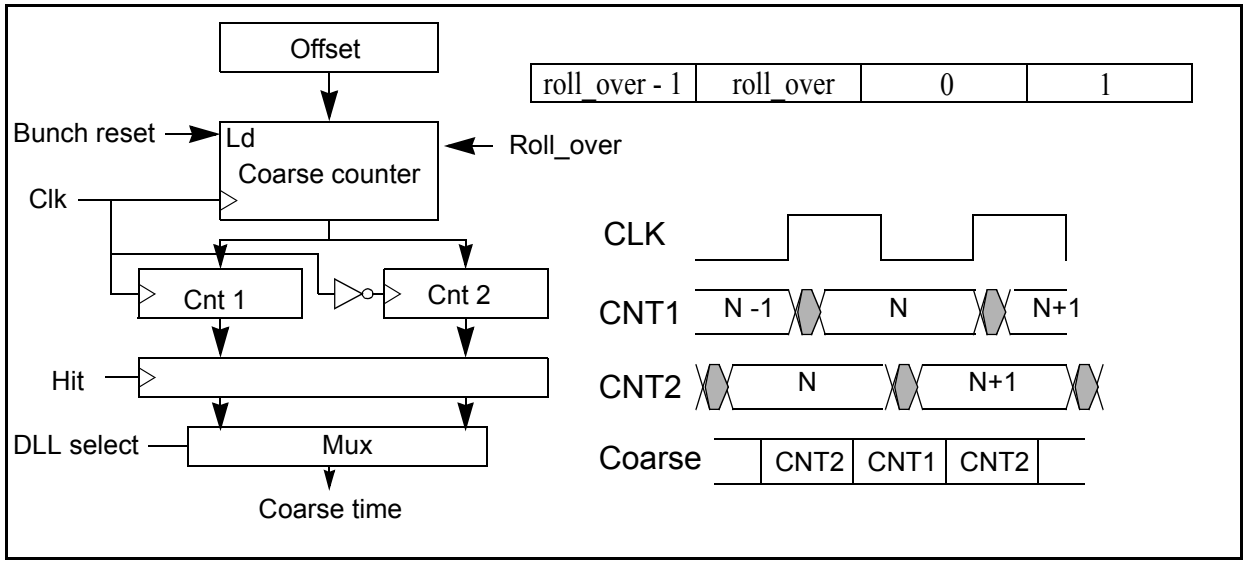


Fig. 6 Phase shifted coarse time counters loaded at hit.

7. Channel buffer.

Each channel can store 4 TDC measurements before being written into the L1 buffer shared between 8 channels. The channel buffer is implemented as a FIFO controlled by an asynchronous hit controller.

A hit controller determines when a time digitization must be performed on a channel. The hit controller can be programmed to digitize individual leading and/or trailing edges of the hit signal. Alternatively the hit controller can produce paired measurements consisting of one leading edge and the corresponding trailing edge (not available in very high resolution R-C mode). If the channel buffer is full, when a new hit arrives, it will be ignored. The hit inputs can be configured to use differential LVDS or single ended LV TTL signal levels. The minimum time between two

consecutive time measurements is typically 5 ns (guaranteed 10 ns). A programmable dead time (~5ns - 10ns - 30ns - 100ns) can be assigned to the TDC channels to reject possible oscillations from the analog front-end.

For the time measurements stored in the channel buffers to be written into the clock synchronous L1 buffer a synchronisation of the status signals from the channel buffers is performed. Double synchronisers are used to prevent any metastable state to propagate to the rest of the chip running synchronously. When paired measurements of leading and a trailing edges are performed, the two measurements are taken off the channel buffer and combined into one combined measurement containing the leading edge time and the extracted pulse width.

The latching of time measurements in the channel buffers can be performed in two alternative ways. A low power scheme minimises the power consumption but may limit the obtained time resolution in the high resolution modes. The higher power alternative increases the power consumption of the HPTDC significantly but should obtain the best possible timing performance. It is recommended to use the low power mode as no significant advantage has been seen with the other mode.

8. Writing into L1 buffer.

When a hit has been detected in a channel buffer the corresponding channel is selected, the fine time measurement is encoded into binary form, the correct coarse time count value is selected and the complete time measurement is written into the L1 buffer together with a channel identifier.

The fine time measurement extracted from the DLL (5bits) and the coarse time count is merged into a complete time measurement in a format which is independent of the clocking speed of the DLL. The 12 MSB of the measurement contains a count of 40 MHz clock cycles and the remaining 8 bits contains an interpolation within the 40 MHz clock cycle. The detailed mapping from the coarse time counter and the encoded DLL interpolation is indicated in the figure below.

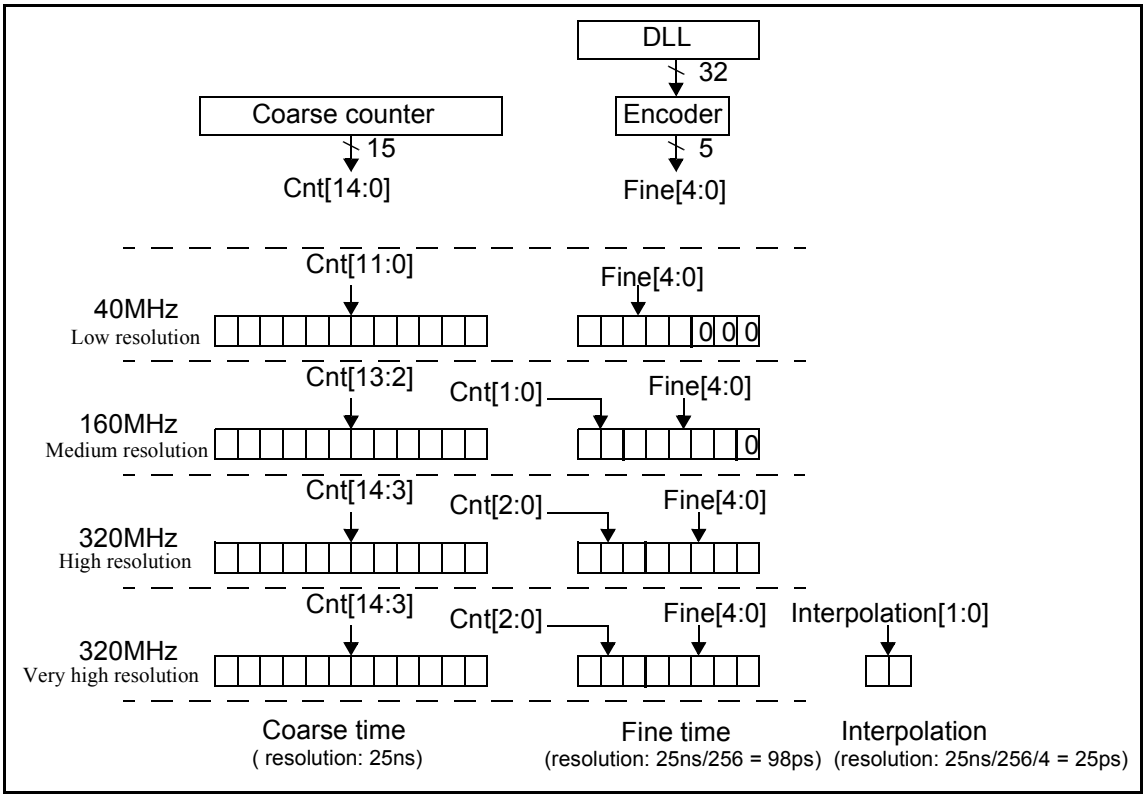


Fig. 7 Mapping coarse count and DLL interpolation into complete time measurement.

In case a paired measurement of leading and trailing edge has been performed the complete time measurement of the leading edge plus a 7 bit pulse width is written into the L1 buffer. The 7 bit pulse width is extracted from the leading and trailing edge measurement, taking into account the programmed roll-over value. The resolution of the width measurement is programmable. In case the pulse width is larger than what can be represented with a 7 bit number, the width will saturate to a value of 7F Hex.

When several hits are waiting in the channel buffers an arbitration between pending requests is performed. New requests are only allowed to enter into the active request queue when all pending requests in the queue have been serviced. Arbitration between channels in the active request queue is done with a simple hardwired priority (channel 0 highest priority, channel 7 lowest priority). The fact that new requests only are accepted in the active request queue when the queue is empty enables all 8 channels in a group to get fair access to their shared L1 buffer (see chapter: Limitations of HPTDC architecture on page 82). The maximum bandwidth available per channel is limited to 40MHz/4 = 10 MHz because of the synchronisation between the asynchronous channel buffers and the clock synchronous logic (at 40MHz logic clock). If a synchronization error occurs and two channels are selected concurrently to be written into the L1 buffer a channel select error status bit is set.

A 9 bit (1 bit coarse + 8bit fine) channel dependant offset is added to the measurement before writing it into the L1 buffer. This enables the following trigger matching to be performed on time measurements which have been corrected for time offsets in the detector and its analog front-end. Large offsets are handled for all channels in common using the coarse count offset. In very high resolution mode identical offsets must be used for the 4 interpolation channels (0-3, 4-7, , 28-31)

9. L1 buffer.

The L1 buffers are 256 words deep and are written into like circular buffers. Reading from the buffers are random access such that the trigger matching can search for data belonging to the received triggers. If a L1 buffer runs full the latest written hit will be marked with a special full flag and subsequent hits from the channel buffers will be rejected until the buffer recovers from this state. When the buffer recovers from being full the first arriving hit will be marked with a full recovery flag. These flags are used by the following trigger matching to identify events that may have lost hits caused by the buffer being full.

10. Trigger Matching.

Trigger matching is performed as a time match between a trigger time tag and the time measurements themselves. The trigger time tag is taken from the trigger FIFO and the time measurements are taken from the L1 buffer. Hits matching a trigger are passed to the common read-out FIFO shared by the four channel groups. Optionally the trigger time tag can be subtracted from the measurements such that all time measurements read out are referenced to the time (bunch crossing) when the event of interest occurred.

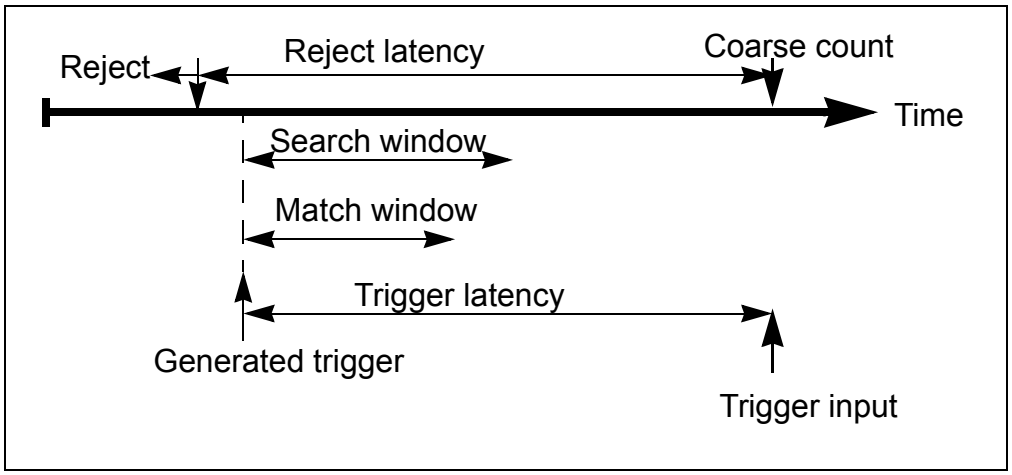


Fig. 8 Trigger, trigger latency and trigger window related to hits on channels

A match between a trigger and a hit is detected within a programmable time window. The trigger is defined as the bunch ID (coarse count) when the event of interest occurred. All hits from this trigger time until the trigger time plus the trigger matching window will be considered as matching the trigger. The trigger matching being based on the bunch ID means that the "resolution" of the trigger matching is one clock cycle (25ns) and that the trigger matching window is also specified in steps of clock cycles. A matching window set to eight equals a time matching window of (8+1)x25ns = 225ns. The absolute maximum obtainable trigger latency which can be accommodated by this scheme equals half the maximum coarse time count = $2^{12}/2$ = 2048 clock cycles. To ensure that data never needs to be stored more than this maximum value during operation it is advised not to use a trigger latency of more than 1024 clock cycles = 25us. The trigger matching function is capable of working across roll-over in all its internal time counters. For a paired measurement the trigger matching is performed on the leading edge of the input pulse. For a detailed description of the trigger matching parameters see: Time alignment between hits, clock and trigger matching on page 49.

The search for hits matching a trigger is performed within an extended search window to guarantee that all matching hits are found even when the hits have not been written into the L1 buffer in strict temporal order. The search for hits matching a trigger stops when a hit older than the search limit is found or no more data is available in the L1 buffer. For normal applications it is sufficient to make the search window eight clock cycles larger than the match window. The search window should be extended for applications where paired measurements of wide pulses are performed, as a paired measurement is not written into the L1 buffer before both leading and trailing edge have been digitized.

A specific requirement of the trigger matching function implemented in the HPTDC is that the trigger latency must be longer than the width of the match window. This is required to ensure that all hits matching a trigger are already in the L1 buffer when a trigger matching search is started. If this is not the case the trigger search may end (by detecting an empty L1 buffer) before the hits to match have actually been written into the L1 buffer. In cases where this is a potential problem it may be required to artificially delay the trigger.

A unique feature of the trigger matching in the HPTDC is its capability to assign hit measurements to multiple triggers. This becomes important in applications with large drift times and closely spaced triggers. Hits are only removed from the L1 buffers if they are older (not within matching window) than the latest processed trigger or have been found "rejectable" by a special reject function.

To prevent buffer overflows and to speed up the search time an automatic reject function reject hits older than a specified limit when no triggers are waiting in the trigger FIFO. A separate reject counter runs with a programmable offset to detect hits to reject. The reject latency should be set to be 1 clock cycle longer than the trigger latency to ensure that no hits of interest are rejected.

For some applications it is required to limit the number of hits assigned to each trigger, to prevent buffer overflows in the following data acquisition system. The trigger matching can be programmed to match a defined maximum number of hits to each trigger. The hits assigned to an event is in this case done on a “first come first served” basis. If the limiting function has rejected hits this will be signalled via an error marking word in the end of the event.

In case an error condition (L1 buffer overflow, Trigger FIFO overflow, memory parity error, limiting number of hits matched to event, etc.) has been detected during the trigger matching a special word with error flags is generated for events which may have suffered loss of hits.

All data belonging to an event are written into the common read-out FIFO with a header and a trailer. The header contains an event id and a bunch id (trigger time tag) and the event trailer contains the same event id plus a word count.

The number of bits available for the time measurements in the defined 32bit read-out format is limited to 19 bits (see chapter: Read-out Interface. on page 16). It is therefore possible to select the time resolution of the time measurements read out (setup[86:84]).

When hit measurements are performed as pairs there is only a limited number of bits available for the read-out. 12 bits have been assigned to represent the leading edge of the pair and 7 bits for the width. This makes it necessary in most cases to subtract the trigger time tag to get a relative measurement of the leading edge. In addition it is possible to program the resolution of the leading edge measurement to gain dynamic range if required (similar function available for width extracted when being written into the L1 buffer).

The trigger matching function may also be completely disabled whereby all data from the L1 buffers are passed directly to the read-out FIFO. In this mode the TDC has an effective FIFO buffering capability of 4x256 + 256 = 1280 measurements. In this mode a special error word will be generated when ever an error condition has been detected in the TDC (see read-out). When trigger matching is disabled all functions related to hits being grouped into events can not be used (headers, trailers, subtraction of trigger time tag, etc.).

When hits from the four channel groups are written into the common read-out FIFO the priority between the individual groups is given in a round robin fashion to guarantee a fair bandwidth sharing. The maximum bandwidth available from each group (8 channels) into the readout fifo is hereby limited to one quarter of the internal logic clock frequency

A special test mode is available where the trigger matching generates hit measurements with a fixed data content encapsulated within normal headers and trailers. The hit data pattern is programmable from JTAG and the number of hits generated per event is determined by the programmed maximum number of hits per event.

11. Control/trigger Interface.

The control/trigger interface takes care of receiving the trigger signal and generate the required trigger time tag to load into the trigger FIFO. In addition it takes care of generating and distributing all signals required to keep the TDC running correctly during data taking.

The TDC needs to receive a global reset signal that initialises and clears all buffers and error flags in the chip before data taking. Separate bunch count reset and event count resets are available

for applications where the event ID and the bunch ID of accepted events must be controlled separately. These signals can either be generated on separate pins of the TDC or be coded on a combined serial line at 40 MHz using LVDS or LV TTL signal levels.

11.1. Encoded trigger and resets

Four basic signals are encoded using three clock periods. The simple coding scheme is restricted to only distribute one command over a period of three clock cycles. A command is signalled with a start bit followed by two bits determining the command (MSB sent first).

Trigger:	1 0 0
Bunch count reset:	1 1 0
Event count reset:	1 0 1
Global reset:	1 1 1

When using encoded trigger and resets an additional latency of three clock periods is introduced by the decoding compared to the use of the direct individual trigger and resets.

11.2. Event count reset

An event count reset loads the programmed event count offset into the event id counter. In addition a separator can be injected into the L1 buffers and the trigger FIFO, if enabled in the programming (described later).

11.3. Bunch count reset.

The bunch count reset loads the programmed offsets into the coarse time counter, the trigger time tag (bunch id) counter and the reject counter. In addition a separator can be injected into the L1 buffer and the trigger FIFO, if enabled in the programming (described later).

The bunch count reset signal should only be asserted in a single clock cycle as different hits in a time reagin around the reset may otherwise get assigned the same coarse count time.

From a bunch reset is given to the TDC until this is seen in the hit measurements them selves a latency of the order of 2 clock cycles is introduced by internal pipelining. The definition of time 0 in relation to the bunch reset is described in more detail in the chapter: Time alignment between hits, clock and trigger matching.

The use of the bunch count reset and the programmed Roll_over value is to support the use of the HPTDC in a fully continous mode in applications where the machine cycle of an accellerator enforces a specific bunch identification scheme (e.g. LHC). With a correctly programmed Roll_over value it is not compulsary to assert the bunch count reset signal for each machine cycle but it is though recommended to still do it ensure a reliable system synchronisation.

11.4. Global reset.

A global reset clears all buffers in the TDC, initialises all internal state machines and counters to their initial state and clears all detected error conditions. A global reset does not re-initialise the PLL and the DLL.

11.5. Trigger

The basis for the trigger matching is a trigger time tag locating in time where hits belong to an event of interest. The trigger decision must be given as a constant latency yes/no trigger signal. The trigger time tag is generated from a counter with a programmable offset. When a trigger is

signalled the value of the trigger time tag counter (bunch id) is loaded into the trigger FIFO. The effective trigger latency using this scheme equals the difference between the coarse time count offset and the trigger time tag offset (see chapter: Time alignment between hits, clock and trigger matching).

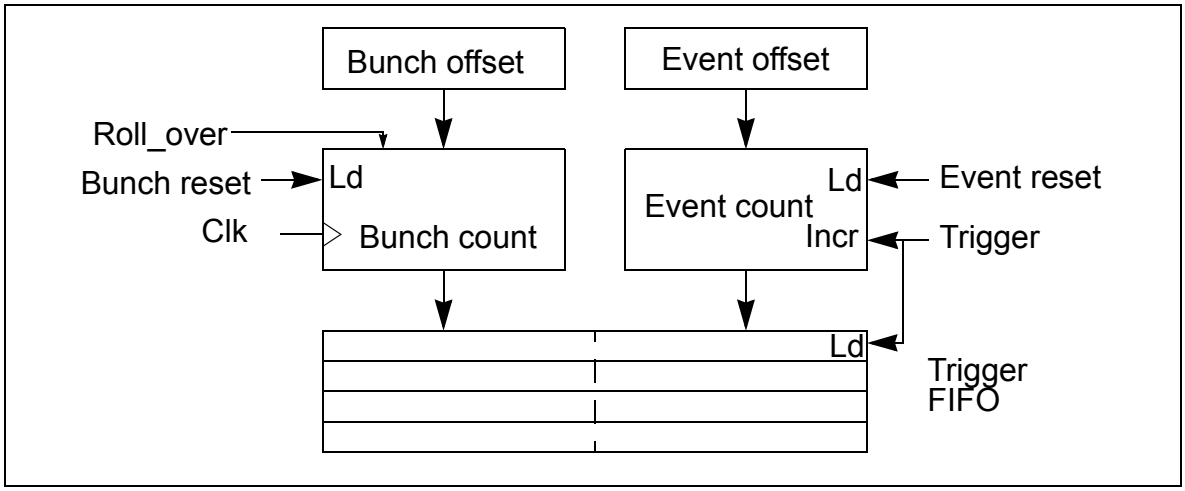


Fig. 9 Generation of trigger data

If the trigger FIFO runs full, then trigger time tags of following events will be lost. The trigger interface keeps track of how many triggers have been lost, so the event synchronisation in the trigger matching and the DAQ system is not lost. For each event with a lost trigger time tag the trigger matching will generate an event with correct event id and a special error flag signalling that the whole event has been lost.

11.6. Separators

The TDC is capable of running continuously even when bunch count resets and event count resets are issued. Matching of triggers and hits across bunch count resets (different machine cycles) are handled automatically if the correct roll-over value (max bunch ID value) has been programmed. Alternatively it is possible to insert special separators in the trigger FIFO and the L1 buffers when a bunch count reset or an event count reset has been issued. These will make sure that hits and triggers from different event count or bunch count periods (machine cycles) never are mixed. In this mode it is not possible to match hits across bunch count periods.

12. Read-out FIFO.

The read-out FIFO is 256 words deep and its main function is to enable events to be read out while others are being processed in the trigger matching. If the read-out FIFO runs full there are two options of how this will be handled.

Back propagate: The trigger matching process will be blocked until new space is available in the read-out FIFO. When this occurs the L1 buffers and the trigger FIFO will be forced to buffer more data. If this situation is maintained for extended periods the L1 buffers or the trigger FIFO will finally overflow.

Reject: As soon as the read-out FIFO is full, event data (not event headers and trailers) will be rejected. Any loss of data will be signalled with a special error word for each event having lost data.

In some cases it is advantageous to start rejecting hits if large amounts of data start to accumulate in the read-out buffer. The effective size of the read-out FIFO can therefore be

"artificially" reduced via the programming.

13. Read-out Interface.

All accepted data from the TDC can be read out via a parallel or serial read-out interface in words of 32 bits. Up to 16 chips can be coupled together using a clock synchronous token passing scheme to perform local event building.

The read-out of an event is started by the master chip sending a group event header (if enabled). The master then sends the token to the first slave chip in the chain which then starts to send its data. The event data from each chip typically consists of a single chip event header (if enabled), accepted time measurements, error flags (if any error detected for event being read out) and finally a single chip event trailer (if enabled). The token is then passed on to the following slave TDCs in the chain until it finally arrives at the master. The master chip asserts its own event data and ends the whole event with a group event trailer (if enabled). A similar scheme can be implemented with a separate read-out controller acting as the master and all TDCs in the chain configured as slaves. If a Token is given to a TDC chip that has not yet received the corresponding trigger, it will keep the token until the trigger has been given and the trigger matching has completed.

For a master chip in a token ring it is in principle intended to use either global headers/trailers or local headers/trailers. A master chip can though be programmed to use both global and local headers/trailers. If using both global and local trailers from a master chip the word count in the global trailer will be one too small as the local trailer from the master chip has not been counted (See: Design bugs and problems on page 99)

A slightly different read-out protocol is available when trigger matching is not enabled, as data is not grouped in events and the related event headers and trailers can not be generated. In this case each TDC having data can be programmed to wait for the token and then send one measurement and immediately pass the token to the next chip even if it has more data to send. This ensures that all TDC's get equal access to send their data and prevents a TDC with a noisy channel to block all the other TDCs in the read-out chain. The cost of this is that the token is circulated continuously which implies some overhead. The TDC can also be programmed to keep the token until all its data has been read out (smaller overhead).

13.1. Readout of debugging information

To enable efficient debugging at the system level internal information about buffer occupancies in the TDC can be read out on a event by event basis (only useful when trigger matching is enabled). Special bunch reset separators can be read out if needed. For very basic system testing the TDC can also be forced to read out a fixed number of data words with a programmable content per received trigger (independent of actual hits).

13.2. Parallel read-out

Read-out of parallel data from the TDC is performed via a clock synchronous bus. Several TDC's may share one read-out bus controlled by the circulating token. The read-out of individual words are controlled by a Data_ready / Get_data handshake and the event synchronisation is controlled by the circulating token. The effective read-out speed can be slowed down by using the Data_ready / Get_data handshake protocol to introduce wait cycles. The number of clock periods that the get_data signal is asserted is used by the master TDC to determine the word count for the event, available in the global trailer.

The Data_ready output signal from the TDC is also tristated when the TDC does not have

the read-out token. This allows the use of a read-out configuration where the Data_ready signals from several TDCs can be connected to the same physical wire and perform a kind of "wired or" function. For this to work reliably, an external pull-down resistor will be required on the Data_ready signal, even when using individual Data_ready signals from each TDC. The TDC will always actively drive the Data_ready signal low before releasing its token and tristate its Data_ready driver. The pull-down resistor does therefore never need to actively pull down a high signal level to a low signal level and a resistance value of 10 Kohm will in most cases be sufficient. The pull-down resistor is only required to maintain a low signal on Data_ready when no TDCs are actively driving the line during the transfer of a token from one TDC to the next. Using this "wired or" configuration it is also possible to connect the wired-or Data_ready directly to the Get_data inputs of the TDC's, if no wait cycles are required to slow down the read-out rate. If the total capacitance of the wire (incl. chip pins) used for the wired-or connection is more than 10 pf an additional delay of the Data_ready driver of 0.04ns/pf must be taken into account.

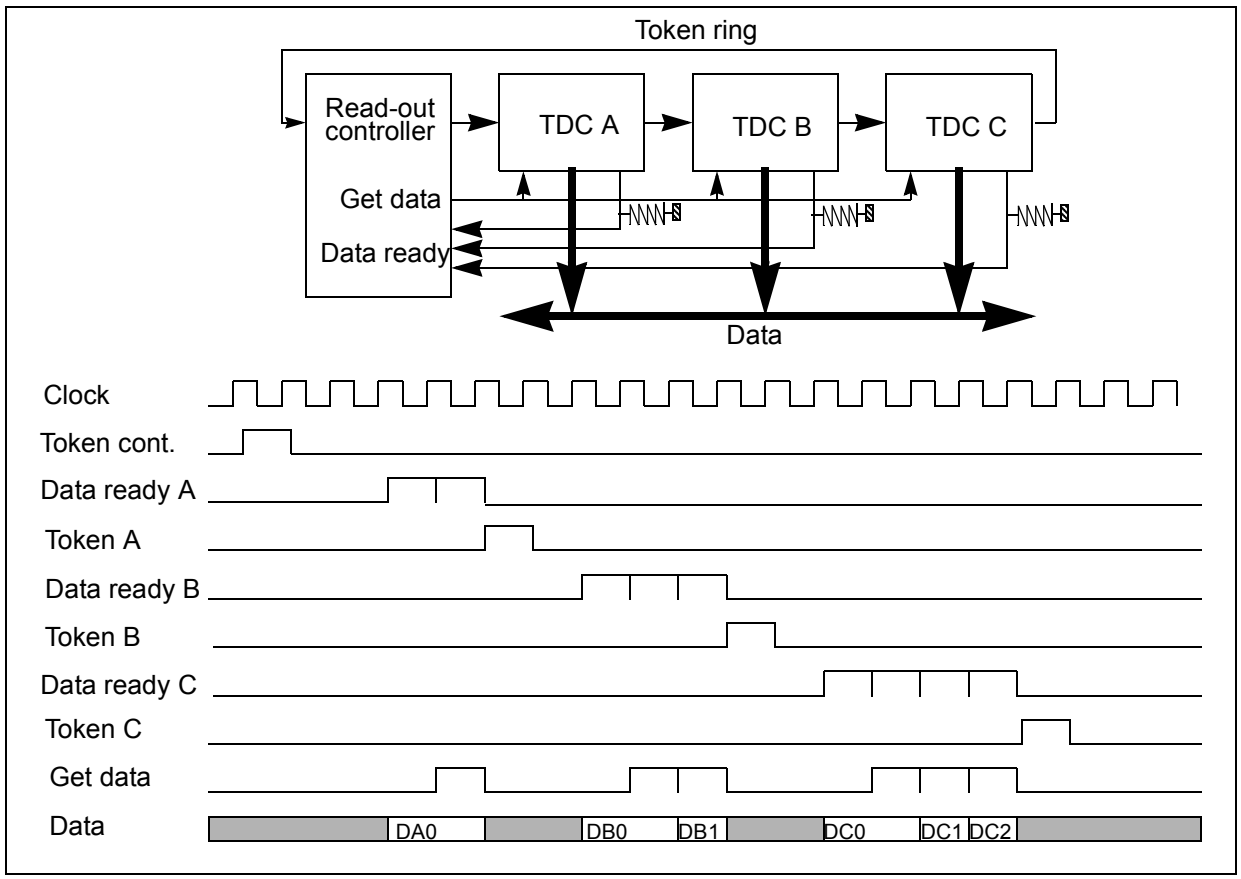


Fig. 10 Token based parallel read-out with all TDCs configured as slaves controlled by an external controller.

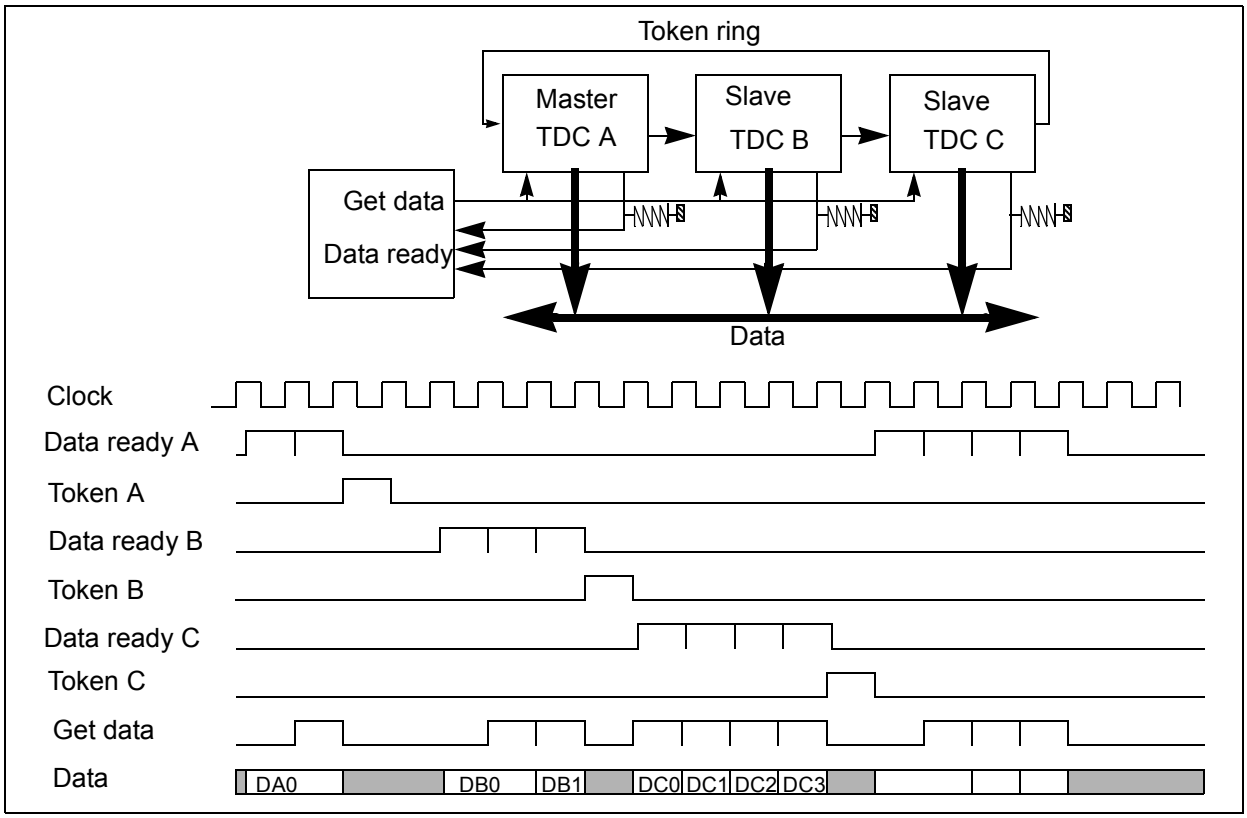


Fig. 11 Token based parallel read-out with one master TDC and a simple read out controller.

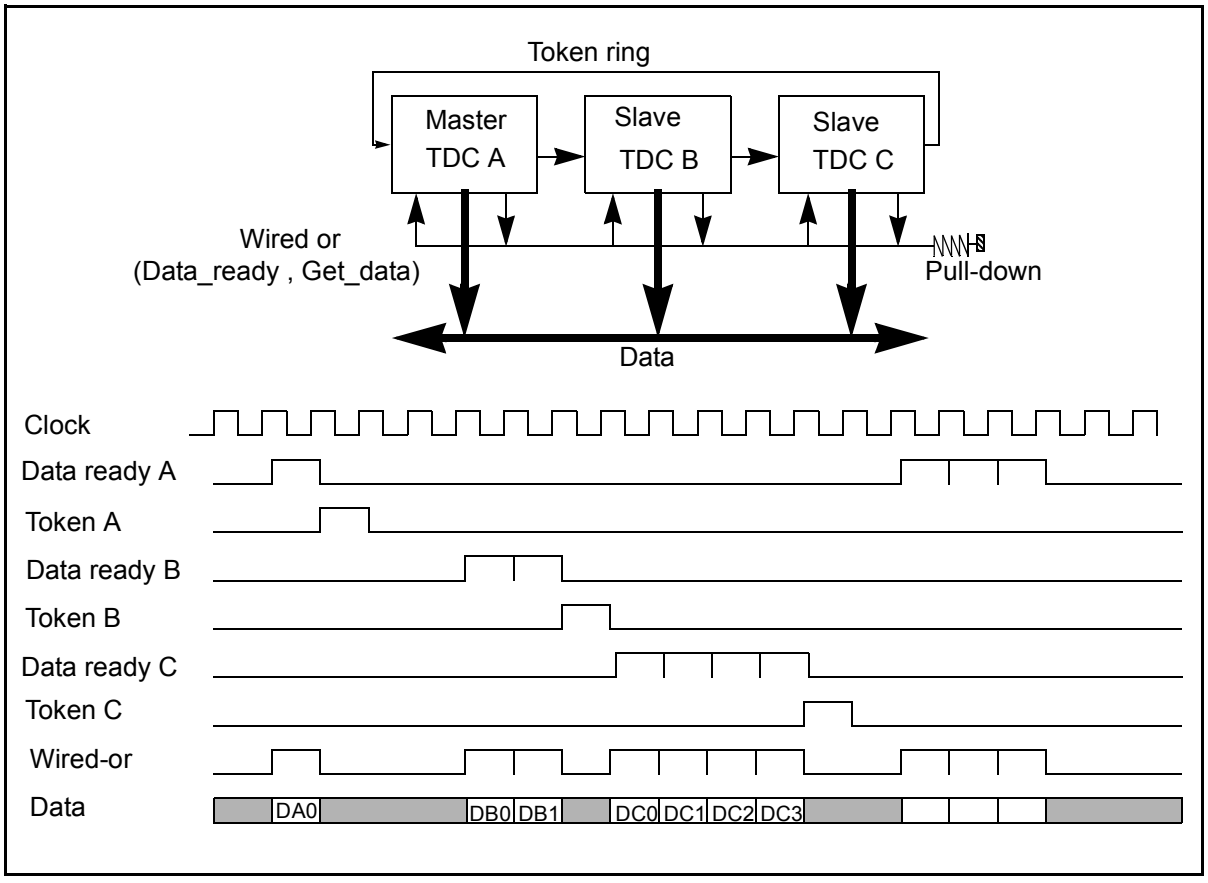


Fig. 12 Using Wired-or connection of Data_ready and Get_data for fast simple read-out

To prevent several TDCs connected to the same bus to drive against each other, before they have been properly initialized, a given initialization pattern has to be loaded as a part of the JTAG control data. Before reloading new programming data, this control data should be written with a pattern different from the enable pattern to prevent possible bus contentions during reloading new programming data. At power up the TDC will be in a state where no TDCs will enable their bus drivers.

13.3. Byte-wise read-out

A byte-wise read-out scheme is available to enable a set of TDC chips to drive their data directly into a set of commercial serializer chips (Hot-link, LVDS links, etc.). The protocol is in principle identical to the parallel read-out mode except that the individual bytes are available on the 8 least significant bits of the parallel data bus. The MSB byte is sent first to enable the receiving end to immediately identify the type of data coming (header/trailer/data). Each byte must be handled by the data_ready/get_data handshake described previously. A header/data maker bit is available on bit 8 of the data bus which is only set high during the 4 bytes of header and trailer information. In addition a 2 bit Byte identifier is available on bit 9 and 10 of the data bus. A parity bit (even parity) of the 8 bit data is available on bit 11.

13.4. Serial read-out

The accepted TDC data can be transmitted serially over twisted pairs using LVDS or LV TTL signals. Data is transmitted in words of 32 bits with a start bit set to one and followed by a parity bit (even parity) and a stop bit set to zero. When no data is transmitted the serial line is kept at zero. The serialization speed is programmable from 80 to 0.3215 Mbits/s.

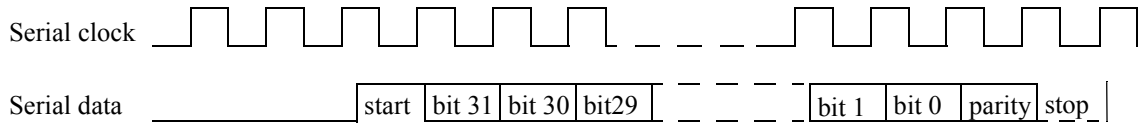


Fig. 13 Serial frame format with start bit and parity bit

In addition to the serialized data a strobe signal can carry timing information in a programmable format.

Clock: Direct serializing clock to strobe data on rising edge.

Edge: Edge strobe to strobe data on both rising and falling edge.

DS: DS strobe format as specified for transputer serial links. DS strobe only changes value when no change of serial data is observed. DS keeps toggleing when no data to send with the serial data = 0.

None: No strobe signal.

NOTE: at serial readout speeds below 40MHz the strobe signal is shifted 25ns after the serial data (see Design bugs and problems on page 99)

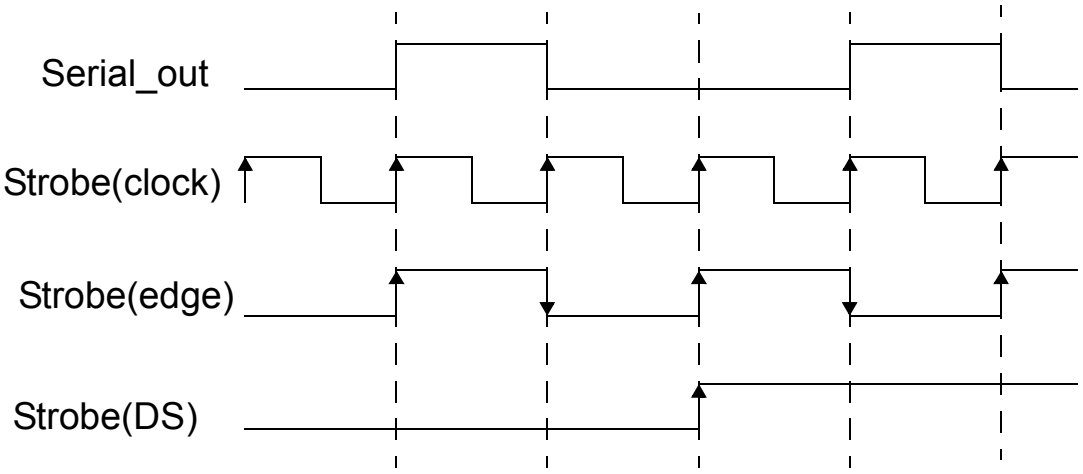


Fig. 14 Different strobe types.

The serial data is connected directly from one slave TDC into the following TDC. When a TDC does not have the token, the serial data is just re-timed with the serializing clock and directed to its output. The internal clocking of serial data prevents excessive delays to build up when passing through several slave TDC's. Programmable delays are available for the serial data input and token input to allow timing adjustments of the signal transmission between TDC's in critical cases. Only the master chip drives the serial data to the DAQ system, potentially via a twisted pair cable as shown in Fig. 16.

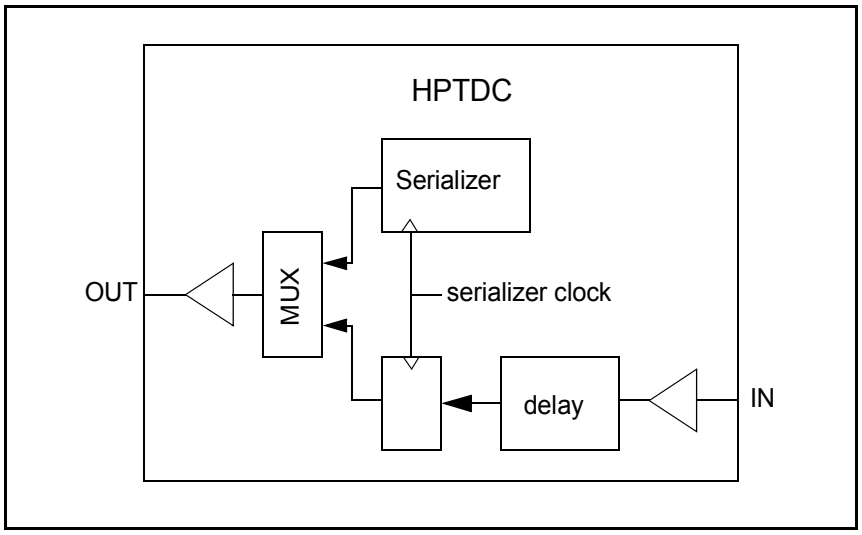


Fig. 15 Programmable delays on inputs of token and serial data

A serial read-out at 80MHz requires the PLL to be properly locked as the internal 80 MHz serialization clock is generated by the PLL.

13.5. Bypass

The use of a token ring to perform local event building is simple and effective but is sensitive to failures in single components in the chain. A set of additional inputs for the token and the serial data are available to enable a slave TDC to be bypassed in case of failure.

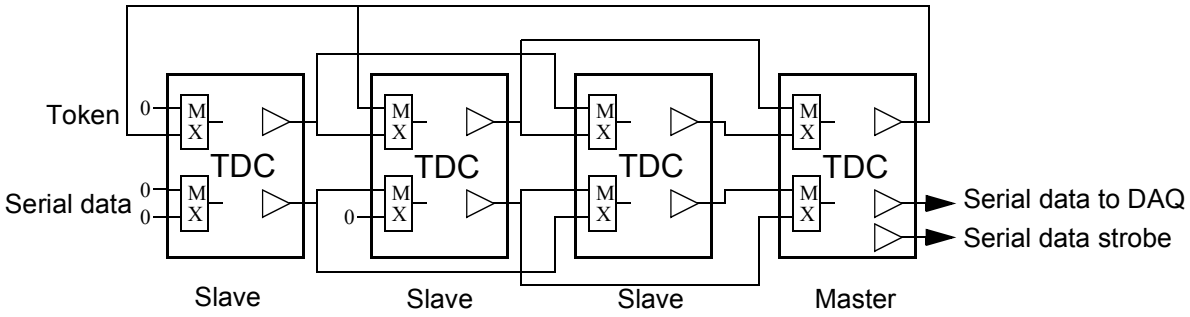


Fig. 16 Serial connection of TDC's with option of bypassing failing slave TDC

13.6. JTAG read-out

For debugging purposes a read-out interface is also available via JTAG. In this read-out mode the token passing between TDCs are disabled and individual words can be read from the JTAG read-out path. A status bit in the data read indicates if a data word was in fact ready for read-out. Each word read is removed from the read-out fifo to be capable of reading subsequent data.

13.7. Packet format

Data read out of the TDC is contained in 32 bit data packets. The first four bits of a packet are used to define the type of data packet. The following 4 bits are used to identify the ID of the TDC chip (programmable) generating the data. Only 8 out of the possible 16 packet types are defined for TDC data (bit 31 always set to zero by TDC). The remaining 8 packet types are available for data packets added by higher levels of the DAQ system.

Group header: Event header from master TDC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	TDC				Event ID												Bunch ID											

TDC: Programmed ID of master TDC.
Event ID: Event ID from event counter.
Bunch ID: Bunch ID of trigger (trigger time tag).

Group trailer: Event trailer from master TDC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	TDC				Event ID												Word count											

TDC: Programmed ID of master TDC.
Event ID: Event ID from event counter.
Word count: Total number of words in event (incl. headers and trailers).

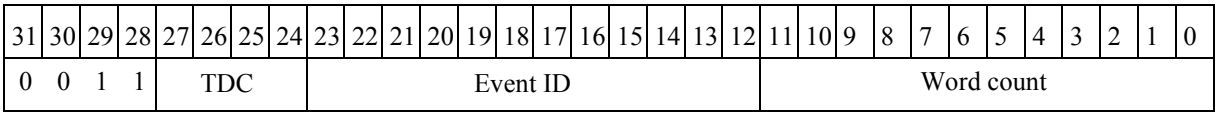
TDC header: Event header from TDC (master and slaves)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	TDC				Event ID												Bunch ID											

TDC: Programmed ID of TDC.
Event ID: Event ID from event counter.

Bunch ID: Bunch ID of trigger (trigger time tag).

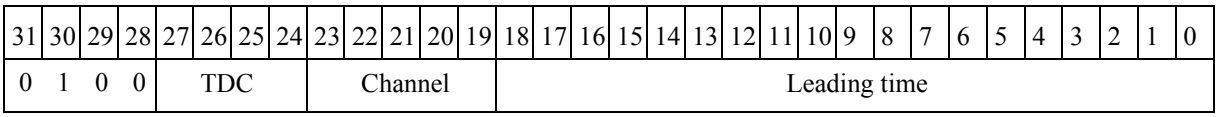
TDC trailer: Event trailer from TDC (master and slaves)



TDC: Programmed ID of TDC.
Event ID: Event ID from event counter.
Word count: Number of words from TDC (incl. headers and trailers).

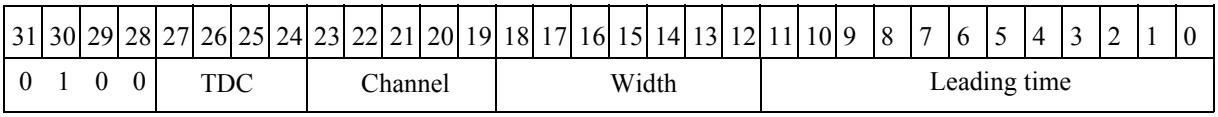
Leading measurement:

Single edge



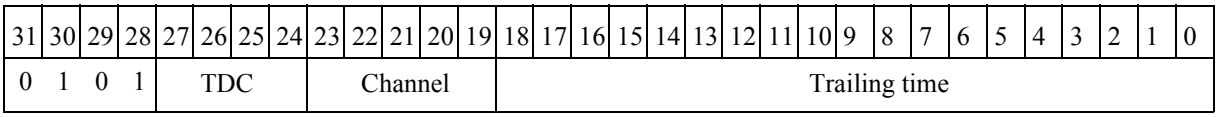
TDC: Programmed ID of TDC.
Channel TDC channel number.
Leading: Leading edge measurement in programmed resolution

Combined measurement of leading and trailing edge (pairing mode)



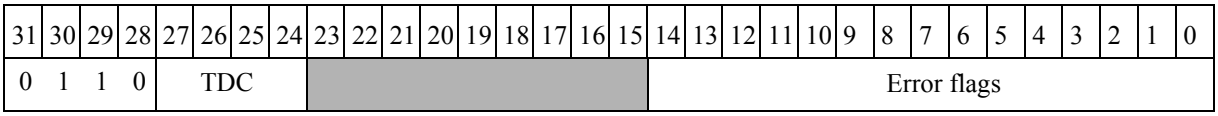
TDC: Programmed ID of TDC.
Channel: TDC channel number.
Width: Width of pulse in programmed time resolution.
Leading: Leading edge in programmed time resolution

Trailing measurement:



TDC: Programmed ID of TDC.
Channel TDC channel number.
Trailing: Trailing edge measurement in programmed resolution

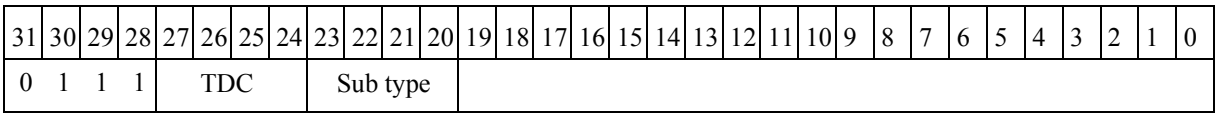
Errors: Error flags sent if an error condition has been detected

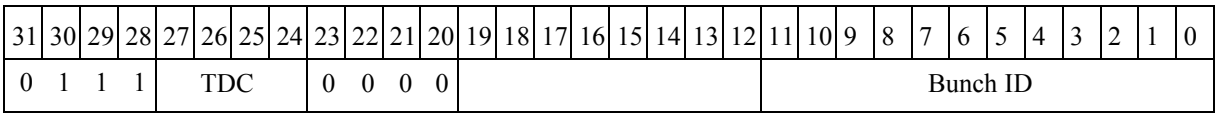


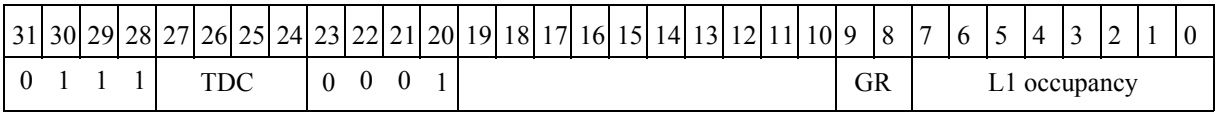
TDC: Programmed ID of TDC.
Error flags:

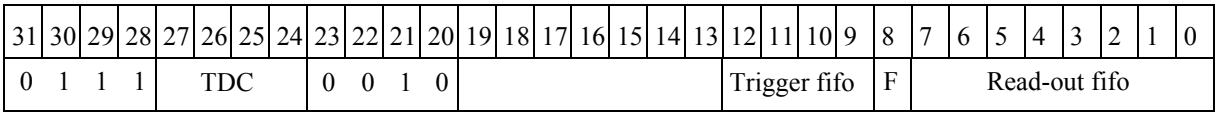
[0]: Hit lost in group 0 from read-out fifo overflow.
[1]: Hit lost in group 0 from L1 buffer overflow
[2]: Hit error have been detected in group 0.
[3]: Hit lost in group 1 from read-out fifo overflow.
[4]: Hit lost in group 1 from L1 buffer overflow
[5]: Hit error have been detected in group 1.
[6]: Hit data lost in group 2 from read-out fifo overflow.
[7]: Hit lost in group 2 from L1 buffer overflow
[8]: Hit error have been detected in group 2.
[9]: Hit lost in group 3 from read-out fifo overflow.
[10]: Hit lost in group 3 from L1 buffer overflow
[11]: Hit error have been detected in group 3.
[12]: Hits rejected because of programmed event size limit
[13]: Event lost (trigger fifo overflow).
[14]: Internal fatal chip error has been detected.

Debugging data: Additional information for system debugging









TDC: Programmed ID of TDC.
Sub type: Debugging data sub type:
0000: separator
0001: L1 Buffer occupancy (one for each of the four groups)
0010: Trigger and read-out fifo occupancy
Bunch ID: Trigger time tag counter when separator was generated.
L1 occupancy: L1 buffer occupancy.
GR: Channel Group.
Trigger fifo: Trigger fifo occupancy

Read-out fifo: Read-out fifo occupancy
F: Read-out fifo Full have been detected during matching.

13.7.1. Read-out format of very high resolution measurements.

When the TDC is configured to perform very high resolution time measurements, using four normal TDC channels per very high resolution channel, the read-out data can be formatted in two alternative ways.

If no on-chip compression is used each very high resolution measurement will generate four individual measurements (one from each sampling tap of the RC delay chain). From these four measurements the interpolation can be performed by determining between which of the four samples the time measurement changes one bin. The four individual measurements triggered by a single hit are not guaranteed to be read out of the TDC in consecutive sequence because of the round robin arbitration between the four channel groups at the input of the read-out FIFO. This read-out mode is only supposed to be used at low hit rates during special calibrations of the TDC.

The interpolation (data reduction) can also be performed on-chip, before being written into the L1 buffer, (when the RC delay chain is correctly calibrated) and in this case the last two bits of the channel number is used to indicate the interpolation factor. This will in practice mean that the two least significant bits of the time measurement in this case are mapped into the two least significant bits of the channel number.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	E	TDC				Channel			Inter.		Edge time																		

E: 0: Leading edge
1: Trailing edge
TDC: Programmed ID of TDC.
Channel TDC channel number (0 - 7).
0 = channel 0, 1 = channel 4, , , 7 = channel 28
Inter. Interpolation factor (2 least significant bits of time measurement)
Edge: Edge measurement in programmed resolution.

14. Error monitoring.

A high level of error monitoring is implemented in the TDC. Two principally different types of errors are dealt with in different ways. Errors that only have effects on a single (or a few) event is clearly indicated for each event, by sending a special error word just before the event trailer. When no trigger matching is used this error information is sent as soon as it has been identified within the TDC. Other types of errors fatal to the correct function of the TDC chip can be used to force the TDC chip into a special error state.

All functional blocks in the TDC are continuously monitored for error conditions. Memories are continuously checked with parity on all data. All internal state machines have been implemented with a “one hot” encoding scheme and are checked continuously for any illegal state. DLL signals captured when a hit is detected is checked to verify that the DLL is in a correct locking state. The JTAG programming data and the JTAG instruction register also have a parity check to

detect if any of the bits have been corrupted during loading or by a Single Event Upset (SEU). The error status of the individual parts of the TDC can be accessed via the JTAG status scan path.

Vernier error:	DLL signals in a hit measurement indicates that the DLL is not in correct lock. (or excessive jitter present on input clock)
Coarse error:	A parity error in the coarse count has been detected in a channel buffer.
Select error:	A synchronisation error has been detected in the priority logic used to select the channel being written into the L1 buffer.
L1 buffer error:	Parity error detected in a L1 buffer.
Trigger FIFO error:	Parity error detected in trigger FIFO.
Matching state error:	Illegal state detected in trigger matching logic.
Read-out FIFO error:	Parity error detected in read-out FIFO.
Read-out state error:	Illegal state detected in read-out logic.
Setup error:	Parity error detected in JTAG setup data.
Control error:	Parity error detected in JTAG control data.
JTAG error:	Parity error in JTAG instruction.

Any detected error condition in the TDC sets its corresponding JTAG error status bit which remains set until a global reset of the TDC is performed. All error status bits are or-ed together with programmable mask bits to generate a global error signal. When the global error signal becomes active the TDC can respond in a number of different ways:

Ignore:	No special action will be performed
Mark events:	All events being generated after the global error has been detected will be marked with a special error flag in the error word.
Bypass:	From the following event the TDC will suppress all its own data and directly pass the token and serial data.

The implemented error monitoring features will to a high extent ensure that a SEU in any part of the TDC will be detected. In most cases the TDC will be capable of reacting to this kind of failure without seriously affecting the function of the system. Certain SEU's will though be capable of changing the operating mode of the TDC (e.g. test modes, disabling of error checking, selection of read-out protocol, selection of clock source, etc.). In these cases the function of the TDC will become unpredictable, but the error status from JTAG can always be read to identify the cause of a failure.

15. Internal clocking of HPTDC

Seen from the outside, the HPTDC only works with a 40 MHz clock delivered with LVDS or LV TTL signal levels. Internally the HPTDC works with up to four different clocks at different frequencies to obtain the maximum performance. All internal clocks are driven directly from the external clock or from the on-chip PLL performing clock multiplication and jitter filtering. To ensure the correct function of the TDC, when using different clocks for different parts of the chip, a very tight synchronisation between clock domains is required and this can only be obtained if all clocks originate from the same clock reference.

For testing purpose (not for normal use) the different clocks can be skewed in relation to each other and can in addition be sourced from an independent external clock (aux_clock).

The DLL and the coarse time counter can be driven by a 40MHz, 160MHz or a 320MHz clock to obtain the different timing resolution modes of the TDC. This part of the TDC should run with the lowest possible clock frequency compatible with the timing requirements in a given application to minimize the power consumption of the TDC.

The serialization of read-out data at a bit rate of 80Mbits/s requires the use of a 80MHz serialization clock from the PLL.

The logic core of the TDC performing data buffering and trigger matching of obtained time measurements will normally run directly from the external 40 MHz clock. For applications with very high hit or trigger rates the data processing performance of the TDC can be improved by increasing the internal core logic clock to 80 MHz or possibly 160MHz. Using a higher clock frequency for the internal core of the chip will result in a significant increase in power consumption and should only be used if really required. Standard versions of the HPTDC chip will be capable of working with a core clock frequency up to 80MHz. 160MHz operation can only be guaranteed for special speed graded chips which have passed a special testing procedure (this will imply a significant price increase).

The IO interface of the HPTDC runs directly with the external clock source to ensure correct synchronisation with external logic.

The power consumption of the HPTDC depends strongly on the chosen clocking mode of the different parts of the chip. At power up, the TDC will have undefined data in its configuration data and may therefore startup in a high frequency clocking mode. A special power up function have been implemented to prevent a possible high power dissipation in this case.

16. Power up mode

When power is applied to the HPTDC all configuration data is in a undefined state and the behaviour of the chip is to a large extent unpredictable. To prevent potential dangerous situations, before the TDC has been fully configured and initialized, a set of special configuration bits (control[3:0] = enable_pattern[3:0]) in the control scan path has been defined. Two potentially problematic cases are handled by the use of these four special control bits.

When using a shared parallel readout bus it must be prevented that several TDCs on the same bus asserts their drivers at the same time. This is normally handled by the token passing mechanism, but this can not be expected to be fully working before the TDC is fully initialized. The output drivers are therefore disabled unless a specific pattern of 0101 is loaded into the enable_pattern[3:0].

In the first version of the HPTDC it was seen that the TDC in some cases after power up consumed power as if it was configured in a high power mode. This is in principle for the HPTDC

a legal working mode, but for low power applications of the HPTDC this may pose a significant problem as the cooling and power supply system may not be designed to handle this. In the second version of the HPTDC the enable_pattern has therefore also been used to force the HPTDC into a power-down mode at power up. The 3 MSB bits of the enable pattern must have a fixed pattern of 010 to get the chip out of this power-down mode. The fact that the LSB bit of the enable pattern is not included allows to control the output enable independently from the forcing of the power-down mode. When the HPTDC is in the power-down mode the PLL is powered down preventing it from generating any high frequency clocks for the DLL and the internal logic. The LVDS hit receivers will also be powered down. This ensures that the power consumption of the HPTDC in this power-down mode is always lower than any normal working mode.

The fact that the active enable patterns contains both logic 0 and logic 1 bits makes it very unlikely that any of the valid enable patterns will be present after a power up. In addition enable_pattern[3] has an asynchronous set from the JTAG trst signal. If the trst signal is kept low (trst is active low) for a short time after power has been applied, the enable_pattern[3:0] will all be forced into a pattern that forces the HPTDC into a fully passive state.

0101: Full power mode, Output drivers enabled

0100: Full power mode, Output drivers disabled

Any other pattern: power down mode and output drivers disabled

The correct sequence to initialize and power up the HPTDC after power has been applied is the following:

1: Load valid configuration data into the setup scan path.

2: Apply valid clock to HPTDC clock input

3: Load control data with PLL_reset = 1, DLL_reset = 1 and enable_pattern = 0100 (reset PLL and DLL)

4: Load control data with PLL_reset = 0, DLL_reset = 1 and enable_pattern = 0100 (PLL locking)

5: Load control data with PLL_reset = 0, DLL_reset = 0 and enable_pattern = 0100 (DLL locking)

6: Reset all chips on shared readout bus with global reset signal

7: Load control data with PLL_reset = 0, DLL_reset = 0 and enable_pattern = 0101 (enable output drivers)

If the PLL is not used step 4 can be skipped. If not using the parallel readout, step 7 can be skipped.

17. JTAG Test and Programming Port.

A JTAG (Joint Test Action Group, IEEE 1149.1 standard) port is used to program the programmable features in the TDC, and also to get access to test facilities built into the TDC. Full boundary scan is supported to be capable of performing extensive testing of TDC modules. In addition special JTAG registers have been included in the data path of the chip to be capable of performing effective testing of registers and embedded memory structures.

Programming of the device is separated into two scan path groups. The setup group consists of setups which can not be changed while the system is actively running and a control group which can be changed during a run (enable and disable of noisy channels). To save silicon area for the large shift register for the setup data it does not have an update register as normally seen on JTAG scan paths. The setup data can still be read, but the read will be "destructive" and new setup data must be written at the same time as it is read (simple shift register).

A set of system test features have been added to the JTAG functions. The normal read-out of the TDC chip can be disabled in a special test mode and the output data from the TDC can be read via JTAG on a word by word basis.

The instruction and setup/control scan paths have a parity bit which is continuously monitored for potential parity errors caused by single event upsets. Any detected parity error will set the respective error bit in the JTAG status register. After writing new information to these scan paths (especially setup scan path as this has no update register) the error status bits must be cleared by a global HPTDC reset to get a valid reading of the parity errors, as the status bits remember any detected error condition since the last general HPTDC reset.

It is brought to the attention that the JTAG signals do not have pull-up resistors as defined in the JTAG standard as the IO library used did not have such an option.

17.1. JTAG tap controller

The JTAG TAP (Test Access Port) controller uses the standard JTAG state diagram as shown in the figure below.

Asserting the JTAG TRST signal (active low) will asynchronously force the TAP controller into the test logic reset state and also enforce a low power state of the HPTDC (see also Power up mode on page 26)

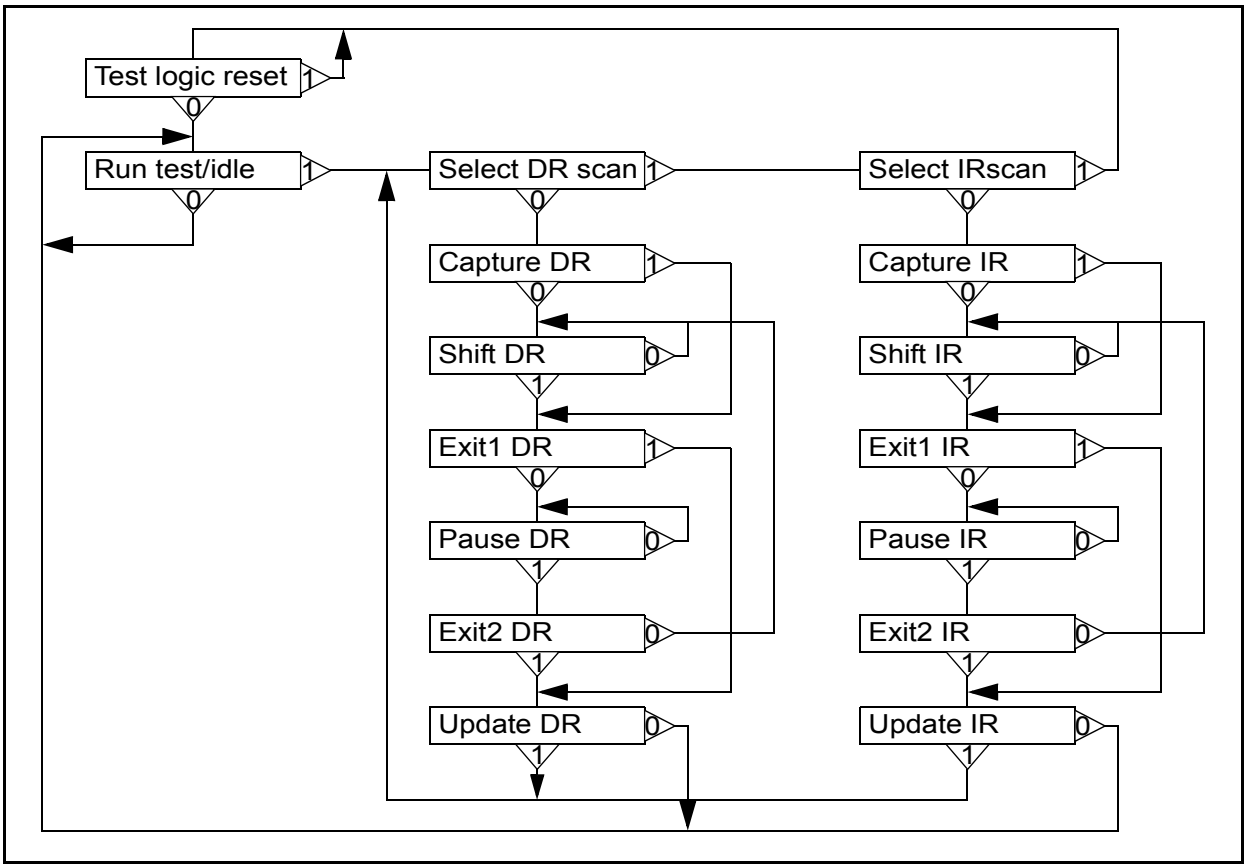


Fig. 17 TAP state diagram as function of JTAG TMS

17.2. JTAG instructions

The JTAG instruction register is 4 bits long plus a parity bit. The JTAG instruction register can be read back (Often not possible in many JTAG implementations)

[3:0]	ins[3:0]	JTAG instruction
[4]	parity[0]	parity of JTAG instruction (even parity)

Instruction:	**Name**	**Description**
0000:	EXTEST.	Boundary scan for test of inter-chip connections on module.
0001:	IDCODE.	Scan out of chip identification code.
0010:	SAMPLE.	Sample of all chip pins via boundary scan registers.
0011:	INTEST.	Using boundary scan registers to test chip itself.
0100:	BIST	Memory self tests (production testing)
0101:	SCAN	Scan of all internal flip-flops (production testing)
0110:	NOT IMPLEMENTED.	
0111:	Readout	Read of TDC output data
1000:	Setup	Load/read of setup data.
1001:	Control	Load/read of control information.
1010:	Status	Read of status information.

1011: Coretest Access to internal coretest scan registers (production testing)

1100 - 1110: NOT IMPLEMENTED.

1111 BYPASS.

During normal operation the JTAG instruction register must not contain the EXTEST, INTEST, BIST or SCAN instructions as these changes the function of the HPTDC into special test modes related to scan path testing.

17.3. Boundary scan register.

All signal pins of the TDC are passed through JTAG boundary scan registers. All JTAG test modes related to the boundary scan registers are supported (EXTEST, INTEST, SAMPLE).

BSR #	**Pin name**	**Description**
[0]	token_out	LVTTL or LVDS outputs depending on setup
[1]	strobe_out	LVTTL or LVDS outputs depending on setup
[2]	serial_out	LVTTL or LVDS outputs depending on setup
[3]	test	
[4]	error	
[5]	data_ready	
[6]	parallel_enable	enable of parallel_data_out (not direct pin)
[38:7]	parallel_data_out[31:0]	
[39]	encoded_control	LVTTL or LVDS mode depending on setup
[40]	trigger	LVTTL or LVDS mode depending on setup
[41]	event_reset	LVTTL or LVDS mode depending on setup
[42]	bunch_reset	LVTTL or LVDS mode depending on setup
[43]	get_data	
[44]	serial_bypass_in	LVTTL or LVDS mode depending on setup
[45]	serial_in	LVTTL or LVDS mode depending on setup
[46]	token_bypass_in	LVTTL or LVDS mode depending on setup
[47]	token_in	LVTTL or LVDS mode depending on setup
[48]	reset	LVTTL or LVDS mode depending on setup
[49]	aux_clock	LVTTL or LVDS mode depending on setup
[50]	clk	LVTTL or LVDS mode depending on setup
[82:51]	hit	LVTTL or LVDS mode depending on setup

17.4. ID code

A 32 bit chip identification code can be shifted out when selecting the ID shift chain.

ID[31:0] 10000100011100001101101011001110 Version 1.3 of HPTDC

17.5. Setup registers

The JTAG setup scan path is used to load programming data that can not be changed while the TDC is actively running. No separate update register is implemented on this scan path so reading the information is "destructive". Proposed default values are given in parenthesis when applicable.

[3:0]	test_select[3:0]	Selection of test output signal (for testing) normal use = 1110 0000 clock_40 0001 pll_clock_40

		0010 pll_clock_80 0011 pll_clock_160 0100 pll_phase_detector 0101 pll_frequency_detector 0110 pll_lock 0111 dll_end inverted 1000 dll_phase_detector 1001 dll_lock 1010 dll_clock 1011 core_clock 1100 io_clock 1101 serial_clock 1110 constant_low 1111 constant_high
[4]	enable_error_mark	Mark events with error if global error signal set (1)
[5]	enable_error_bypass	Bypass TDC chip if global error signal set (0) (overrides enable_error_mark)
[16:6]	enable_error[10:0]	Enable of internal error types for generation of global error signal (all set) bit # 0: vernier error (DLL unlocked or excessive jitter) 1: coarse error (parity error on coarse count) 2: channel select error (synchronisation error) 3: l1 buffer parity error 4: trigger fifo parity error 5: trigger matching error (state error) 6: read-out fifo parity error 7: read-out state error 8: setup parity error 9: control parity error 10: jtag instruction parity error
[19:17]	readout_single_cycle_speed[2:0]	Serial transmission speed in single cycle mode (0) 000: 40 Mbits/s 001: 20 Mbits/s 010: 10 Mbits/s 011: 5 Mbits/s 100: 2,5 Mbits/s 101: 1.25 Mbits/s 110: 0.625 Mbits/s 111: 0.3125 Mbits/s
[23:20]	serial_delay[3:0]	Programmable delay of serial input (0) Time unit ~ 1ns
[25:24]	strobe_select[1:0]	Selection of serial strobe type (0): 00: no strobe 01: DS strobe 10: leading and trailing edge (edge) 11: leading edge (clock)
[26]	readout_speed_select	Selection of serial read-out speed (0) 0: single cycle

		(as defined by setup[19:17]) 1: 80 Mbits/s (PLL lock required)
[30:27]	token_delay[3:0]	Programmable delay of token input (0) Time unit ~ 1ns
[31]	enable_local_trailer	Enable of local trailers in read-out (1)
[32]	enable_local_header	Enable of local headers in read-out (1)
[33]	enable_global_trailer	Enable of global trailers in read-out (0) (only valid for master TDC)
[34]	enable_global_header	Enable of global headers in read-out (0) (only valid for master TDC)
[35]	keep_token	Keep token until end of event or no more data (1) otherwise pass token after each word read. Must be enabled when using trigger matching.
[36]	master	Master chip in token ring (0)
[37]	enable_bytewise	Enable of byte-wise readout (0) (setup[38] must be 0)
[38]	enable_serial	Enable of serial read-out (otherwise parallel read-out) (0)
[39]	enable_jtag_readout	Enable of readout via JTAG (0) (overrides other readout modes)
[43:40]	tdc_id[3:0]	TDC identifier in readout
[44]	select_bypass_inputs	Select serial in and token in from bypass inputs (0)
[47:45]	readout_fifo_size[2:0]	Effective size of readout fifo (111) 000: 2 001: 4 010: 8 011: 16 100: 32 101: 64 110: 128 111: 256
[59:48]	reject_count_offset[11:0]	Reject counter offset defines reject latency together with coarse count offset
[71:60]	search_window[11:0]	Search window in number of clock cycles
[83:72]	match_window[11:0]	Matching window in number of clock cycles (0 = 25ns, 1 = 50ns, 2 = 75ns , ,)
[86:84]	leading_resolution[2:0]	Resolution of leading/trailing edge (001) used to select bits read out. 000: 100ps (coarse count[11] lost in readout) 001: 200ps 010: 400ps (only useful in pair mode) 011: 800ps (only useful in pair mode) 100: 1.6ns (only useful in pair mode) 101: 3.12ns (only useful in pair mode) 110: 6.25ns (only useful in pair mode) 111: 12.5ns (only useful in pair mode)
[114:87]	fixed_pattern[27:0]	Fixed readout pattern (for debugging) [27:24] readout_data[31:28] illegal pattern = 0010

		illegal pattern = 0011
		[23:0] readout data [23:0]
		readout data[27:24] taken from tdc_id
[115]	enable_fixed_pattern	Enable readout of fixed pattern (for debugging)
[119:116]	max_event_size[3:0]	Maximum number of hits per event (1001) only valid when using trigger matching 0000: 0 0001: 1 0010: 2 0011: 4 0100: 8 0101: 16 0110: 32 0111: 64 1000: 128 1001: no limit 1010 - 1111: illegal
[120]	reject_readout_fifo_full	Reject hits when read-out fifo full. (1) otherwise back propagate to L1 buffers
[121]	enable_readout_occupancy	Enable readout of buffer occupancies (0) for each event (debugging). Only allowed when using trigger matching.
[122]	enable_readout_separator	Enable read-out of separators (debugging) (0) only valid if generation of separators enabled
[123]	enable_overflow_detect	Enable overflow detect of L1 buffers (1) (should always be enabled)
[124]	enable_relative	Enable read-out of relative time to trigger time tag only valid when using trigger matching
[125]	enable_automatic_reject	Enable of automatic rejection (1) (should always be enabled if trigger matching)
[137:126]	event_count_offset[11:0]	Event number offset (0)
[149:138]	trigger_count_offset[11:0]	Trigger time tag counter offset used to set effective trigger latency
[150]	enable_set_counters_on_bunch_reset	Enable all counters to be set on bunch count reset (1)
[151]	enable_master_reset_code	Enable master reset code on encoded_control (0)
[152]	enable_master_reset_on_event_reset	Enable master reset of whole TDC on event reset (0)
[153]	enable_reset_channel_buffer_when_separator	Enable reset channel buffers when separator (0)
[154]	enable_separator_on_event_reset	Enable generation of separator on event reset (0)
[155]	enable_separator_on_bunch_reset	Enable generation of separator on bunch reset (0)
[156]	enable_direct_event_reset	Enable of direct event reset input pin (1) otherwise taken from encoded control
[157]	enable_direct_bunch_reset	Enable of direct bunch reset input pin (1) otherwise taken from encoded control
[158]	enable_direct_trigger	Enable of direct trigger input pin (1)

		otherwise taken from encoded control
[167:159]	offset31[8:0]	Offset adjust for channel 31
[176:168]	offset30[8:0]	Offset adjust for channel 30
[185:177]	offset29[8:0]	Offset adjust for channel 29
[194:186]	offset28[8:0]	Offset adjust for channel 28
[203:195]	offset27[8:0]	Offset adjust for channel 27
[212:204]	offset26[8:0]	Offset adjust for channel 26
[221:213]	offset25[8:0]	Offset adjust for channel 25
[230:222]	offset24[8:0]	Offset adjust for channel 24
[239:231]	offset23[8:0]	Offset adjust for channel 23
[248:240]	offset22[8:0]	Offset adjust for channel 22
[257:249]	offset21[8:0]	Offset adjust for channel 21
[266:258]	offset20[8:0]	Offset adjust for channel 20
[275:267]	offset19[8:0]	Offset adjust for channel 19
[284:276]	offset18[8:0]	Offset adjust for channel 18
[293:285]	offset17[8:0]	Offset adjust for channel 17
[302:294]	offset16[8:0]	Offset adjust for channel 16
[311:303]	offset15[8:0]	Offset adjust for channel 15
[320:312]	offset14[8:0]	Offset adjust for channel 14
[329:321]	offset13[8:0]	Offset adjust for channel 13
[338:330]	offset12[8:0]	Offset adjust for channel 12
[347:339]	offset11[8:0]	Offset adjust for channel 11
[356:348]	offset10[8:0]	Offset adjust for channel 10
[365:357]	offset9[8:0]	Offset adjust for channel 9
[374:366]	offset8[8:0]	Offset adjust for channel 8
[383:375]	offset7[8:0]	Offset adjust for channel 7
[392:384]	offset6[8:0]	Offset adjust for channel 6
[401:393]	offset5[8:0]	Offset adjust for channel 5
[410:402]	offset4[8:0]	Offset adjust for channel 4
[419:411]	offset3[8:0]	Offset adjust for channel 3
[428:420	offset2[8:0]	Offset adjust for channel 2
[437:429]	offset1[8:0]	Offset adjust for channel 1
[446:438]	offset0[8:0]	Offset adjust for channel 0
[458:447]	coarse_count_offset[11:0]	Offset for coarse time counter
[554:459]	dll_tap_adjust[95:0]	Adjustment of DLL taps (0) resolution: ~10ps [2:0]: tap 0 [5:3]: tap1 - - - - - - - - [95:93]: tap 31 normal use: all adjustments can be set to zero
[566:555]	rc_adjust[11:0]	Adjustment of R-C delay line.. rc_adjust[2:0] = tap1 (7) rc_adjust[5:3] = tap2 (7) rc_adjust[8:6] = tap3 (4) rc_adjust[11:9] = tap4 (2) only needed in very high resolution RC mode
[569:567]	not used	(rc_adjust 14:12)
[570]	low_power_mode	Low power mode of channel buffers (1)

		(rc_adjust[15])
[574:571]	width_select[3:0]	Pulse width resolution when paired measurements 0000: 100ps 0001: 200ps 0010: 400ps 0011: 800s 0100: 1.6ns 0101: 3.2ns 0110: 6.25ns 0111: 12.5ns 1000: 25ns 1001: 50ns 1010: 100ns 1011: 200ns 1100: 400ns 1101: 800ns 1110: not valid 1111: not valid
[579:575]	vernier_offset[4:0]	Offset in vernier decoding (0) fixed value = 00000
[583:580]	dll_control[3:0]	Control of dll (update) DLL charge pump levels (0001)
[585:584]	dead_time[1:0]	Channel dead time between hits (0) 00: ~5ns 01: ~10ns 10: ~30ns 11: ~100ns
[586]	test_invert	Automatic inversion of test pattern (0) only used during production testing normal use = 0
[587]	test_mode	Test mode where hit data taken from coretest. (0) Only used during production testing normal use = 0
[588]	enable_trailing	Enable of trailing edges
[589]	enable_leading	Enable of leading edges
[590]	mode_rc_compression	Perform RC interpolation on-chip. only valid in very high resolution mode
[591]	mode_rc	Enable of R-C delay line mode (very high res. mode) only channel 0,4,8,12,16,20,24,28 active
[593:592]	dll_mode[1:0]	Selection of DLL speed mode (update) 00: 40MHz 01: 160MHz 10: 320MHz 11: illegal must match selected clock speed for DLL
[601:594]	pll_control[7:0]	Control of PLL (update) [4:0]: charge pump current (00100) [5]: power down mode (0) [6]: enable test outputs (0)

		[7]: invert connection to status[61]
[605:602]	serial_clock_delay[3:0]	Delay of internal serial clock (update) (0) [3]: 0 direct clock 1 delayed clock [2:0]: delay in step of typical 0.13 ns normal use: fixed pattern = 000
[609:606]	io_clock_delay[3:0]	Delay of internal io clock (update) (0) [3]: 0 direct clock 1 delayed clock [2:0]: delay in step of typical 0.13 ns normal use: fixed pattern = 000
[613:610]	core_clock_delay[3:0]	Delay of internal core clock (update) (0) [3]: 0 direct clock 1 delayed clock [2:0]: delay in step of typical 0.13 ns normal use: fixed pattern = 000
[617:614]	dll_clock_delay[3:0]	Delay of internal dll clock (update) (0) [3]: 0 direct clock 1 delayed clock [2:0]: delay in step of typical 0.13 ns normal use: fixed pattern = 000
[619:618]	serial_clock_source[1:0]	selection of source for serial clock (update) (0) 00: pll_clock_80 01: (pll_clock_160), not valid 10: (pll_clock_40), not valid 11: aux_clock (only for testing) normal use: fixed pattern = 00
[621:620]	io_clock_source[1:0]	Selection of clock source for IO signals (update) (0) 00: clock_40 01: (pll_clock_80), not valid 10: (pll_clock_160), not valid 11: aux_clock (only for testing) normal use: fixed pattern = 00
[623:622]	core_clock_source[1:0]	Selection of clock source for internal logic (update) 00: clock_40 01: pll_clock_80 10: pll_clock_160 (only valid for speed graded ICs) 11: aux_clock (only for testing) normal use: fixed pattern = 00
[626:624]	dll_clock_source[2:0]	Selection of clock source for DLL (update) 000: clock_40 (low resolution mode with direct clock) 001: pll_clock_40 (low resolution with PLL clock) 010: pll_clock_160 (medium resolution) 011: pll_clock_320 (high resolution) 100: aux_clock (only for testing)
[638:627]	roll_over[11:0]	Counter roll over value. (FFF hex) defines max count value from where counters will go to zero.
[639]	enable_matching	Enable of trigger matching

[640]	enable_pair	Enable pairing of leading and trailing edges (overrides individual enable of leading/trailing edges) not allowed in very high resolution RC mode
[641]	enable_ttl_serial	Enable LV TTL input on: serial_in serial_bypass_in token_in token_bypass_in Otherwise uses LVDS input levels Disables LVDS drivers on: serial_out strobe_out token_out
[642]	enable_ttl_control	Enable LV TTL input on: trigger bunch_reset event_reset encoded_control otherwise uses LVDS input levels
[643]	enable_ttl_reset	Enable LV TTL input on: reset otherwise uses LVDS input levels
[644]	enable_ttl_clock	Enable LV TTL inputs on: (update) clk aux_clock otherwise uses LVDS input levels
[645]	enable_ttl_hit	Enable LV TTL input on: hit[31:0] otherwise uses LVDS input levels
[646]	setup_parity	Parity of setup data (even parity)

17.6. Control registers

The JTAG control scan path is used to enable/disable channels which can be done while the TDC is actively running. The control scan path is also use to initialize the PLL and DLL after power up. A global reset (equivalent to asserting the reset pin) can be issued to initialize the global state of the TDC after power up and reconfiguring.

[3:0]	enable_pattern[3:0]	Specific pattern = 0101 required to enable drivers on parallel readout bus. Specific pattern = 010x required to get out of power-down mode. JTAG trst sets enable_pattern[3] to 1. see Power up mode on page 26
[4]	Global_reset	Global reset of TDC via JTAG. (must be set to one and returned to zero)
[36:5]	enable_channel[31:0]	Enable/disable of individual channels
[37]	dll_reset	Reset of DLL (must be set to one and returned to zero)
[38]	pll_reset	Reset of PLL (must be set to one and returned to zero)
[39]	control_parity	Parity of control data (even parity)

17.6.1.Reset of PLL and DLL.

The PLL and the DLL is not initialized when a global reset of the TDC chip is performed. The lock tracing of the PLL/DLL is a rather slow process (few ms) and is only required to be performed once power and a stable clock have been applied.

A reset of the PLL and DLL must be performed via the JTAG control scan path after power up. The PLL must first be initialized and obtain lock before the DLL can be initialized. Their correct parameters in the setup scan path must have been loaded before they can be initialized via their respective reset bits in the control scan path. Reloading new setup data may result in the DLL and PLL losing lock.

17.7. Status registers

The JTAG status scan path is used to get access to the status of the TDC while it is running (or after a run). It is possible to monitor all internal buffers by sampling their occupancies when the JTAG status registers are read. This sampling is performed using the JTAG clock as a reference. As the JTAG clock in most cases is uncorrelated to the main clock of the TDC this sampling of occupancies may in some cases result in incorrect buffer occupancies to be read out (if sampling occupancies when they are in the middle of changing)

[10:0]	error[10:0]	status of error monitoring bit # 0:vernier error 1:coarse error 2:channel select error 3:l1 buffer parity error 4:trigger fifo parity error 5:trigger matching state error 6:readout fifo parity error 7:readout state error 8:setup parity error 9:control parity error 10:jtag instruction error
[11]	have_token	TDC have read-out token
[19:12]	readout_fifo_occupancy[7:0]	occupancy of read-out fifo
[20]	readout_fifo_full	read-out fifo full
[21]	readout_fifo_empty	read-out fifo empty
[29:22]	group3_l1_occupancy[7:0]	occupancy of L1 buffer of channel group 3
[37:30]	group2_l1_occupancy[7:0]	occupancy of L1 buffer of channel group 2
[45:38]	group1_l1_occupancy[7:0]	occupancy of L1 buffer of channel group 1
[53:46]	group0_l1_occupancy[7:0]	occupancy of L1 buffer of channel group 0
[57:54]	trigger_fifo_occupancy[3:0]	occupancy of trigger fifo
[58]	trigger_fifo_full	trigger fifo full
[59]	trigger_fifo_empty	trigger fifo empty
[60]	dll_lock	DLL in lock.
[61]	not used	(inverted setup[601])

17.8. Read-out via JTAG

When the TDC has been configured to be in the JTAG read-out mode (setup[39]) the normal read-out interfaces will be disabled and the data available in the read-out fifo can be read out on a word by word basis from JTAG. Each time the JTAG update signal is generated the JTAG will access and remove a single data word from the read-out FIFO (if any data available).

[0]	valid_data	valid read-out data (otherwise read-out FIFO empty)
[32:1]	readout_data[31:0]	read-out data

17.9. Memory self test

With the BIST instruction all internal memories will be put in a special self test mode. The self test of the memories will be started when entering the JTAG runbist state. A minimum number of 3200 JTAG clocks must be given in the runbist state to finalize the memory tests. After this the memory self test results can be read out on the serial JTAG output.

[0]	readout_end	Test of readout fifo ended
[1]	readout_fail	Test of readout fifo failed.
[2]	group3_end	Test of group 3 L1 buffer ended
[3]	group3_fail	Test of group 3 L1 buffer failed.
[4]	group2_end	Test of group 2 L1 buffer ended
[5]	group2_fail	Test of group 2 L1 buffer failed.
[6]	group1_end	Test of group 1 L1 buffer ended
[7]	group1_fail	Test of group 1 L1 buffer failed.
[8]	group0_end	Test of group 0 L1 buffer ended.
[9]	group0_fail	Test of group 0 L1 buffer failed.

The end signal is initialized to low when the runbist state is entered and is only asserted when sufficient JTAG clocks has been generated to finalize the memory tests. The fail signal is initialized to high when the runbist state is entered and is only set low when sufficient JTAG clocks has been generated to finalize the memory test and no errors have been detected.

17.10.Internal coretest registers

The JTAG coretest scan path is used to perform extended testing of the TDC chip. The internal scan path gives direct access to the interface between the channel buffers and first level buffer logic. It is used in connection with the test mode select bits in the setup scan path and is only intended for verification and production tests of the TDC chip.

[8:0]	common_matching_state[8:0]	idle	000000001
		header	000000010
		lost_header	000000100
		error	000001000
		trailer	000010000
		lost_trailer	000100000
		separator	001000000
		occupancy	010000000
		matching	100000000
[35:9]	trigger_data[26:0]	[11:0]	bunch id
		[23:12]	event_id
		[24]	separator
		[25]	trigger lost
		[26]	parity

[36]	trigger_ready		
---------- GROUP 3 ----------			
[42:37]	group3_matching_state[5:0]	idle	00001
		write_occupancy	00010
		active	00100
		waiting_for_separator	01000
		wait_end	10000
[77:43]	group3_l1_data[34:0]		
	COMBINED MEASUREMENT		
		[7:0]	leading edge fine time
		[19:8]	leading edge coarse time
		[26:20]	width
		[29:27]	channel
		[30]	error
		[31]	overflow start
		[32]	overflow stop
		[33]	separator
		[34]	parity
	SINGLE MEASUREMENT		
		[7:0]	edge fine time
		[19:8]	edge coarse time
		[20]	edge type
		[26:21]	undefined (0)
		[29:27]	channel
		[30]	error
		[31]	overflow start
		[32]	overflow stop
		[33]	separator
		[34]	parity
[78]	group3_l1_empty		
[79]	group3_l1_ready		
---------- GROUP 2 ----------			
[85:80]	group2_matching_state[5:0]	idle	00001
		write_occupancy	00010
		active	00100
		waiting_for_separator	01000
		wait_end	10000
[120:86]	group2_l1_data[34:0]		
	COMBINED MEASUREMENT		
		[7:0]	leading edge fine time
		[19:8]	leading edge coarse time
		[26:20]	width
		[29:27]	channel
		[30]	error
		[31]	overflow start
		[32]	overflow stop
		[33]	separator
		[34]	parity
	SINGLE MEASUREMENT		

		[7:0]	edge fine time
		[19:8]	edge coarse time
		[20]	edge type
		[26:21]	undefined (0)
		[29:27]	channel
		[30]	error
		[31]	overflow start
		[32]	overflow stop
		[33]	separator
		[34]	parity
[121]	group2_l1_empty		
[122]	group2_l1_ready		
---------- GROUP 1 ----------			
[128:123]	group1_matching_state[5:0]	idle	00001
		write_occupancy	00010
		active	00100
		waiting_for_separator	01000
		wait_end	10000
[163:129]	group1_l1_data[34:0]		
	COMBINED MEASUREMENT		
		[7:0]	leading edge fine time
		[19:8]	leading edge coarse time
		[26:20]	width
		[29:27]	channel
		[30]	error
		[31]	overflow start
		[32]	overflow stop
		[33]	separator
		[34]	parity
	SINGLE MEASUREMENT		
		[7:0]	edge fine time
		[19:8]	edge coarse time
		[20]	edge type
		[26:21]	undefined (0)
		[29:27]	channel
		[30]	error
		[31]	overflow start
		[32]	overflow stop
		[33]	separator
		[34]	parity
[164]	group1_l1_empty		
[165]	group1_l1_ready		
---------- GROUP 0 ----------			
[171:166]	group0_matching_state[5:0]	idle	00001
		write_occupancy	00010
		active	00100
		waiting_for_separator	01000
		wait_end	10000
[206:172]	group0_l1_data[34:0]		

	COMBINED MEASUREMENT		
		[7:0]	leading edge fine time
		[19:8]	leading edge coarse time
		[26:20]	width
		[29:27]	channel
		[30]	error
		[31]	overflow start
		[32]	overflow stop
		[33]	separator
		[34]	parity
	SINGLE MEASUREMENT		
		[7:0]	edge fine time
		[19:8]	edge coarse time
		[20]	edge type
		[26:21]	undefined (0)
		[29:27]	channel
		[30]	error
		[31]	overflow start
		[32]	overflow stop
		[33]	separator
		[34]	parity
[207]	group0_l1_empty		
[208]	group0_l1_ready		
---------- GROUP 3 ----------			
[338:209]	hit_data[129:0]	[31:0]	first vernier
		[46:32]	first coarse 1
		[47]	first coarse 1 parity
		[62:48]	first coarse 2
		[63]	first coarse 2 parity
		[64]	first edge type
		[96:65]	second vernier
		[111:97]	second coarse 1
		[112]	second coarse 1 parity
		[127:113]	second coarse 2
		[128]	second coarse 2 parity
		[129]	second edge type
[341:339]	hit_channel[2:0]	channel	
[342]	hit_select_error[0]	select error detected	
[343]	hit_load[0]	load hit data	
---------- GROUP 2 ----------			
[473:344]	hit_data[129:0]	[31:0]	first vernier
		[46:32]	first coarse 1
		[47]	first coarse 1 parity
		[62:48]	first coarse 2
		[63]	first coarse 2 parity
		[64]	first edge type
		[96:65]	second vernier
		[111:97]	second coarse 1
		[112]	second coarse 1 parity

		[127:113]	second coarse 2
		[128]	second coarse 2 parity
		[129]	second edge type
[476:474]	hit_channel[2:0]	channel	
[477]	hit_select_error[0]	select error detected	
[478]	hit_load[0]	load hit data	
---------- GROUP 1 ----------			
[608:479]	hit_data[129:0]	[31:0]	first vernier
		[46:32]	first coarse 1
		[47]	first coarse 1 parity
		[62:48]	first coarse 2
		[63]	first coarse 2 parity
		[64]	first edge type
		[96:65]	second vernier
		[111:97]	second coarse 1
		[112]	second coarse 1 parity
		[127:113]	second coarse 2
		[128]	second coarse 2 parity
		[129]	second edge type
[611:609]	hit_channel[2:0]	channel	
[612]	hit_select_error[0]	select error detected	
[613]	hit_load[0]	load hit data	
---------- GROUP 0 ----------			
[743:614]	hit_data[129:0]	[31:0]	first vernier
		[46:32]	first coarse 1
		[47]	first coarse 1 parity
		[62:48]	first coarse 2
		[63]	first coarse 2 parity
		[64]	first edge type
		[96:65]	second vernier
		[111:97]	second coarse 1
		[112]	second coarse 1 parity
		[127:113]	second coarse 2
		[128]	second coarse 2 parity
		[129]	second edge type
[746:744]	hit_channel[2:0]	channel	
[747]	hit_select_error[0]	select error detected	
[748]	hit_load[0]	load hit data	

17.11. Scan

In this special mode all flip-flops in the TDC will be configured as a large shift register which is used to perform exhaustive production tests. To enable all flip-flops in the synchronous logic to be clocked by the JTAG clock during shifting, the source of all internal clocks must be programmed to be the aux_clock (serial_clock_source[1:0] = 10, io_clock_source[1:0] = 10, core_clock_source[1:0] = 10, dll_clock_source[2:0] = 100). During the active shifting of the internal scan path the internal aux_clock will be derived from the JTAG clock.

To load the state of the internal logic into the flip-flops in this test mode the external aux_clock must be used. During shifting data in the scan path, the aux_clock must be driven high.

To load the response of the internal logic into the internal flip-flops the JTAG TAP controller in the TDC must exit the shift_dr state. The aux_clock must be driven low and again driven to its passive high state. The response of the internal logic can now be read out from the scan path.

The frozen state of the TDC can also be read out by stopping all clocks at a given point (this can not be done on clocks generated by the PLL), program all internal clocks to be driven from the aux_clock (which is driven actively high) and finally shift out the internal scan path.

The total number of flip-flops in the internal scan path is 1829.

18. Calibration of very high resolution mode

Before the TDC can be assumed to perform correct time measurements in the very high resolution mode the RC delay line must have been properly calibrated. A set of default calibration constants for 40 MHz operation (320 MHz from PLL) and nominal Vdd is proposed. Differences between chips and differences in operational conditions may though require corrections to these to be made. When appropriate calibration constants have been determined they can be considered stable if less than +/- 10% Vdd and less than +/- 25 ^{0}C variations in operational conditions are maintained.

To perform a correct calibration it is advantageous to read out the four individual time measurements made from the four taps of the RC delay chain (no on-chip interpolation). The four individual measurements enable a more detailed view of the four samples to be obtained. The interpolation point under normal operation is determined as the point where the LSB bit of the time measurement changes, as indicated in the table below.

Time measurements				Interpolation
tap0 = xxxx0	tap1 = xxxx0	tap2 = xxxx0	tap3 = TDC0	3
tap0 = xxxx0	tap1 = xxxx0	tap2 = xxxx0	tap3 = TDC1	0
tap0 = xxxx0	tap1 = xxxx0	tap2 = TDC1	tap3 = xxxxx	1
tap0 = xxxx0	tap1 = TDC1	tap2 = xxxxx	tap3 = xxxxx	2

Exchange of binary zero with binary one are also valid patterns.
TDC1= Time measurement read out with LSB bit marked (or TDC0)

Fig. 18 RC interpolation conversion

A few simple observations can be made from the given interpolation table. If measurements are seen, where more than one change of LSB is detected, then the total time span of the RC delay line is too long. On the other hand the observation of no LSB change in a single measurement can not be used to conclude that the time span of the RC delay line is too short, as this condition equals a valid interpolation point.

A description of the calibration procedure for the R-C delay line using a statistical code density test can be found in ref[4]. A code density test consists of histogramming the occurrence of each interpolation point over a set of random hits. For a sufficiently large population of random hits (few thousand) this histogram is a direct measure of the time span of each tap of the RC delay chain. A flat histogram is the ideal case where all time taps have equal size and covers uniformly the delay of one delay cell in the DLL.

The adjustable RC delay chain has in the HPTDC been implemented as a distributed RC delay line with fixed tap points. The adjustment of the taps is implemented as a bank of capacitors which can be connected to each tap point. This implementation is attractive from point of view of layout of the RC delay chain, but results in an implementation where the adjustment of a time tap also influences the others. When changing an adjust value it will influence the RC-delays in segments before this point but not the delays in following segments (see Fig. 19). This means that adjust1 affects bin2 only, Adjust2 affects bin2 and bin1 and adjust3 and adjust4 affects bin2, bin1 and bin0. Bin3 covers the time difference from the last RC delay tap (tap3) to tap0 of the following delay element in the DLL and any increase in bin0, bin1 or bin2 will result in a decrease in bin3.

Table 1: RC bin changes when changing adjust by 1.

	Bin0	Bin1	Bin2
Adjust1	~0	~0	0.56ps
Adjust2	~0	0.61ps	0.74ps
Adjust3	0.92ps	0.84ps	0.92ps
Adjust4	1.37ps	1.09ps	0.88ps

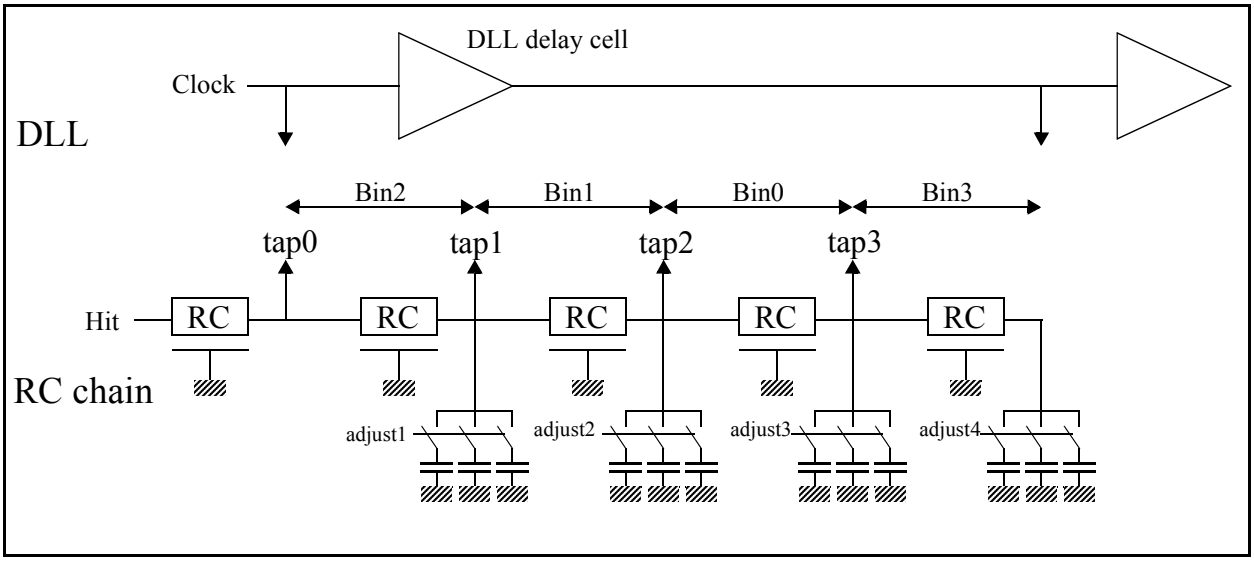


Fig. 19 RC delay chain with switched capacitor loads.

A first order calibration of the delay line is obtained by starting with all adjustments set to zero. Then the end of the RC-delay is adjusted, using the last two adjustable loads, as this has a strong influence on all time bins. If the adjustment for this is performed using either adjust 3 or adjust4 have no practical difference. It can be advised to make the first adjustments to the delay chain by incrementing these two as a pair with a common adjustment value. The common calibration constant for adjust 3 and adjust4 is chosen in a simple incremental fashion with typical code density histograms generated for each value as illustrated in Fig. 20. The optimal point to finish this first step is when Bin0 in the histogram has a size close to 1/4. From this point a second step working on the two remaining time bins can be performed.

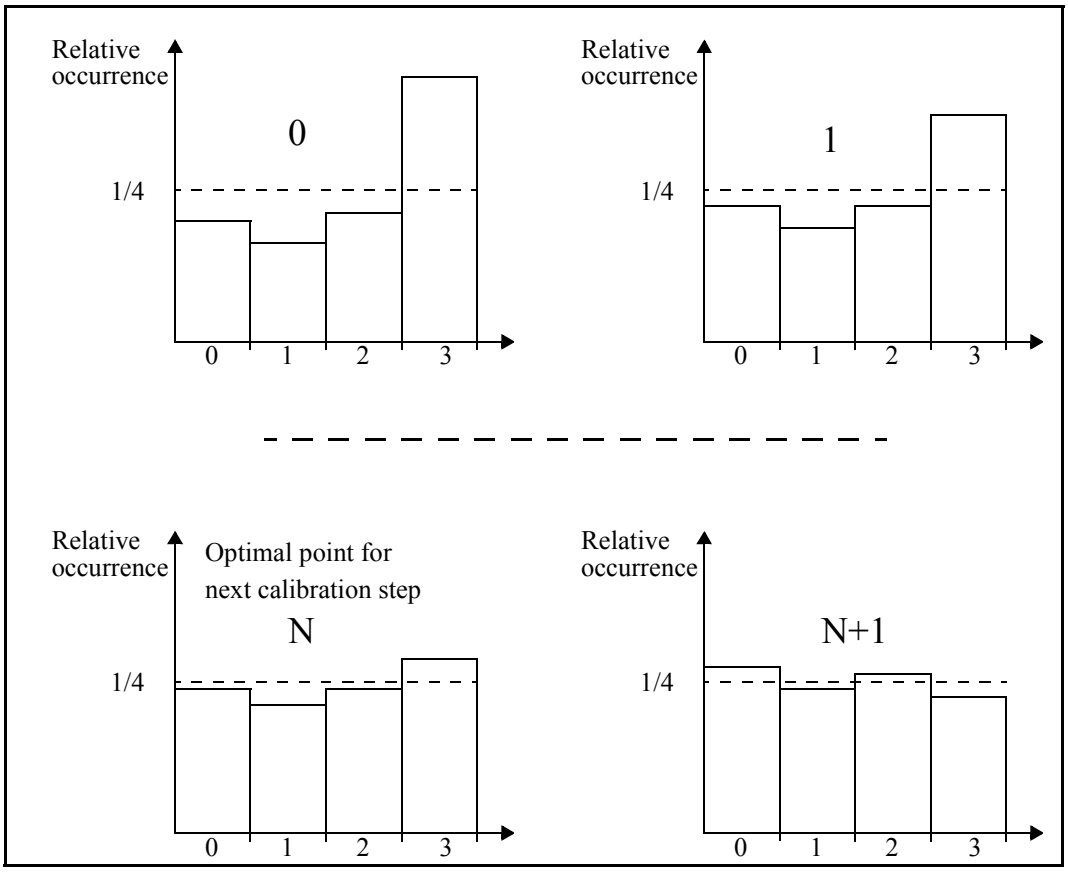


Fig. 20 Initial calibration of RC delay line

The final goal is to obtain a height equal (or as close as possible) to one quarter of the total histogram population in each bin as shown in Fig. 21. To increase bin1 adjust2 can be increased until it gets close to its optimal value of 1/4. Finally bin2 can be increased by increasing adjust1. To obtain the best possible adjustment an iterative procedure using information in table 1 can be used.

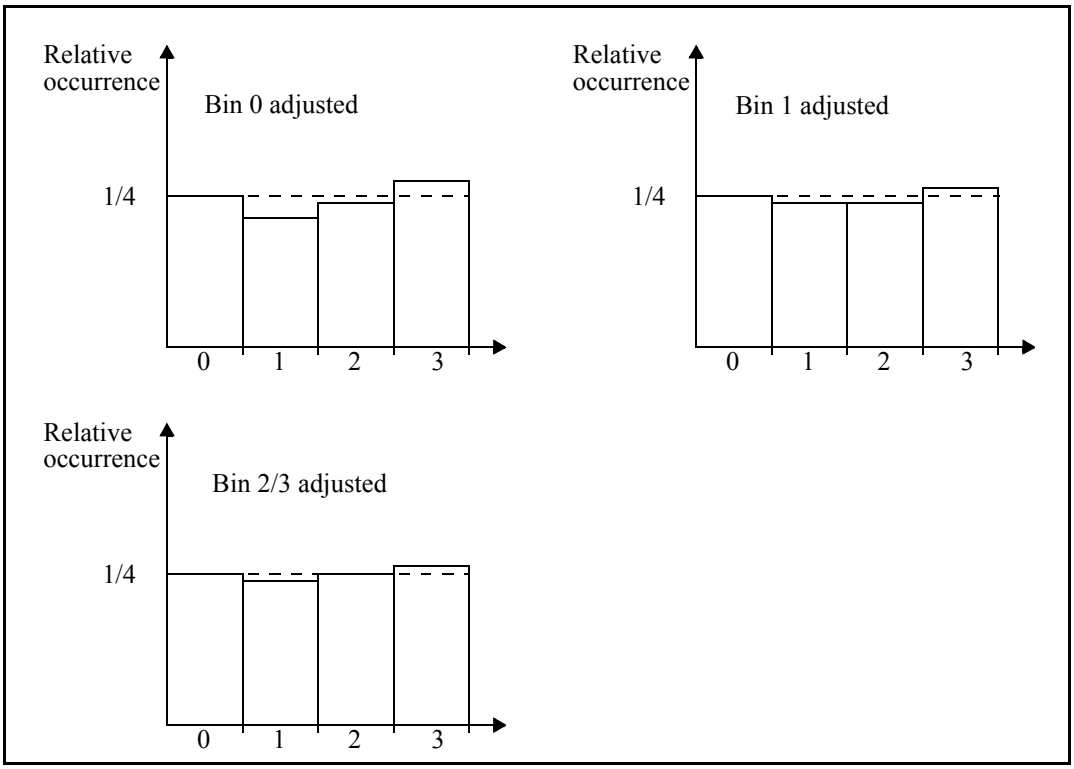


Fig. 21 Final calibration of RC delay line.

It has been found to be difficult to use the described calibration scheme in many cases because of an unfortunate emphasis of Bin0 and differences between channels on the same chip. Only one set of common RC-adjustments exist to calibrate the 8 RC-delay lines. To obtain the best possible performance of all channels on the same chip the RC-adjustment should be performed as an average over all 8 channels. A set of recommeneded adjustment values have been determined from a sample of chips and is proposed as default values (See: Setup registers on page 30)

To perform the described code density test it is required to feed the TDC with hits from a source which is uncorrelated to the reference clock (40MHz). As the code density test only is used to calibrate the RC delay line spanning a time window of 100ps, it is in principle sufficient to require that the source and the reference clock is uncorrelated seen over a time interval of 100ps. If the hits and the reference clock are completely uncorrelated, the RC adjustment is performed as an average, over all delay cells in the DLL. If some kind of coarse correlation exists, the calibration of the RC delay will be performed over a limited/weighted set of DLL delay cells. Such a weighting will though in most cases not be significant and is still a valid basis for the calibration.

A simple source of completely random hits is an independent oscillator. Real physics hits from detectors can in most cases also be used as the source of random hits because of the natural spread of interactions, and the small correlation interval required (this must though be carefully evaluated in each case). For detectors with significant drift times the time spectrum of arriving hits will naturally be quite flat and wide.

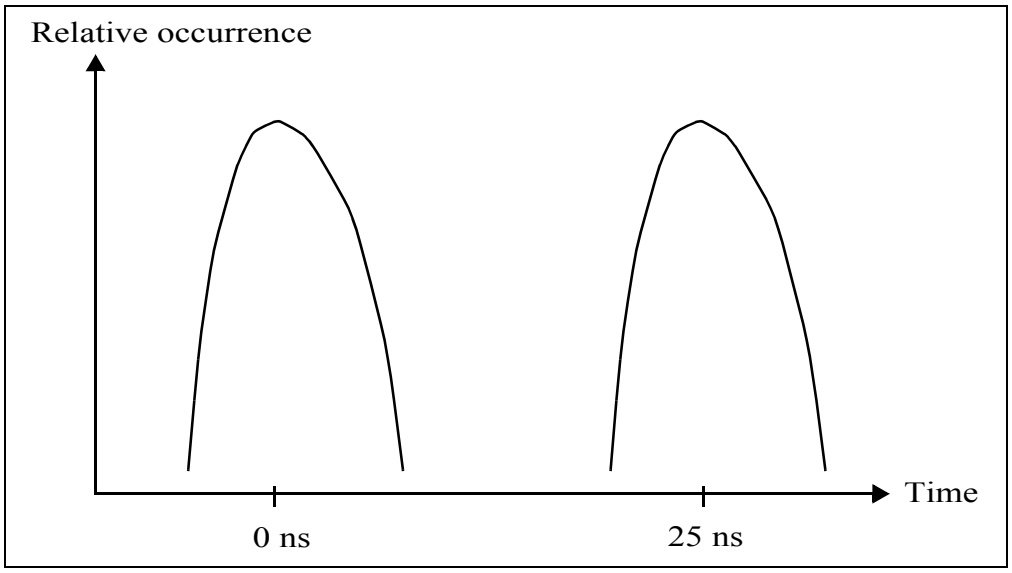


Fig. 22 Illustration of possible time correlation between hits and 40 MHz clock in a LHC experiment.

It is obviously also possible to calibrate the RC delay line against a time sweep from a high precision pulse generator. Pulse generators with sufficient stability and resolution is though hard to get or very expensive and can normally not be used for in-situ calibrations.

19. Adjustment of DLL taps

The time taps from the DLL have the option of being adjusted with high resolution via the programming (see Fig. 5). For low resolution modes this additional calibration of the DLL will not bring any significant improvement and can be left with all adjustment constants at zero. For high resolution modes (100 ps, 25 ps binning) the additional calibration of the DLL delay cells can in certain cases bring some improvements. For applications where very good differential or integral linearity is required each TDC chip must be independently calibrated and adjusted. An individual in-situ calibration can, as the calibration of the RC delay line, be performed with a simple code density test using hits from a "random" source. In this case, where the time interval covered by the delay line is much longer (3125 ps compared to 100ps), it must though be ensured that the hit and the reference are uncorrelated seen over a much larger time interval.

The adjustment of the DLL taps is performed on the outputs from the delay chain and not on the delay cells themselves. This means that if one DLL tap adjust is increased that one time bin will get longer and the next bin will get smaller. The mapping between the DLL delay elements and the time bins seen after encoding has been optimized to perform a correct selection between the two coarse time counters running on the two phases of the clock. An increase of DLL_adjust[0] will increase time bin27 and decrease bin28 when no channel offset correction is used. The time adjustment step is of the order of 10ps.

Used in very high resolution mode it has been seen that the largest imperfection in the differential non linearity is seen from the end of the DLL to the beginning of the DLL. This can best be compenstaed for using increasing adjust values down the delay chain: Dll_adjust 0 - 3: 0, Dll_adjust 4 - 7: 1, Dll_adjust 8 - 11: 2, Dll_adjust 12 - 15: 3, Dll_adjust 16 - 19: 4, Dll_adjust 20 - 23: 5, Dll_adjust 24 - 27: 6, Dll_adjust 28 - 31: 7.

20. Time alignment between hits, clock and trigger matching

The TDC contains many programmable setups which have effects on the performed time measurements and their optional trigger matching. The main time reference of the TDC is the clock

and the bunch reset which defines the T0 time. The time alignment can basically be divided into three different areas as shown in the figure below.

A: Time relationship between the hits, clock and the bunch count reset generating the basic timing measurements of the TDC.

B: Time relationship between the performed time measurements and the trigger time tag.

C: Time relationship between the time measurements and the automatic reject.

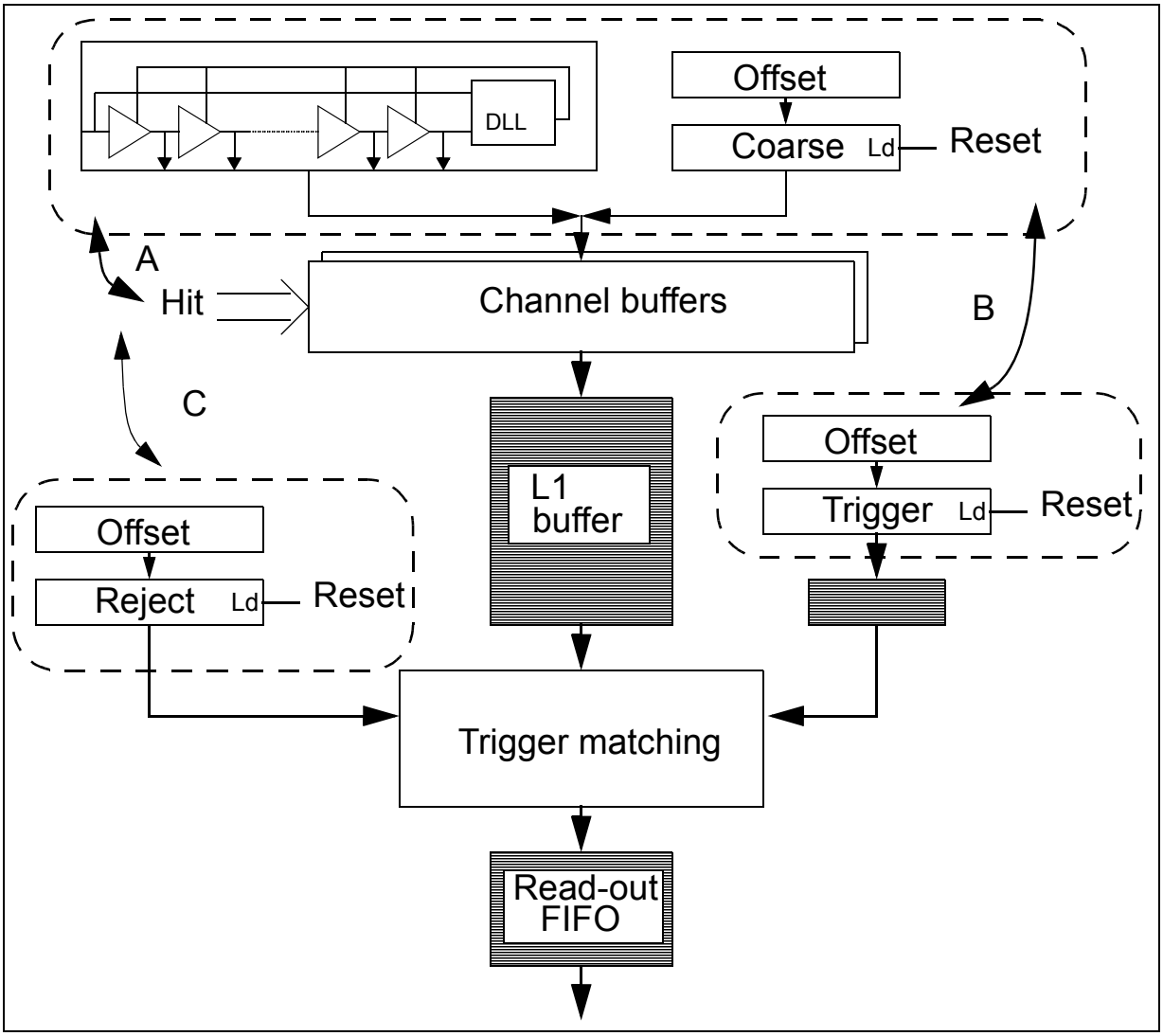


Fig. 23 Time alignment between A: hits, B: trigger and C: automatic reject

20.1. Basic time measurement.

As previously stated, the basic time reference of the TDC measurements is the rising edges of the clock. The bunch count reset defines the T0 reference time when the coarse time count is loaded with its offset value. The TDC also contains internal delay paths of the clock and its channel inputs which influence the actual time measurement obtained. These effects can be considered as a time shift (offset) in relation to the ideal measurement. The time shifts of individual channels may be slightly different but care has been taken to ensure that the channel offset variation is below 1ns. The time shift of the measurements also has some variation from chip to chip and variations with supply voltage and temperature. These variations have also been kept low by balancing the delay paths of the clock and the channels.

In the figure below a hit signal is defined such that the time measurement equals zero (coarse_count_offset = 0, channel adjust = 0). The delay from the rising edge of the clock where the bunch reset signal was asserted to the rising edge of the hit signal is for a typical case shown in the table below for the different TDC modes.

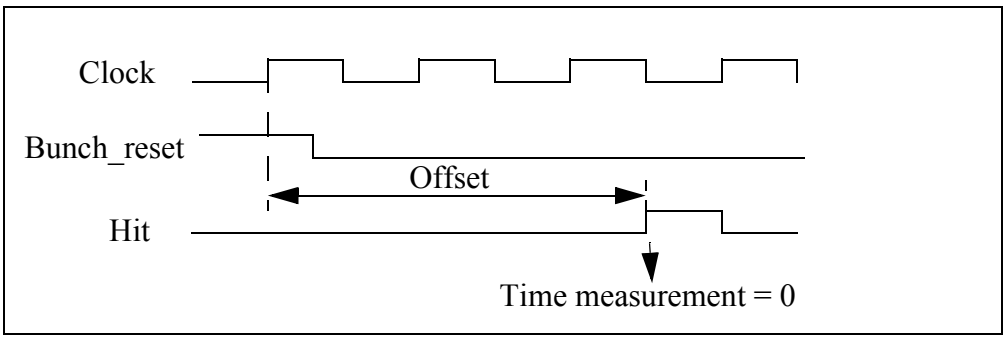


Fig. 24 Definition of reference time measurement.

Table 2: Typical time offsets as function of TDC mode with 40MHz reference clock

TDC mode	Offset
Low resolution	57.5ns
Medium resolution	23.5ns
High resolution	11.8ns
Very high resolution	11.1ns

The final time measurement being used in the trigger matching or being read out is affected by the programmed coarse time offset and the individual fine time channel adjustments.

20.2. Alignment between coarse time count and trigger time tag.

To perform an exact trigger matching the basic time measurement must be aligned with the positive trigger signal taking into account the actual latency of the trigger decision.The effective trigger latency in number of clock cycles equals the distance between the coarse count offset and the trigger count offset as illustrated in Fig. 25. When no counter roll-over is used (roll-over = FFF Hex) the relation ship is simply: latency = (Coarse_count_offset - Trigger_count_offset) modulus (2^{12}). A simple example is given to illustrate this: A Coarse_count_offset of 100 Hex (decimal 256) and a Trigger_count_offset of 000 Hex gives an effective trigger latency of 100 Hex (decimal 256). Normally it is preferable to have a coarse count offset of zero and in this case the trigger count offset must be chosen to (000 Hex - 100 Hex) modulus (2^{12}) = F00 Hex (decimal 3840). If the channel adjustment feature is used then this effect must also be taken into account.

The calculation of the trigger time tag counter offset is made more complicated when the counters are not “allowed” to overflow to zero at their natural binary value (FFF Hex). In many LHC experiments it is required to enforce the counters to be reset (using bunch reset) for each machine cycle to enforce them to represent the Bunch identification in the machine structure. In this case the trigger time tag offset must be chosen so the “effective” distance to the coarse time offset equals the required latency. This relationship can not be expressed in a straight forward equation and is best illustrated graphically as shown in Fig. 26.

As can be seen, the trigger latency is the effective distance between the trigger offset and the coarse offset. In case the excluded roll-over region is not between the two offset values then the roll-over does not need to be taken into account for the offset calculations. If it is located between the two, then the distance must be extended by the length of the excluded region.

In typical cases it is preferable to use a coarse count offset of zero and then the trigger offset can simply be calculated as: roll_over - latency + 1.

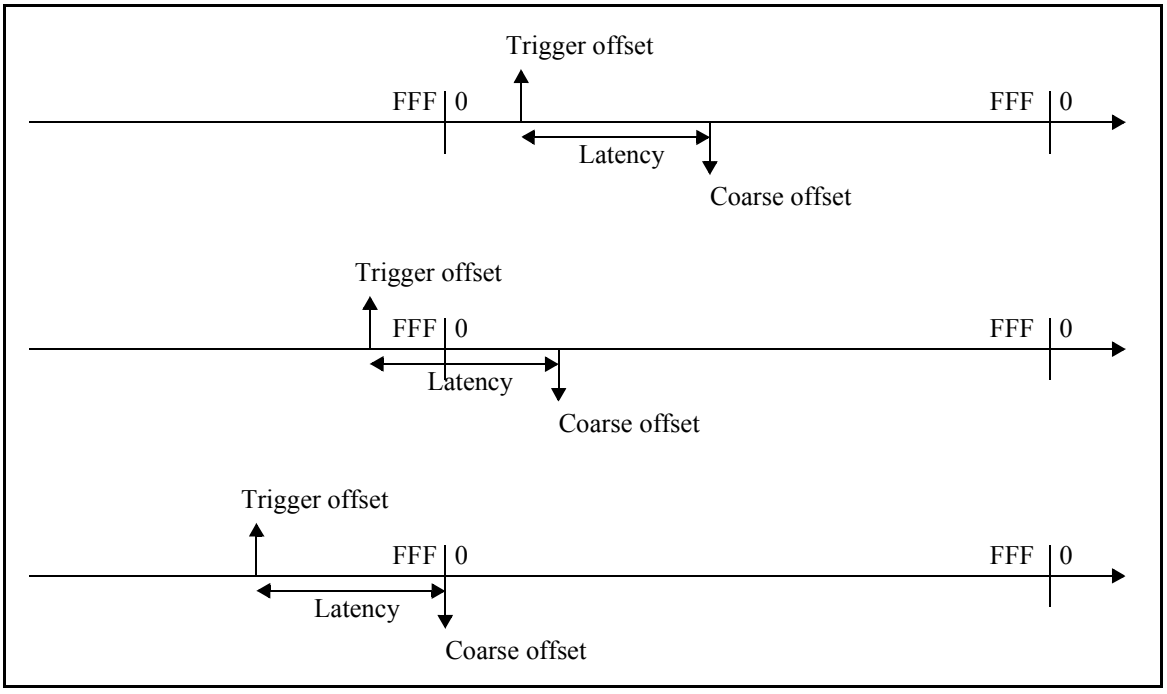


Fig. 25 Trigger latency with normal binary roll-over

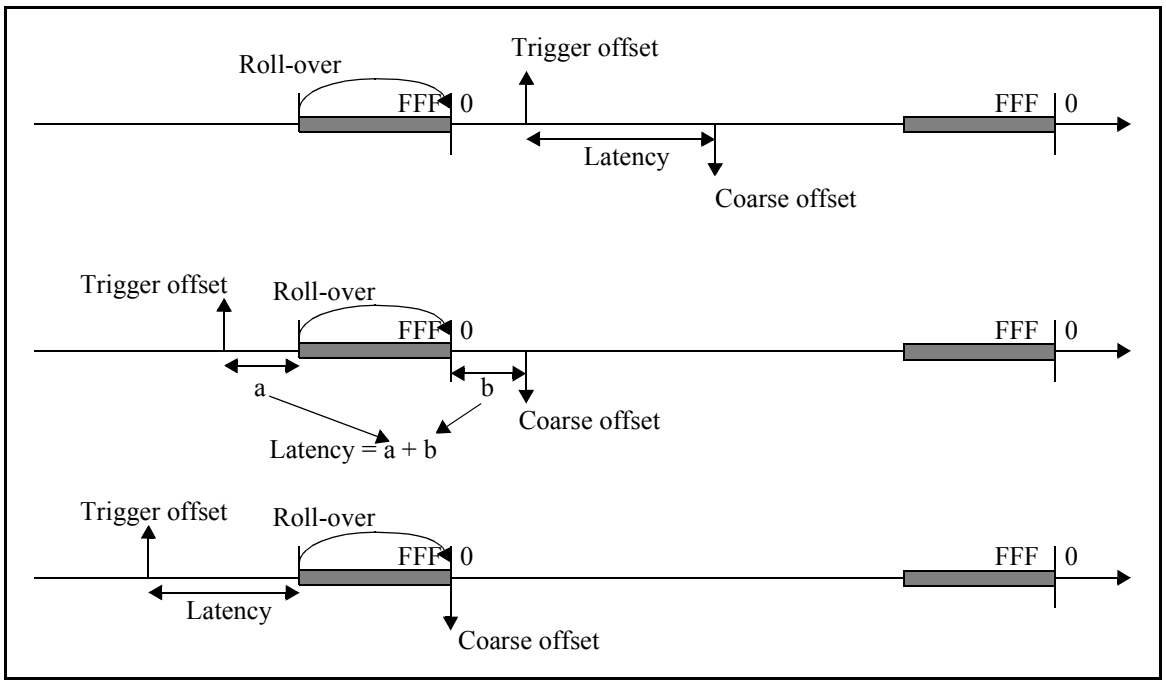


Fig. 26 Trigger latency with roll-over

20.3. Alignment between trigger time tag and reject count offset.

The workings of the reject counter is very similar to the trigger time tag counter. The difference between the coarse time counter offset and the reject counter offset is used to detect when a measurement has become older than a specified reject limit. The rejection limit should be set equal to the trigger latency plus a small safety margin (2). For a trigger latency of 256 (same as in example in the section above) a reject limit of 258 can be considered a good choice. This transforms into a reject count offset of 3837 (EFD Hex) when a coarse count offset of zero and a natural binary roll_over of FFF hex is used. When the roll over function is used, the excluded count region must be taken into account as explained for the trigger count offset calculation.

21. Signals.

Signal	Input/Output	Definition
Clk:	I	Master clock for TDC LVDS mode: clock TTL mode: clock
Clkb:	I	Master clock inverted. LVDS mode: clock inverted TTL mode: not used.
aux_clock:	I	Auxiliary clock input. LVDS mode: clock TTL mode: clock
aux_clockb:	I	Auxiliary clock inverted. LVDS mode: clock inverted TTL mode: not used.
reset:	I	master reset of state machines and buffers LVDS mode: reset TTL mode: reset
resetb:	I	master reset inverted LVDS mode: reset inverted TTL mode: not used
bunch_reset:	I	reset of coarse count, trigger time tag count, reject count. LVDS mode: bunch reset TTL mode: bunch reset
bunch_resetb:	I	bunch reset inverted. LVDS mode: bunch reset inverted TTL mode: not used
event_reset:	I	reset of event counter LVDS mode: event reset TTL mode: event reset
event_resetb:	I	event reset inverted. LVDS mode: event reset inverted TTL mode: not used
trigger:	I	load of trigger into trigger fifo LVDS mode: trigger TTL mode: trigger
triggerb:	I	trigger inverted. LVDS mode: trigger inverted TTL mode: not used
encoded_control:	I	encoded control (trigger, bunch reset, event reset, master reset) LVDS mode: encoded control TTL mode: encoded control
encoded_controlb:	I	encoded control inverted.

		LVDS mode: encoded control inverted TTL mode: not used
hit[31:0]	I	hit inputs LVDS mode: hit inputs TTL mode: hit inputs
hitb[31:0]	I	hit inputs inverted LVDS mode: hit inputs inverted TTL mode: not used
token_in	I	token input LVDS mode: token input TTL mode: token input
token_inb	I	token input inverted LVDS mode: token input inverted TTL mode: not used
token_bypass_in	I	token bypass input LVDS mode: token bypass input TTL mode: token bypass input
token_bypass_inb	I	token bypass input inverted LVDS mode: token bypass input inverted TTL mode: not used
serial_in	I	serial input LVDS mode: serial input TTL mode: serial input
serial_inb	I	serial input inverted LVDS mode: serial input inverted TTL mode: not used
serial_bypass_in	I	serial bypass input LVDS mode: serial bypass input TTL mode: serial bypass input
serial_bypass_inb	I	serial bypass input inverted LVDS mode: serial bypass input inverted TTL mode: not used
token_out	O	token output in LVDS
token_outb	O	token inverted output in LVDS
serial_out	O	serial output in LVDS
serial_outb	O	serial inverted output in LVDS
strobe_out	O	strobe output in LVDS
strobe_outb	O	strobe inverted output in LVDS
token_out_TTL	O	token output (LVTTL). 12 mA drive capability
serial_out_TTL	O	serial output (LVTTL).

		12 mA drive capability
strobe_out_TTL	O	strobe output (LVTTL). 12 mA drive capability
error	O	global error output (LVTTL). 8 mA drive capability
data_ready	O/Z	data ready for read-out (parallel read-out mode) (LVTTL). 12 mA drive capability
get_data	I	get parallel data (LVTTL)
parallel_data[31:0]	O/Z	parallel data out (LVTTL) 8 mA drive capability
tck	I	JTAG test clock (LVTTL).
trst	I	JTAG reset (active low) (LVTTL). NO PULL-UP resistor
tms	I	JTAG test mode select.(LVTTL)
tdi	I	JTAG test data in.(LVTTL)
tdo	O/Z	JTAG test data out (LVTTL). 8 mA drive capability
test	O	test signal output (LVTTL) 8 mA drive capability
gnd	I	Common ground
vdd_pll	I	VDD for PLL (2.5 v).
vdd_dll	I	VDD for DLL (2.5 v).
vdd_core	I	VDD for internal core (2.5v)
vdd_pre	I	VDD for IO pre-drivers (2.5)
vdd_TTL_out	I	VDD for TTL outputs (3.3v).
vdd_TTL_in	I	VDD for TTL inputs (3.3v)
vdd_lvds_out	I	VDD for LVDS outputs (2.5 v).
vdd_hit_in	I	LVDS receiver power supply for hit inputs (2.5v)
vdd_clock_in	I	LVDS receiver power supply for clock input (2.5v)
NOTE:	LVDS inputs do not have on-chip termination resistors.	

22. Signal timing.

TTL output delays are specified with a maximum load of 50 pf. For a loading exceeding 50 pf an additional delay of 0.04ns/pf and 0.06ns/pf must be added for 12mA and 8 mA output buffers respectively.

#	NAME	Description	Reference edge	Min.	Max	units
0	Tpclk	Clock period		25	50	ns
		Jitter	CLK+		0.1	ns
		Duty cycle		40/60	60/40	%
1	Tstok	Token_in setup	CLK+	5		ns
2	Thtok	Token_in hold	CLK+	2		ns
3	Tdtok	Token_out delay	CLK+	2	10	ns
4	Tdrdy	Data_ready delay	CLK+	2	10	ns
5	Tsget	Get_data setup	CLK+	5		ns
6	Thget	Get_data hold	CLK+	2		ns
7	Tdpdat	parallel_data delay	CLK+	2	10	ns
8	Tzpdat	parallel_data tristate delay	CLK+	2	10	ns

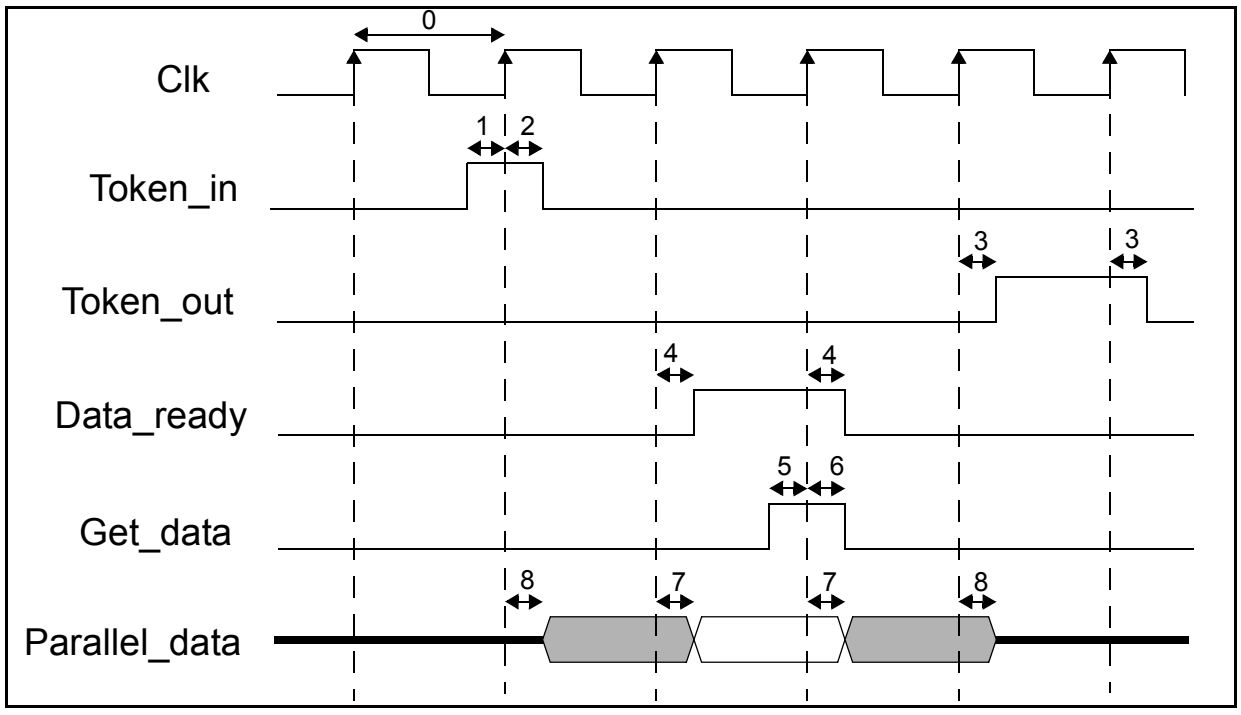


Fig. 27 Timing of parallel read-out signals

#	Name	Description	Reference edge	Min	Max	units
10	Tsres	Reset setup	CLK+	5		ns
11	Thres	Reset hold	CLK+	2		ns
12	Tsbres	Bunch_reset setup	CLK+	5		ns
13	Thbres	Bunch reset hold	CLK+	2		ns
14	Tseres	Event_reset setup	CLK+	5		ns
15	Theres	Event reset hold	CLK+	2		ns
16	Tstrg	Trigger setup	CLK+	5		ns
17	Thtrg	Trigger hold	CLK+	2		ns
18	Tsenc	Encoded_control setup	CLK+	5		ns
19	Thenc	Encoded_control hold	CLK+	2		ns
20	Tderr	Error delay	CLK+	2	10	ns

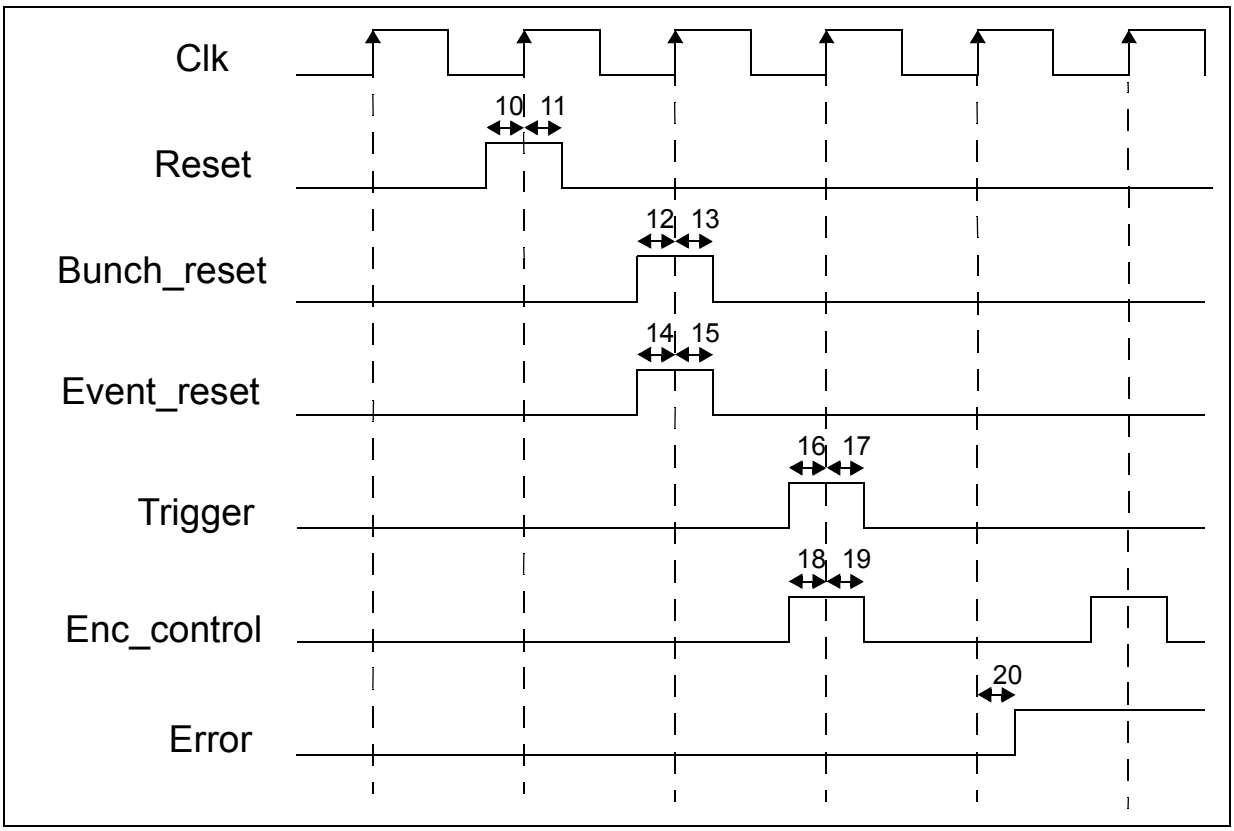


Fig. 28 Timing of resets, trigger and error.

#	Name	Description	Reference edge	Min	Max	units
30	Tsser	Serial_in setup (f <=40 MHz)[1]	CLK+	5		ns
31	Thser	Serial_in hold (f <=40 MHz)[1]	CLK+	2		ns
32	Tdser	Serial_out delay	CLK+	2	10	ns
33	Tdstr	Strobe_out delay	CLK+	2	10	ns
34	Tsstrser	Strobe_out, serial_out skew[2,3]	Strobe+-		+/-1	ns
35	Tjser	Serial_out jitter			+/-1	ns
36	Tsskew	Serial transfer skew[1] @40mbits/s			10	ns
		@80mbits/s			5	ns

Note1: When connecting the serial output of one TDC into the serial input of the next TDC it can be assumed that the internal delays in the two chips are nearly equivalent. The serial transfer between two TDC's is "guaranteed" to work if ("clock skew between the two TDC's" + "delay from serial_out to serial_in") is below Tsskew [36]. If this can not be guaranteed it is possible to artificially skew the serial_in via the programming (serial_delay[3:0]).

Note 2: The strobe signal is generated such that the strobe and the serial data changes at the same time within a skew of +/- 1ns when running at 80Mbits/s or 40Mbits/s. It is up to the receiver of the data to time the sampling of the serial-data using the available strobe. The specified skew is with identical loads on the two output pads.

Note 3: At serial readout speeds below 40MHz the strobe signal has by mistake been shifted by one clock cycle after the change of the serial data. This may though in some applications be advantageous as it may ease the correct sampling of the serial data on the receiver side.

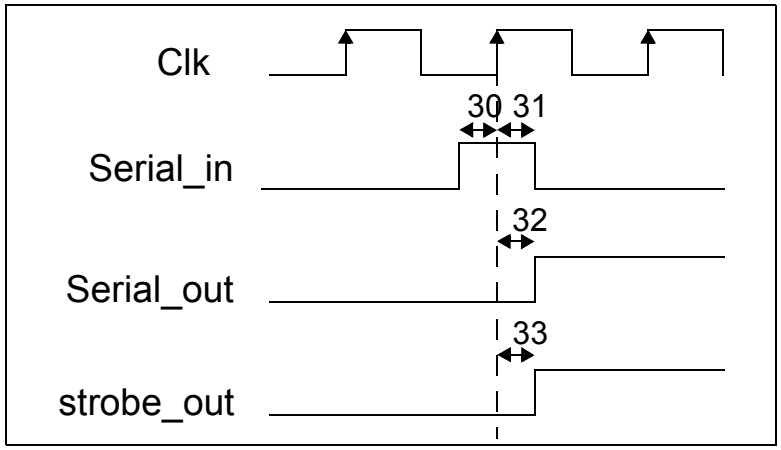


Fig. 29 Timing of serial data in relation to external TDC clock

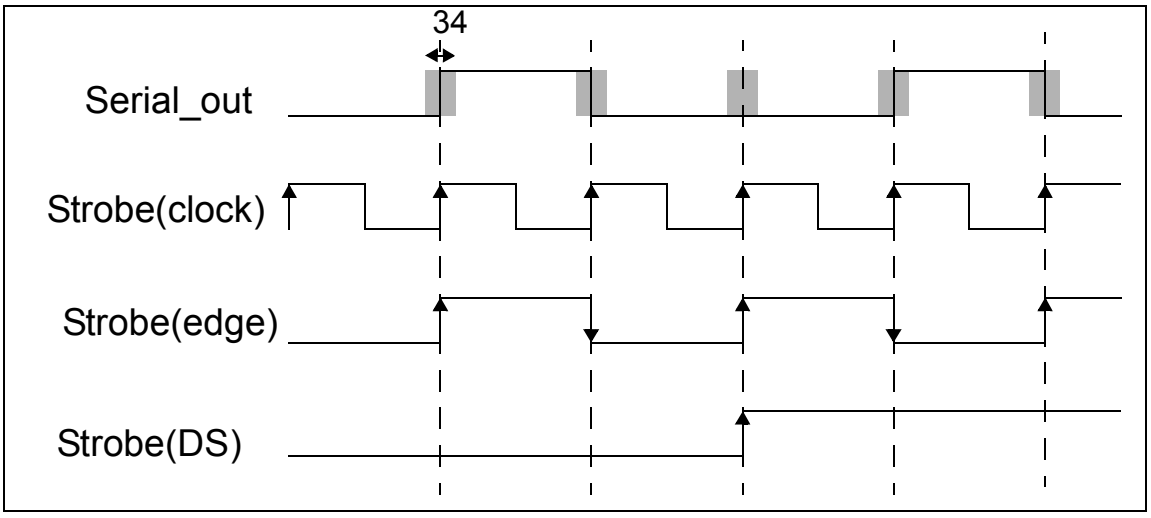


Fig. 30 Timing of strobe and serial data signals at 80 and 40MHz

#	Name	Description	Reference edge	Min	Max	units
40	Tptck	tck clock period		25		ns
41	Twtrst	trst width		10		ns
42	Trtrst	trst recovery	TCK+	5		ns
43	Tstms	tms setup	TCK+	5		ns
44	Thtms	tms hold	TCK+	2		ns
45	Tstdi	tdi setup	TCK+	5		ns
46	Thtdi	tdi hold	TCK+	2		ns
47	Tdtdo	tdo delay	TCK-	2	10	ns
48	Tztdo	tdo tristate delay	TCK-	2	10	ns

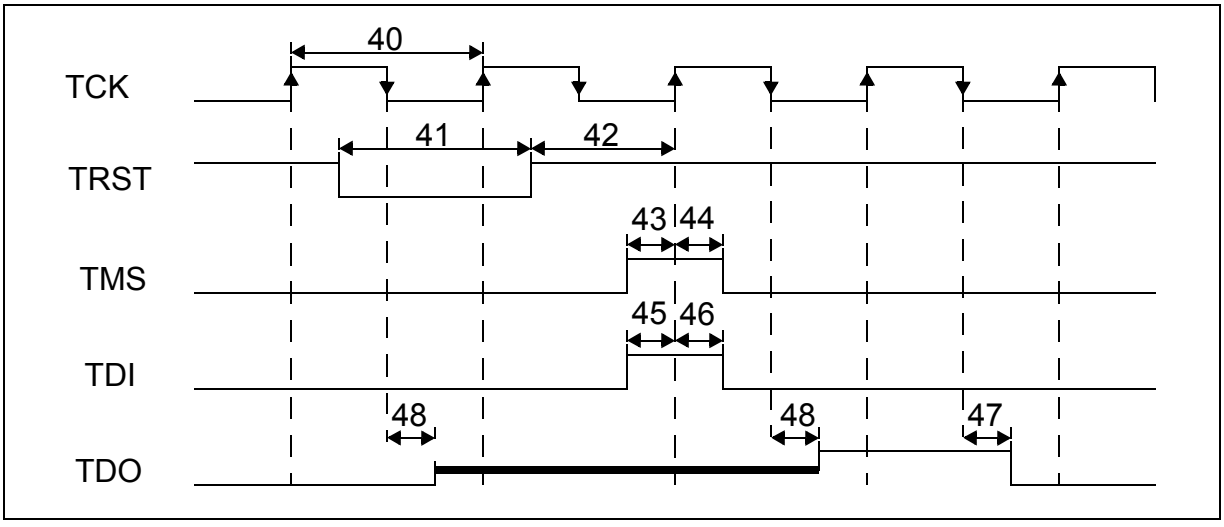


Fig. 31 Timing of JTAG signals.

23. DC characteristics

For all TTL IO signals Vdd refer to the 3.3v power supply.

For all LVDS IO signals Vdd refer to the 2.5 volt power supply

Table 3: TTL IO

Symbol	Parameter	Condition	Min	Typ.	Max	Unit
V_{ih}	Input high voltage		$0.7V_{dd}$			V
V_{il}	Input low voltage				$0.3V_{dd}$	V
I_{ih}	Input high current	$V_{in}=V_{dd}$	-10		10	uA
I_{il}	Input low current	$V_{in}=V_{ss}$	-10		10	uA
V_{OH}	Output high voltage					
	8 mA type	I_{OH}=-8mA	2.4			V
	12 mA type	I_{OH}=-12mA	2.4			V
V_{OL}	Output low voltage					
	8 mA type	I_{OL}=8mA			0.4	V
	12 mA type	I_{OL}=12mA			0.4	V
I_{OZ}	3-state leakage		-10		10	uA

Table 4: LVDS IO

Symbol	Parameter	Condition	Min	Typ.	Max	Unit
V_{idiff}	Input differential voltage		25			mV
V_{icom}	Input common-mode voltage		0.1		$0.9V_{dd}$	V
Vi	Input single-ended voltage		0		Vdd	V
I_i	Input current		-10		10	uA
Iodiff	Output differential current		3.0	3.5	4.0	mA
Vocom	Output common mode voltage		1.1	1.2	1.3	V

24. Power consumption

The typical power consumption of the HPTDC in the different operating modes are shown below. For special applications where the logic clock must run at 160MHz (special speed graded circuits) an additional power consumption of 500mW must be added to numbers for the 80MHz logic clock operation.

Table 5: Power consumption of HPTDC

Resolution mode	Low power mode	Logic clock	Hit inputs	Power
Low resolution	Yes	40 MHz	TTL	450 mW
Low resolution	Yes	40 MHz	LVDS	650mW
Low resolution	Yes	80MHz	TTL	700 mW
Low resolution	Yes	80 MHz	LVDS	900 mW
Medium resolution	Yes	40 MHz	TTL	550 mW
Medium resolution	Yes	40 MHz	LVDS	750 mW
Medium resolution	Yes	80 MHz	TTL	800 mW
Medium resolution	Yes	80 MHz	LVDS	1000 mW
High resolution	Yes	40 MHz	TTL	800 mW
High resolution	Yes	40 MHz	LVDS	1000 mW
High resolution	Yes	80 MHz	TTL	1100 mW
High resolution	Yes	80 MHz	LVDS	1300 mW
High resolution	No	40 MHz	TTL	1100 mW
High resolution	No	40 MHz	LVDS	1300 mW
High resolution	No	80 MHz	TTL	1300 mW
High resolution	No	80 MHz	LVDS	1500 mW

A temperature difference of 30 ^{o}C from ambient temperature (Max 70 ^{o}C) to the temperature of the silicon die (Max 100 ^{o}C) has been taken into account in the definition of the operating conditions of the HPTDC. The Thermal resistance of the Plastic BGA is specified to be 26 ^{o}C/W when 13 thermal via's are connected to a ground plane in the printed circuit board. For the high resolution mode where the power consumption is relatively high the ambient temperature must be limited to 60 ^{o}C or the thermal conduction to the PCB must be improved by using the thermal conduction from remaining ground/power and signal pins. If used with a 160MHz logic clock special care must be taken to ensure sufficient cooling.

It is recommended to only use the HPTDC in the low power mode (Setup[570]) as no significant advantages have been seen when used in the high power mode.

25. Operating conditions

Table 6: Operating condition of HPTDC

Symbol	Parameter	Rating	Unit
V_{DD} - Core	Core supply voltage	2.3 - 2.7 Recommended = 2.5	V
VDD-pre	IO pre-driver supply	2.3 - 2.7 Recommended = 2.5	V
V_{DD} - IO-TTL	TTL IO supply voltage	3.0 - 3.6 Recommended = 3.3v	V
VIN	Input DC voltage	-0.3 - Vdd-IO-TTL - 0.3	V
T_A	Ambient temperature Low resolution modes	-40 - 70	^oC
T_A	Ambient temperature High resolution mode	-40 - 60	^oC

The HPTDC has in some cases been seen to be sensitive to the power on sequence between the 3.3v IO power supply and the 2.5v core power supply. It is recommended to apply the 3.3v IO power to the chip before the 2.5v power. If the 2.5v power is supplied first it has in some cases been seen that the IC starts up with a relative large power consumption until fully initialized.

26. Packaging

The HPTDC is packaged in a 225 pin plastic BGA package conforming to the JEDEC MO-151 outline.

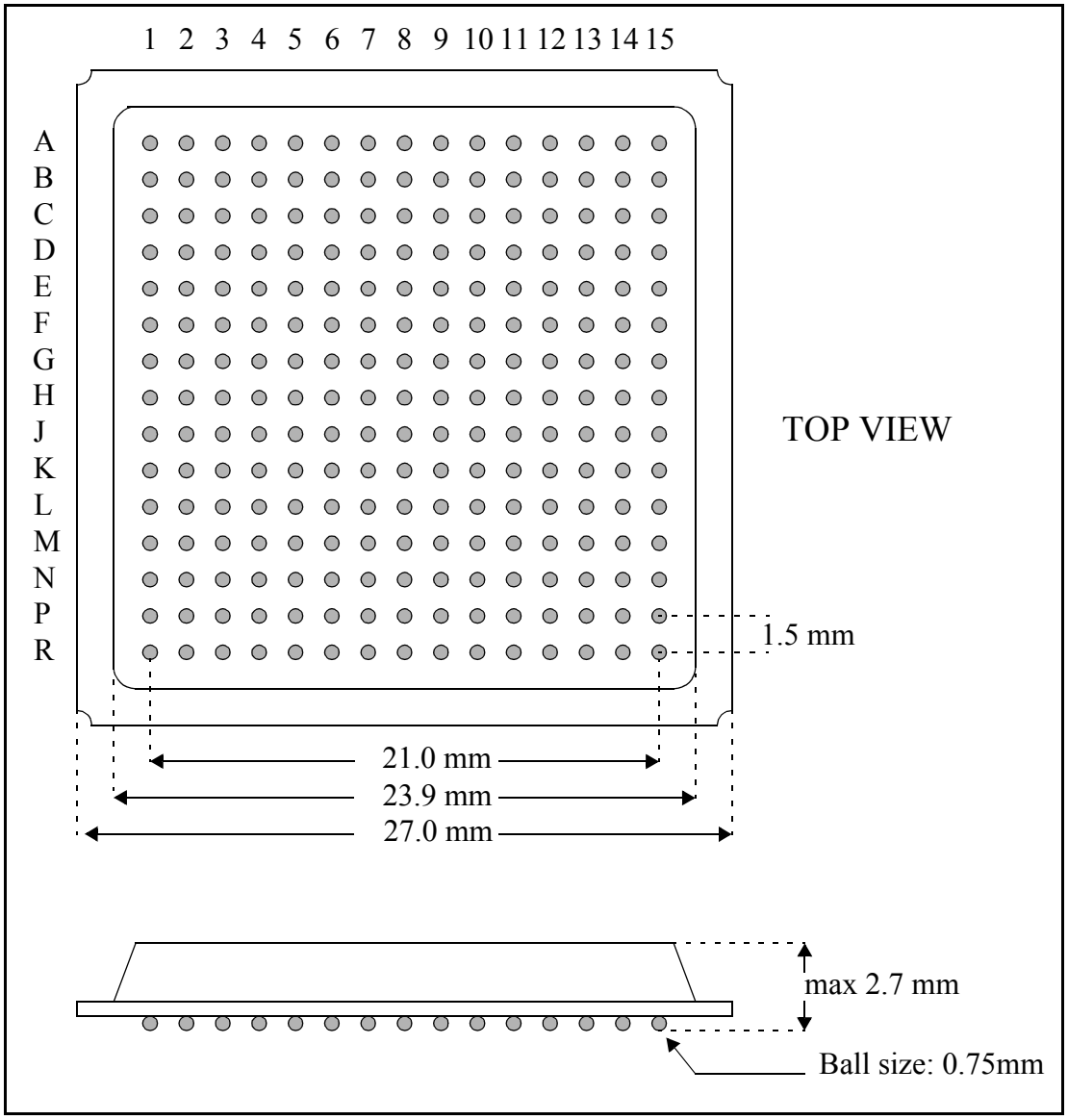


Fig. 32 Mechanical dimensions of plastic BGA 225 package.

The cooling of this type of package is mainly ensured by the thermal conduction from the package through the solder balls to the printed circuit board. It is therefore important to ensure a good thermal conduction from the solder pads on the PCB.

26.1. Pin mapping

A1	gnd	A2	gnd	A3	Vdd_TTL_out
A4	Parallel_data[13]	A5	Parallel_data[9]	A6	Parallel_data[7]
A7	Parallel_data[3]	A8	Vdd_TTL_out	A9	gnd
A10	gnd	A11	serial_out_TTL	A12	Vdd_LVDS_out
A13	serial_outb	A14	serial_inb	A15	gnd
B1	Parallel_data[20]	B2	Vdd_core	B3	Parallel_data[17]
B4	Parallel_data[15]	B5	Parallel_data[10]	B6	gnd
B7	Parallel_data[2]	B8	gnd	B9	Vdd_core
B10	get_data	B11	Vdd_LVDS_out	B12	strobe_out
B13	token_out	B14	Vdd_core	B15	NC (Vdd PLL)
C1	gnd	C2	Parallel_data[21]	C3	Vdd TTL out
C4	Parallel_data[16]	C5	Parallel_data[12]	C6	Vdd TLL out
C7	Parallel_data[4]	C8	Vdd_core	C9	Vdd_core
C10	data_ready	C11	gnd	C12	serial_out
C13	NC (Vdd_PLL)	C14	NC (Vdd_PLL)	C15	token_bypass_in
D1	Parallel_data[26]	D2	Vdd_TTL_out	D3	Parallel_data[23]
D4	Parallel_data[19]	D5	gnd	D6	Parallel_data[11]
D7	Parallel_data[5]	D8	Parallel_data[0]	D9	gnd
D10	strobe_out_TTL	D11	gnd	D12	serial_in
D13	token_bypass_inb	D14	token_in	D15	serial_bypass_in
E1	Parallel_data[30]	E2	Parallel_data[28]	E3	Parallel_data[27]
E4	Parallel_data[25]	E5	Parallel_data[22]	E6	Parallel_data[14]
E7	Parallel_data[6]	E8	Parallel_data[1]	E9	Vdd_core
E10	token_out_TTL	E11	token_outb	E12	token_inb
E13	NC (Vdd_clock_in)	E14	clk_b	E15	clk
F1	Vdd TTL out	F2	gnd	F3	gnd
F4	Parallel_data[31]	F5	Parallel_data[29]	F6	Parallel_data[18]
F7	Parallel_data[8]	F8	gnd (Therm)	F9	strobe_outb
F10	NC (Vdd_PLL)	F11	serial_bypass_inb	F12	gnd
F13	aux_clock	F14	NC (Vdd_pre)	F15	gnd
G1	jtag_tdi	G2	jtag_tms	G3	jtag_tck
G4	jtag_trst	G5	jtag_tdo	G6	Parallel_data[24]
G7	gnd (Therm)	G8	gnd (Therm)	G9	gnd (Therm)
G10	aux_clockb	G11	encoded_controlb	G12	encoded_control
G13	triggerb	G14	event_resetb	G15	trigger
H1	Vdd_TLL_in	H2	gnd	H3	error
H4	test	H5	gnd	H6	gnd (Therm)
H7	gnd (Therm)	H8	gnd (Therm)	H9	gnd (Therm)
H10	gnd (Therm)	H11	event_reset	H12	bunch_resetb
H13	reset	H14	reset_b	H15	bunch_reset
J1	Vdd_core	J2	Vdd_core	J3	gnd
J4	Vdd_core	J5	Vdd_core	J6	NC (Vdd_pre)
J7	gnd (Therm)	J8	gnd (Therm)	J9	gnd (Therm)
J10	hitb[28]	J11	Vdd_core	J12	Vdd_core
J13	Vdd_core	J14	gnd	J15	Vdd_core
K1	NC (Vdd_DLL)	K2	NC (Vdd_DLL)	K3	gnd
K4	hit[1]	K5	hitb[2]	K6	hitb[5]
K7	hitb[9]	K8	gnd (Therm)	K9	gnd
K10	hitb[24]	K11	hit[31]	K12	gnd
K13	NC (Vdd_pre)	K14	gnd	K15	gnd
L1	hit[0]	L2	hitb[0]	L3	hitb[1]
L4	hitb[3]	L5	hitb[7]	L6	gnd
L7	hit[16]	L8	NC (Vdd_hit_in)	L9	hit[20]
L10	Vdd_core	L11	hitb[26]	L12	hit[29]
L13	hit[30]	L14	hitb[30]	L15	hitb[31]
M1	hit[2]	M2	hit[3]	M3	hitb[4]

M4	gnd	M5	hit[10]	M6	hit[12]
M7	hit[14]	M8	gnd	M9	hitb[19]
M10	Vdd_core	M11	hitb[22]	M12	hit[25]
M13	hit[27]	M14	hit[28]	M15	hitb[29]
N1	hit[4]	N2	hit[5]	N3	hitb[6]
N4	hit[8]	N5	hit[11]	N6	hitb[12]
N7	hitb[14]	N8	hitb[16]	N9	hit[19]
N10	hitb[21]	N11	gnd	N12	hitb[23]
N13	NC (Vdd_hit_in)	N14	hit[26]	N15	hitb[27]
P1	hit[6]	P2	Vdd_core	P3	hit[7]
P4	hit[9]	P5	hitb[11]	P6	hit[13]
P7	hit[15]	P8	hit[17]	P9	hit[18]
P10	hit[21]	P11	Vdd_core	P12	hit[22]
P13	hit[24]	P14	Vdd_core	P15	hitb[25]
R1	gnd	R2	NC (Vdd_hit_in)	R3	hitb[8]
R4	hitb[10]	R5	NC (Vdd_hit_in)	R6	hitb[13]
R7	hitb[15]	R8	hitb[17]	R9	hitb[18]
R10	hitb[20]	R11	gnd	R12	Vdd_core
R13	hit[23]	R14	gnd	R15	gnd

NC = Not Connected. Some of these pins were in the first pin definition of the HPTDC by mistake defined to be connected to the 3.3v supply. This mistake was not discovered for the HPTDC version 1.0 as this version could only work with a single 2.5 volt power supply. For the HPTDC version 1.1 these pins were left unconnected to make the new chips usable in existing test setups. The vdd_hit_in, vdd_clock_in and vdd_pre power supplies were therefore connected to the vdd-core power ring in the package. For HPTDC version 1.2 & 1.3 vdd_dll and vdd_pll power was also taken from internal power-gnd plane in the custom made package and the dedicated package pins left unconnected. For new applications using the HPTDC the NC pins should be connected to a 2.5v power supply as future versions of the HPTDC package may need to use these separate 2.5v power supply pins to reduce power supply noise to the PLL, DLL, clock receivers and hit receivers.

Note: Pins marked with Therm are thermal conduction pins which must be connected to a heat conduction ground plane for proper cooling.

27. Technical Specifications.

Number of channels:	32 / 8		
Clock frequency:	40 MHz external 40MHz / 160 MHz / 320 MHz internal		
Time bin size:	781 ps 195 ps 98ps 24 ps	low resolution mode medium resolution mode high resolution mode very high resolution mode (8 channels)	
Differential non linearity:	Typical values +/- 0.2 bin +/- 0.3 bin + 0.60, -0.25 bin +0.35, -0.25 bin +1.3, -0.7 bin	 0.08 bin RMS 0.09 bin RMS 0.10 bin RMS 0.08 bin RMS 0.21 bin RMS	 low resolution mode medium resolution mode high resolution mode high resolution mode (DLL corr.) very high resolution mode
Integral non linearity:	Typical values +/- 0.25 bin +/- 0.50 bin +0.6,-1.4 bin +3.5,-5.0 bin	 0.11 bin RMS 0.24 bin RMS 0.50 bin RMS 2.1 bin RMS	 low resolution mode medium resolution mode high resolution mode very high resolution mode
Time resolution:	Typical values 0.34 bin RMS 0.44 bin RMS 0.65 bin RMS 2.4 bin RMS 0.72 bin RMS	 (265 ps) (86ps) (64ps) (58ps) (17 ps)	 low resolution mode medium resolution mode high resolution mode very high resolution mode very high resolution mode (table corr)
	Time resolution in HPTDC is limited by INL from internal crosstalk caused by logic core running at 40MHz. (See: Design bugs and problems on page 99) This can be corrected for using a simple table correction in the following data processing (see: Time resolution measurements on page 68)		
Difference between channels:	Maximum +/- 1 ns offset		
Variation with temperature:	Maximum 100ps change with 10 Deg. change of IC temperature.		
Cross talk:	Maximum 150 ps from concurrent 31 channels to one channel.		
Dynamic range:	12 + 5 = 17 bit 12 + 7 = 19 bit 12 + 8 = 20 bit (19) 12 + 8 + 2 = 22 bit (21) Numbers in parenthesis are number of bits read out		low resolution mode medium resolution mode high resolution mode very high resolution mode.
Double pulse resolution:	Typical 5 ns. Guaranteed 10ns		
Max. recommended Hit rate:	Core logic at 40 MHz 2 MHz per channel, all 32 channels used 4 MHz per channel, 16 channels used. 2 MHz per channel in very high resolution mode		

	Core logic at 80 MHz: 4 MHz per channel, all 32 channels used 8 MHz per channel, 16 channels used. 4 MHz per channel in very high resolution mode Core logic at 160MHz (only for special speed rated pieces): 8 MHz per channel, all 32 channels used 16 MHz per channel, 16 channels used. 8 MHz per channel in very high resolution mode
Event buffer size:	4 x 256
Read-out buffer size:	256
Trigger buffer size:	16
Power supply:	Core: 2.3 - 2.7 volt TTL IO: 3.0 - 3.6 volt
Power consumption:	450mW - 2000 mW depending on operation mode.
Temperature range:	-40 - 70 Deg. Cent.
Hit inputs:	LVDS or LV TTL
Clock input:	LVDS or LV TTL
Trigger and reset inputs:	LVDS or LV TTL
Serial readout:	LVDS or LV TTL
Parallel readout:	LV TTL

28. Time resolution measurements

The time resolution of the HPTDC have been measured with different schemes to characterize the effective performance.

Differential and Integral non-linearities have been measured with code density tests using a random source of hits. Code density tests are very efficient to measure DNL and INL but must not be taken as a measure of the effective time resolution of a TDC device as all error components of random nature (e.g. jitter) are excluded from the results. Code density tests are also very useful to optimize the RC-delay chain and DLL tap adjustments.Non-linearities are only shown covering a time period of 25ns as the TDC architecture, using a simple counter to extend the dynamic range, ensures that the patterns shown are repeated identically for each 25ns period

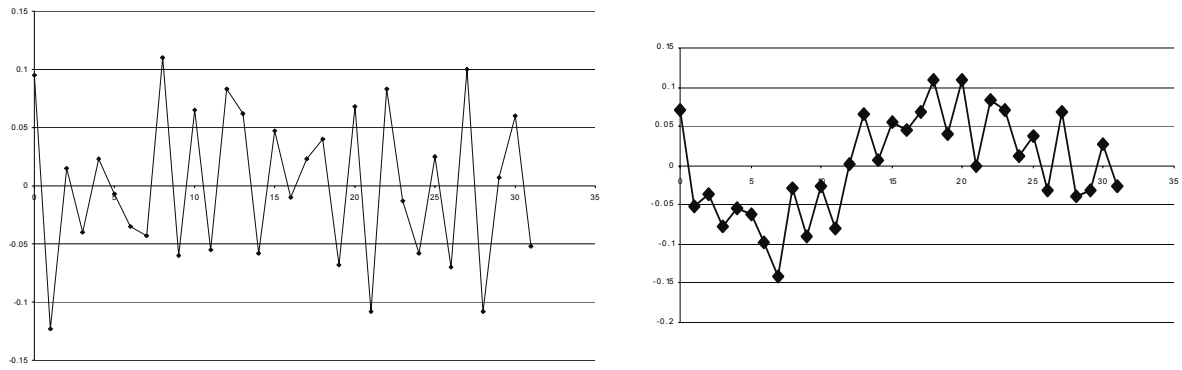


Fig. 33 DNL and INL in low resolution mode

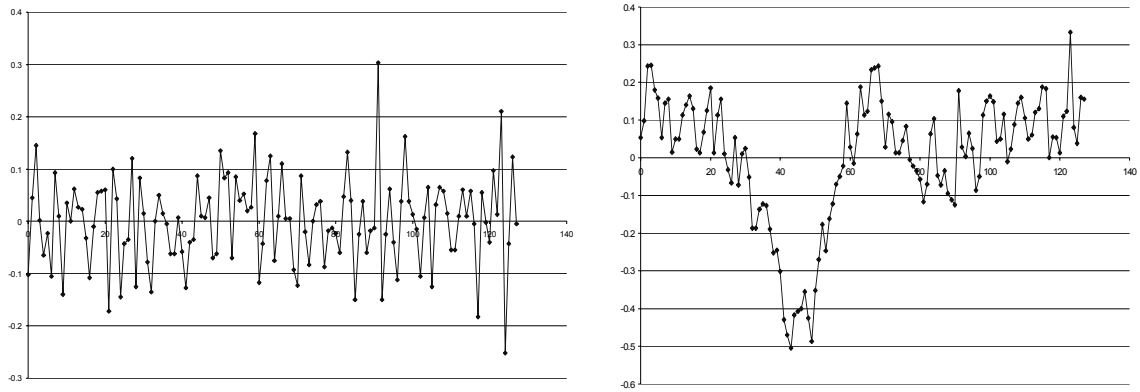


Fig. 34 DNL and INL in medium resolution mode

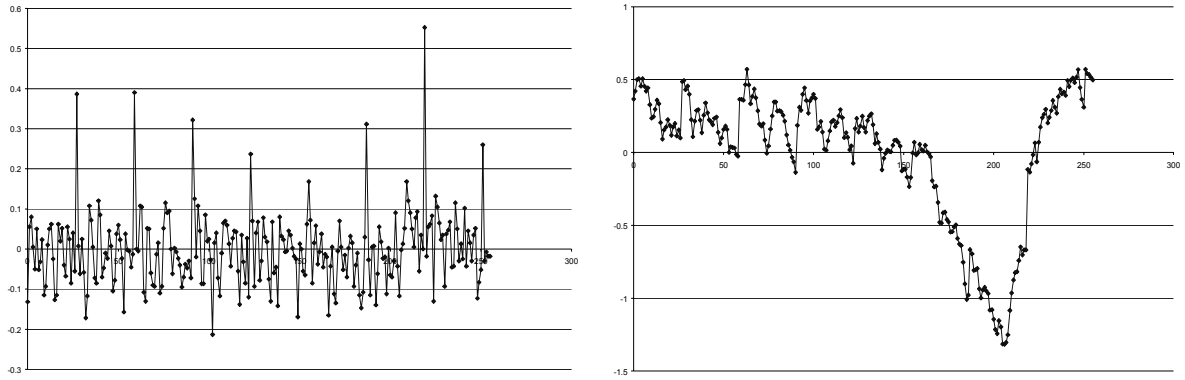


Fig. 35 DNL and INL in high resolution mode without DLL tap adjustments

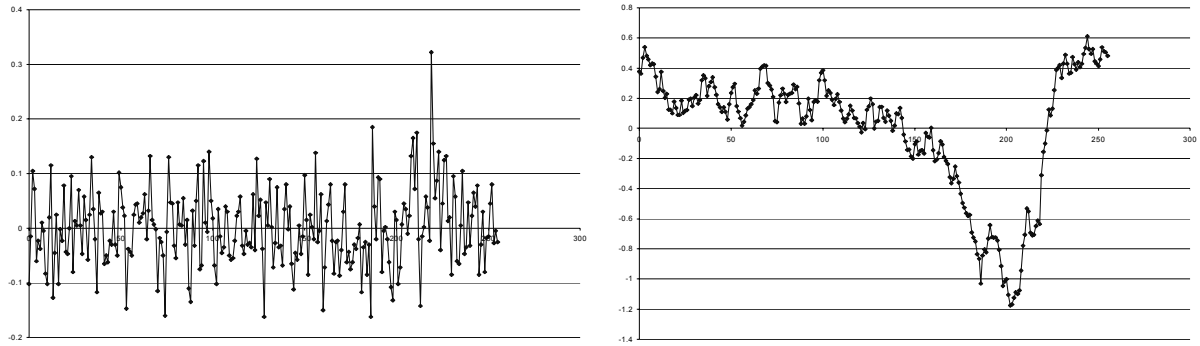


Fig. 36 DNL and INL in high resolution mode with DLL tap correction

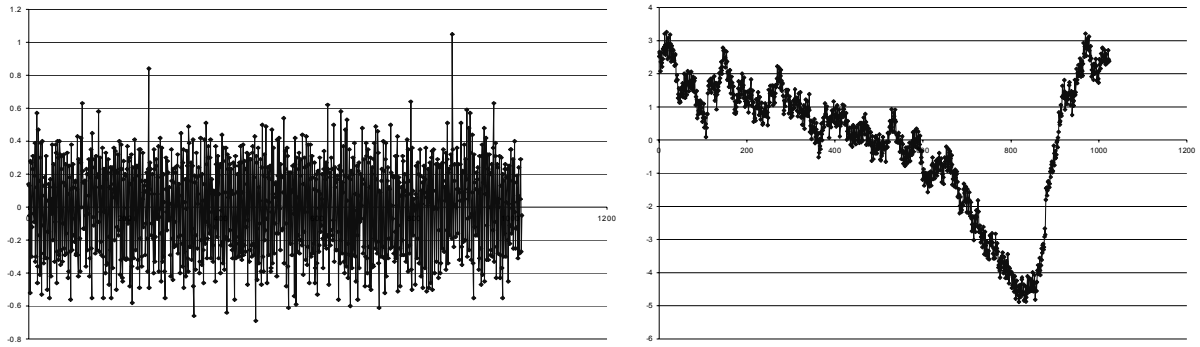


Fig. 37 DNL and INL in very high resolution mode with DLL tap adjust

A fixed pattern in the INL is clearly seen in the high resolution and the very high resolution modes. This is caused by 40 MHz cross talk from the logic part of the chip to the time measurement part and is believed to come from power supply and substrate coupling (see: Design bugs and problems on page 99). As this crosstalk comes from the 40MHz clock, which is also the time reference of the TDC, the integral non linearity have a stable shape between chips and can therefore be compensated for by a simple look up table using the LSB bits of the measurements as an entry (8 bits in high resolution mode, 10 bits in very high resolution mode). The DNL have a clear peak in bin27 plus multiples of 32. Bin27 is the DLL delay tap covering from the end of the delay chain to the beginning of the delay chain and includes possible phase detector errors (see: Design bugs and problems on page 99).

The INL have also been characterized using a 40 and 80 MHz core clock and it can be clearly seen on Fig. 38 that the INL is directly related to the core clock.

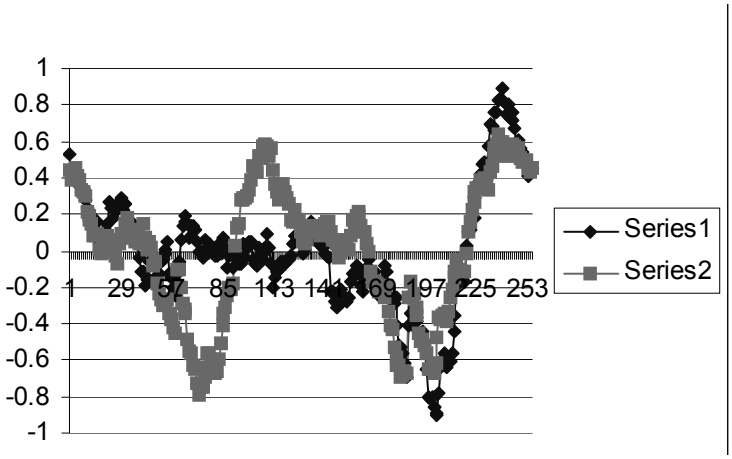


Fig. 38 INL with 40MHz and 80MHz core logic clock in high resolution mode.

The effective time resolution of the HPTDC has been measured using time sweeps generated by an IC tester. The limited time resolution (100ps) and linearity of the IC tester have though prevented these measurements to be used as a real timing characterization of the HPTDC except for the low resolution mode.

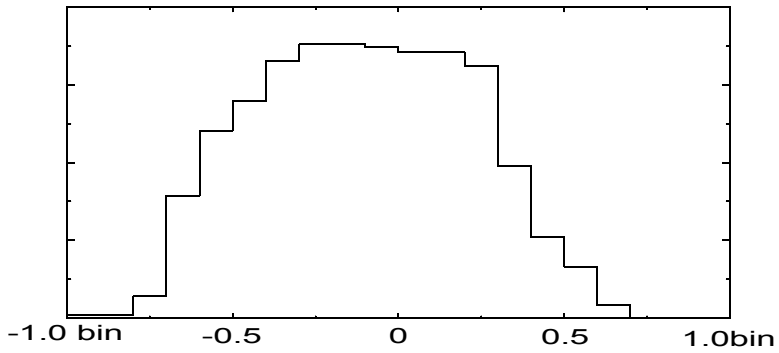


Fig. 39 Error histogram of Low resolution mode using time sweep of IC tester

An alternative scheme using simple cable delays have been used to measure the effective time resolution of the HPTDC excluding possible imperfections from instruments used. A random source of hits is connected to two TDC channels with a fixed cable delay between the two channels. Measurements have been made with different cable delays to ensure that the fixed time difference between the two channels does not give artifacts in the measurements (e.g. a cable delay of 25ns would give artificially good results). The variation in the observed time difference is a measure of the RMS resolution of the TDC between two individual time measurements. Assuming that the two time measurements are uncorrelated (ensured by using random hits and doing measurements with multiple cable delays) the effective time resolution of one TDC measurement can be determined by dividing the measured RMS variation of the time difference by $2^{1/2}$. For the high resolution and the very high resolution modes the effective time resolution can be significantly improved using a simple table correction of each individual time measurement for the Integral nonlinearity shown in Fig. 36 and Fig. 37.

Table 7: Effective HPTDC resolution based on cable delay measurements

Mode	Resolution (RMS)
Low resolution	0.34 bin (265 ps)
Medium resolution	0.44 bin (86ps)
High resolution	0.65 bin (64ps)
High resolution DLL tap adjust INL table correction	0.35 bin (34ps)
Very high resolution	2.4 bin (58ps)
Very high resolution DLL tap adjust INL table correction	0.72 bin (17 ps)

28.1. RC delay chain adjustment

The RC-delay chain adjustment have in some cases been seen to be a bit problematic. Channel differences are seen which can in some cases make the the choice of optimal parameters a bit problematic as the RC adjustments are common for all channels. It has been found by testing

a sample of chips that the following RC-adjustment values normally gives the best overall results: RC-adjust1 = 7, RC-adjust2 = 7, Rc-adjust3 = 4, RC-adjust4=2 (for a clocking frequency of 40MHz).

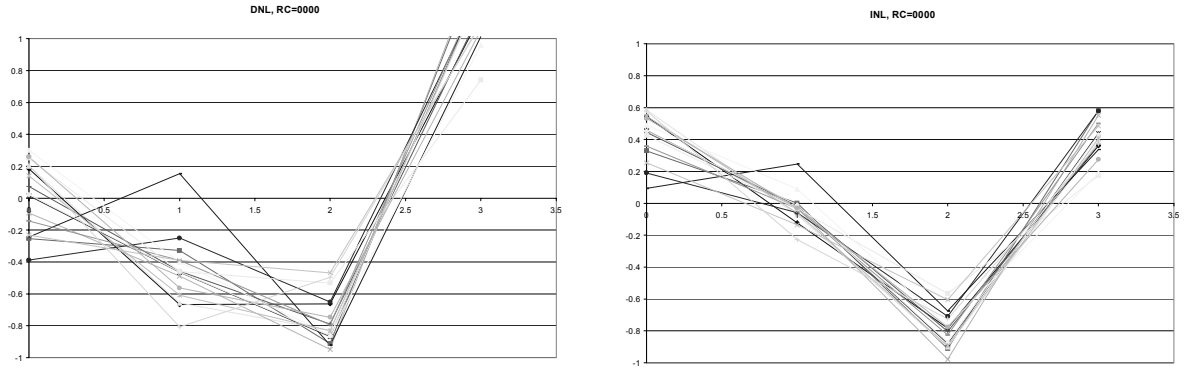


Fig. 40 DNL and INL Variation of RC-delay bins with all RC-adjust set to zero.

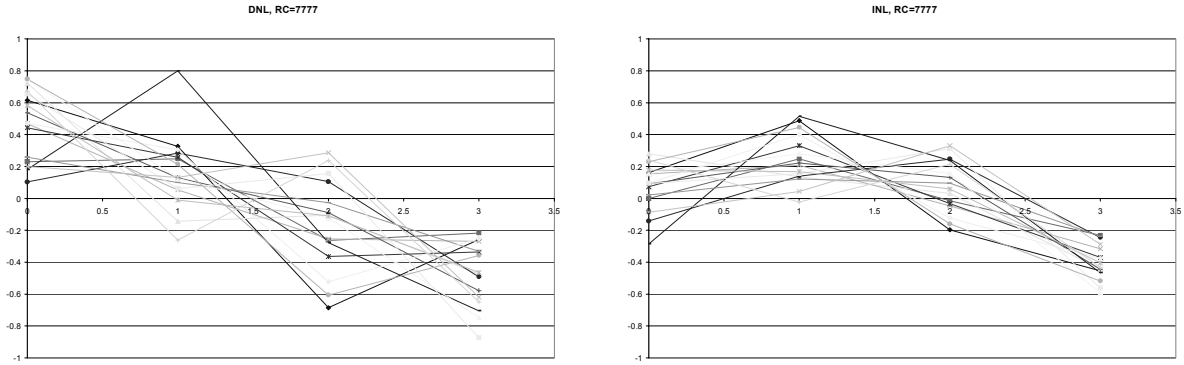


Fig. 41 DNL and INL Variation of RC-delay bins with all RC-adjust set to maximum.

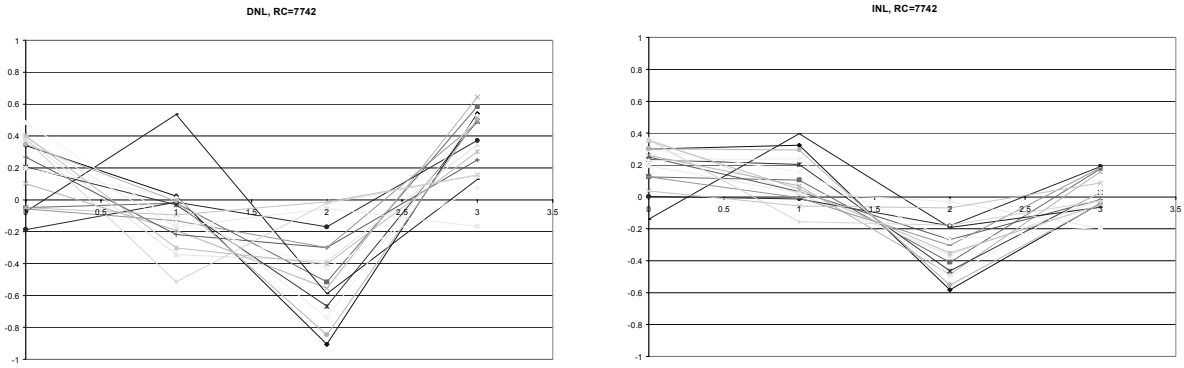


Fig. 42 DNL and INL Variation in RC-delay bins with RC-adjust1=7, RC-adjust2=7, RC-adjust3=4, Rc-adjust4=2.

As can be seen from Fig. 41, Fig. 42 and Fig. 42 containing data from different channels in different chips, there are some differences in the DNL and INL over the RC interpolation delay lines seen by different channels. It can though also be seen that if Bin0 in the DNL is large there is a clear tendency for Bin1 to be small and opposite. This is quite important as the time errors do not accumulate in the INL curves. As the INL stays relatively well centred around nominal values the observed variations in the RC-delay chains do not have a serious impact on the effective time resolution of the TDC in the very high resolution mode. The effective time resolution of the

HPTDC in the very high resolution is limited by a combination of time bin size, RC-bin size variations, DLL bin size variations, jitter in PLL, Jitter in DLL and the stability of the clock reference and the hit signals them selves (excluding observed INL problem from 40MHz crosstalk)

29. Typical applications of HPTDC

The HPTDC has many programmable features which makes it usable in many different applications. The TDC can be used in triggered or non triggered applications. It has mainly been designed to be used in LHC experiments where the bunch collisions are supposed to be synchronised to the TDC reference clock. There is though no principal problem using the TDC in applications where such a synchronisation does not exist. The choice between different time resolutions enables the TDC performance to be optimized to the requirements of a specific detector. For the highest resolution mode only 8 channels are available and should therefore only be used if absolutely necessary. The choice between the lower resolution modes is mainly a choice between acceptable time resolution and acceptable power consumption (Low resolution modes have significantly lower power consumption).

The (very) large set of programmable features in the HPTDC in some cases makes it difficult to determine quickly if the HPTDC can be used (and how to be used) in a given application. The data driven architecture requires an analyses of hit rates, trigger latency, trigger rate and available buffering resources inside the TDC to be performed. The best way to perform a thorough analysis of the performance of the HPTDC in a given application is to simulate its behaviour under realistic and conservative conditions. A Verilog model of the HPTDC and a simulation environment is available for such simulations and this can be requested from the CERN micro-electronics group. To give a quick overview of the flexibility of the HPTDC a few different examples are described briefly, together with a realistic set of the main programming data.

The fact that the HPTDC is a time stamp TDC instead of a start/stop TDC (common start, common stop) may be a bit foreign to many traditional TDC users. The fact that the TDC works with time stamps does by no means prevent it from being used in traditional start/stop configurations. As the HPTDC does not have a specific common start or stop channel this function must be handled by a dedicated time stamp channel per TDC or TDC module. The time difference between the start and stop measurements can then be obtained by simple subtractions of time stamp measurements. In this case it must be reminded that the effective RMS time resolution of such a time interval measurement, made from two conversions, is de-rated by a factor of square root of two (~1.4).

29.1. Fixed target experiment.

A fixed target is bombarded with a pulsed beam. Generated particles are detected behind the fixed target by a “wall” of detector elements. A beam detector is used to detect each beam pulse which is used as a time reference (start time) of each event. All detector channels are connected to TDC channels (stop time) measuring leading edges of discriminated detector signals. To extract the time difference between the start time and the stop times a simple subtraction is required in the DAQ system or alternatively off-line. All time measurements are read out to a DAQ system as it is assumed that the data rates are sufficiently low that no triggering is required to reduce the data rate from the TDCs.

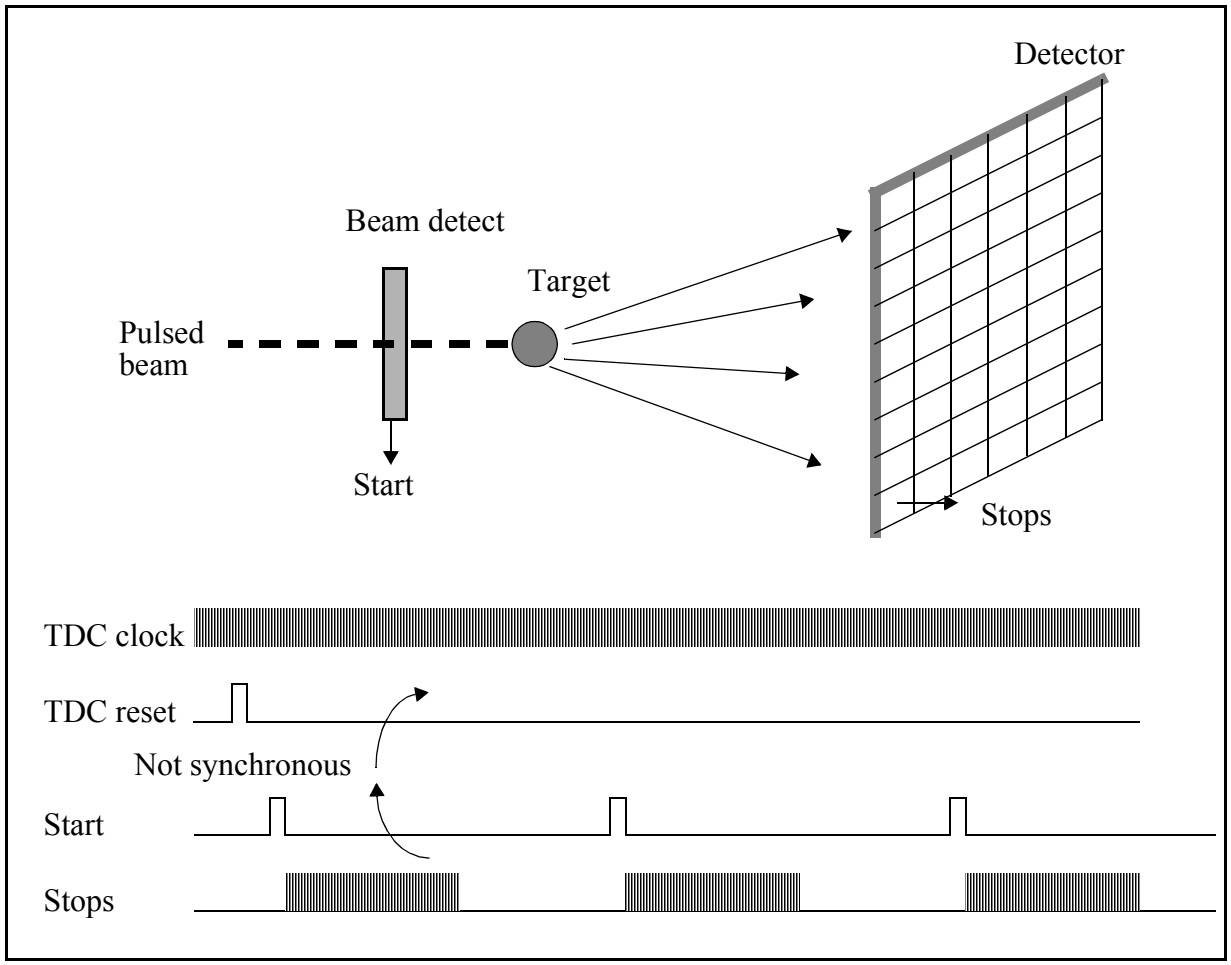


Fig. 43 Generic fixed target experiment using the HP TDC.

In the given application a high time resolution is required but for cost reasons it is required to maintain 32 TDC channels per TDC. The best time resolution the TDC can obtain on 32 channels is a 100ps binning where the PLL is used to perform clock multiplication from an external 40MHz clock to an internal DLL clock of 320MHz.

The detailed configuration of the measurement of the start time reference depends on the configuration of TDC modules and the generation of the reference clock to the TDC's. If a global clock is distributed to all TDCs in the system it is in principle only required to measure the start time on one TDC channel in the whole system. If local time references (local TDC clocks) are used on each TDC module it will be required to measure the start time on a TDC channel on each TDC module. It can also be decided to measure the start time on one channel of each TDC to have redundancy in its measurement and use this for additional system verifications.

The TDCs only needs to be reset once at the start of a run at any chosen moment as the absolute time stamps of the individual time measurements are of no importance. If the coarse time roll over is programmed to occur at the natural binary boundary (roll_over = hex FFF) the extraction of relative times between start and stops only needs to take this natural overflow into account (a two's complement subtraction can easily be made to do this automatically). Dedicated Event and bunch resets are not required and must be wired to passive states

As no trigger matching is used in the given application, time measurements from the TDC's are not grouped into events. All time measurements must therefore be read out as individual measurements and the grouping into events must be performed in the DAQ system (or local

processing on TDC modules). The read-out protocol from the TDCs used in this case can be a read-out controller starting the read-out of the TDCs after each beam pulse, where each TDC is allowed to keep the read-out token until all its data has been read out via a shared 32 bit bus.

Main programming data:

Enable_leading:	1	Only leading edges detected
Enable_trailing:	0	
Enable_pair:	0	
Dead_time:	01	10 ns dead time to remove potential ringing from analog front-end.
Dll_clock_source:	011	320MHz from PLL
Dll_mode	10	320MHz DLL mode
Leading_resolution	000	100ps
Enable_matching	0	No trigger matching
Enable_serial	0	Parallel readout
Enable_bytewise	0	32 bit readout
Keep_token	1	TDC allowed to keep token until no more data
Master	0	All TDCs configured as slaves.

29.1.1. Trigger matching

It can be advantageous to use the trigger matching function in the TDC, to either reduced the data rates by a real trigger, or just have an event based read-out. The configuration of TDC channels can remain as for the non triggered case.

It is assumed that the beam detect (also used as start time) is used as the basis for a kind of trigger. If no event rate reduction is required, the start can be used nearly directly as a trigger, keeping in mind that the latency must be larger than the matching window. If a real trigger function is required a separate trigger system must determine which events are to be triggered on. By defining values of the trigger latency and the trigger matching window it is possible to precisely select the time window, in relationship to the trigger, from where time measurements will be extracted.

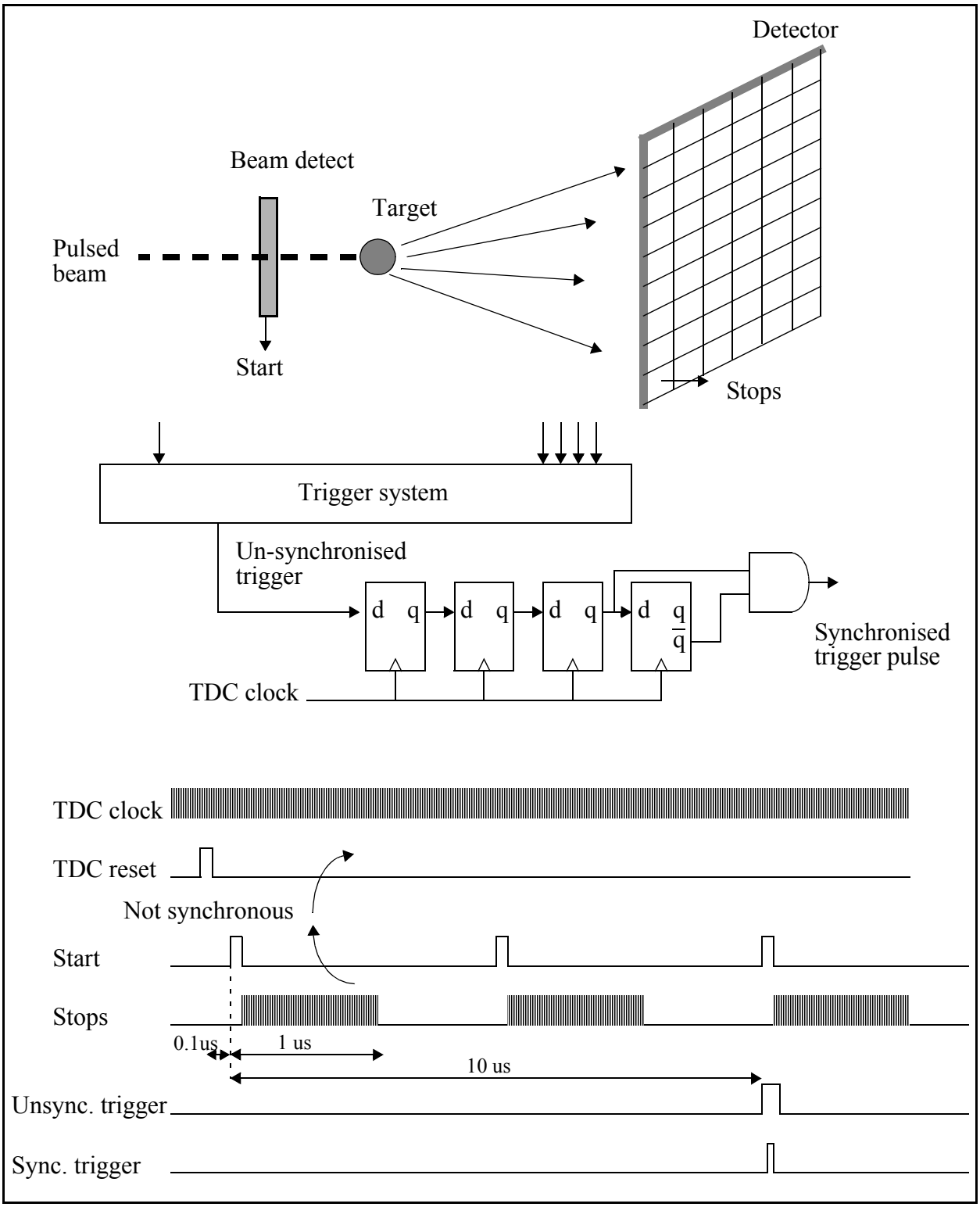


Fig. 44 Use of trigger matching based on non synchronous trigger signal.

An important point in the use of the trigger matching function of the HPTDC is that the trigger accept signal must be given as a clock synchronous signal with a fixed latency. If the trigger system is not synchronised to the TDC clock it will be necessary to synchronise it. Such a synchronisation can be performed either centrally or on local TDC modules. It is important to perform the synchronisation minimising the risk of propagating meta-stable conditions in the system, by using a multistage synchronisation scheme. As the HPTDC is capable of accepting new

triggers in consecutive clock periods it is also important to shape the trigger signal such that the TDC only sees a synchronous trigger accept signal for one clock period for each trigger.

For the further study it is assumed that a trigger system has a latency of 10 us and the maximum time spread of hits related to an event is 1 us.

In the given application, where it is important that the measured start times from the beam detection is also extracted, it is necessary to also extract hits which may have happened a short time before the triggered event. It is assumed that an additional pre-trigger window of 100ns is sufficient to guarantee this. In Fig. 44 where the different parameters are indicated graphically it is assumed that no new events are generated before a trigger has been generated for the preceding event, for the simplicity of the figure. The HPTDC can handle buffering multiple events during the trigger latency without any problems. It can in addition handle extracting overlapping events where hits belong to multiple events.

An effective trigger matching window of 1.1 us equals a programmed matching window of (1100ns/25ns - 1) = 43 clock cycles at 40 MHz. To ensure that all hits corresponding to triggers are correctly extracted, even if they are written into the L1 buffer slightly out of order, a search window of 43 + 8 = 51 is appropriate.

The effective trigger latency taking into account the 100 ns pre-trigger window is 10000ns + 100ns = 10100ns. This requires an offset between the coarse time counter and the trigger time tag counter of 404. Assuming a coarse time count offset of zero and a natural binary roll-over at FFF Hex the trigger count offset must be set to (0 - 404 modulus (2^{12}) or (FFF hex - 404 + 1) = 3692. To ensure the removal of old hits from the L1 buffer during periods with no triggers the automatic reject function must be enabled with a reject count offset of (FFF hex - (404 + 4) + 1) = 3688.

A parallel read-out interface is maintained. With the use of trigger matching, data is grouped into events and it is decided to use local headers and trailers from each TDC.

Main programming data:

Enable_leading:	1	Only leading edges detected
Enable_trailing:	0	
Enable_pair:	0	
Dead_time:	01	10 ns dead time to remove potential ringing from analog front-end.
Dll_clock_source:	011	320MHz from PLL
Dll_mode	10	320MHz DLL mode
Leading_resolution	000	100ps
Coarse_offset	0	
Enable_matching	1	Trigger matching enabled
Trigger_offset	3692	Trigger latency of 404 clock cycles
Enable_reject	1	Always used when trigger macthing
Reject_offset	3688	Reject latency of 408 clock cycles
Match_window	43	Matching window of 1.1 us
Search_window	51	Matching_window + 8
Enable_serial	0	Parallel readout
Enable_bytewise	0	32 bit readout
Keep_token	1	TDC allowed to keep token until no more data
Enable_global_header	0	Only for master
Enable_global_trailer	0	Only for master
Enable_local_header	1	Local headers enabled

Enable_local_trailer	1	Local trailers enables
Master	0	All slaves

29.2. LHC type experiments

The HPTDC has to a large extent been optimized for the use in LHC experiments with high interaction rates from 25ns spaced bunch collisions. A generic muon drift chamber detector is a typical application where a significant set of the available features in the HPTDC is used. A simplified layout of the generic muon detector is shown in Fig. 45 (only one layer). Each bunch crossing, muons from multiple interactions finally cross the drift chambers after the majority of other particles have been removed by the calorimeters and the muon filter. Leading edges are used for drift time measurements and the pulse width (10ns - 100ns) is used to perform a simple time-over-threshold measurement for slewing corrections. To ensure the correct corresponding leading time and pulse width, the option of pairing leading and trailing edges is used. For the drift time measurements the lowest time resolution is sufficient without degrading the position measurement. The drift chambers are assumed to have drift times of up to 500ns. Because of the very high interaction rate, muons from multiple interactions and multiple bunch crossings, will overlap in such a large time window. For each trigger the TDC must extract any potential candidate from the triggered event. The real muons from the event of interest will be identified at later stages of the DAQ when track reconstruction is performed. The minimum spacing between positive triggers are much smaller than the necessary matching window which implies that the TDC must support overlapping triggers where hits may belong to multiple events. Hits which belong to multiple events are extracted multiple times, once for each event. This prevents the following event processing to handle data sharing between events.

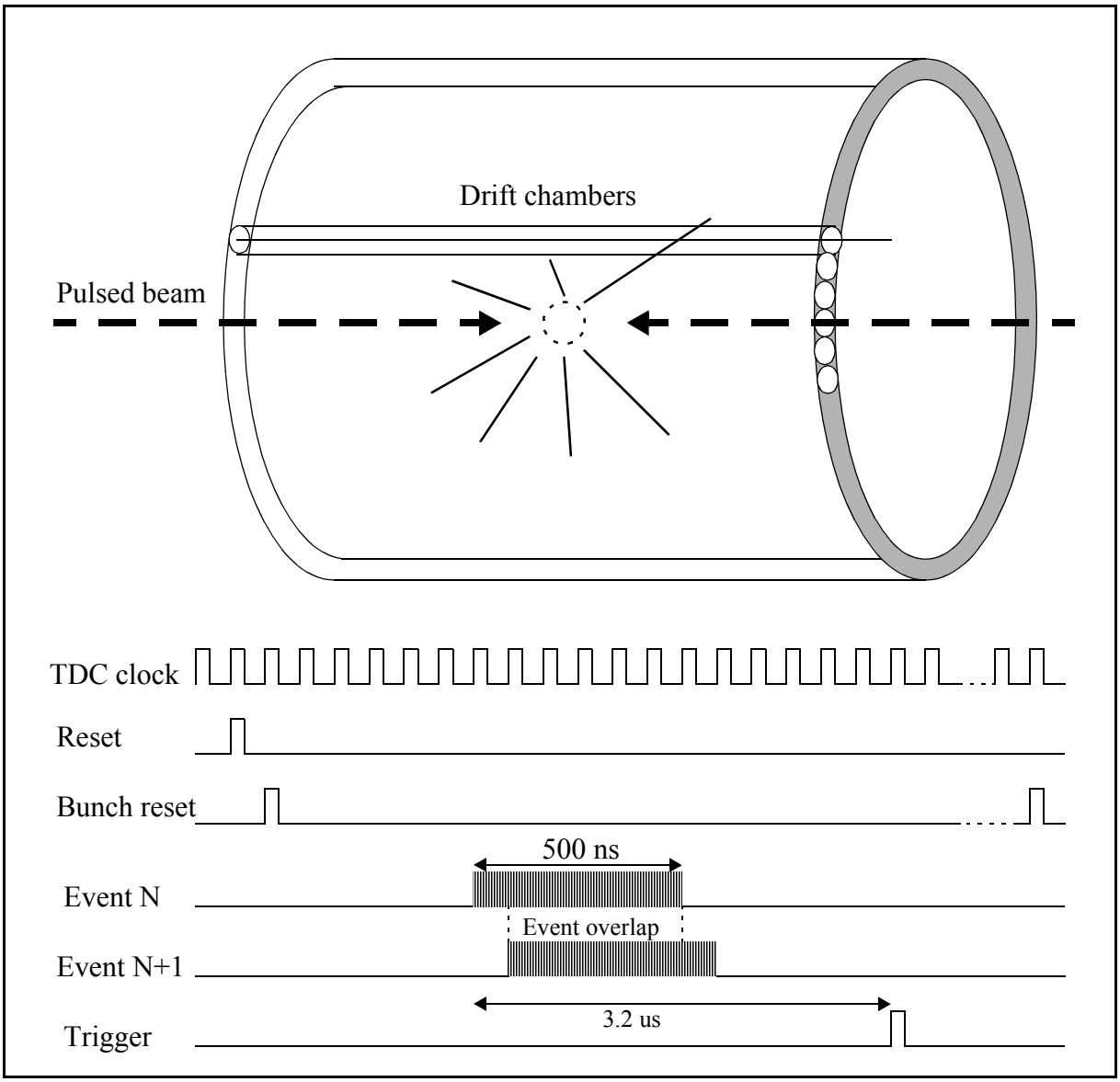


Fig. 45 LHC type application of HPTDC

In the majority of the LHC experiments the front-end electronics is clocked with a sampling clock locked to the bunch interactions. Each interaction has a bunch identification given as its relative position in the LHC machine cycle covering 3564 periods of 25ns. This bunch identification is used trough out the front-end and DAQ system to identify the correct timing of event data and must be appended to data as early as possible. The 12 bit coarse count and trigger count is made with this purpose in mind, to enable time measurements and triggers to be identified directly with the bunch identification.

For the generic muon detector a trigger latency of 3.2 us (128 clock cycles) is defined. To perform correct bunch identification, the coarse counters in the TDC are reset at the beginning of each LHC machine cycle. To enable a correct trigger matching to be performed across machine cycles (in fact not required in most LHC detectors because of a large bunch gap in end of the LHC machine cycle) the roll-over must be set to 3563 (max bunch ID). Lets assume for further discussions that the coarse count offset is set to zero. With the given roll over value of 3563 the trigger count offset can be found to be (3563 - 128 + 1) = 3437. The reject count offset is set to (3563 -(128+4) + 1) = 3433. If an offset of the coarse count is introduced all other counter offsets

must be shifted accordingly.

The matching time window of 500ns corresponds to a programmed matching window of (500ns/25ns - 1) = 19. In this case where paired measurements are being performed it is advisable to additionally extent the search window by 100ns/25ns = 4 clock cycles. The search window is then, with an additional width of 8, equal to 19 + 4 + 8 = 31.

As both the leading edge and the pulse width measurements are read out in one word, only a limited number of bits is available for each. To gain resolution the leading edge measurement is read out relative to the time of the event (bunch ID). To cover a 100ns dynamic range of the width measurement in 7 bits, the resolution of the width is set to 800ps.

The read-out protocol chosen is a byte-wise read-out shared between 4 TDCs used to drive directly a serial link transmitter (HotLink). One TDC is defined as the read-out master controlling the remaining three slave TDC's. To have a sufficient set of headers and trailers without taking to much bandwidth only global headers and trailers are used.

Main programming data:

Enable_leading:	0	
Enable_trailing:	0	
Enable_pair:	1	Pairing of leading and trailing edges
Dead_time:	01	10 ns dead time to remove potential ringing from analog front-end.
Dll_clock_source:	001	40MHz from PLL
Dll_mode	00	40MHz DLL mode
Leading_resolution	011	781 ps
Width_resolution	0011	781 ps
Coarse_offset	0	
Enable_matching	1	Trigger matching enabled
Roll-over	3563	
Trigger_offset	3437	Trigger latency of 128 clock cycles
Enable_reject	1	Always used when trigger macthing
Reject_offset	3433	Reject latency of 132 clock cycles
Match_window	19	Matching window of 500 ns
Search_window	31	Matching_window + 8 + 100ns/25ns
Enable_serial	0	Parallel readout
Enable_bytewise	1	8 bit readout
Keep_token	1	TDC keeps token until end of event
Enable_global_header	1	Only for master
Enable_global_trailer	1	Only for master
Enable_global_header	0	Remaining slaves
Enable_global_trailer	0	Remaining slaves
Enable_local_header	0	Local headers disabled
Enable_local_trailer	0	Local trailers disabled
Master	1	One master
Master	0	Remaining slaves

30. Limitations of HPTDC architecture

The data driven architecture of the HPTDC poses a set of limitations which must be taken into account for each application. Several levels of data merging are performed before data are finally read out. Each level of data merging has a related set of de-randomizer buffers to minimize the effect of such potential bottlenecks. In addition it must be verified that sufficient buffering is available in the L1 buffer to store data during the trigger latency.

The effect of data merging and shared buffering depends strongly on the statistical properties of the hits and the triggers, and their correlation. If little correlation exists it can in general be considered safe to use half the bandwidth/buffering available and use the remaining half for statistical variations. For cases with hits coming in bursts, or hits highly correlated between channels, it will often be required to perform a more detailed study.

A set of statistical simulations, using the simulation model available, has been performed to illustrate the behaviour of the HPTDC under different conditions. The HPTDC is in its baseline configuration considered to run with an internal logic clock frequency of 40 MHz. The performance improvement obtained using a higher clock speed is dealt with separately.

30.1. Channel merging

Hits from 8 channels are merged into one L1 buffer. A 4 deep derandomizer is available in each channel to smoothen the bandwidth of hits and allow channel measurements to wait before being written into the L1 buffer. The Arbitration between channels is performed such that the available bandwidth is used the most efficiently and most fair possible. When a channel buffer is full, subsequent hits will simply be rejected until free buffering space is available. The absolute maximum bandwidth which can be used by one channel is 10MHz (@ 40 MHz clocking) limited by the synchronisation logic. When several channels are used the total bandwidth available (40MHz) is distributed such that each channel gets its fair share (max 10 MHz or 1/8 of total bandwidth). The first channels in each group will be serviced quicker than the last ones because of the simple arbitration scheme used. This will for high hit rates give an advantage to the high priority channels which can use their channel derandomizers more efficiently. Channels with lower priority will under such circumstances appear like they have a smaller derandomizer capability and therefore have a slightly higher rejection.

For 8 completely uncorrelated sources of hits, with an assumed dead time of 5ns, the channel derandomizers are quite efficient and a total hit rate per channel of up to 2.5 MHz can be sustained with low losses. The hit loss as function of the hit rate per channel is shown in Fig. 46. The channel buffer occupancies and the total number of hits waiting in the channel buffers of a group are shown in Fig. 47 for a selected set of conditions. The ratio of hits being rejected is clearly seen in the channel buffer occupancy histograms. A channel buffer occupancy of 4, seen when a new hit arrives, implies that the hit is lost (seen between x=4 and x=5 in the channel buffer histogram). The group occupancy histograms gives an indication of the total number of hits waiting, in the 8 channel buffers, to be written into the L1 buffer via the shared data path.

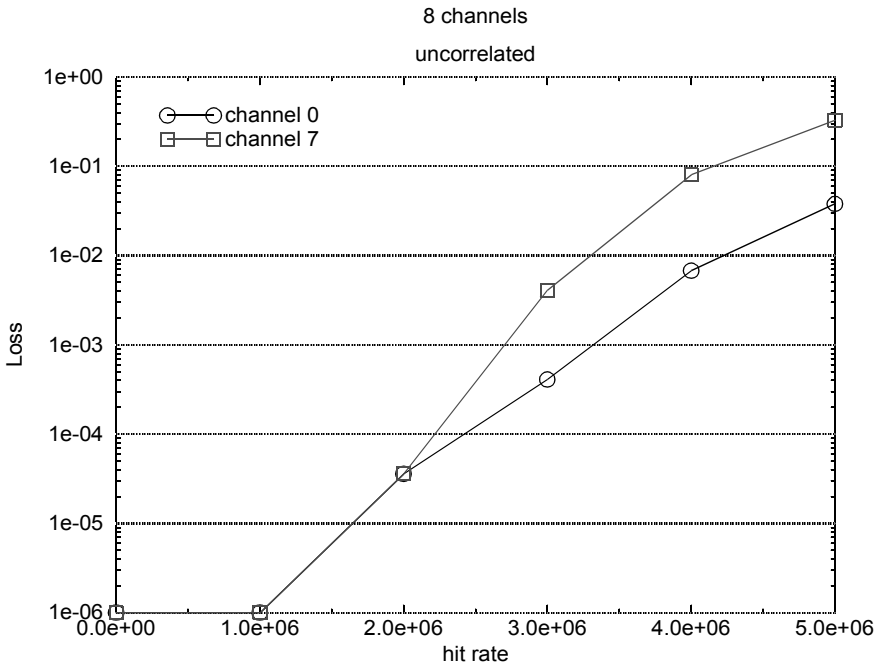


Fig. 46 Hit loss of uncorrelated hits with no dead time.

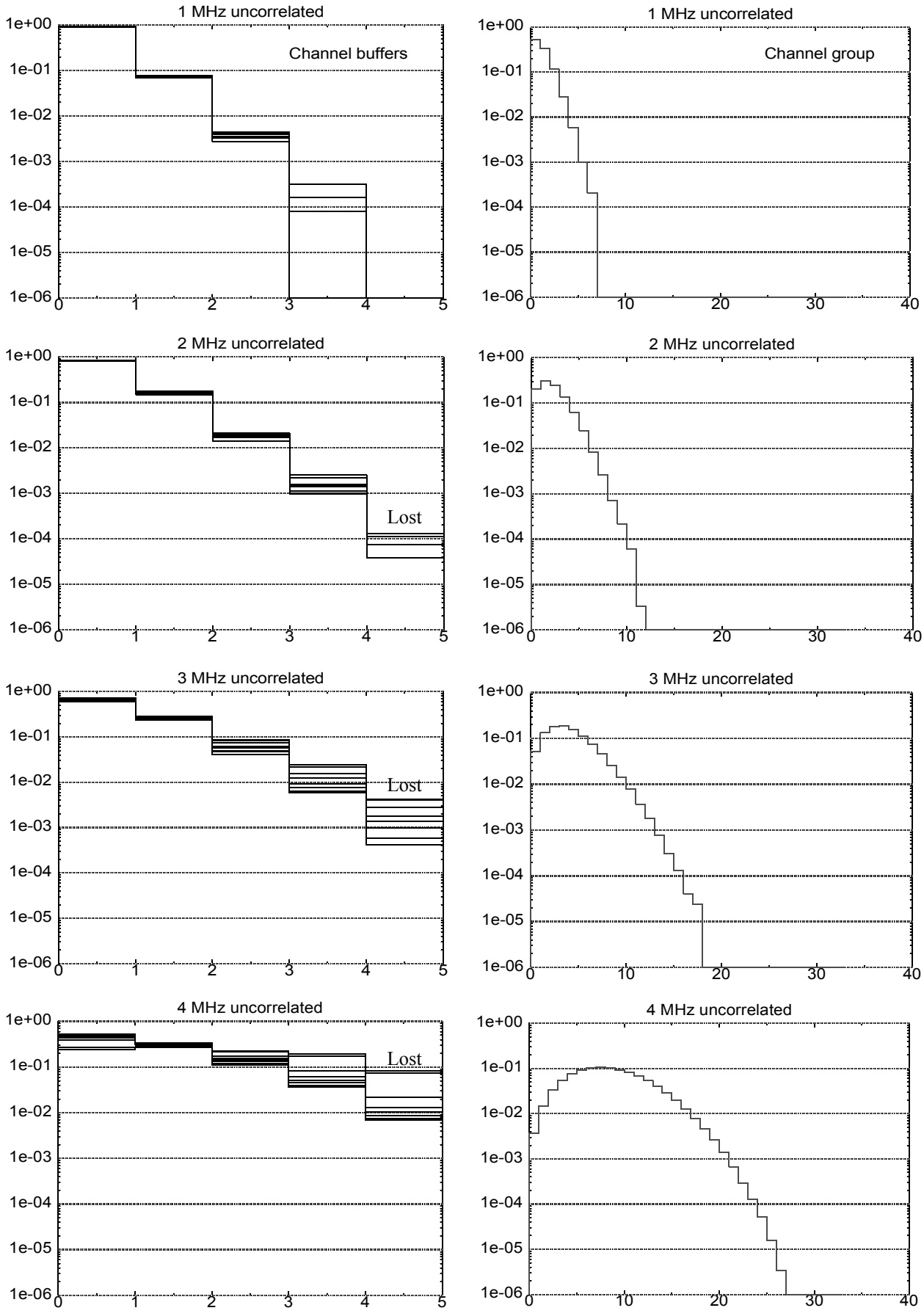


Fig. 47 Channel buffers and group occupancies at different hit rates with no channel dead time

The efficiency of the channel derandomizers also depends on the intrinsic dead time on each channel. In case of a dead time of 100ns in the detector and the associated analog front-end, the hit loss in the TDC is significantly affected as shown in Fig. 48. This can be explained from the simple fact that closely spaced hits are the most demanding from point of view of de-randomization. If such closely spaced hits are excluded by the detector itself or the analog front-end, the derandomizers in the TDC channels are under much less "stress".

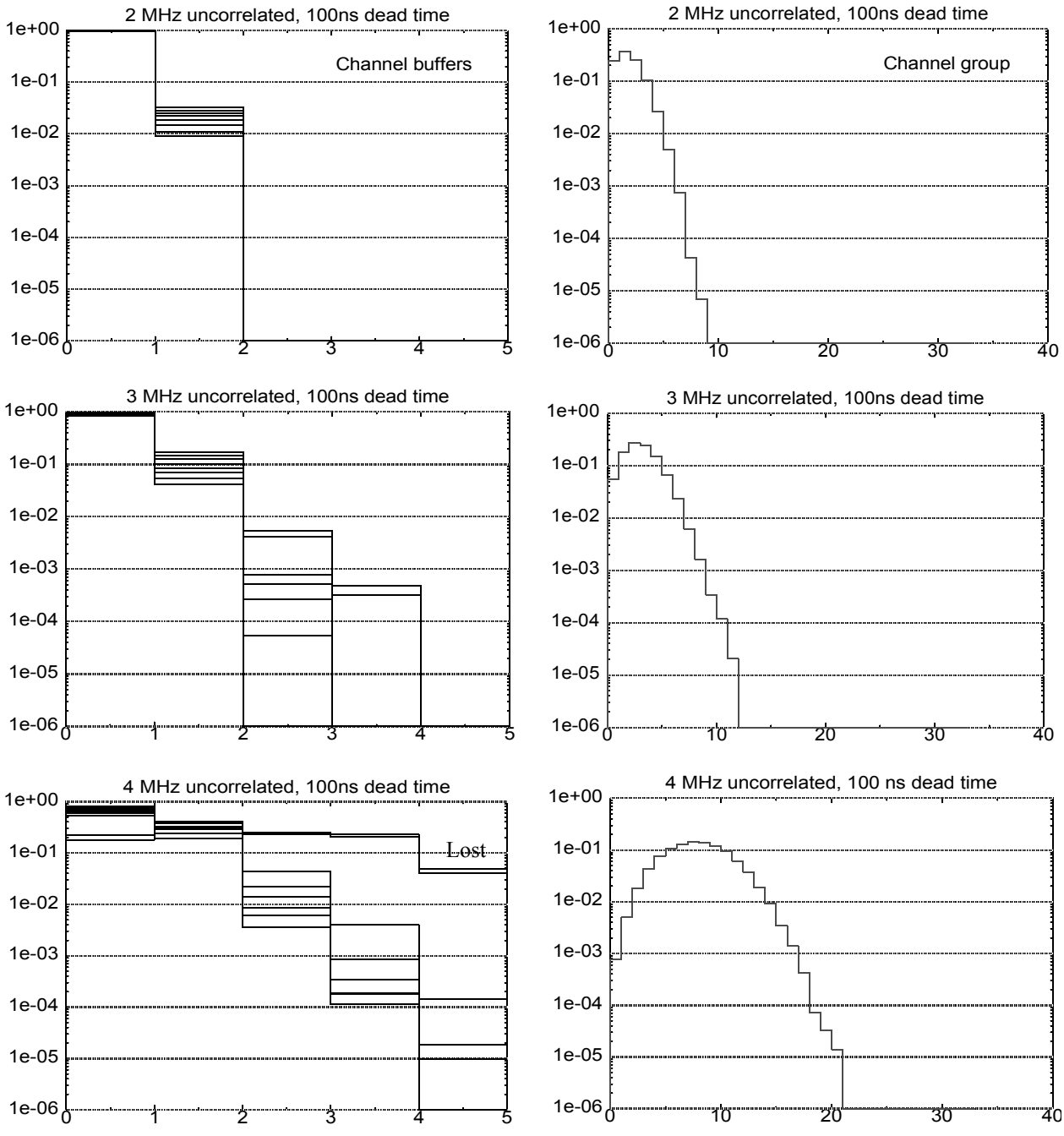


Fig. 48 Channel buffers and group occupancies at different hit rates with a channel dead time of 100ns

At hit rates close to saturate the total bandwidth available, it is clearly seen that the last two channels of a channel group suffers significantly higher losses than the remaining channels. This is caused by a combination of effects from the synchronisation logic, delays in the channel controllers and the final arbitration. At saturation, channel 6 and 7 in a channel group is mainly

serviced last, just before the request queue becomes empty and can be reloaded. Because of delays in the synchronisation logic and the channel controllers the arbitration logic can not “see” if more hits are waiting in channel 6 and 7 when reloading the arbitration queue. This implies that these hits will be forced to wait an additional arbitration period of up to 8 clock cycles (time required to service full request queue). This effect only occurs when all TDC channels have high rates concurrently and starts to saturate the arbitration bandwidth. If only a few channels in a channel group have high rates (noisy channels) then this effect is insignificant.

A 100ns dead time must be considered quite large, for cases where the hit rate approaches several MHz. When using a more realistic dead time of 25ns, hit rates up to 3 MHz per channel on all 8 channels can as shown in Fig. 49 be acceptable in many applications.

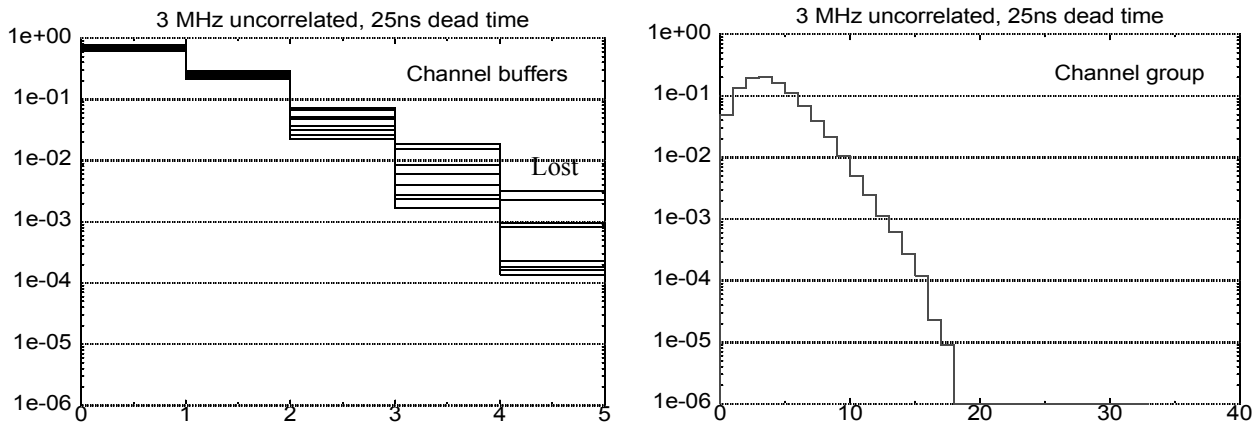


Fig. 49 Channel buffers and group occupancies at 3 MHz hit rate with a channel dead time of 25ns.

To sustain higher hit rates per channel it is possible to use only four channels in each channel group. In this case the hit loss for a given hit rate is decreased as indicated in Fig. 50. As the maximum bandwidth available per channel is 10 MHz (at 40 MHz clocking speed), the hit loss is in this case not dominated by the competition between channels, but is mainly determined by the time de-randomization needed in each individual channel. The time de-randomization is improved if a 25ns dead time is enforced as indicated in Fig. 51. Using fewer than four channels per group does not bring significant improvements.

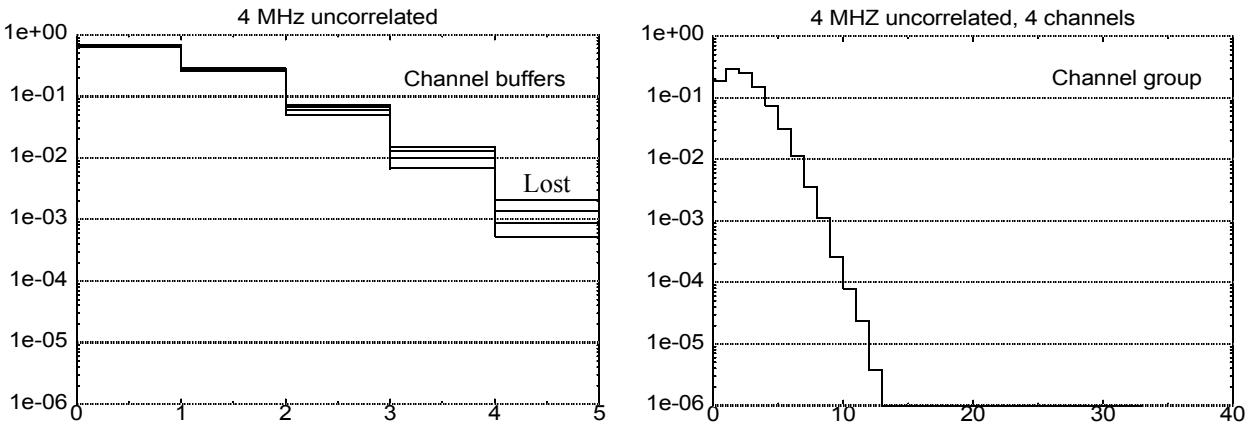


Fig. 50 Channel buffers and group occupancies when using 4 channels per group with no dead time

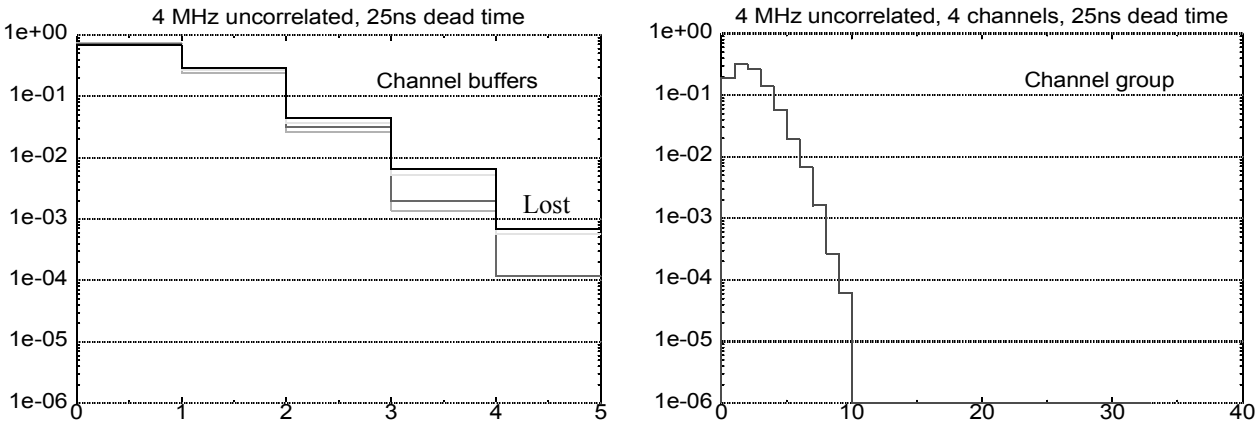


Fig. 51 Channel buffers and group occupancies when using 4 channels per group with a channel dead time of 25ns.

For applications with very strong correlations between channels the effective hit rates must be decreased. In case of a 100% correlation between channels, where all channels always fires at the same time, the efficiency of the channel derandomizers are shown in Fig. 52.

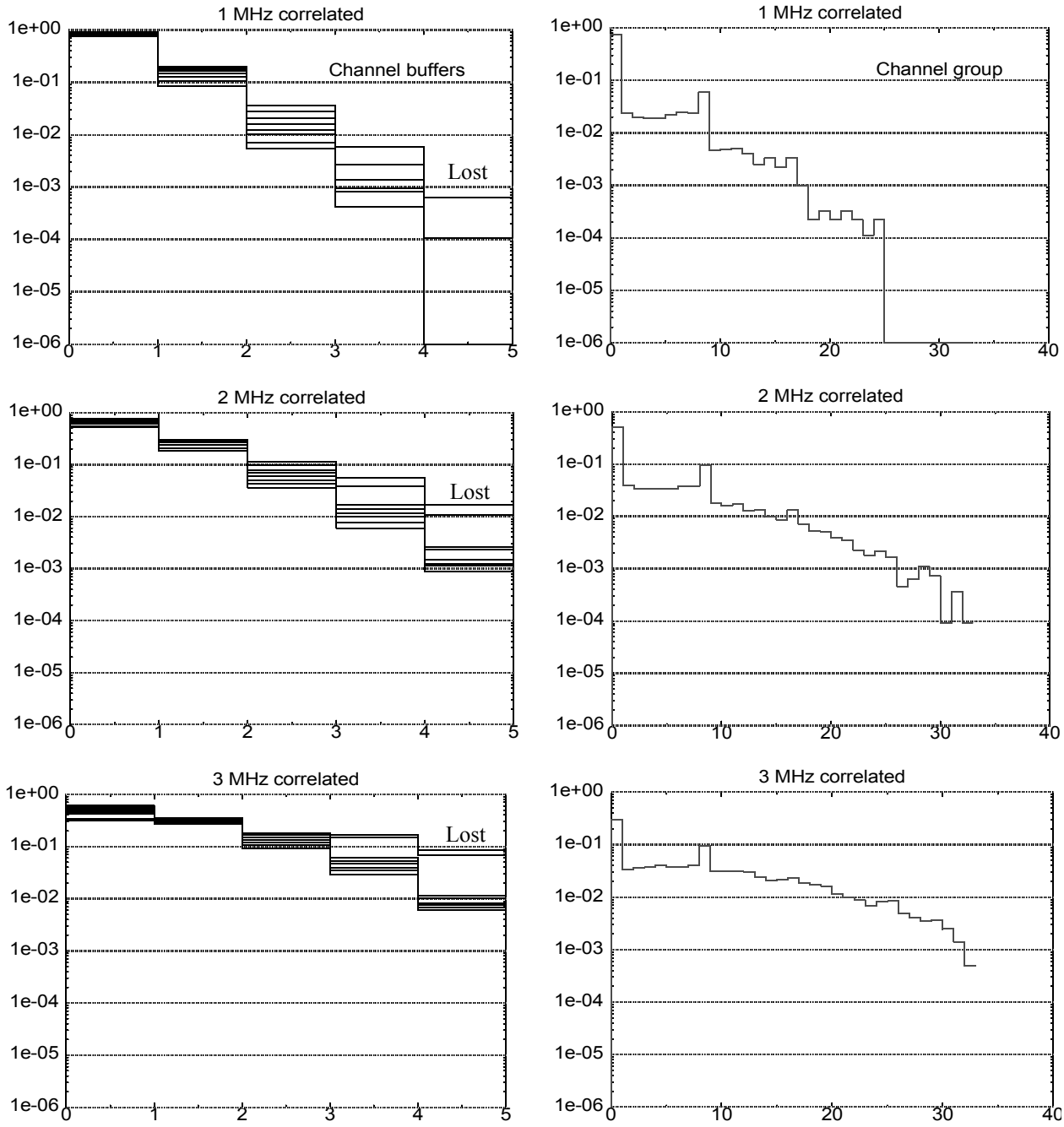


Fig. 52 Channel buffers and group occupancies with 100% correlated hits on 8 channels with no dead time

When the pairing mode of the TDC is used, where a leading edge and its corresponding falling edge are handled as a measurement pair, the channel buffers can only store 2 measurement pairs. In this case the hit rates must in general be lowered to half as indicated in Fig. 53. The fact that two measurements are stored per signal pulse is clearly seen in the occupancies. The hit losses in the pairing mode is not as in the previous case mainly determined by the competition between the 8 channels in a group. The main limitation in this case is that fact that a de-randomization of only two pairs is quite small, when the maximum processing rate per channel in the arbitration is limited to 10MHz. The introduction of dead-time in the channels (e.g. 100ns) will to large extent resolve this problem and the losses becomes insignificant.

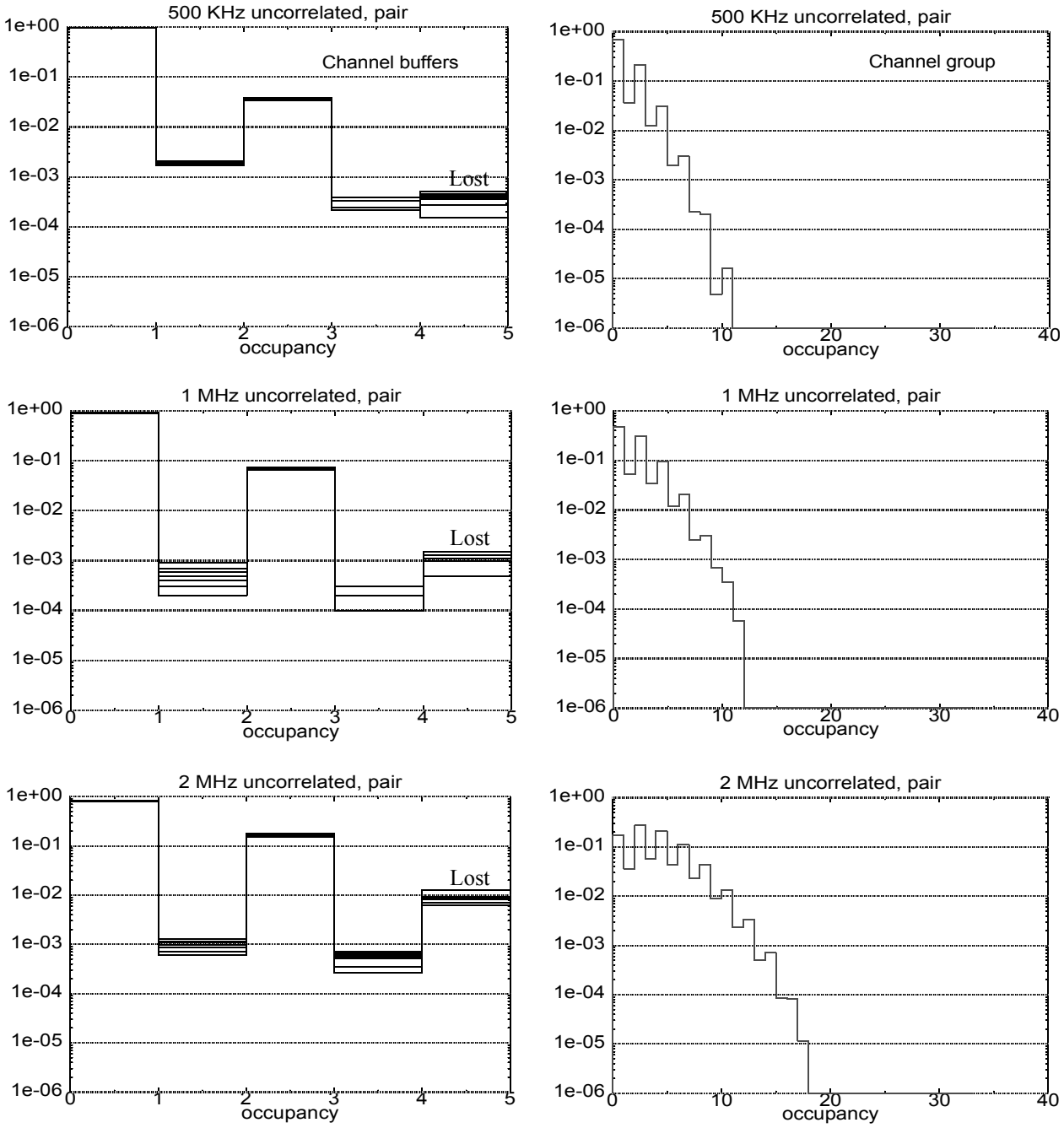


Fig. 53 Channel buffers and group occupancies in pairing mode on 8 channels

All simulations of hit losses assumes that hits are evenly distributed in time (using an exponential distribution function of the time interval between hits as shown in Fig. 54). For applications where the hits are very bursty in nature it will be required to make more specific investigations (simulations).

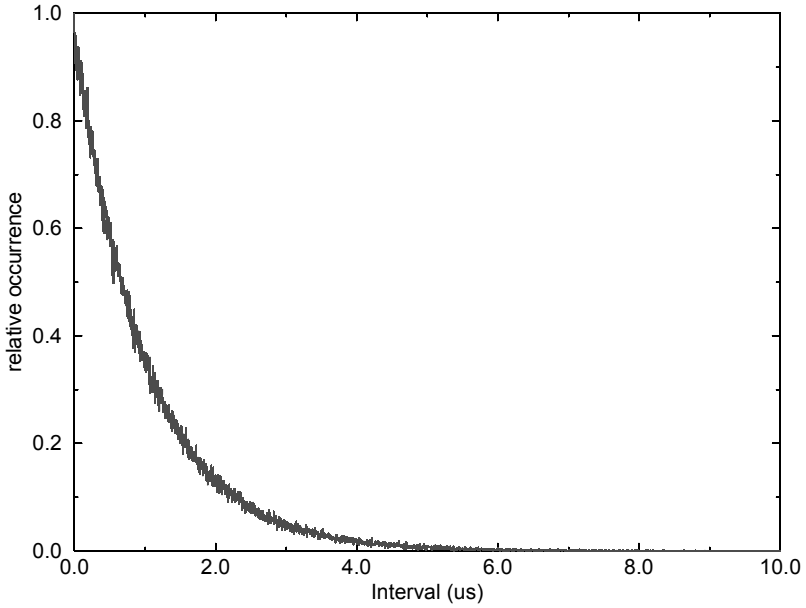


Fig. 54 Distribution of interval between hits on same channel at 1 MHz rate

The conclusion which can be drawn from the shown simulations is that hit rates of up to 2 MHz can be accommodated with low losses, even considering the case where a 100% correlation between channels exists. When pairing of leading and trailing edge measurements are performed the channel derandomizers are less efficient and hit rates should not exceed 1 MHz on all channels as each hit pulse occupies two locations in the channel buffer. In case of using only 4 channels per channel group, hit rates of up to 4 MHz can be handled in the non pairing mode.

When the internal core of the TDC runs with a higher clock frequency, the acceptable hit rates can be multiplied with the same factor (see: Improved performance at higher internal clock frequency on page 95).

30.2. L1 buffer

The L1 buffer must store hit measurements during the latency of the trigger, when the trigger matching function is used. The required buffering is proportional to the product of the average hit rate on the 8 channels, feeding one L1 buffer, and the trigger latency. The L1 buffer occupancy is basically insensitive to the correlations between channels but is obviously sensitive to the general time distribution of the hits. The occupancy of the level 1 buffer is shown for a few selected cases in Fig. 55, assuming evenly distributed hits at a rate of 1 MHz. It can in general be considered safe to work under conditions where the average buffer occupancy is less than half the available buffering capability. In case the L1 buffer runs full, the HPTDC will keep track of the time window where hits have been lost and mark events which have suffered losses from this.

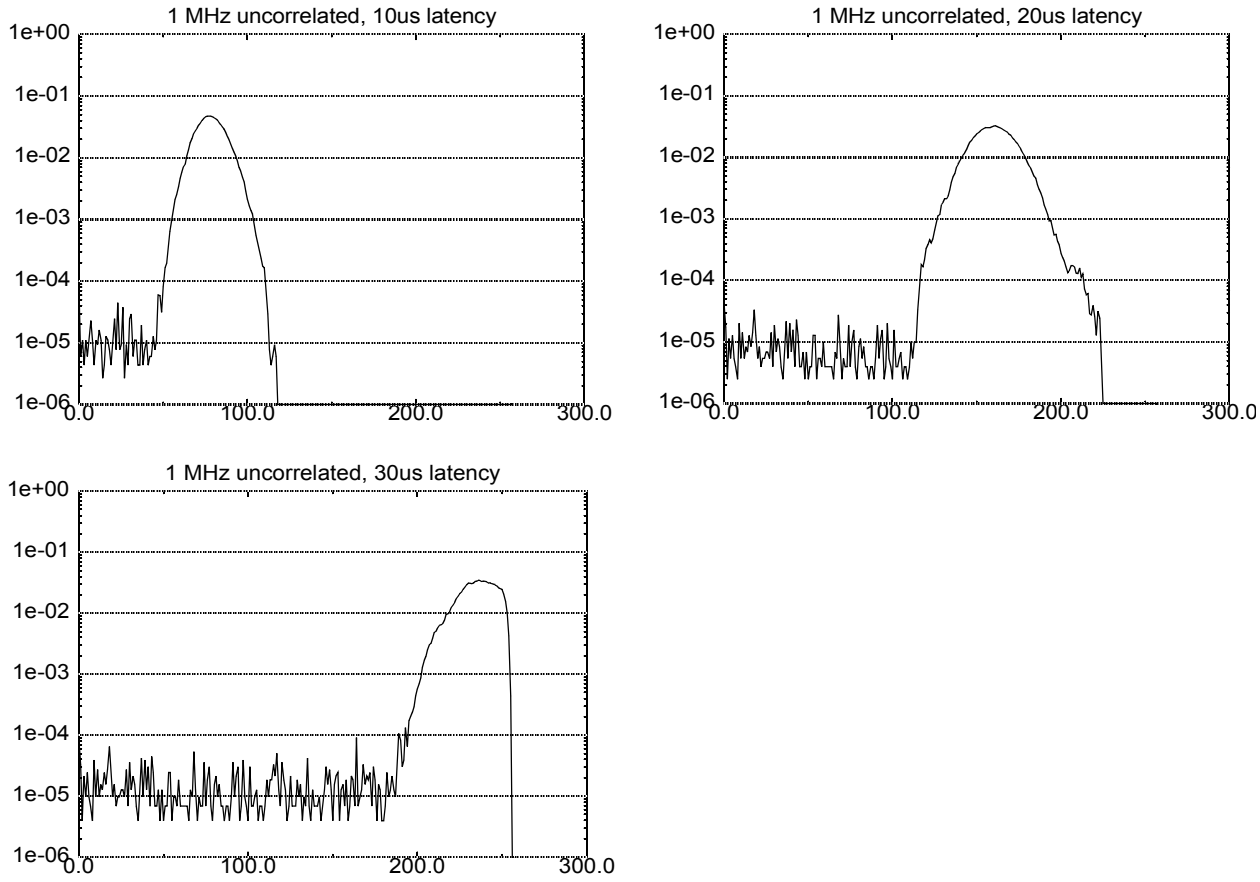


Fig. 55 L1 buffer occupancy at 1MHz hit rate and different trigger latencies

In the shown cases each L1 buffer must buffer an average number of hits given as: 8 channels x 1 MHz x 10us = 80 hits, per 10 us of trigger latency. In case of a trigger latency of 30us it is clearly seen that the L1 buffer distribution is truncated by the fact that the buffer runs full, and hit measurements have been lost

The L1 buffer occupancy can in certain cases be influenced by the characteristics of the trigger matching. During a burst of trigger accepts, the L1 buffer can be forced to buffer data for extended periods, while the trigger matching is performed. This is especially the case if a large trigger matching window is used and many hits are matched to multiple triggers during a trigger burst. To illustrate this, a set of L1 buffer occupancy histograms are shown in Fig. 56, for varying trigger rates using a large trigger matching window of 1 us and a trigger latency of 10 us.

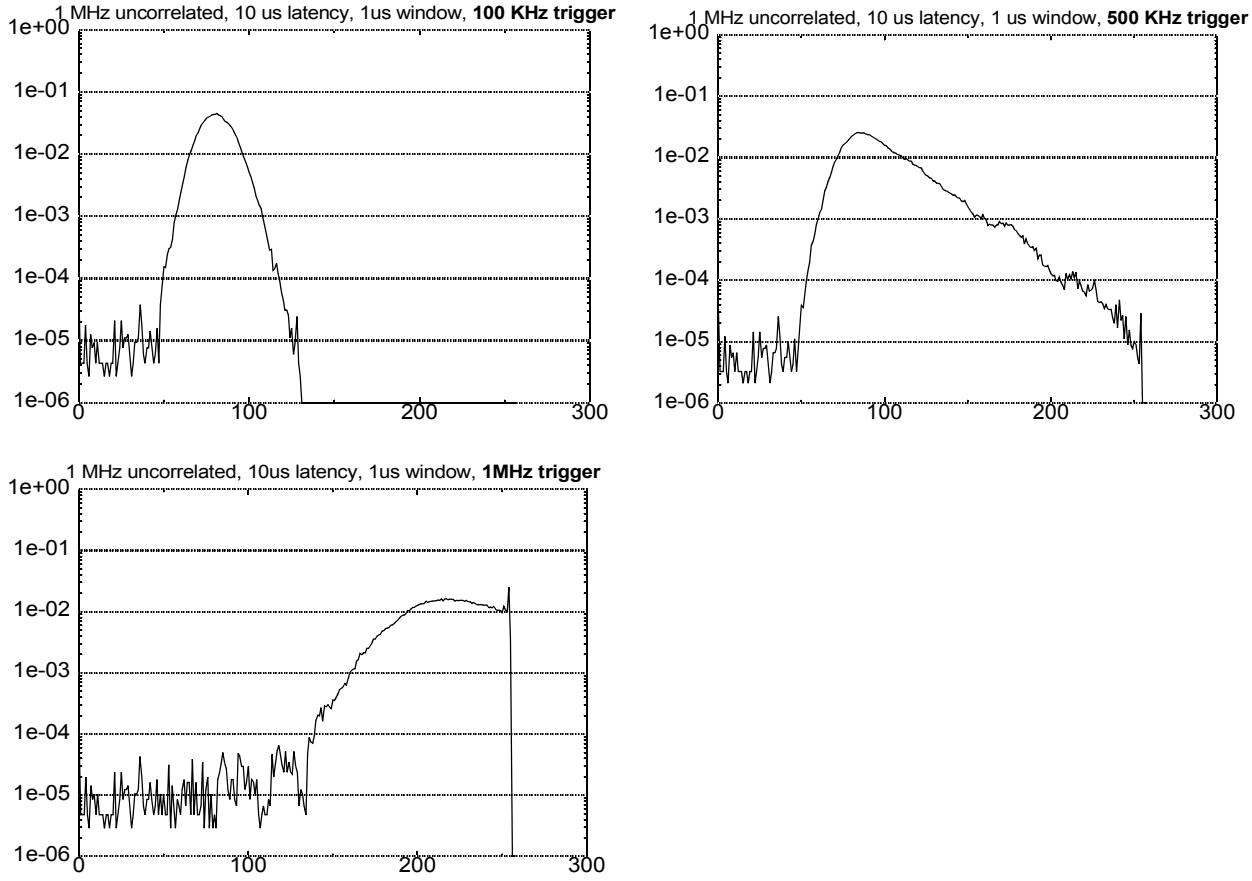


Fig. 56 Trigger matching effect on L1 buffer occupancy.

The conditions where such limitations in the trigger matching occur are related to the limited bandwidth of 40MHz into the read out FIFO shared by 4 channel groups. With a trigger matching window of 1 us and 32 channels with 1 MHz hit rates the average number of matched hits per event equals 32 x 1MHz x 1us = 32. A header and a trailer is required for each event and the trigger matching has a processing overhead of the order of 4 clock cycles. This means that the trigger matching of each event will on average consume 32+2+4 clock cycles = 950ns. The absolutely maximum trigger rate is therefore in this case of the order of 1 MHz. To allow for statistical fluctuations of the trigger rates and the hit rates, the trigger rate should though not exceed ~500KHz. This matches well the L1 buffer occupancy data presented in Fig. 56. At 500 KHz trigger rate a tail on the L1 buffer occupancy is clearly seen. At 1 MHz trigger rate the trigger matching is completely saturated and a significant number of hits have been lost because of L1 buffer overflows.

The trigger rate in most experiments is centrally controlled in a such a manner that many closely spaced triggers are prevented. This is normally required to prevent large amounts of data to accumulate in front-end and DAQ buffers, which may finally overflow. Such a trigger rate control mechanism can consist of a minimum trigger spacing of 100ns and a limitation of maximum 16 triggers in a time window equal to 16 times the average trigger spacing (e.g. maximum 16 triggers in 16 us with a trigger rate of 1MHz). In this case the large tail see in the previous case is significantly reduced as shown in Fig. 57.

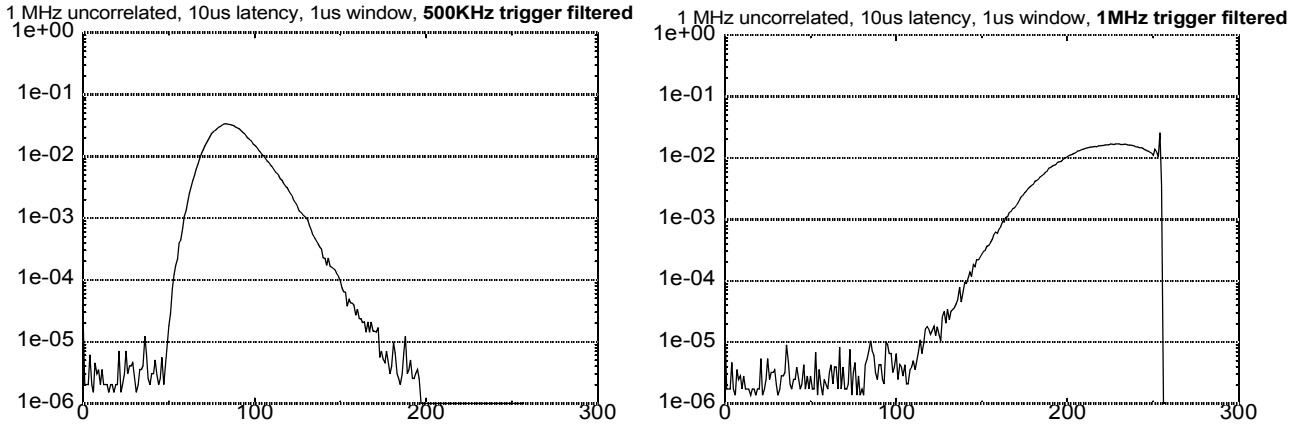


Fig. 57 L1 buffer occupancy with central trigger limitations

30.3. Trigger FIFO

The trigger FIFO is intended to buffer incoming triggers when the trigger matching is busy processing previous triggers. Its occupancy is sensitive to a large set of parameters: The trigger rate, the "burstyness" of the trigger and finally the time required to perform the trigger matching of an event. The time required to perform trigger matching in a channel group is proportional to the product of the trigger matching search window and the total hit rate of the 8 channels. In most cases the trigger matching is limited by the merging of matched hits from 4 channel groups into one read-out fifo as explained previously. When the trigger fifo needs to store significant amounts of triggers, the L1 buffer occupancy will in most cases also be affected. The trigger FIFO occupancy, for the same conditions as shown in the L1 buffer occupancy simulation (Fig. 56), is shown in Fig. 58. In case that the trigger FIFO overflows, during a period of extreme conditions, the trigger matching logic will take care of regenerating empty dummy events for each trigger lost, to prevent the loss of event synchronisation in the system.

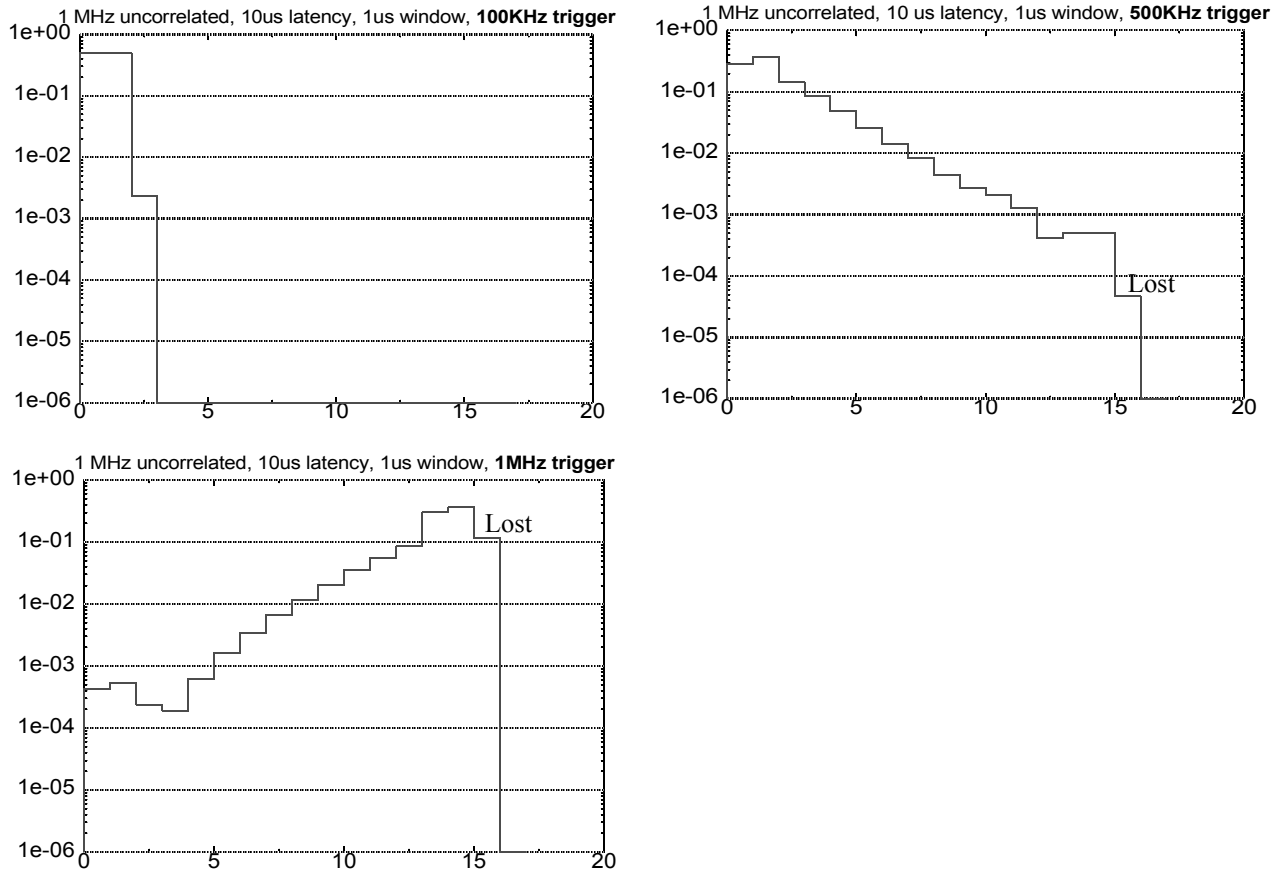


Fig. 58 Trigger FIFO occupancy for same conditions as in Fig. 56.

The trigger FIFO occupancy is influenced by any trigger limitations in a similar way as the L1 buffer. The trigger FIFO occupancy is shown in Fig. 59 with the same trigger restrictions as used in Fig. 57.

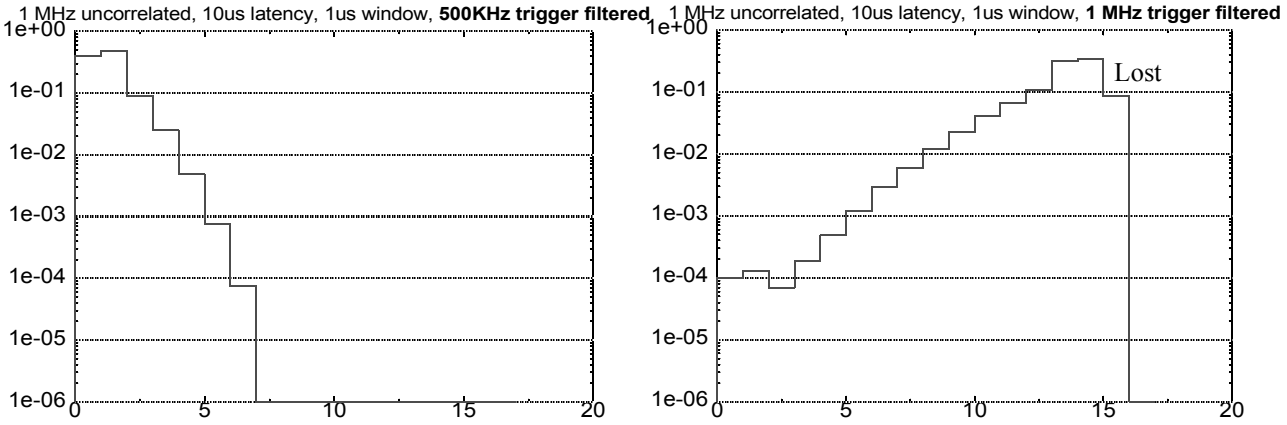


Fig. 59 Trigger FIFO occupancy when using trigger limitations as in Fig. 57

30.4. Read-out FIFO

The use of the read-out FIFO is mainly determined by the read-out speed used. If a slow read-out is shared between many TDCs the read-out FIFO may be required to store significant amounts of data. A case is shown in Fig. 60 where 4 TDCs share a slow 40 Mbits/s serial read-out where the average bandwidth required is 25 Mbits/s (((4 TDCs x 32 channels x 100KHz hit rate x 1us

match window) + 2 header/trailer) x 50KHz trigger x (32 + 3 bits per word) = 25.6 Mbits/s). It is in general not recommended to run the read-out speed too close to the required average bandwidth. As previously mentioned, it is in general advisable to have an available read-out bandwidth close to the double of the required average bandwidth, to prevent data to pile-up inside the TDC during local bursts. If large amounts of data start to accumulate in the read-out FIFO it may in some configurations influence the occupancy of other buffers in the TDC. It is recommended to use the option to let the trigger matching reject hits if the read-out FIFO is full. This will protect other buffers in the TDC from problems related with a too slow read-out.

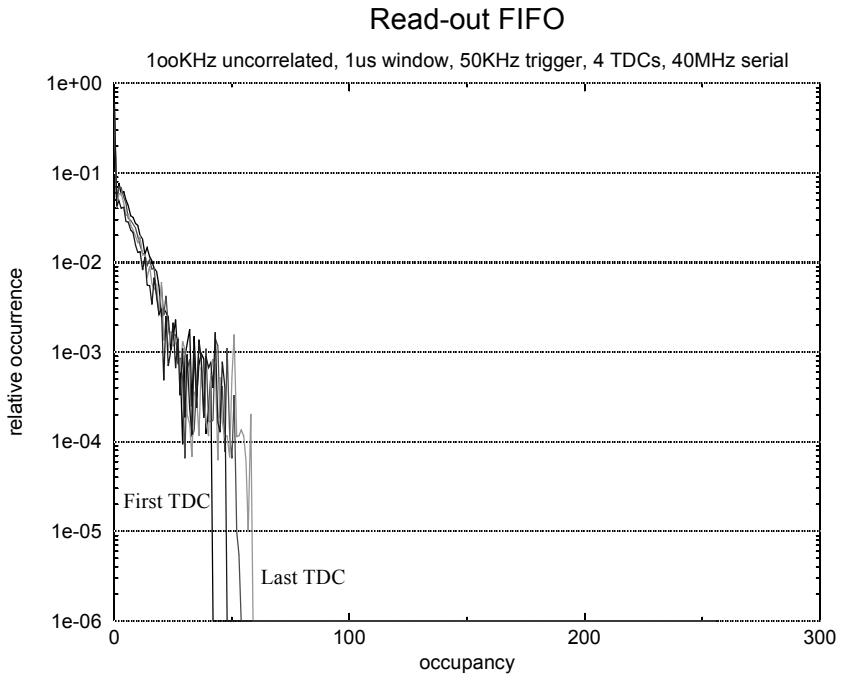


Fig. 60 Read-out FIFO occupancies for four TDCs sharing a slow serial read-out.

30.5. Improved performance at higher internal clock frequency

The shown limitations of the HPTDC architecture assumed a clocking frequency of 40 MHz of the internal logic (lowest power consumption). At higher internal clock frequencies (80 or 160 MHz) the performance of the HPTDC can in certain cases be significantly improved.

At a higher internal clock frequency, the arbitration between channels in a channel group and the transfer from the channel buffers to the L1 buffer are significantly improved. A general guideline is that the sustained hit rates per channel is proportional to the clocking speed used. To demonstrate this the channel buffer occupancies at 80 MHz and 160MHz clocking frequency are shown in Fig. 61 and Fig. 62 for a representative set of hit rates.

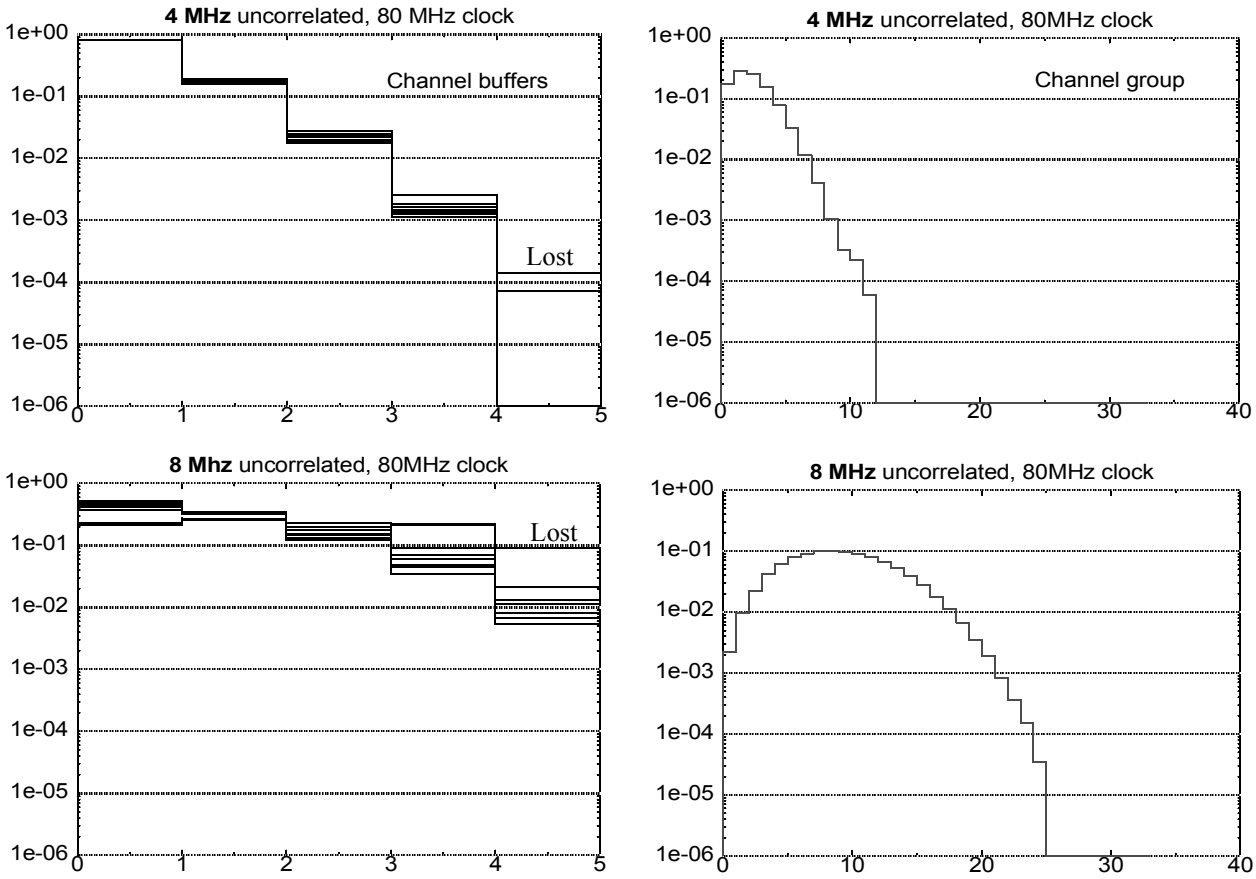


Fig. 61 Channel and group buffer occupancies at 80 MHz clock speed.

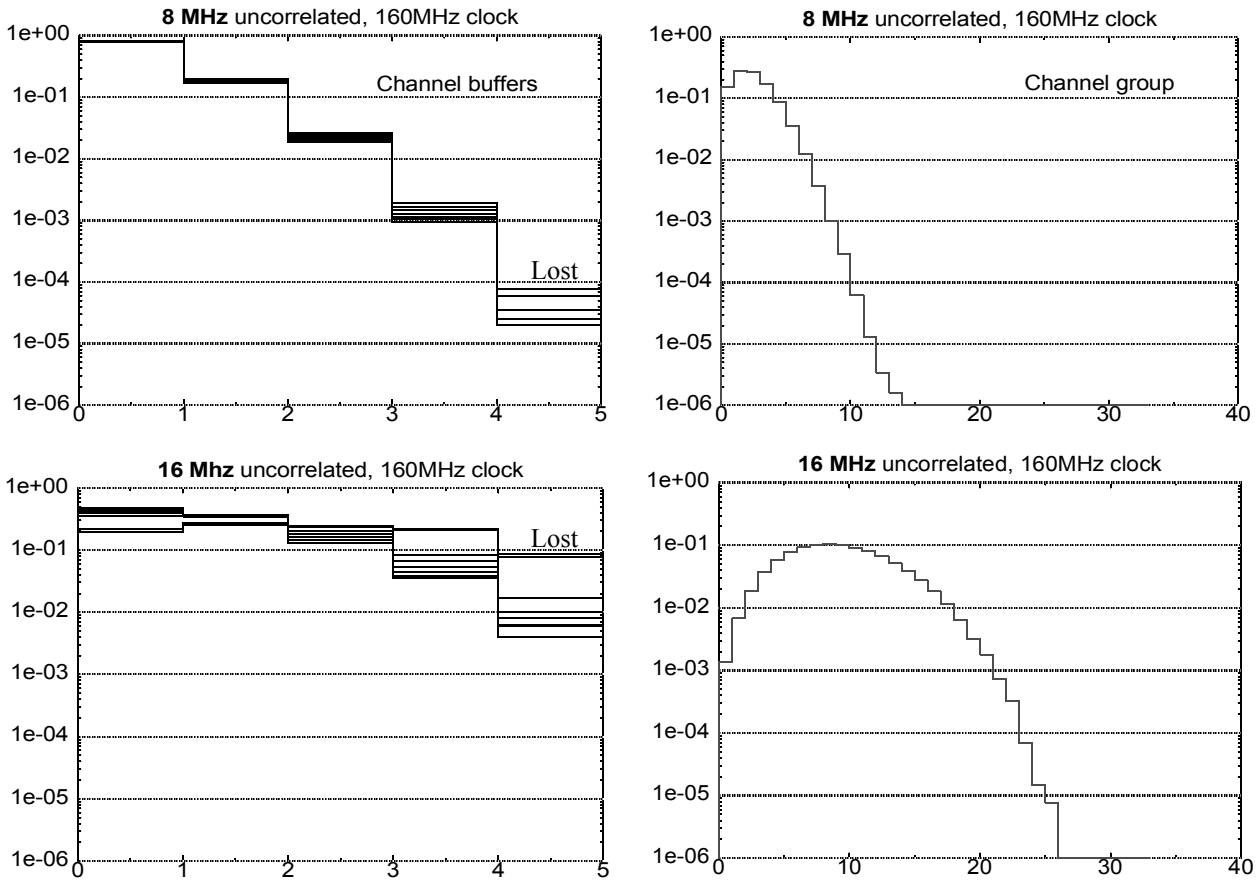


Fig. 62 Channel and group buffer occupancies at 160 MHz clock speed.

The effect on the L1 buffer occupancy of an increased clocking frequency is under normal conditions insignificant. The number of hits to be stored in the L1 buffer is determined by the hit rates and the trigger latency and is therefore normally independent of the clocking speed.

At increased clocking speed the trigger matching function can be performed faster which has a direct effect on the trigger fifo occupancy. In cases where the L1 buffer occupancy is influenced by limitations in the trigger matching (Fig. 56) the L1 buffer occupancy can also be decreased as shown in Fig. 63.

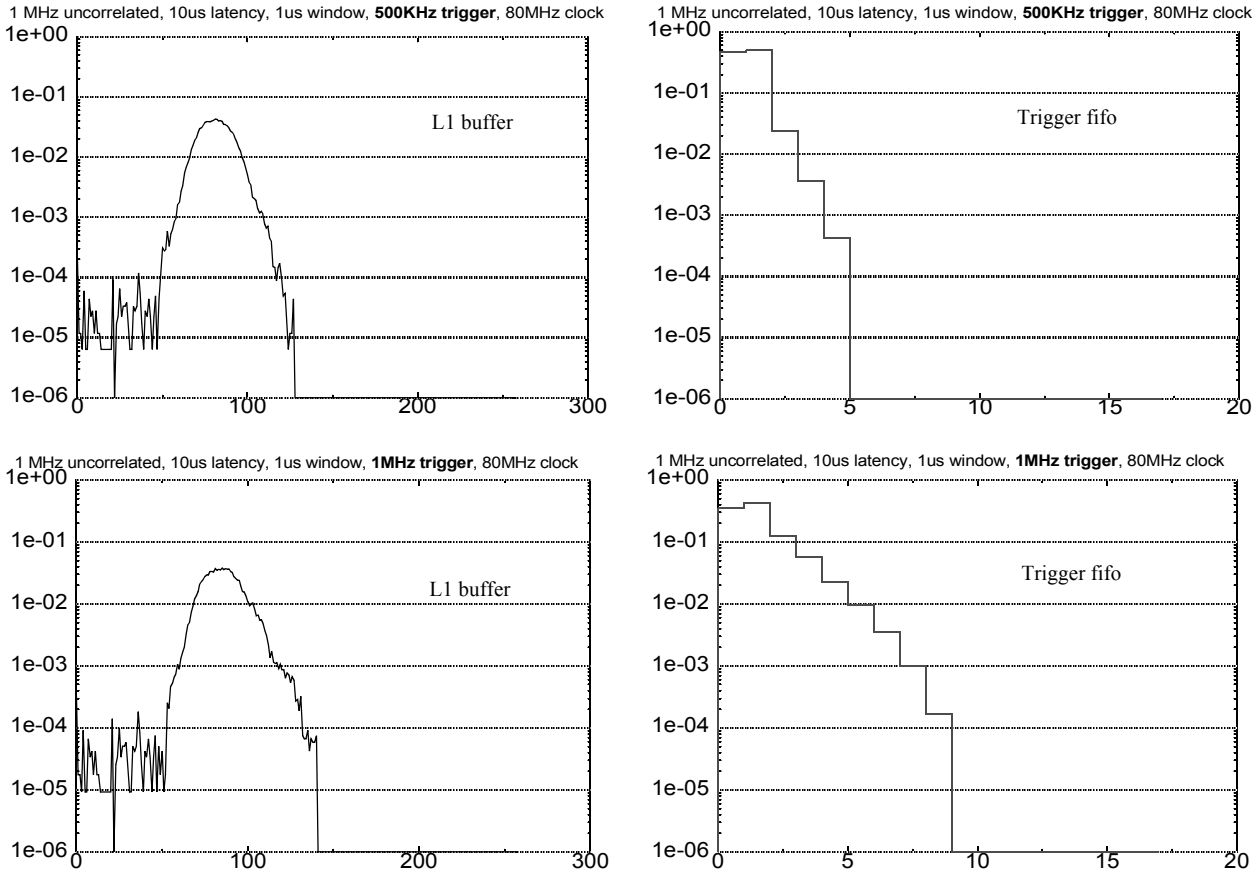


Fig. 63 Improvements of Trigger fifo and L1 buffer occupancies using 80MHz internal clocking (compared to Fig. 56 and Fig. 58)

31. References

[1]: J. Christiansen, C. Ljuslin, A. Marchioro, "an integrated 16 channel CMOS time to digital converter". Nuclear Science Symp. 1993, pp. 625 - 629.

[2]: J. Christiansen. "32 channel TDC with on-chip buffering and trigger matching", Third workshop on electronics for LHC experiments CERN/LHCC/97-60. pp. 333 - 337.

[3]: Y. Arai & J. Christiansen. "Requirements and specifications of the TDC for the ATLAS precision muon tracker", ATLAS MUON note 179

[4]: M. Mota & J. Christiansen. "A high resolution time interpolator based on a delay locked loop and an RC delay line". IEEE journal of solid-state circuits, Vol. 34, No. 10, October 1999.

32. Design bugs and problems

It is currently known that the HPTDC 1.3 has the following bugs/problems:

No PLL lock detect: The PLL lock detect is not implemented.

Serial read-out at 40 MHz and below: The strobe signal related to the serial read-out is specified to change at the same time as the serial data itself. At serialization speeds below 40 MHz the strobe signal changes value 25 ns after the serial data.

Group and TDC trailers: If a readout master TDC is programmed to generate both group and TDC trailers the word count in the group trailer will not count the TDC trailer from the master.

INL from 40 MHz logic: The INL in the high resolution and the very high resolution have a fixed pattern non linearity caused by crosstalk from the logic part of the chip running at 40 MHz. As the logic part of the chip runs with the same clock reference as the time measurements this fixed non linearity can be compensated for by a sample table look-up. Several attempts have been made to improve this by changing signal routing and powering scheme but only limited improvements have been obtained. It is belived that the problem comes from substrate coupling from the digital logic of the chip and this can only be significantly improved by a complete redesign of the digital logic using a digital library where groundbounce is not coupled directly to the substrate.

DNL at bin27: The DNL in the high resolution modes have a clear non linearity in timing bin27 plus multiples of 32. Timing bin 27 is generated from the end of the DLL to the beginning of the DLL and therefore includes possible timing errors from the phase detector, charge pump and the signal routing to the phase detector. Detailed simulations of the DLL with all its parasitics have not enabled a clear understanding of this problem. The problem has quite large variations between chips but a clear tendency to have bin27 too large is present. The problem is most significant in the region where the INL have also been seen to have problems from crosstalk from the logic clock and the DNL problem can possibly also be caused by the crosstalk effect. The problem can partly be compensated for by using the DLL tap adjustment feature: DLL_tap_ajust0 - 3 = 0, DLL_tap_ajust4 - 7 = 1, DLL_tap_ajust8 - 11 = 2, DLL_tap_ajust12 - 15 = 3, DLL_tap_ajust16 - 19 = 4, DLL_tap_ajust20 - 23 = 5, DLL_tap_ajust24 - 27 = 6, DLL_tap_ajust28 - 31 = 7.