TECHNICAL SPECIFICATION FLT TRACK PARAMETER UNIT V1.1

Changed 7.9.98: Geographical address register, Cache + Comparator
Changed 2.11.98: Signaturregister, LED description

This document describes the technical specification for the Track-Parameter-Unit (TPU) from the First Level Trigger (FLT) of the HERA-B experiment.

FEATURES OF THE TPU
*** TPU DESIGNED AS PIPELINED PROCESSOR WITH 50 MHz GLOBAL CLOCK**
*** DESIGN WITH ONLY 12 TPU PIPELINE STEPS TO REDUCE TRIGGER LATENCY**
*** DESIGN USING LUTs FOR HIGH FLEXIBILITY**
*** SERIAL TEST INTERFACE FOR ALL PIPELINE STAGES**
*** INTERFACE FOR READOUT OF TPU INPUT + OUTPUT MESSAGE DURING TPU RUN**
*** 21-STAGE COMPARATOR LOGIC TO SUPRESS DOUBLE TRACKS**
*** TRACK SUPPRESSION USING TRACK HISTOGRAM CACHE**
*** RANDOM GENERATOR + SIGNATUR REGISTER FOR TESTS DURING TPU RUN**
*** SERIAL INTERFACE FOR READOUT OF TPU INPUT + OUTPUT MESSAGE DURING TPU RUN**

*** CPU MC68020 25/33MHz FOR TPU CONTROL**
*** 4 MBYTE DUAL-PORTED-DRAM**
*** CPU VME MASTER INTERFACE**
*** DRAM VME SLAVE INTERFACE**
*** CPU INTERFACE TO MESSAGE BOARD**

TECHNICAL DATA CPU SUMMARY

CPU	: MC68020 (MOTOROLA), intern instruction cache
FPU (optional)	: MC68882 (MOTOROLA), floating point coprocessor
CPUCLK	: 25 MHz (33MHz) (only 25 MHz tested))
INTERN INSTR CACHE SIZE	: 256 BYTE

For detailed information see Motorola MC68020 user's manual

GLOBAL CPU ADDRESS SPACE:

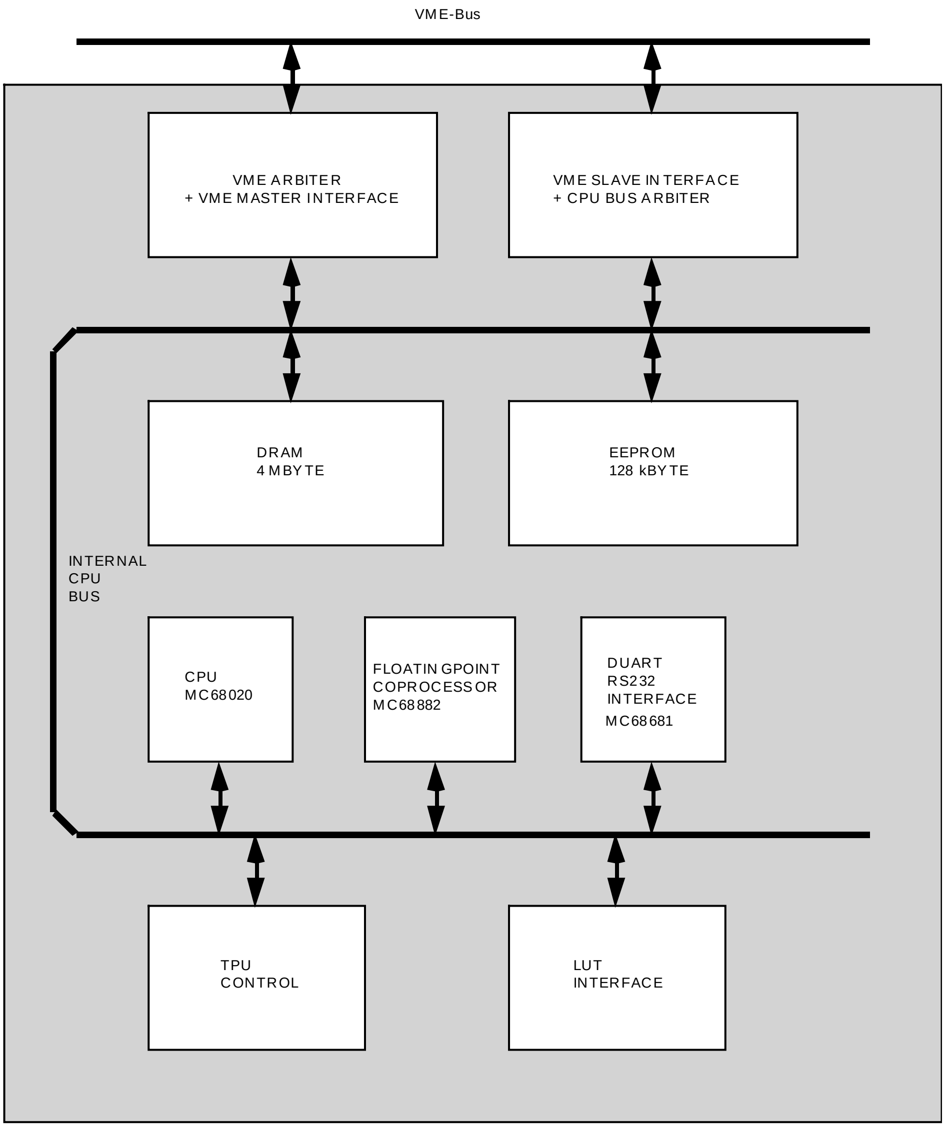
`00000000 - 003FFFFF`: DRAM (cacheable)
(For loading of SSP + PC after reset EEPROM address space until the first access to `$008XXXXX`)
`00400000 - 007FFFFF`: CSR SPACE (not cacheable)
`00800000 - 0081FFFF`: EEPROM (cacheable)
`00820000 - 00BFFFFF`: NOT USED (reserved for expanding of the EEPROM)
`00C00000 - 00FFFFFF`: LUT Interface (not cacheable)
`01000000 - 0FFFFFFF`: VME EXTENSION ADDRESS SPACE (not cacheable)

CPU PORT SIZE:

DRAM: 32 bit
DUART: 16 bit
CSR: 8 + 32 bit
EEPROM: 8 bit
LUT: 8 bit
VME: 32 bit

CPU INTERRUPT LEVEL CONTROL

INT7: INTERRUPT LEVEL7 FROM TERM CONTROL LOGIC, AUTOVECTOR INTERRUPT
INT6: INTERRUPT FROM VME SLAVE INTERFACE, AUTOVECTOR INTERRUPT
INT5: not used
INT4: INTERRUPT FROM DUART MC68681, AUTOVECTOR INTERRUPT
INT3: INTERRUPT FROM MESSAGE CONTROL, AUTOVECTOR INTERRUPT
INT2: not used
INT1: not used

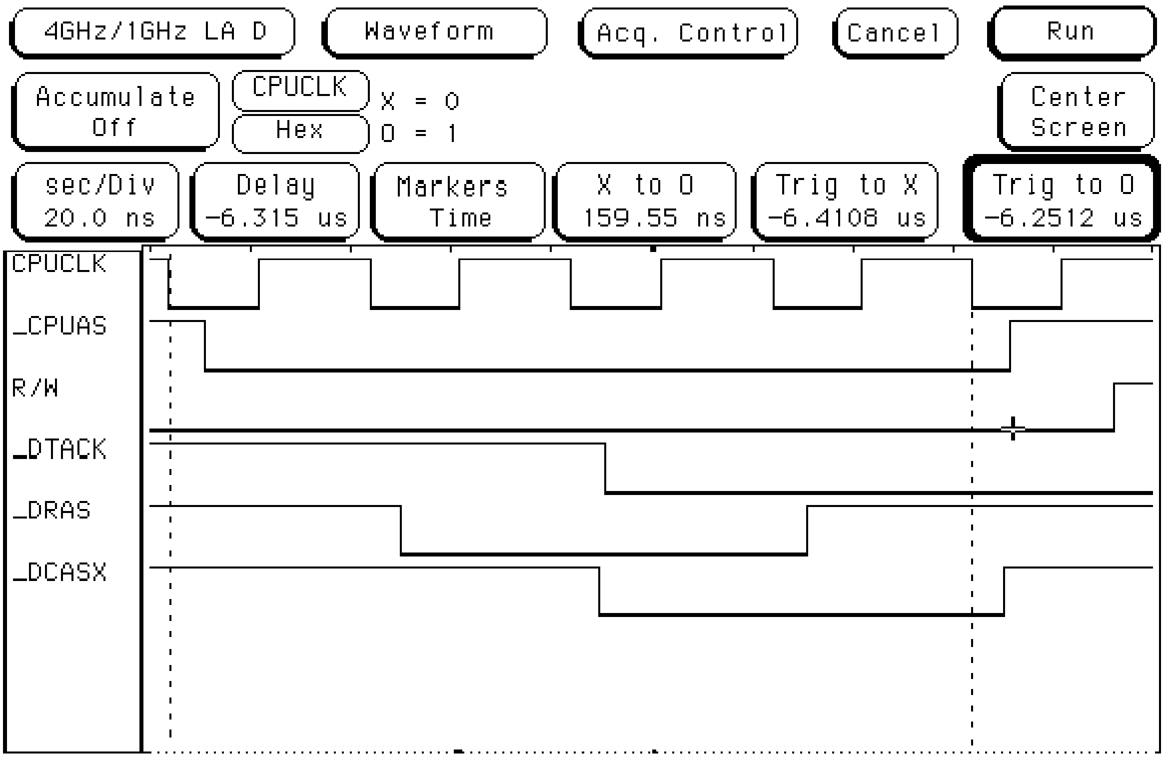


ON-BOARD CPU ORGANIZATION

DRAM TECHNICAL DATA SUMMARY

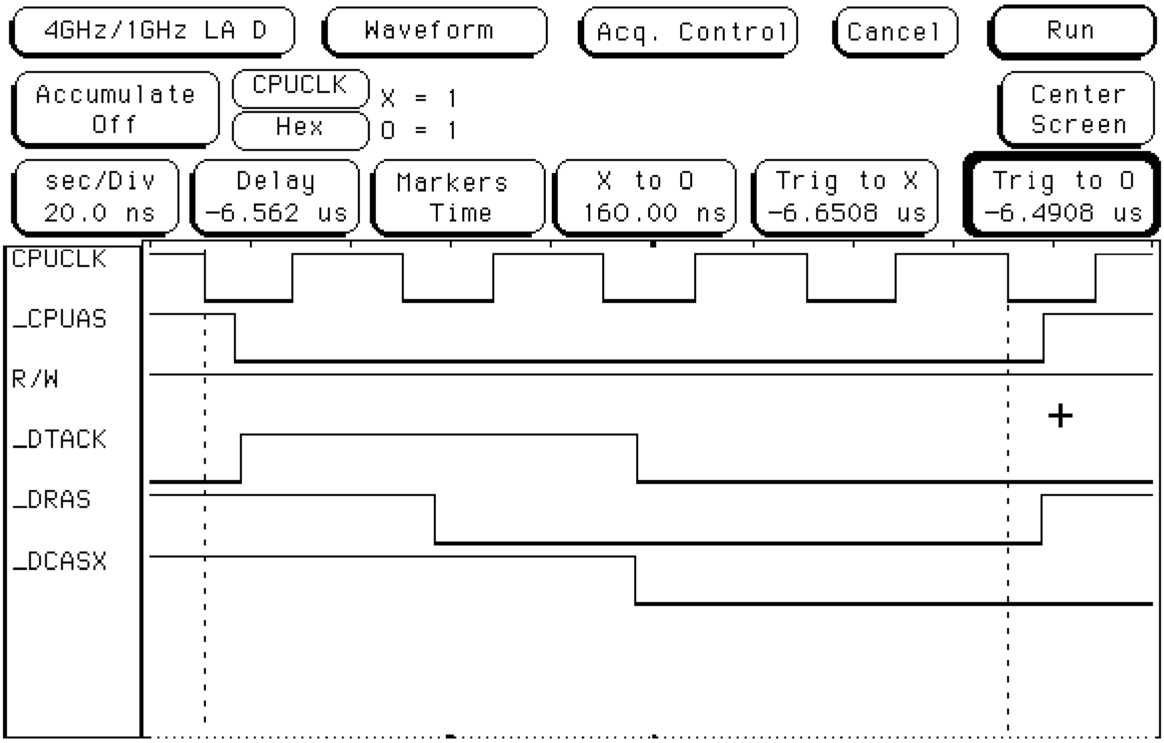
CPU ADDRESS SPACE: **00000000 - 003FFFFF**
(cacheable only for CPU-on-chip cache)
(For loading of SSP + PC after power-up EEPROM address space is mapped to the CPU address space until the first CPU access to $008XXXXX)
Realisation: 2 * 16 M-BIT DRAMS 1M *16 -60 ns == 4 MBYTE:
TRANSFER SIZE: BYTE, WORD, LWORD, TBYTE

After a reset, the CPU expects the supervisor stack pointer (SSP) and the program counter value (PC) and the addresses $00000000 and $00000004. After a power up the DRAM has no defined value at this addresses. Therefore the EEPROM address space ($008xxxxx) is mapped to the DRAM address space $000xxxxx. In this way it is possible for the CPU to read the supervisor stack pointer and the program counter from the EEPROM for a correct program start. After the first CPU access to the correct EEPROM address space the address decoding logic of the DRAM and the EEPROM is changed to the normal way.
To improve system performance, the DRAM is cacheable for the on-chip cache of the MC68020CPU. To avoid problems with the cache during loading and start of programs (no snoop controller available!), it is recommended to clear and enable the cache after the begin of the program and to disable(!) the cache at the end of the program.



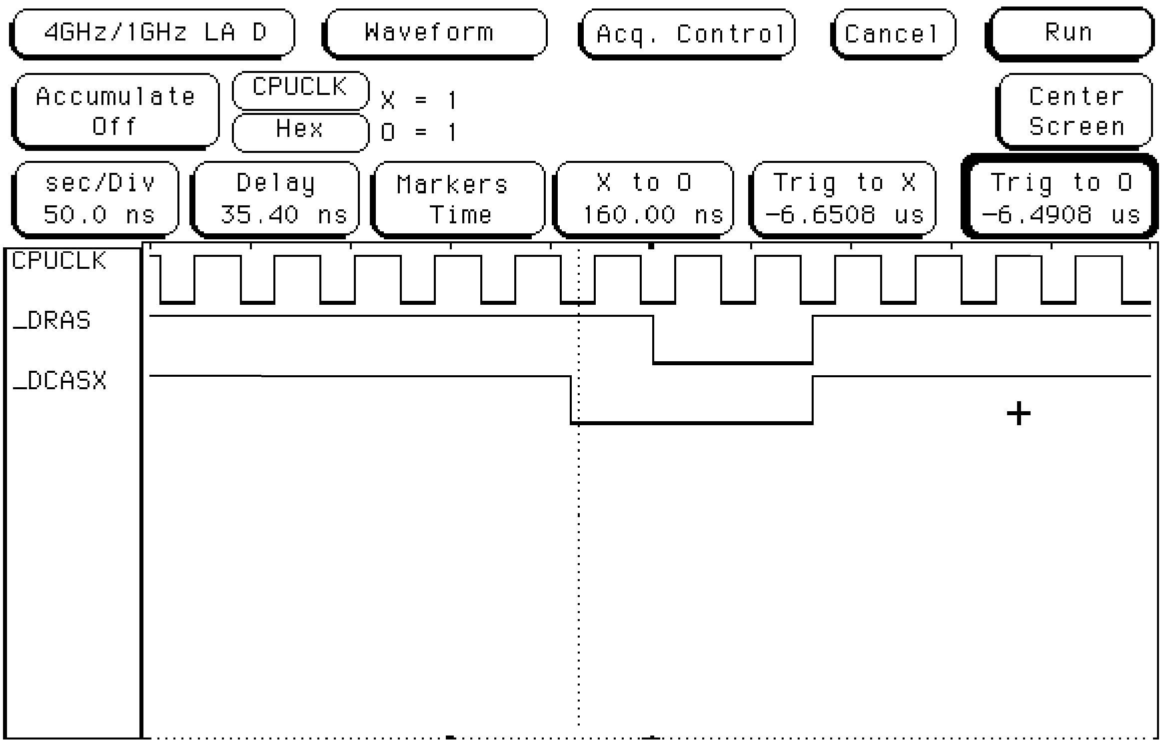
CPU write access to DRAM: Access time = 160ns = 4 clock cycles = 2 wait states

CPUCLK: CPU clock = 25 MHz
_CPUAS: Address strobe from CPU
R/W: Read-write CPU
_DTACK: Data acknowledge from DRAM CONTROL (CPUC1)
_DRAS: DRAM row address strobe
_DCASX: DRAM column address strobe



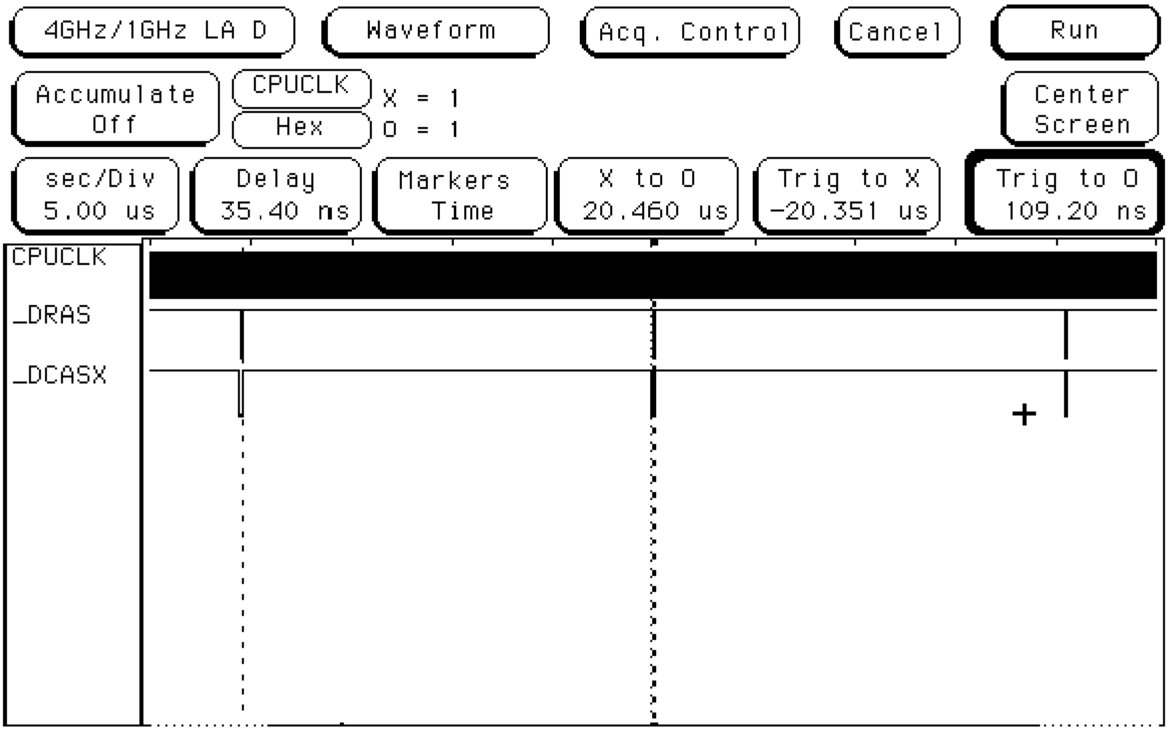
CPU read access to DRAM: Access time = 160ns = 4 clock cycles = 2 wait states

CPUCLK:	CPU clock = 25 MHz
_CPUAS:	Address strobe from CPU
R/W:	Read-write CPU
_DTACK:	Data acknowledge from DRAM CONTROL (CPUC1)
_DRAS:	DRAM row address strobe
_DCASX:	DRAM column address strobe



DRAM CAS-BEFORE-RAS REFRESH

CPUCLK:	CPU clock = 25 MHz
_DRAS:	DRAM row address strobe
_DCASX:	DRAM column address strobe



DRAM refresh cycle time: ≈ **20.5 µs**

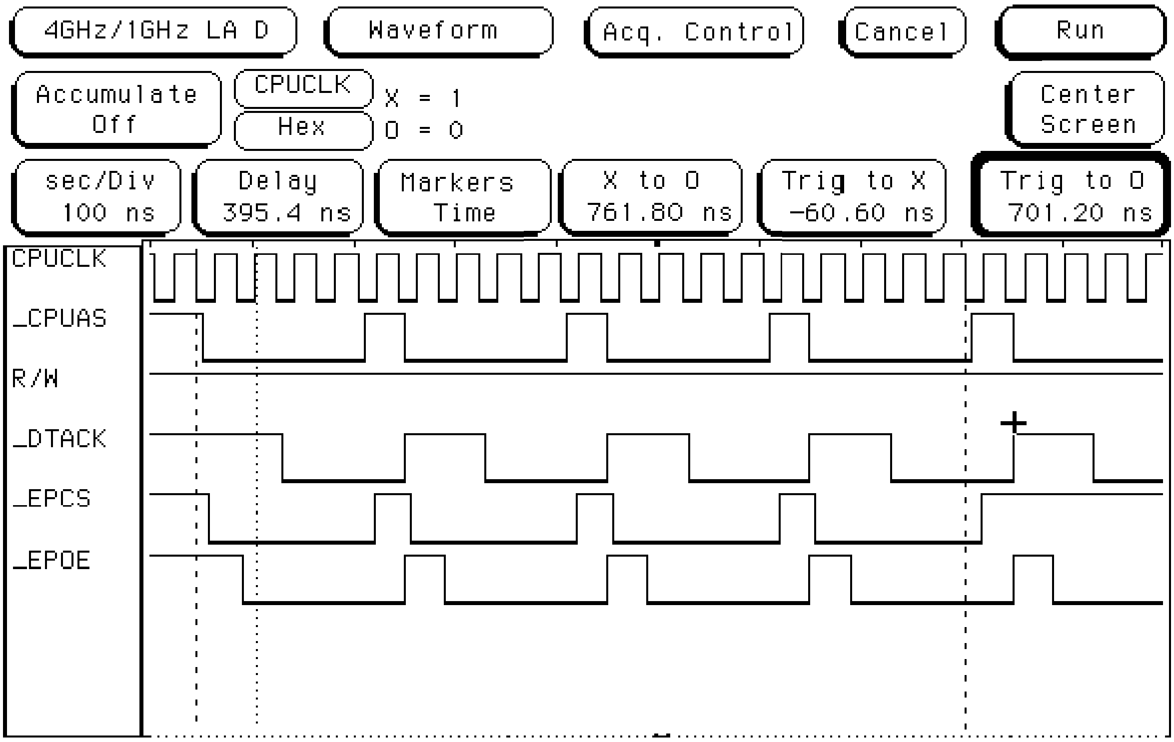
CPUCLK: CPU clock = 25 MHz
_DRAS: DRAM row address strobe
_DCASX: DRAM column address strobe

Comment: In the DRAM data sheet it is recommended, to choose a DRAM refresh cycle time of about 16 µs. The refresh cycle time is only critical near the upper limit of the DRAM temperature space (≈ 70 °C). Under normal temperature conditions a refresh cycle time of 20 µs is completely sufficient.

TECHNICAL DATA EPROM SUMMARY

CPU ADDRESS SPACE **:00800000 - 0083FFFF (cacheable for internal cache)**
REALISATION: 1* 1MBIT EEPROM 128k*8 -120ns == 128 kByte
TRANSFER-SIZE: BYTE,WORD,LWORD (only read transfers possible)
TRANSFER TIME:
(For details of the EEPROM command and programming definitions see databook AMD FLASH MEMORY PRODUCTS 1994/1995 page 1-11)

Note: The limit of erase/program cycles of the used EEPROM AM29F010 is 100.000 (typical 1.000.000). Therefore a testloop for EEPROM programming with many passes may cause permanent errors in the EEPROMs.



CPU longword read from EEPROM (4 byte transfers): cycle time ≈ 760 ns
CPU byte read from EEPROM: cycle time 160 ns

CPUCLK: CPU clock = 25 MHz
_CPUAS: Address strobe CPU bus from CPU or from VME SLAVE CONTROL
R/W: CPU read-write
_DTACK: Data acknowledge from EEPROM CONTROL (CPUC1)
_EPCS: EEPROM chip select
_EPOE: EEPROM output enable

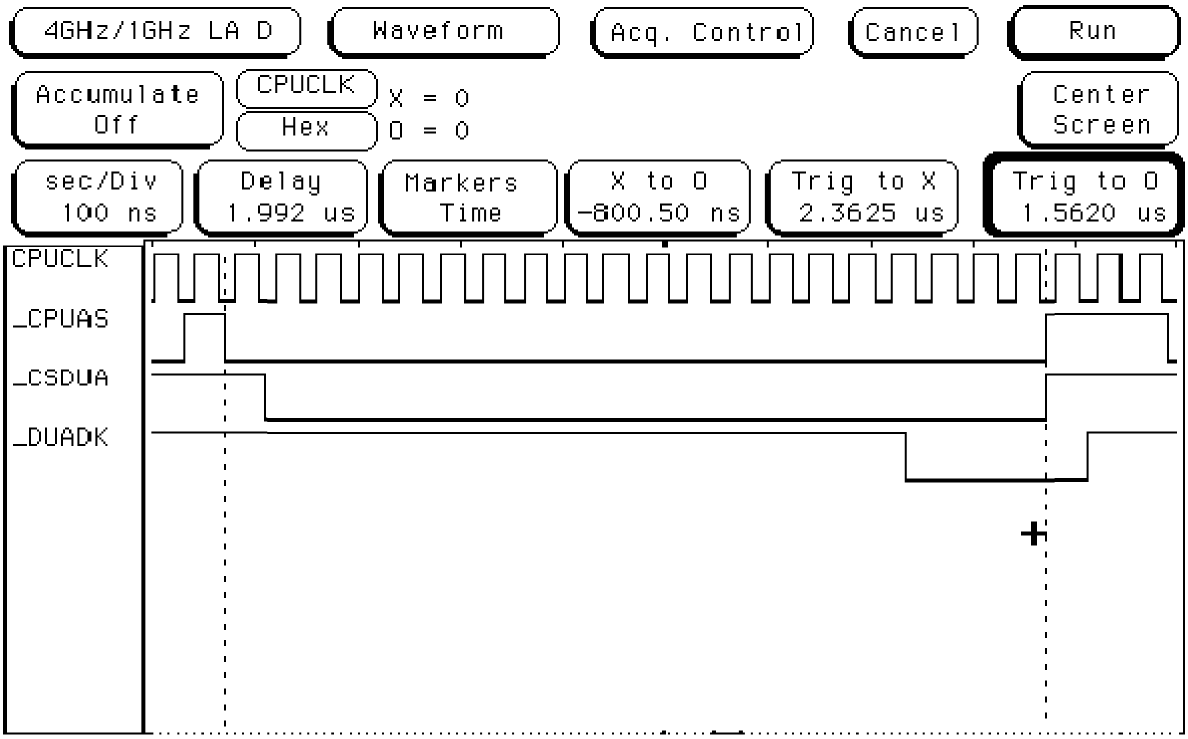
CSR ADDRESS SPACE

00400000 - 00401FFF	RESERVED (PIO)
00402000 - 00403FFF	DUART MC68681 (RS232)
00404003	VME INTERRUPT VECTOR REGISTER
00406040 – 00406080	MESSAGE BOARD
00408000 - 0040800C	CPU CONTROL REGISTER
0040A000 - 0040A005	TPU CONTROL REGISTER
0040A007	TPU CONTROL REGISTER
00410000 – 00410007	ADC1
00412000 – 00412007	ADC2
00414006	TPU COUNTER CONTROL
00414008	TPU MESSAGE COUNTER
00414009	TPU MESSAGE COUNTER
0041400A	TPU MESSAGE COUNTER
0041400B	TPU MESSAGE COUNTER
0041600D	CACHE CUT REG
0041800C	TPU STATUS (LED) REG

DUART MC68681 (RS232 Interface) Register

	Read	Write
00402001	Mode Reg A	Mode Reg.A
00402003	Status Reg A	Clock Select Reg A
00402005	Reserved	Command Register A
00402007	RX Holding Reg A	TX Holding Reg A
00402009	Inp. Port Change Reg.	Aux Control Reg
0040200B	Interrupt Status Reg.	Interrupt Mask Reg.
0040200D	Counter/Timer Upper	C/T Upper Reg.
0040200F	Counter/Timer Lower	C/T Lower Register
00402011	Mode Reg B	Mode Reg B
00402013	Status Reg B	Clock Select Reg B
00402015	Reserved	Command Register B
00402017	RX Holding Reg B	TX Holding Reg B
00402019	Interrupt Vector Reg	Interrupt Vector Reg
0040201B	Input Port	Output Port Config Reg
0040201D	Start Counter Command	Set Output-Port Bits Command
0040201F	Stop Counter Command	Res Output-Port Bits Command

For detailed information see Motorola MC68XXX family data book.



CPU access to DUART: Access time ≈ 2.4 µs

CPUCLK:	CPU clock = 25 MHz
_CPUAS:	Address strobe CPU bus from CPU or from VME SLAVE CONTROL
_CSDUA:	Chip-select DUART
_DUADK:	Data acknowledge DUART

VME INTERRUPT VECTOR REGISTER

`00404003` VME INTERRUPT VECTOR REGISTER
This register is used, to put an 8-Bit vector on the VME-Bus during interrupt acknowledge cycle. Only byte-write operation is possible.

MESSAGE BOARD INTERFACE

00406040	RECREAD CTRL
00406044	RECREAD INTERRUPT
00406050	RECEIVER1 CTRL
00406054	RECEIVER1 DATA
00406058	RECEIVER2 CTRL
0040605C	RECEIVER2 DATA
00406060	RECEIVER3 CTRL
00406064	RECEIVER3 DATA
00406068	RECEIVER4 CTRL
0040606C	RECEIVER5 DATA
00406070	TRANSMITTER CTRL
00406074	TRANSMITTER DATA
00406080	T_AEF LOAD

Note: Details see in message board description.

CPU CONTROL REGISTER FUNCTION

00408003 + Slave Address	RESET CONTROL
00408007 (+ Slave Address)	VME interrupt /IRQ7 control
0040800B + Slave Address	VME Slaveinterrupt
0040800F (+ Slave Address)	Watchdog Enable

RESET CONTROL
This bit is used for reset control of the microprocessor MC68020.
Write transfer with D0 = 1 sets the microprocessor in reset mode.
Write transfer with D0 = 0 sets the microprocessor in the normal operation mode.
Note: Do not activate the reset control from the intern microprocessor!

IRQ7 CONTROL
This bit is used to force an interrupt /IRQ7 on the VME bus.
Write transfer with D0 = 1 activates the interrupt /IRQ7 on the VME bus.
Write transfer with D0 = 0 disables the interrupt /IRQ7 on the VME bus.

VME SLAVEINTERRUPT
This bit is used to force an interrupt level 6 to the internal microprocessor MC68020.
Write transfer with D0 = 1 activates the interrupt.
Write transfer with D0 = 0 disables the interrupt.

WATCHDOG ENABLE (not tested)
This bit is used for enable/disable the Maxim watchdog timer MAX691.
This feature is used to recognize a system hang up and start a reinitialization of the system by the activation of a reset.
Write transfer with D0 = 1 enables the watchdog timer. The watchdog timer checks the transitions of the data line D0. If the watchdog timer observes no change during a time of about one second, it initiates the CPU reset operation.
Write transfer with D0 = 0 disables the watchdog timer. After power-up the watchdog timer is disabled.

TPU GEOGRAPHICAL ADDRESS REGISTER

00408002 Geographical address register (read only)

[1 ,/IRQ1, /IRQ2, /IRQ3, /IRQ4, /IRQ5, /IRQ6, /IRQ7]
[D7, D6 , D5 , D4 , D3 , D2 , D1 , D0]

This register is used to identifier the geographical address of the TPU board. Only byte read operations are allowed.

Example:
```
GEO_REG   EQU $408002
          MOVE.B GEO_REG,D0     ;MOVE GEOGRAPHICAL ADDRESS TO D0
```

VME SLAVE INTERFACE

ADDRESS SPACE DRAM: `00000000 - 003FFFFF` + Slave Address Offset (4 MByte)
TRANSFER SIZE: BYTE, WORD, LWORD
READ-MODIFY-WRITE transfers implemented (not tested)

Only for tests:
ADDRESS SPACE CSR: `00400000 - 007FFFFF` + Slave Address Offset (4 MByte)
ADDRESS SPACE EEPROM: `00800000 - 00BFFFFF` + Slave Address Offset (4 MByte)
ADDRESS SPACE LUT: `00C00000 - 00FFFFFF` + Slave Address Offset (4 MByte)

VME address modifier decode:

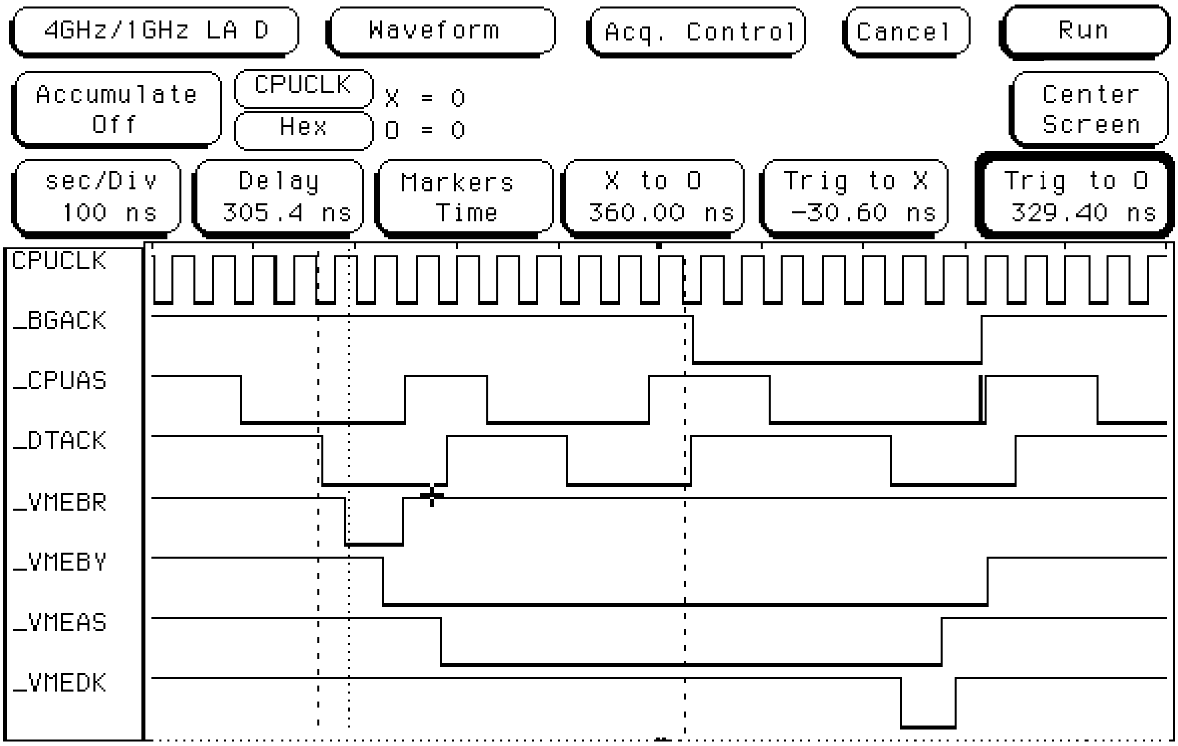
```
VSLSEL = (!VAM5&!VAM4&VAM3&
        ((!VAM2&!VAM1& VAM0)#
         ( VAM2&!VAM1& VAM0)#
         ( VAM2& VAM1&!VAM0)#
         (!VAM2& VAM1&!VAM0)));
```

Geographical slave address decode:

```
VME_ADR = [VA29,VA28,VA27,VA26,VA25,VA24];
VME_IRQ = [!VME_IRQ6,!VME_IRQ5,!VME_IRQ4,!VME_IRQ3,!VME_IRQ2,!VME_IRQ1]
VSLSEL = (VME_ADR == VME_IRQ) & ([VA31,VA30] == SLAVEADR);
SLAVEADR == ^B10;
```

Example: `VME_IRQ = 32` -> slave address offset = `$B2000000`
Example for VME slave access from another VME master:
Langword read transfer from memory address `$00100000` to CPU register D0:

```
  MOVE.L  $00100000 + VME_SLAVE_ADDRESS,D0
```



DRAM read from extern VME master

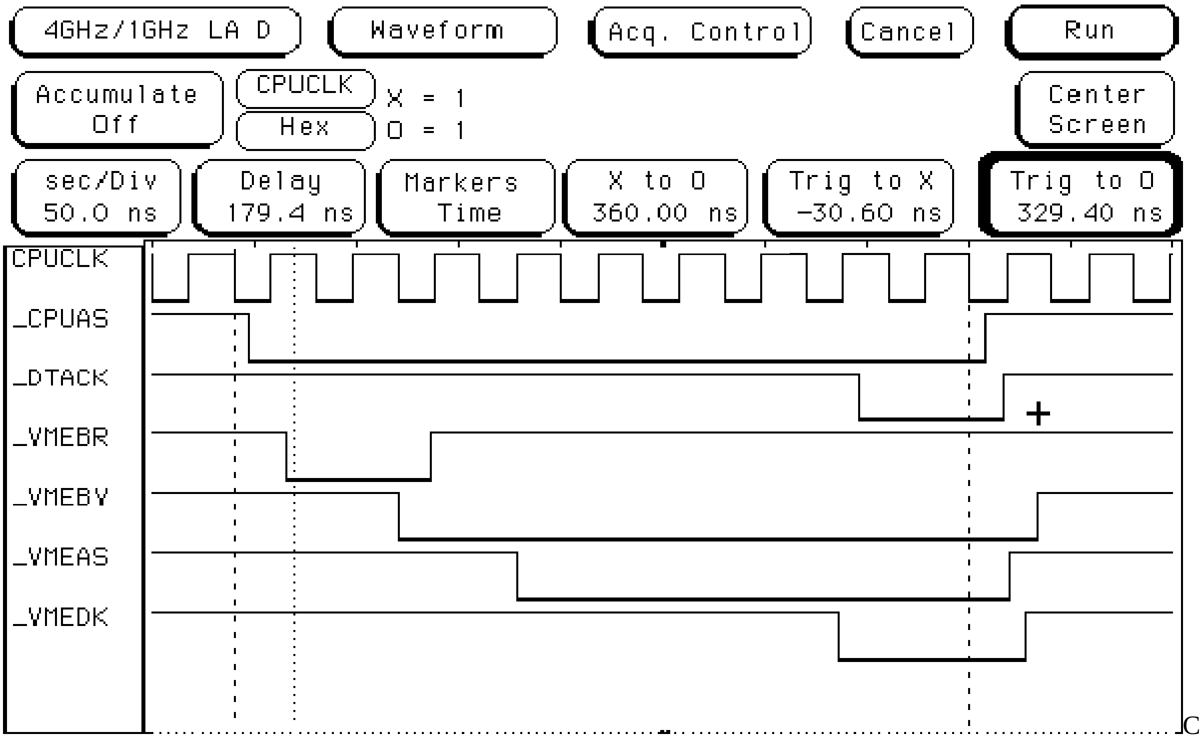
CPUCLK: CPU clock = 25 MHz
_CBACK: CPU bus acknowledge from CPU CONTROL PAL (CPUC1)
_CPUAS: Address strobe CPU bus from CPU or from VME SLAVE CONTROL
_DTACK: Data acknowledge from DRAM CONTROL (CPUC1)
_VMEBR: VME BUS REQUEST
_VMEBY: VME BUS BUSY
_VMEAS: VME address strobe
_VMEDK: VME data acknowledge

VME MASTER INTERFACE

CPU ADDRESS SPACE :04000000 - FFFFFFFF (not cacheable)
TRANSFER SIZE: BYTE, WORD, LWORD
ADDRESS-MODIFIER: SUPDAT, SUPPRO, USER DATA
VME-ARBITER ON BOARD: ACTIVATION WITH JUMPER
JUMPER SET: VME ARBITER ENABLED
JUMPER NOT SET: VME ARBITER DISABLED
ARBITRATION LEVEL 3 USED: BG3, BR3

FUNCTION CODE ADDRESS_MODIFIER VME
CPU 68020

```
([CFC2,CFC1,CFC0]->[VAM5,VAM4,VAM3,VAM2,VAM1,VAM0]);
 [ 0  , 0  , 1  ]->[ 0  , 0  , 1  , 0  , 0  , 1  ]; "EXTENDED NON-PRIVILEGED DATA ACCESS
 [ 1  , 0  , 1  ]->[ 0  , 0  , 1  , 1  , 0  , 1  ]; "EXTENDED SUPERVISOR DATA ACCESS
 [ 1  , 1  , 0  ]->[ 0  , 0  , 1  , 1  , 1  , 0  ]; "EXTENDED SUPERVISOR PROGRAM ACCESS
 [ 0  , 0  , 0  ]->[ 1  , 1  , 1  , 1  , 1  , 1  ]; "INVALID ACCESS
 [ 0  , 1  , 0  ]->[ 1  , 1  , 1  , 1  , 1  , 1  ]; "INVALID ACCESS
 [ 0  , 1  , 1  ]->[ 1  , 1  , 1  , 1  , 1  , 1  ]; "INVALID ACCESS
 [ 1  , 0  , 0  ]->[ 1  , 1  , 1  , 1  , 1  , 1  ]; "INVALID ACCESS
 [ 1  , 1  , 1  ]->[ 1  , 1  , 1  , 1  , 1  , 1  ]; "INVALID ACCESS
```



CPU read transfer to DRAM of FLT-TEST-CPU via VME: Access time = 360 ns

CPUCLK: CPU clock = 25 MHz
_CPUAS: Address strobe CPU bus from CPU or from VME SLAVE CONTROL
_DTACK: Data acknowledge from DRAM CONTROL (CPUC1)
_VMEBR: VME BUS REQUEST
_VMEBY: VME BUS BUSY
_VMEAS: VME address strobe
_VMEDK: VME data acknowledge

TPU CONTROL REGISTER

`0040A000:`   TPU CONTROL (global control of TPU pipeline)

```
[CLEARVALID,TIMTEST,CNTEN,MRES,SHIFT,TEST,RUNSINGLE,RUN]
[    D7     ,   D6   ,   D5 , D4 ,   D3 , D2 ,    D1    , D0]
```

CLEARVALID:
This bit is used to clear all data valid bits in the TPU pipelines.
Write transfer with D7 = 1: All valid bits are cleared
Write transfer with D7 = 0: Disable CLEARVALID
Note: This bit is necessary for TPU pipeline reset after power up or for reinitialization. During normal TPU pipeline operation CLEARVALID must be disabled.

TIMTEST:
This bit is used for hardware timing tests (eye-diagrams of TPU pipelines).
Write transfer with D6 = 1: Enable timing testmode.
Write transfer with D6 = 0: Disable timing testmode
Note: During normal TPU pipeline operation TIMTEST must be disabled.

CNTEN:
This bit is used to enable/disable the message counter.
Write transfer with D5 = 1: Enable message count.
Write transfer with D5 = 0: Disable message count (counter hold).

MRES:
This bit is used for reset of the message interface.
Write transfer with D4 = 1: Set message board reset.
Write transfer with D4 = 0: Disable message board reset
Note: During normal TPU pipeline operation MRES must be disabled.

SHIFT
This bit is used to perform a pipeline shift operation (one byte)
Write transfer with D3 = 1: Enable pipeline shift for 8 clock cycles.
Write transfer with D3 = 0: No shift operation.
Note: Do not set RUN and CLEARVALID during shift operation.
Pipeline select with SHIFT PIPELINE LEVEL REGISTER

TEST
This bit is used to trigger the transfer of test data through the TPU pipeline
Write transfer with D2 = 1: Send test message through TPU pipeline to signature register (not to message output!).
Write transfer with D2 = 0: No action

RUNSINGLE
This bit is used for single step TPU pipeline operation
Write transfer with D1 = 1: Enable TPU pipeline for N + 1 clock cycles
(N = clock cycle counter)
Write transfer with D1 = 0: No action

RUN
This bit is used for RUN/STOP control of TPU pipeline operation
Write transfer with D0 = 1: Normal TPU pipeline operation
Write transfer with D0 = 0: Stop TPU pipeline and stall all pipeline registers.

0040A001: COMPARATOR + HISTOGRAM CACHE CONTROL

```
[ X , X , X , X ,CACHERES,TIMEOUTEN,COMPRES,COMPEN]
[ D7, D6, D5, D4,   D3   ,    D2   ,   D1  ,  D0  ]
```

CACHERESET
This bit is used to clear all histogram cache entries.
Write transfer with D3 = 1: All entries are cleared; cache disabled
Write transfer with D3 = 0: Cache enabled
Note: Use this signal for histogram cache initialization

TIMEOUTEN
This bit is used to enable/disable timeout clear for old histogram cache entries.
Write transfer with D2 = 1: Old entries are cleared after timeout
Write transfer with D2 = 0: Timeout clear disabled

COMPRES
This bit is used to clear all track comparator entries.
Write transfer with D3 = 1: Clear all comparator entries; comparator disabled
Write transfer with D3 = 0: Disable clear
Note: Use this signal for comparator initialization

COMPEN
This bit is used to enable/disable the comparator for double track recognition
Write transfer with D2 = 1: Comparator enabled
Write transfer with D2 = 0: Comparator disabled

0040A002: LUTSEL (LUT SELECT REGISTER)

```
[ X , X ,LUTSEL5,LUTSEL4,LUTSEL3,LUTSEL2,LUTSEL1,LUTSEL0]
[ D7, D6,   D5  ,   D4  ,   D3  ,  D2   ,   D1  ,  D0   ]
```

This register is used to select a LUT for the CPU access.
Example: Select LUT3 -> `MOVE.B #3,LUTSEL`

0040A003: TPU DATA FLOW CONTROL: RND GENERATOR + SIGNATUR REGISTER

```
[SIG_VALID,SIG_RES,SIG_SHIFT,RNDINIT,RNDSHFT,RNDREAD,   X   ,  X    ]
[    D7   ,   D6  ,    D5   ,   D4  ,   D3  ,  D2   ,   D1  ,  D0   ]
```

SIG_VALID
This bit is used as a indicator for a valid output message in the signature register (read only)
Read transfer with D7 = 1: No message valid
Read transfer with D7 = 0: Message valid for readout

SIG_RES + SIG_SHIFT
These Bits are used for signature register operation
Write `!D6&!D5`: Load signature register with a message. This operation is used for message readout (CPU check!) during TPU run.
Write `D6&!D5`: Reset signature register
Write `!D6& D5`: Serial shift of signature register for readout
Write `D6& D5`: No operation of signature register

RNDINIT
This bit is used for the initialization of the random generator in the first TPU pipeline stage (P1)
(write only)
Write transfer with D4 = 1: Initialization of the random generator
Write transfer with D4 = 0: No operation

RNDSHFT
This bit is used for serial shift of the random generator registers in the first TPU pipeline stage (P1). This operation is used for message readout of the first TPU pipeline stage during run (CPU check!) (write only)
Write transfer with D3 = 1: Shift random generator registers 8 clock cycles
Write transfer with D3 = 0: No operation

RNDREAD
This bit is used to load the random generator registers in the first TPU pipeline stage (P1) with the input message. This operation is used for message readout of the first TPU pipeline stage during run (CPU check!)
(write only)
Write transfer with D2 = 1: Load random generator register with the next input message
Write transfer with D2 = 0: No operation

0040A004: PIPESEL (PIPELINE SELECT REGISTER)

```
[ X , X , X ,SPLSCD4,SPLSCD3,SPLSCD2,SPLSCD1,SPLSD0]
[ D7, D6, D5,   D4   ,   D3  ,  D2   ,   D1  ,  D0  ]
```

This register is used to select the pipeline stage for serial shift.
Example: Select P7 -> `MOVE.B #7,PIPESEL`

0040A005: SFDATA (SHIFT DATA REGISTER)

```
[SCD7,SCD6,SCD5,SCD4,SCD3,SCD2,SCD1,SCD0]
[ D7 , D6 , D5 , D4 , D3 , D2 , D1 , D0 ]
```

This register is used as an interface register for serial shift operations. During serial shift operations this register appears as an additional 8-bit register, connected with the serial data stream of the selected pipeline stage. During CPU read/write access this registers is connected as an 8-bit parallel port to the CPU.

00414006: COUNTCONTROL (CONTROL OF MESSAGE COUNTERS)

```
[CACHECNTRES,COMPCNTRES,MINCNTRES,TPUCNTRES,LDCACHECNT,LDCOMPCNT,LDMINCNT,LDTPUCNT]
[     D7    ,    D6    ,    D5   ,    D4   ,    D3    ,    D2   ,    D1  ,   D0   ]
```

CACHECOUNTRES
This bit is used for the initialization (reset) of the cache hit counter
(write only). The cache hit counter is a 32-bit counter.
Write transfer with D7 = 1: Initialization of the counter
Write transfer with D7 = 0: No operation

COMPCOUNTRES
This bit is used for the initialization (reset) of the double track comparator hit counter (write only). The comparator hit counter is a 32-bit counter.
Write transfer with D6 = 1: Initialization of the counter
Write transfer with D6 = 0: No operation

MINCOUNTRES
This bit is used for the initialization (reset) of the Message input counter (write only). The message input counter is a 32-bit counter.
Write transfer with D5 = 1: Initialization of the counter
Write transfer with D5 = 0: No operation

TPUCOUNTRES
This bit is used for the initialization (reset) of the TPU processor hit counter(write only). The TPU hit counter is a 32-bit counter.
Write transfer with D4 = 1: Initialization of the counter
Write transfer with D4 = 0: No operation

LDCACHECOUNT
This bit is used for loading of the histogram cache hit counter to the count buffer.(for CPU readout, write only)
Write transfer with D3 = 1: Load counter to buffer register
Write transfer with D3 = 0: No operation

LDCOMPCOUNT
This bit is used for loading of the double track comparator hit counter to the count buffer.(for CPU readout, write only)
Write transfer with D2 = 1: Load counter to buffer register
Write transfer with D2 = 0: No operation

LDMINCOUNT
This bit is used for loading of the TPU message input counter to the count buffer.(for CPU readout, write only)
Write transfer with D1 = 1: Load counter to buffer register
Write transfer with D1 = 0: No operation

LDTPUCOUNT
This bit is used for loading of the TPU hit counter to the count buffer.(for CPU readout, write only)
Write transfer with D0 = 1: Load counter to buffer register
Write transfer with D0 = 0: No operation

MESSAGE COUNTER BUFFER REGISTER
00414008: COUNTREG0 (MESSAGE COUNTER BUFFER REGISTER)
```
[CNT7,CNT6,CNT5,CNT4,CNT3,CNT2,CNT1,CNT0]
[ D7 , D6 , D5 , D4 , D3 , D2 , D1 , D0 ]
```
00414009: COUNTREG1 (MESSAGE COUNTER BUFFER REGISTER)
```
[CNT15,CNT14,CNT13,CNT12,CNT11,CNT10,CNT9,CNT8]
[  D7 ,  D6 ,  D5 ,  D4 ,  D3 ,  D2 , D1 , D0 ]
```
0041400A: COUNTREG2 (MESSAGE COUNTER BUFFER REGISTER)
```
[CNT23,CNT22,CNT21,CNT20,CNT19,CNT18,CNT17,CNT16]
[  D7 ,  D6 ,  D5 ,  D4 ,  D3 ,  D2 , D1  , D0  ]
```
0041400B: COUNTREG3 (MESSAGE COUNTER BUFFER REGISTER)
```
[CNT31,CNT30,CNT29,CNT28,CNT27,CNT26,CNT25,CNT24]
[  D7 ,  D6 ,  D5 ,  D4 ,  D3 ,  D2 , D1  , D0  ]
```

This registers are used for CPU readout of the message counters (read only).
Direct access of the CPU to the message counters is not convenient, since the CPU needs several clock cycles (and possibly several accesses) for a counter readout. The load operation of the message counter buffer needs only one clock cycle.
Example0: Readout of the histogram cache counter.
```
      MOVE.B #$04,COUNTCONTROL  ;load cache counter to buffer register
      MOVE.L COUNTREG3,D0       ;move buffer register to D0
      ROL.L #8,D0               ;shift byte
      MOVE.L COUNTREG2,D0       ;move buffer register to D0
      ROL.L #8,D0               ;shift byte
      MOVE.L COUNTREG1,D0       ;move buffer register to D0
      ROL.L #8,D0               ;shift byte
      MOVE.L COUNTREG0,D0       ;move buffer register to D0
```

Example1: Readout of message in counter.
```
      MOVE.B #$02,COUNTCONTROL  ;load cache counter to buffer register
      MOVE.L COUNTREG3,D0       ;move buffer register to D0
      ROL.L #8,D0               ;shift byte
      MOVE.L COUNTREG2,D0       ;move buffer register to D0
      ROL.L #8,D0               ;shift byte
      MOVE.L COUNTREG1,D0       ;move buffer register to D0
      ROL.L #8,D0               ;shift byte
      MOVE.L COUNTREG0,D0       ;move buffer register to D0
```

0041800C: LED_READBACK (LED READBACK REGISTER)

```
[X,X,X,_DOUBLE_TRACK,_CACHE_HIT,_TPU_MESSAGE_OUT,_TPU_MESSAGE_IN,_TPURUN]
[D7,D6,D5,    D4    ,     D3   ,      D2        ,      D1       ,   D0  ]
```

_DOUBLE_TRACK (not tested)
This bit is used as indicator for double tracks (read only).
Read transfer with D4 = 1: No double tracks found in the last 100 ms
Read transfer with D4 = 0: Double tracks found in the last 100 ms

_CACHE_HIT (not tested)
This bit is used as indicator for histogram cache hits (read only).
Read transfer with D3 = 1: No tracks suppression from cache in the last 100 ms
Read transfer with D3 = 0: Track suppression found in the last 100 ms

_TPU_MESSAGE_OUT (not tested)
This bit is used as indicator for output messages to the TDU (read only).
Read transfer with D2 = 1: No messages to TDU in the last 100 ms
Read transfer with D2 = 0: The TPU sends messages to TDU in the last 100 ms

_TPU_MESSAGE_IN (not tested)
This bit is used as indicator for input messages from the TFUs (read only).
Read transfer with D1 = 1: No messages from TFUs in the last 100 ms
Read transfer with D1 = 0: The TFU sends messages to TPU in the last 100 ms

_TPURUN (not tested)
This bit is used as indicator for TPU run (read only). Indicator is a monoflop with a pulse with of 100 ms.
Read transfer with D0 = 1: TPU is not running
Read transfer with D0 = 0: The TPU runs

0041600D: CUT_REGISTER (Histogram cache cut register)

```
[X ,X ,X ,CUTREG4,CUTREG3,CUTREG2,CUTREG1,CUTREG0]
[D7,D6,D5,   D4   ,   D3   ,   D2   ,   D1   ,   D0   ]
```

This register is used to define the cut limit for the number of tracks suppressed by the BX histogram cache.
Example: Write $10 to cut register

```
MOVE.B #$10,CUT_REGISTER
```

0040A00E: RCNT (Run cycle counter) (write only)

```
[RCNT7,RCNT6,RCNT5,RCNT4,RCNT3,RCNT2,RCNT1,RCNT0]
[ D7  , D6  , D5  , D4  , D3  , D2  , D1  , D0  ]
```

This counter is used to define the number of clock cycles for TPU pipeline operations during the RUNSINGLE instruction. The counter can be set from the CPU by applying the desired value in the RCNT register. If the RCNT register is programmed with N, the TPU runs N+1 clock cycles. During RUNSINGLE instruction the RCNT register is operating as a down counter. The stage of the counter changes on the positive transition of the TPU pipeline clock. If the value of the RCNT register is zero, the RUNSINGLE bit is cleared.

Example: Write 22 to RCNT register

```
MOVE.B #22,RCNT            ;load 22 to RCNT
```

-> The TPU runs 22+1 = 23 clock cycles during the next RUNSINGLE operation
Note: For single step TPU pipeline operations, clear the RCNT register (N=0). During RUNSINGLE operations, the TPU operates N+1 = 1 clock cycles.

0040A00F: RCNT STATUS (status of run cycle counter) (read only)

```
[RCNTZERO, X   , X   , X   , X   , X   , X   , X   ]
[ D7     , D6  , D5  , D4  , D3  , D2  , D1  , D0  ]
```

This bit is used to check the status of the RCNT register.
Read transfer with D7 = 1: RCNT == 0
Read transfer with D7 = 0: RCNT != 0

Note: It is possible (only for testing) to access the TPU control registers via VME slave interface.
Address space: TPU_CONTROL_REGISTER ADDRESS + VME SLAVE ADDRESS
Use byte transfers for even address registers and word transfers for odd address registers
Byte transfers to even address:

```
MOVE.B #DATA,TPU_CONTROL_REGISTER + VME_SLAVE_ADDRESS
DATA = [DATA7..0]
```

Byte transfers to odd address:

```
MOVE.W #DATA,(TPU_CONTROL_REGISTER - 1) + VME_SLAVE_ADDRESS
DATA = [DATA7..0,X,X,X,X,X,X,X,X]
```

It is not recommended to use this feature for applications.

VOLTAGE + TEMPERATURE CONTROL

`00410000`: ADC1 channel 0: 5V control (TTL power)
`00410001`: ADC1 channel 1: 5V control (TTL power)
`00410002`: ADC1 channel 2: 5V control (TTL power)
`00410003`: ADC1 channel 3: 5V control (TTL power)
`00410004`: ADC1 channel 4: 5V control (TTL power)
`00410005`: ADC1 channel 5: 5V control (TTL power)
`00410006`: ADC1 channel 6: 5V control (TTL power)
`00410007`: ADC1 channel 7: 3V control (termination voltage for PECL)
`00412000`: ADC2 channel 0: not used
`00412001`: ADC2 channel 1: 2.2V control (termination voltage for TTL)
`00412002`: ADC2 channel 2: Temperature control (lower back)
`00412003`: ADC2 channel 3: Temperature control (middle back)
`00412004`: ADC2 channel 4: Temperature control (upper front)
`00412005`: ADC2 channel 5: Temperature control (middle front)
`00412006`: ADC2 channel 6: Temperature control (lower front)
`00412007`: ADC2 channel 7: Temperature control (upper back (CPU))

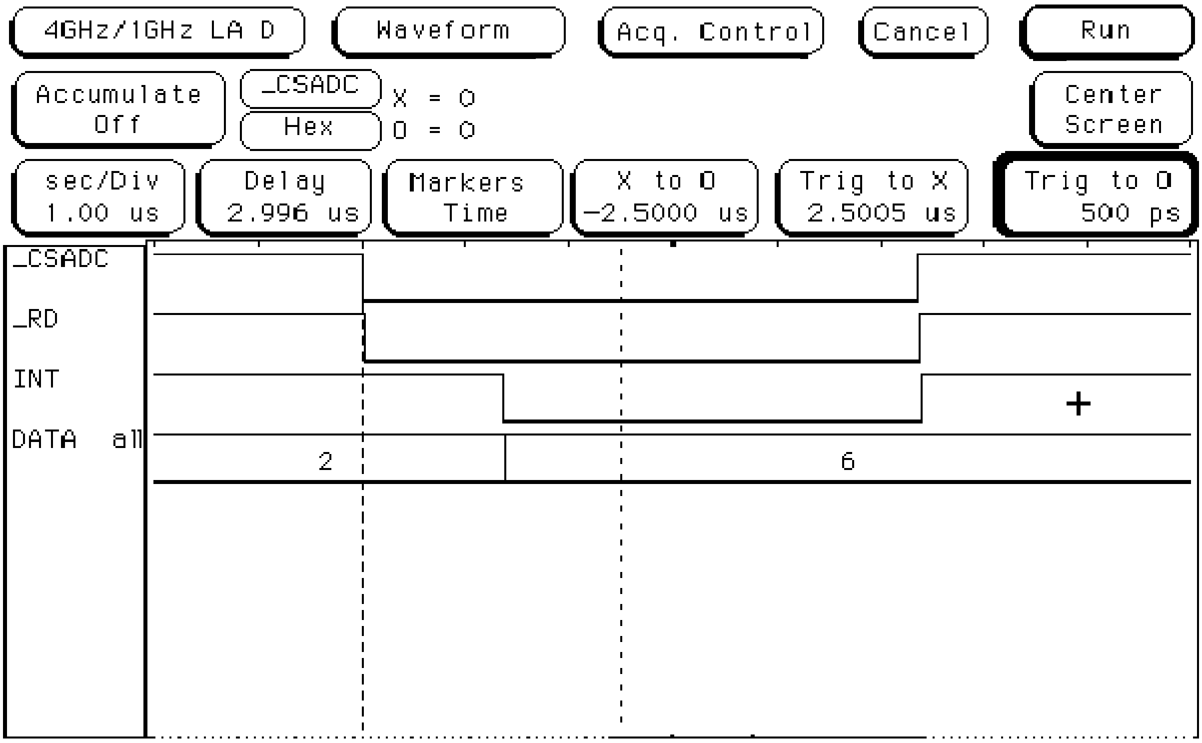
Two 8 channel 8-Bit ADCs are used for voltage and temperature control on the TPU. For ADC readout only byte transfers are allowed.
Example (ADC2, channel 3):

```
ADC2_3 EQU $412002
MOVE.B  ADC2_3,D0   ;read ADC2 channel 3 to D0
```

Calculation for 2V control:
Voltage(2V) := (ADCOUT(8-Bit) * ADC_REF_VOLTAGE * 2)/ADC-RESOLUTION;
:= (ADCOUT * 2.5V * 2) / 256;
Calculation for 3V control:
Voltage(3V) := (ADCOUT(8-Bit) * ADC_REF_VOLTAGE * 2)/ADC-RESOLUTION;
:= (ADCOUT * 2.5V * 2) / 256;
Calculation for 5V control:
Voltage(5V) := (ADCOUT(8-Bit) * ADC_REF_VOLTAGE * 14.7/4.7)/ADC-RESOLUTION;
:= (ADCOUT * 2.5V * 14.7 / 4.7) / 256;
Calculation for temperature:
Temperature(°C):= (((ADCOUT * ADC_REF_VOLTAGE * 2)/256)*5/VOLTAGE - 1.375) / 0.0225
:= (ADC_OUT * 5) / 256 * 5 / VOLTAGE - 1.375) / 0.0225.

Factor 2: Insertion loss of input resistor network (10 kΩ + 10kΩ)/10kΩ.
Factor 14.7 / 4.7: Insertion loss of input resistor network (10 kΩ + 4.7 kΩ)/4.7 kΩ.
VOLTAGE: Supply voltage of temperature sensor (≈ 5V).

Note: Because of the pipelined architecture of the ADCs it is recommended for correct initialization, to readout all ADC channels after power up.

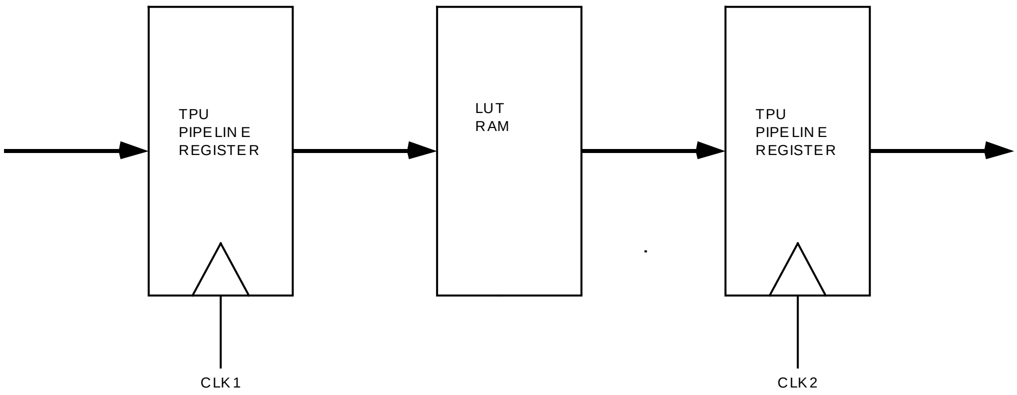


CPU access to ADC: Access time ≈ 6.5 µs

TPU PROCESSOR OPERATION

The design of the TPU processor was largely influenced by the following conditions:

1. 50 MHz clock frequency for each pipeline stage
2. Minimize the number of pipeline stages
3. Design optimisation for LUTs 32k*8, 128k*8, 512k*8
4. Design optimisation for LUTs with 15 ns access time
5. Serial test interface for each pipeline stage
6. Serial address load interface for CPU access



TPU pipeline stage

Timing calculation for CLK1 == CLK2:
$T_{ges} = T_{CO}$(Register) + T_{access} (RAM) + T_{setup}(Register) ≤ 20 ns (for 50 MHz)
T_{CO}(MACH210,211) = 4.5 ns (clock to output valid)
T_{setup}(MACH210,211) = 5.5 ns (data setup time before clock)
T_{access}(RAM) ≤ **10 ns** (RAM access time)

Since 4 Mbit SRAMs with an access time less than 10 ns are not available and to reduce costs, CLK2 was delayed by 5 ns to CLK1 in some pipeline stages.

Timing calculation for pipeline stage n/n+1 (CLK1 + 5 ns = CLK2):
$T_{ges} = T_{CO}$(Register) + T_{access} (RAM) + T_{setup}(Register) ≤ 25 ns (for 50 MHz)
T_{CO}(MACH210,211) = 4.5 ns (clock to output valid)
T_{setup}(MACH210,211) = 5.5 ns (data setup time before clock)
T_{access}(RAM) ≤ **15 ns** (RAM access time)

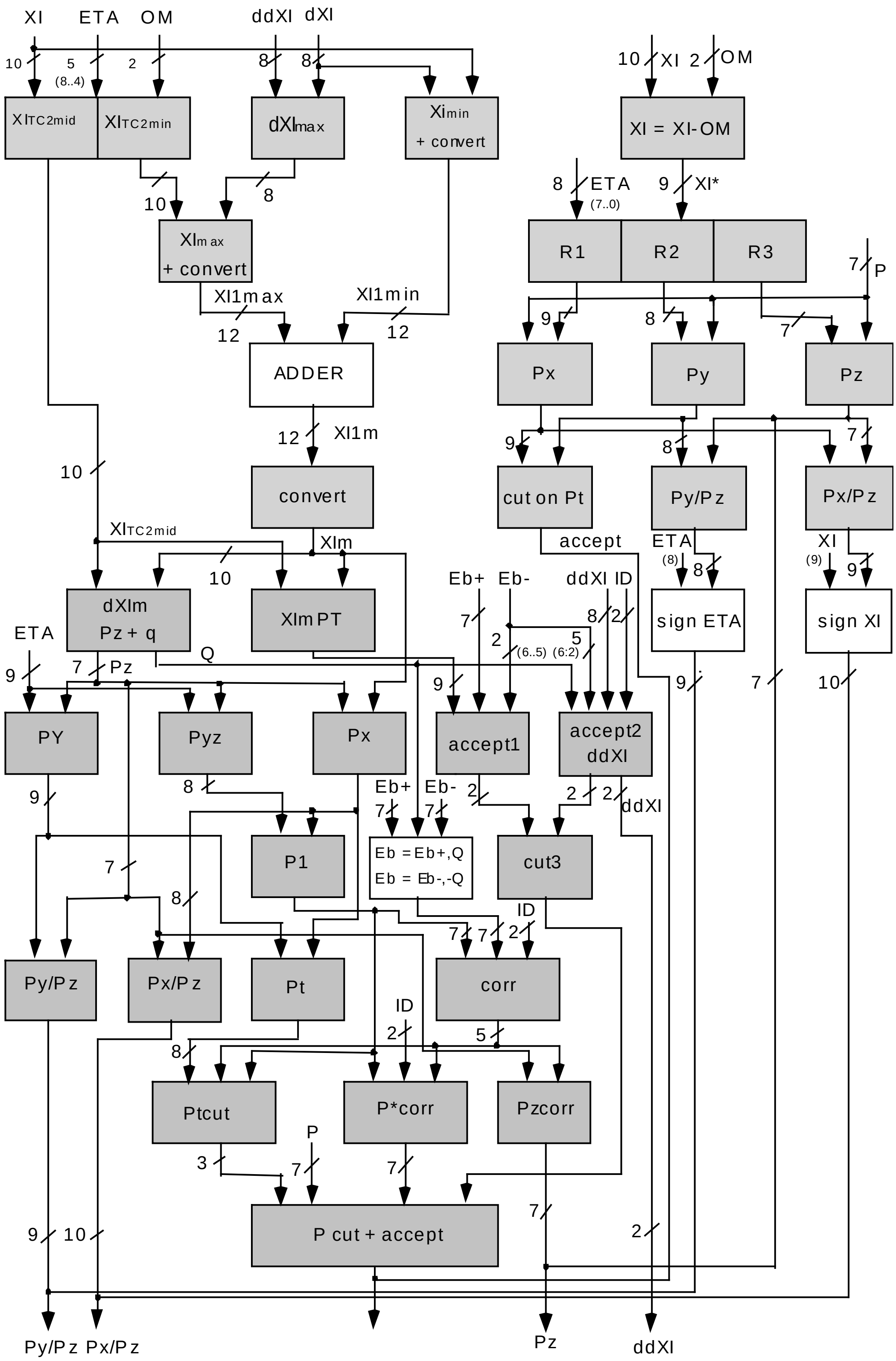
Timing calculation for pipeline stage n+1/n+2 (CLK2 + 5 ns = CLK3 = CLK1 + 10 ns):
$T_{ges} = T_{CO}$(Register) + T_{access} (RAM) + T_{setup}(Register) ≤ 25 ns (for 50 MHz)
T_{CO}(MACH210,211) = 4.5 ns (clock to output valid)
T_{setup}(MACH210,211) = 5.5 ns (data setup time before clock)
T_{access}(RAM) ≤ **15 ns** (RAM access time)

Timing calculation for pipeline stage n+2/n+3 (CLK4 == CLK1):
$T_{ges} = T_{CO}$(Register) + T_{setup}(Register) ≤ 10 ns (for 50 MHz)
= 4.5 ns + 5.5 ns ≤ 10 ns
(Between the pipeline stages n+2 and n+3 no LUT operation is possible!)

Typical logical equation of pipeline register variables (for example P1XI0) with an LUT output:

```
P1XI0 := CLKEN & P0XI0                          “normal pipeline operation : P1XI1 := P0XI0
       #!CLKEN &  P1_SHIFT & ADRLD & P1XI0   “pipeline stall: P1XI1 := P1XI1
       #!CLKEN & !P1_SHIFT & ADRLD & SER     “serial shift
       #!CLKEN &  P1_SHIFT &!ADRLD & SERADR; “serial address load
```

P1XI0	: Register pipeline stage P1
P0XI0	: Input variable to pipeline register
CLKEN	: Clock enable; high active
P1_SHIFT	: Control of serial shift; low active
SER	: Input variable for serial shift
ADRLD	: Control of serial address load; low active
SERADR	: Serial address input (from address serializer)



Dataflow in the Track-Parameter-Unit

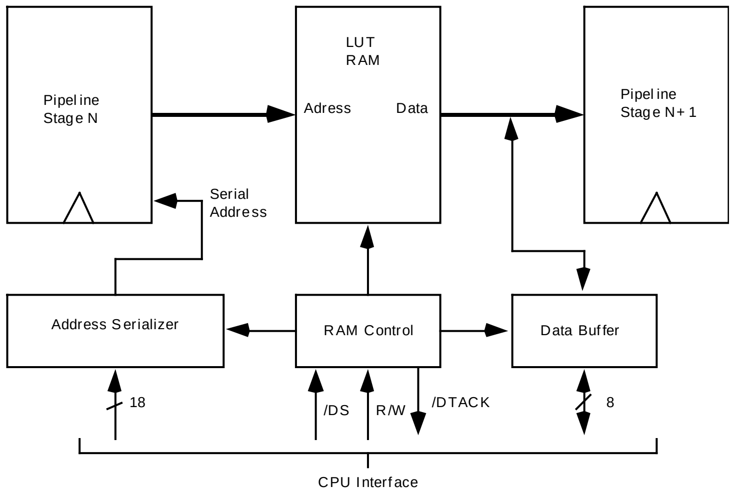
LUT READ/WRITE OPERATION

The necessary TPU registers are:

TPU LUTSEL REGISTER (`$0040A002`) : LUTSEL5..LUTSEL0 (D5..D0)
TPU LUT ADDRESS SPACE (depending on LUT size) : `$00C00000 - $00C3FFFF`

The LUTSEL REGISTER selects the LUT number for CPU accesses. The address window for CPU access to the LUT begins at `$00C00000`. The end of the address window is depending on the LUT size. Read and write transfers with byte, word or longword operand sizes are possible.

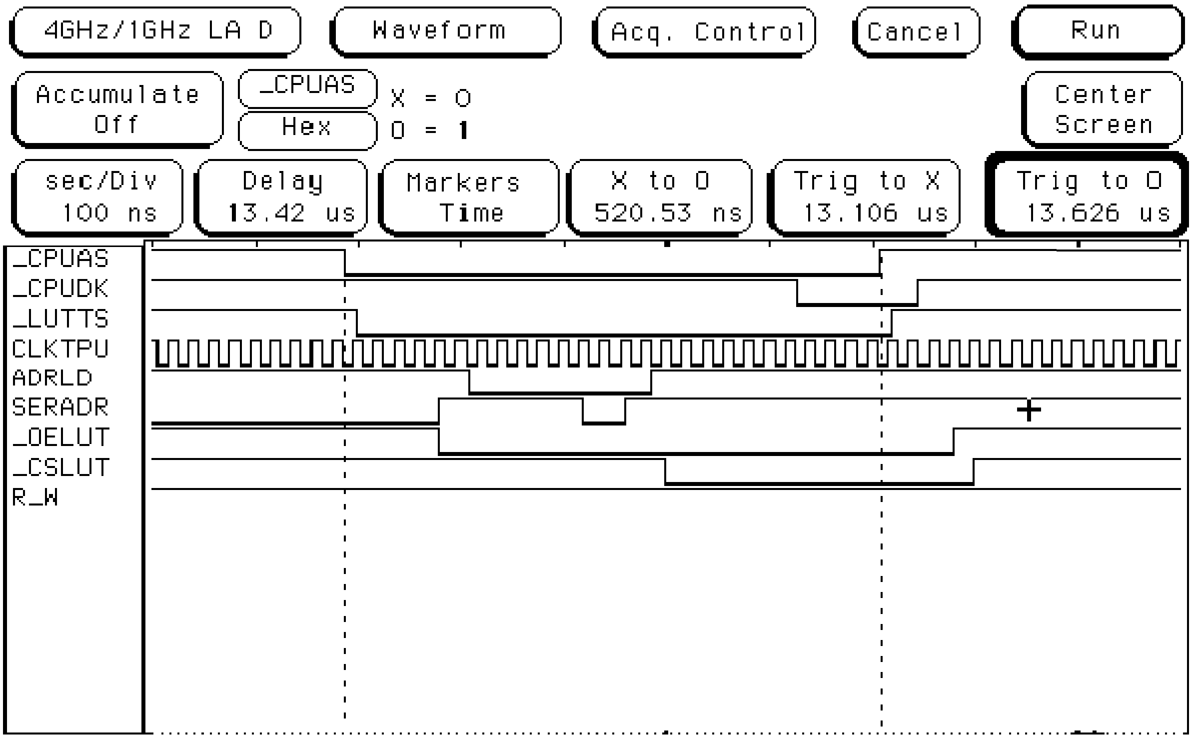
Operation flow for LUT access:
1. Select LUT using LUTSEL REGISTER.
2. CPU read/write operation in the LUT ADDRESS SPACE



LUT-CPU Interface

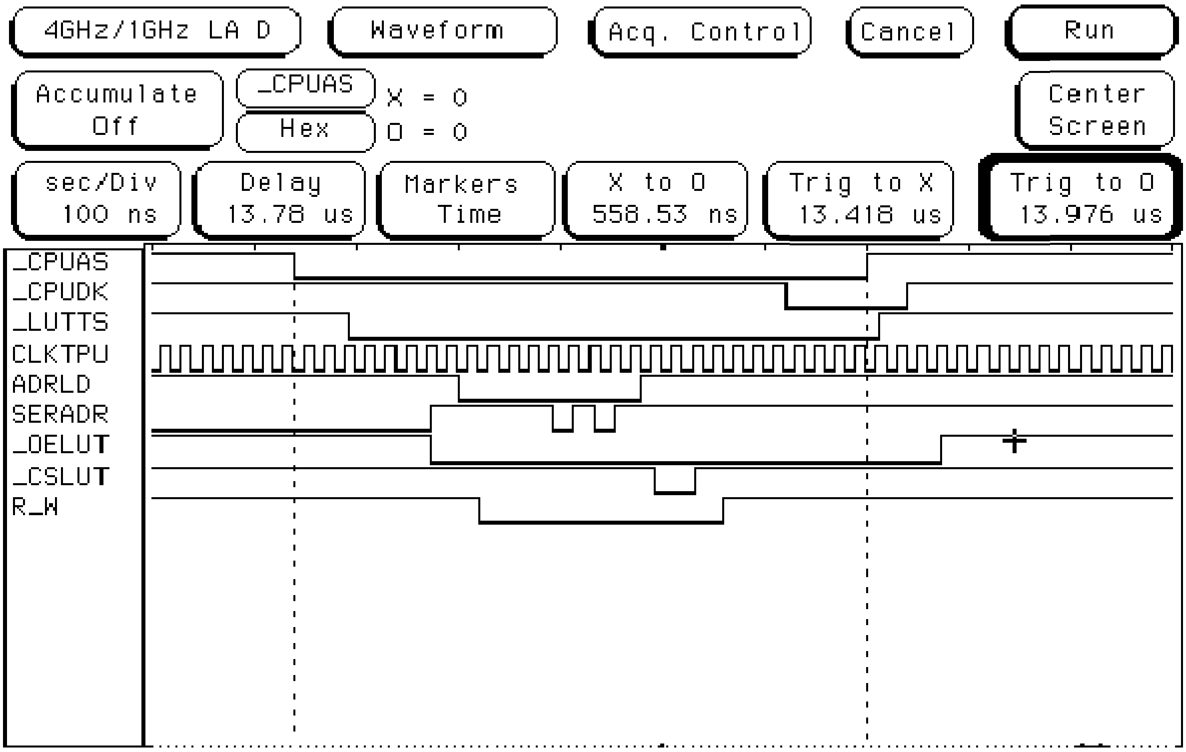
Note: When the TPU is operating in LUT access mode, it is absolute necessary to disable the normal TPU pipeline operation and to reset the pipeline (write $80 in TPU CONTROL REGISTER).

Note: It is possible (only for testing) to access the LUTs with another VME masters via VME slave interface.
Address space: `$00C00000 - $00C03FFF + VME SLAVE ADDRESS`
Byte transfer to even address:
`MOVE.B #DATA,LUTADR + VME_SLAVE_ADDRESS`
DATA = [LUTDATA7..0]
Byte transfer to odd address :
`MOVE.W #DATA,(LUTADR - 1) + VME_SLAVE_ADDRESS`
DATA = [LUTDATA7..0,X,X,X,X,X,X,X,X]
It is not recommended to use this feature for applications.



CPU read access to LUT RAM: Access time ≈ 520 ns

_CPUAS:	CPU address strobe	_CPUDK:	CPU transfer acknowledge
_LUTTS:	LUT transfer strobe	CLKTPU:	Clock TPU = 50 MHz
ADRLD:	Load serial address	SERADR:	Serial address (SERHADR/SERLADR)
_OELUT:	Enable data (LUT-CPU) buffer	_CSLUT:	Chip select LUT RAM
R_W:	Read-write LUT RAM		



CPU write access to LUT RAM: Access time ≈ 520 ns

_CPUAS:	CPU address strobe	_CPUDK:	CPU transfer acknowledge
_LUTTS:	LUT transfer strobe	CLKTPU:	Clock TPU = 50 MHz
ADRLD:	Load serial address	SERADR:	Serial address (SERHADR/SERLADR)
_OELUT:	Enable data (LUT-CPU) buffer	_CSLUT:	Chip select LUT RAM
R_W:	Read-write LUT RAM		

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P1-P2

CPU ADR	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	LUT7
	128k8-15	128k8-15	128k8-15	512k8-15	512k8-15	128k8-15	32k8-15
A17				dxi7	dxi7	dxi7	
A16	eta7	eta7	eta7	dxi6	dxi6	dxi6	
A15	eta6	eta6	eta6	dxi5	dxi5	dxi5	
A14	eta5	eta5	eta5	dxi4	dxi4	dxi4	
A13	eta4	eta4	eta4	dxi3	dxi3	dxi3	
A12	eta3	eta3	eta3	dxi2	dxi2	dxi2	
A11	om1	om1	om1	dxi1	dxi1	dxi1	om1
A10	om0	om0	om0	dxi0	dxi0	dxi0	om0
A9	xi9	xi9	xi9	xi9	xi9		xi9
A8	xi8	xi8	xi8	xi8	xi8		xi8
A7	xi7	xi7	xi7	xi7	xi7	ddxi7	xi7
A6	xi6	xi6	xi6	xi6	xi6	ddxi6	xi6
A5	xi5	xi5	xi5	xi5	xi5	ddxi5	xi5
A4	xi4	xi4	xi4	xi4	xi4	ddxi4	xi4
A3	xi3	xi3	xi3	xi3	xi3	ddxi3	xi3
A2	xi2	xi2	xi2	xi2	xi2	ddxi2	xi2
A1	xi1	xi1	xi1	xi1	xi1	ddxi1	xi1
A0	xi0	xi0	xi0	xi0	xi0	ddxi0	xi0
CPU DATA							
D7	ximid7	x	ximint7	xi1min7	x	dximax7	xil7
D6	ximid6	x	ximint6	xi1min6	x	dximax6	xil6
D5	ximid5	x	ximint5	xi1min5	x	dximax5	xil5
D4	ximid4	xil8	ximint4	xi1min4	x	dximax4	xil4
D3	ximid3	ximint9	ximint3	xi1min3	xi1min11	dximax3	xil3
D2	ximid2	ximint8	ximint2	xi1min2	xi1min10	dximax2	xil2
D1	ximid1	ximid9	ximint1	xi1min1	xi1min9	dximax1	xil1
D0	ximid0	ximid8	ximint0	xi1min0	xi1min8	dximax0	xil0

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P2-P3

CPU ADR	LUT8	LUT9
	512k8-15	512k8-15
A17	Dximax7	dximax7
A16	Dximax6	dximax6
A15	Dximax5	dximax5
A14	Dximax4	dximax4
A13	Dximax3	dximax3
A12	Dximax2	dximax2
A11	Dximax1	dximax1
A10	Dximax0	dximax0
A9	Ximin9	ximin9
A8	Ximin8	ximin8
A7	Ximin7	ximin7
A6	Ximin6	ximin6
A5	Ximin5	ximin5
A4	Ximin4	ximin4
A3	Ximin3	ximin3
A2	Ximin2	ximin2
A1	Ximin1	ximin1
A0	Ximin0	ximin0
CPU DATA		
D7	Ximax7	X
D6	Ximax6	X
D5	Ximax5	X
D4	Ximax4	X
D3	Ximax3	Ximax
D2	Ximax2	Ximax
D1	Ximax1	ximax9
D0	Ximax0	ximax8

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P4-P5

CPU ADR	LUT10	LUT11	LUT12	LUT13	LUT14
	128k8-15	128k8-15	128k8-15	32k8-10	32k8-10
A17					
A16	eta7	eta7	eta7		
A15	eta6	eta6	eta6		
A14	eta5	eta5	eta5		
A13	eta4	eta4	eta4		
A12	eta3	eta3	eta3		
A11	eta2	eta2	eta2	xi1m11	xi1m11
A10	eta1	eta1	eta1	xi1m10	xi1m10
A9	eta0	eta0	eta0	xi1m9	xi1m9
A8	xi8	xi8	xi8	xi1m8	xi1m8
A7	xi7	xi7	xi7	xi1m7	xi1m7
A6	xi6	xi6	xi6	xi1m6	xi1m6
A5	xi5	xi5	xi5	xi1m5	xi1m5
A4	xi4	xi4	xi4	xi1m4	xi1m4
A3	xi3	xi3	xi3	xi1m3	xi1m3
A2	xi2	xi2	xi2	xi1m2	xi1m2
A1	xi1	xi1	xi1	xi1m1	xi1m1
A0	xi0	xi0	xi0	xi1m0	xi1m0
CPU DATA					
D7	r17	r18	r27	xim7	X
D6	r16	r36	r26	xim6	X
D5	r15	r35	r25	xim5	X
D4	r14	r34	r24	xim4	X
D3	r13	r33	r23	xim3	X
D2	r12	r32	r22	xim2	X
D1	r11	r31	r21	xim1	xim9
D0	r10	r30	r20	xim0	xim8

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P5 - P6

CPU ADR	LUT16,17,18,19	LUT20,21,22,23	LUT24,25,26,27
	512k8-15	512k8-15	1M1-15
	xim9, xim8	Xim9, xim8	Xim9, xim8
	00, 01, 10, 11	00, 01, 10, 11	00, 01, 10, 11
A17	xim7	Xim7	Xim7
A16	xim6	Xim6	Xim6
A15	xim5	Xim5	Xim5
A14	xim4	Xim4	Xim4
A13	xim3	Xim3	Xim3
A12	xim2	Xim2	Xim2
A11	xim1	Xim1	Xim1
A10	xim0	Xim0	Xim0
A9	ximid9	Ximid9	Ximid9
A8	ximid8	Ximid8	Ximid8
A7	ximid7	Ximid7	Ximid7
A6	ximid6	Ximid6	Ximid6
A5	ximid5	Ximid5	Ximid5
A4	ximid4	Ximid4	Ximid4
A3	ximid3	Ximid3	Ximid3
A2	ximid2	Ximid2	Ximid2
A1	ximid1	Ximid1	Ximid1
A0	ximid0	Ximid0	Ximid0
CPU DATA			
D7	q	Ximpt7	X
D6	pz_6	Ximpt6	X
D5	pz_5	Ximpt5	X
D4	pz_4	Ximpt4	X
D3	pz_3	Ximpt3	X
D2	pz_2	Ximpt2	X
D1	Pz_1	Ximpt1	X
D0	pz_0	Ximpt1	Ximpt8

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P6 - P7

CPU ADR	LUT28	LUT29	LUT30	LUT31	LUT32	LUT33	LUT34	LUT35	LUT36	LUT37	LUT38
											ximpt8
											0: D0,D1
	128k8-15	128k8-15	128k8-15	128k8-15	128k8-15	128k8-15	128k8-15	128k8-15	128k8-15	128k8-15	1: D2,D3
A17											128k8-15
A16								xim9	xim9	ddxi7	Ximpt7
A15	r18	r18			eta8	eta8	eta8	xim8	xim8	ddxi6	Ximpt6
A14	r17	r17	r27		eta7	eta7	eta7	xim7	xim7	ddxi5	Ximpt5
A13	r16	r16	r26	r36	eta6	eta6	eta6	xim6	xim6	ddxi4	Ximpt4
A12	r15	r15	r25	r35	eta5	eta5	eta5	xim5	xim5	ddxi3	Ximpt3
A11	r14	r14	r24	r34	eta4	eta4	eta4	xim4	xim4	ddxi2	Ximpt2
A10	r13	r13	r23	r33	eta3	eta3	eta3	xim3	xim3	ddxi1	Ximpt1
A9	r12	r12	r22	r32	eta2	eta2	eta2	xim2	xim2	ddxi0	Ximpt0
A8	r11	r11	r21	r31	eta1	eta1	eta1	xim1	xim1	eb_6	eb_6
A7	r10	r10	r20	r30	eta0	eta0	eta0	xim0	xim0	eb_5	eb_5
A6	p6	p6	p6	p6	pz_6	pz_6	pz_6	pz_6	pz_6	eb_4	eb6
A5	p5	p5	p5	p5	pz_5	pz_5	pz_5	pz_5	pz_5	eb_3	eb5
A4	p4	p4	p4	p4	pz_4	pz_4	pz_4	pz_4	pz_4	eb_2	eb4
A3	p3	p3	p3	p3	pz_3	pz_3	pz_3	pz_3	pz_3	q	eb3
A2	p2	p2	p2	p2	pz_2	pz_2	pz_2	pz_2	pz_2	all	eb2
A1	p1	p1	p1	p1	pz_1	pz_1	pz_1	pz_1	pz_1	id1	eb1
A0	p0	p0	p0	p0	pz_0	pz_0	pz_0	pz_0	pz_0	id0	eb0
CPU DATA											
D7	px7	x	py7	x	py_7	x	pyz_7	px_7	X	acceb1	X
D6	px6	x	py6	pz6	py_6	x	pyz_6	px_6	X	acceb0	X
D5	px5	x	py5	pz5	py_5	x	pyz_5	px_5	X	x	X
D4	px4	x	py4	pz4	py_4	x	pyz_4	px_4	X	x	X
D3	px3	x	py3	pz3	py_3	x	pyz_3	px_3	X	x	Accea3
D2	px2	x	py2	pz2	py_2	x	pyz_2	px_2	X	x	Accea2
D1	px1	x	py1	pz1	py_1	x	pyz_1	px_1	X	ddxi1	Accea1
D0	px0	px8	py0	pz0	py_0	py_8	pyz_0	px_0	px_8	ddxi0	Accea0

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P7 – P8

CPU ADR	LUT39
A17	
A16	
A15	
A14	
A13	
A12	
A11	
A10	
A9	
A8	
A7	
A6	
A5	
A4	
A3	acceb1
A2	acceb0
A1	accea1
A0	accea0
CPU DATA	
D7	X
D6	X
D5	X
D4	X
D3	X
D2	X
D1	X
D0	cut3

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P8 – P9

CPU ADR	LUT40	LUT41	LUT42	LUT43	LUT44	LUT45	LUT46	LUT47	LUT48	LUT49
	512k8-15	128k8-15	128k8-15	128k8-15	128k8-15	128k8-15	128k8-15	32k8-15	128k8-15	128k8-15
A17	px_8									
A16	px_7	pyz_7					py7	py7		
A15	px_6	pyz_6	pz_6	pz_6	pz_6	pz_6	py6	py6	pz6	Pz6
A14	px_5	pyz_5	pz_5	pz_5	pz_5	pz_5	py5	py5	pz5	Pz5
A13	px_4	pyz_4	pz_4	pz_4	pz_4	pz_4	py4	py4	pz4	Pz4
A12	px_3	pyz_3	pz_3	pz_3	pz_3	pz_3	py3	py3	pz3	Pz3
A11	px_2	pyz_2	pz_2	pz_2	pz_2	pz_2	py2	py2	pz2	Pz2
A10	px_1	pyz_1	pz_1	pz_1	pz_1	pz_1	py1	py1	pz1	Pz1
A9	px_0	pyz_0	pz_0	pz_0	pz_0	pz_0	py0	py0	pz0	Pz0
A8	py_8	px_8	py_8	py_8	px_8	px_8	px8		px8	Px8
A7	py_7	px_7	py_7	py_7	px_7	px_7	px7		px7	Px7
A6	py_6	px_6	py_6	py_6	px_6	px_6	px6	pz6	px6	Px6
A5	py_5	px_5	py_5	py_5	px_5	px_5	px5	pz5	px5	Px5
A4	py_4	px_4	py_4	py_4	px_4	px_4	px4	pz4	px4	Px4
A3	py_3	px_3	py_3	py_3	px_3	px_3	px3	pz3	px3	Px3
A2	py_2	px_2	py_2	py_2	px_2	px_2	px2	pz2	px2	Px2
A1	py_1	px_1	py_1	py_1	px_1	px_1	px1	pz1	px1	Px1
A0	py_0	px_0	py_0	py_0	px_0	px_0	px0	pz0	px0	Px0
CPU DATA										
D7	pt7	p1_7	pypz7	x	pxpz7	x	x	pypz7	Pxpz7	X
D6	pt6	p1_6	pypz6	x	pxpz6	x	x	pypz6	Pxpz6	X
D5	pt5	p1_5	pypz5	x	pxpz5	x	x	pypz5	Pxpz5	X
D4	pt4	p1_4	pypz4	x	pxpz4	x	x	pypz4	Pxpz4	X
D3	pt3	p1_3	pypz3	x	pxpz3	x	x	pypz3	Pxpz3	X
D2	pt2	p1_2	pypz2	x	pxpz2	x	x	pypz2	Pxpz2	X
D1	pt1	p1_1	pypz1	x	pxpz1	pxpz9	x	pypz1	Pxpz1	X
D0	pt0	p1_0	pypz0	pypz8	pxpz0	pxpz8	accept	pypz0	Pxpz0	Pxpz8

CPU ADDRESS + DATA MAP FOR THE LUT ON THE PIPELINE STAGE P9 – P10

CPU ADR	LUT50
	128k8-10
A17	
A16	
A15	p6
A14	p5
A13	p4
A12	p3
A11	p2
A10	p1
A9	p0
A8	eb6
A7	eb5
A6	eb4
A5	eb3
A4	eb2
A3	eb1
A2	eb0
A1	id1
A0	id0
CPU DATA	
D7	X
D6	X
D5	X
D4	corr4
D3	corr3
D2	corr2
D1	corr1
D0	corr0

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P10 – P11

CPU ADR	LUT51	LUT52	LUT53
	32k8-10	32k8-10	32k8-10
A17			
A16			
A15			
A14	pt7		
A13	pt6	p1_6	
A12	pt5	p1_5	
A11	pt4	p1_4	pz6
A10	pt3	p1_3	pz5
A9	pt2	p1_2	pz4
A8	pt1	p1_1	pz3
A7	pt0	p1_0	pz2
A6	id1	id1	pz1
A5	id0	id0	pz0
A4	corr4	corr4	corr4
A3	corr3	corr3	corr3
A2	corr2	corr2	corr2
A1	corr1	corr1	corr1
A0	corr0	corr0	corr0
CPU DATA			
D7	x	x	X
D6	x	p_corr6	pzcorr6
D5	x	p_corr5	pzcorr5
D4	x	p_corr4	pzcorr4
D3	x	p_corr3	pzcorr3
D2	ptcut2	p_corr2	pzcorr2
D1	ptcut1	p_corr1	pzcorr1
D0	ptcut0	p_corr0	pzcorr0

CPU ADDRESS + DATA MAP FOR THE LUTS BETWEEN PIPELINE STAGE P11 - P12

CPU ADR	LUT54
	128k8-10
A17	
A16	p6
A15	p5
A14	p4
A13	p3
A12	p2
A11	p1
A10	p0
A9	cut3
A8	p_corr6
A7	p_corr5
A6	p_corr4
A5	p_corr3
A4	p_corr2
A3	p_corr1
A2	p_corr0
A1	id1
A0	id0
CPU DATA	
D7	acc7
D6	acc6
D5	acc5
D4	acc4
D3	acc3
D2	acc2
D1	acc1
D0	acc0

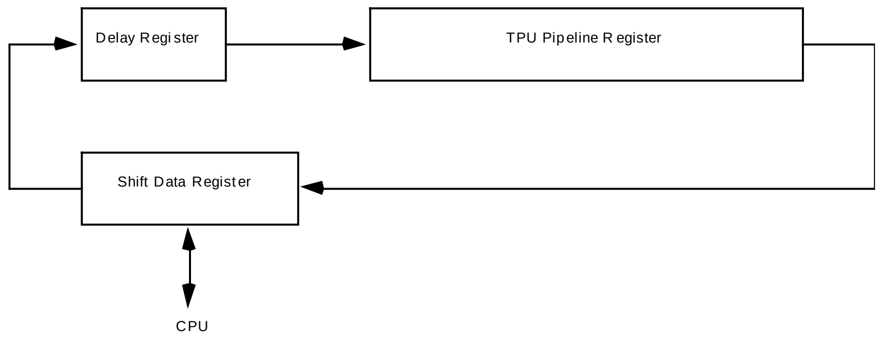
SERIAL SHIFT OPERATION

The necessary TPU registers are:

TPU CONTROL REGISTER ($0040A000): **SHIFT-BIT (D3)**
SHIFT DATA REGISTER ($0040A005): **SCD7 .. SCD0 (D7..D0)**
SHIFT PIPELINE LEVEL REGISTER($0040A004): **SPLSCD4 .. SPLSD0 (D4..D0)**

The SHIFT-BIT is used to perform a shift operation. Write transfer to SHIFT-BIT with 1(CD=3) triggers a byte-lenght (8 register) shift operation. After the end of the shift operation, the SHIFT-Bit is automatic set to zero. The SHIFT-DATA-REGISTER is used for CPU read-write accesses to shift pipeline. Only byte transfers are possible. The SHIFT PIPELINE LEVEL REGISTER selects the pipeline stage for shifting. Read and write byte transfers are possible.

Operation flow for shift operation:
Pipeline write: 1. Select SHIFT PIPELINE LEVEL REGISTER.
2. Write SHIFT DATA REGISTER.
3. Set SHIFT-BIT.
Pipeline read: 1. Select SHIFT PIPELINE LEVEL REGISTER.
2. Set SHIFT-BIT.
3. Read SHIFT DATA REGISTER



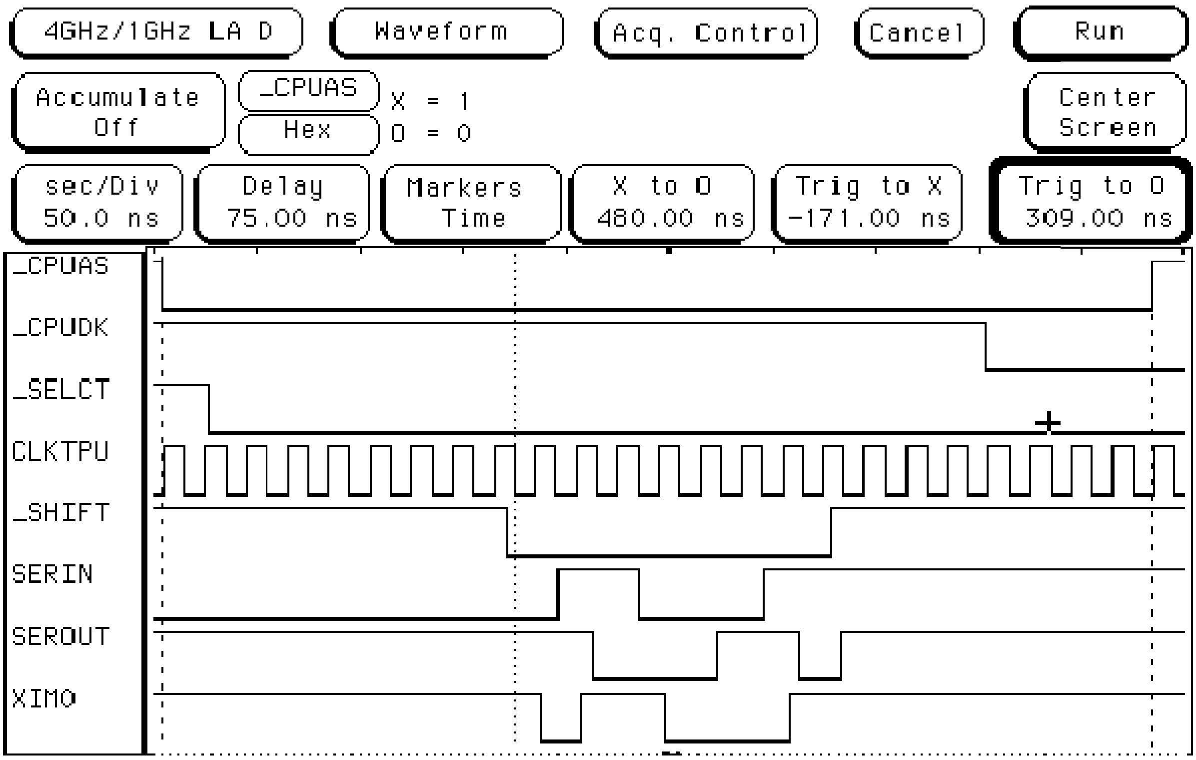
Serial data flow during shift

Delay register: Additional delay register to set the shift register length to a byte boundary.

Note: When the TPU is operating in shift mode, it is absolute necessary to disable the normal TPU pipeline operation (RUN = 0). No RESETVALID may be asserted.
Note: To restore the original pipeline status after serial shift, N +1 shift operations are necessary, if the pipeline lenght is N bytes. Example (pipeline length = 3 bytes):
SHIFT – READ – SHIFT – READ – SHIFT – READ – SHIFT (3 read operations + 4 shift operations)

```
 ; Testprogram for test of the connections between COMP1-3 and COMP4-6

CTRL      EQU $40a000     ;GLOBAL CONTROL REG OF TPU PIPELINE
PIPESEL   EQU $40a004     ;SELECT PIPE STAGE FOR SERIAL SHIFT
LUTSEL    EQU $40a002     ;SELECT LUT FOR CPU ACCESS
RNDSIG    EQU $40a003     ;RND + SIGNATUR CONTROL REG
DATA      EQU $40a005     ;DATA REG FOR SERIAL SHIFT
COMPCACHE EQU $40a001     ;COMPARATOR + CACHE CONTROL


; Testprogram for load of pipeline stage P19
WRITE_PIPE19
;PROGRAMM FOR LOAD OF P19
;INPUT PARAMETERS
; BX7-0       : D0
; VAL,X,ID1,ID0: D1
; PX9..0      ; D2
; PY8..0      ; D3
     MOVEM.L D0-D7,-(A7)
     MOVE.B #19,PIPESEL

;SHIFT ID1.. ID0, BX7 .. BX2
     MOVE.L D1,D7                        ;GET ID1, ID0
     ASL.L  #6,D7
     ANDI.L #$000000C0,D7
     MOVE.L D0,D6                        ;GET BX
     ASR.L  #2,D6
     ANDI.L #$0000003F,D6                ;BX1,BX0 = 0
     ADD.L  D6,D7                        ;ID1..ID0 + BX7..BX2
     MOVE.B D7,DATA                      ;WRITE DATA IN SHIFT REG
     MOVE.B #$08,CTRL                    ;SHIFT DATA ID1..ID0 + BX7..BX2

 ;SHIFT BX1..BX0, VAL, PY8..PY4
     MOVE.L D0,D7                        ;GET BX
     ANDI.L #$00000003,D7                ;DELETE BX7..BX4
     ASL.L  #6,D7
     MOVE.L D1,D6                        ;GET VALID
     ANDI.L #$00000008,D6                ;DELETE ID
     ASL.L  #2,D6
     ADD.L  D6,D7                        ;BX1..BX0 + VALID
     MOVE.L D3,D6                        ;GET PY
     ASR.L  #4,D6
     ANDI.L #$0000001F,D6
     ADD.L  D6,D7                        ; BX3..BX0 + VALID + PY8..PY6
     MOVE.B D7,DATA                      ;WRITE DATA IN SHIFT REG
     MOVE.B #$08,CTRL                    ;SHIFT DATA

;SHIFT PY3.. PY0, PX9 .. PX6
     MOVE.L D3,D7                        ;GET PY
     ANDI.L #$000000F,D7                 ;DELETE PY8..PY4
     ASL.L  #4,D7
     MOVE.L D2,D6                        ;GET PX
     ASR.L  #6,D6
     ANDI.L #$0000000F,D6                ;PX7..PX0 = 0
     ADD.L  D6,D7                        ;PY5..PY0 + PX9..BX8
     MOVE.B D7,DATA                      ;WRITE DATA IN SHIFT REG
     MOVE.B #$08,CTRL                    ;

;SHIFT PX5 .. PX0
     MOVE.L D2,D6                        ;GET PX
     ANDI.L #$0000003F,D6                ;PX9..PX8 = 0
     ASL.L  #2,D6
     MOVE.B D6,DATA                      ;WRITE DATA IN SHIFT REG
     MOVE.B #$08,CTRL                    ;SHIFT DATA ID1..ID0 + BX7..BX2
     MOVEM.L (A7)+,D0-D7
     RTS

READ_PIPE20
;PROGRAM FOR READ OF P20
;OUTPUT PARAMETERS
; BX7-0       : D0
; VAL,X,ID1,ID0: D1
; PX9..0      ; D2
; PY8..0      ; D3
     MOVEM.L D4,-(A7)
     MOVE.B #20,PIPESEL
     MOVE.B #$08,CTRL           ;SHIFT DATA P13WEN, P13BX7..1
     NOP
     MOVE.B #$08,CTRL           ;SHIFT DATA P13BX0, P13WEN, P13BX7..2
     NOP
     MOVE.B #$08,CTRL           ;SHIFT DATA P13BX1..0, P20ID1..0, P20BX7..4
     MOVE.B DATA,D1             ;READ ID1,ID0
     ASR.L  #4,D1
```

```
ANDI.L #$00000003,D1
MOVE.B DATA,D0            ;READ BX7..4
ASL.L  #4,D0
ANDI.L #$000000F0,D0
MOVE.B #$08,CTRL          ;SHIFT DATA P20BX3..0, P20VAL, P20PY8..6
MOVE.B DATA,D3            ;READ  BX3..BX0
ASR.L  #4,D3
ANDI.L #$0000000F,D3
ADD.L  D3,D0              ;BX -> D0
MOVE.B DATA,D3            ;READ VAL
ANDI.L #$00000008,D3
ADD.L  D3,D1              ;VAL -> D1
MOVE.B DATA,D3            ;READ  PY8..6
ASL.L  #6,D3
ANDI.L #$000001C0,D3
MOVE.B #$08,CTRL          ;SHIFT DATA P20PY5..0,P20PX9..8
MOVE.B DATA,D2            ;READ  PY5..0
ASR.L  #2,D2
ANDI.L #$0000003F,D2
ADD.L  D2,D3              ;PY -> D3
MOVE.B DATA,D2            ;READ  PX9..8
ASL.L  #8,D2
ANDI.L #$00000300,D2
MOVE.B #$08,CTRL          ;SHIFT DATA P20PX7..0
MOVE.B DATA,D4            ;READ  PX7..0
ANDI.L #$000000FF,D4
ADD.L  D4,D2              ;PX -> D2
MOVEM.L (A7)+,D4
RTS
```



Serial shift timing: CPU access time ≈ 480 ns

_CPUAS	: CPU address strobe
_CPUDK	: CPU transfer acknowledge
_SELCT	: Select CSR address space
CLKTPU	: Clock TPU = 50 MHz
_SHIFT	: TPU pipeline serial shift control (low active)
SERIN	: TPU pipeline serial in
SEROUT	: TPU pipeline serial out
XIM0	: TPU pipeline register (example)

SERIAL SHIFT PIPELINE STAGE P1

Note: Definition of operations:
„:>“ Register delay;
„->“ combinatorial delay (not interesting for software development)

```
                -> P1SER1
XIBUF1 (12 REG) -> P1SER1   :> P1XI0 :> P1XI1 :> P1XI2 :> P1XI3 :> P1XI4
                            :> P1XI5 :> P1XI6 :> P1XI7 :> P1XI8 :> P1XI9
                            :> P1OM0 :> P1OM1
                -> P1SER2

DDXI1  (12 REG) -> P1SER2 :> P1DDXI0 :> P1DDXI1 :> P1DDXI2 :> P1DDXI3
                          :> P1DDXI4 :> P1DDXI5 :> P1DDXI6 :> P1DDXI7
                          :> P1DXI0 :> P1DXI1 :> P1DX2 :> P1DXI3
                -> P1SER3

ETA1   (13 REG) -> P1SER3 :> P1DXI4 :> P1DXI5 :> P1DXI6 :> P1DX7
                          :> P1ETA0 :> P1ETA1 :> P1ETA2 :> P1ETA3
                          :> P1ETA4 :> P1ETA5 :> P1ETA6 :> P1ETA7 :> P1ETA8
                -> P1SER4

PIPELIN1(33 REG)-> P1SER4 :> P1EB0 :> P1EB1 :> P1EB2 :> P1EB3
                          :> P1EB4 :> P1EB5 :> P1EB6
                          :> P1EB_0 :> P1EB_1 :> P1EB_2 :> P1EB_3
                          :> P1EB_4 :> P1EB_5 :> P1EB_6
                          :> P1ID0 :> P1ID1
                          :> P1ALL
                          :> P1P0 :> P1P1 :> P1P2 :> P1P3
                          :> P1P4 :> P1P5 :> P1P6
                          :> P1BX0 :> P1BX1 :> P1BX2 :> P1BX3
                          :> P1BX4 :> P1BX5 :> P1BX6 :> P1BX7
                          :> P1FLAG
                -> P1FLAG
LUTC1  (2 REG)  -> P1FLAG :> P1VALIDOUT :> P1VALID
                    -> P1VALID
TOTAL LENGHT = 12 + 12 + 13 + 33 + 2 = 72 REG (-> Delay Reg = 0)
```

Pipe 1	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	\VALID	\VALIDOU	FLAG	BX7	BX6	BX5	BX4	BX3
shift 2	BX2	BX1	BX0	P6	P5	P4	P3	P2
shift 3	P1	P0	ALL	ID1	ID0	EB_6	EB_5	EB_4
shift 4	EB_3	EB_2	EB_1	EB_0	EB6	EB5	EB4	EB3
shift 5	EB2	EB1	EB0	ETA8	ETA7	ETA6	ETA5	ETA4
shift 6	ETA3	ETA2	ETA1	ETA0	DXI7	DXI6	DXI5	DXI4
shift 7	DXI3	DXI2	DXI1	DXI0	DDXI7	DDXI6	DDXI5	DDXI4
shift 8	DDXI3	DDXI2	DDXI1	DDXI0	OM1	OM0	XI9	XI8
shift 9	XI7	XI6	XI5	XI4	XI3	XI2	XI1	XI0

\VALID, \VALIDOUT:	Indicator for valid data in pipeline
Flag:	Flag, pass to TDU
BX0..BX6:	BX (bunch-crossing) number
P0..6:	Momentum
ALL:	Hit in every TFU
ID0,ID1:	Pretrigger identifier
EB_0..6:	E-,; ECAL energy
EB0..6:	E+; ECAL energy
ETA0..8:	η; y coordinate at TC2
DXI0..7:	dξ; minimal track slope
DDXI0..7:	ddξ; size of track window
OM0,1:	ω; # (adjacent cells) in TC2
XI0..9:	ξ; x coordinte at TC2

SERIAL SHIFT PIPELINE STAGE P2

```
                -> P1SER1
XIBUF2 (10 REG) -> P1SER1 -> P2_VALIDD
                          :> P2XIMID0 :> P2XIMID1 :> P2XIMID2 :> P2XIMID3
                          :> P2XIMID4 :> P2XIMID5 :> P2XIMID6 :> P2XIMID7
                          :> P2XIMID8 :> P2XIMID9
                -> P2SER2
XIMINBF1 (15 REG)->P2SER2 :> P2XIMIN0 :> P2XIMIN1 :> P2XIMIN2 :> P2XIMIN3
                          :> P2XIMIN4 :> P2XIMIN5 :> P2XIMIN6 :> P2XIMIN7
                          :> P2XIMIN8 :> P2XIMIN9
                          :> P2XI1MIN0 :> P2XI1MIN1 :> P2XI1MIN2
                          :> P2XI1MIN3 :> P2XI1MIN4
                -> P2SER3
XIMAXBF1(15 REG)-> P2SER3 :> P2XI1MIN5 :> P2XI1MIN6 :> P2XI1MIN7
                          :> P2XI1MIN8 :> P2XI1MIN9 :> P2XI1MIN10
                          :> P2XIMIN11
                          :> P2DXIMAX0 :> P2DXIMAX1 :> P2DXIMAX2
                          :> P2DXIMAX3
                          :> P2DXIMAX4 :> P2DXIMAX5 :> P2DXIMAX6
                          :> P2DXIMAX7
                -> P2SER4
PIPELIN1(16 REG)->P2SER4  :> P2DDXI0 :> P2DDXI1 :> P2DDXI2 :> P2DDXI3
                          :> P2DDXI4 :> P2DDXI5 :> P2DDXI6 :> P2DDXI7
                          :> P2BX0 :> P2BX1 :> P2BX2 :> P2BX3
                          :> P2BX4 :> P2BX5 :> P2BX6 :> P2BX7
                -> P2BX7
ETABF3 (7 REG)  -> P2BX7  :> P2ETA0 :> P2ETA1 :> P2ETA2 :> P2ETA3
                          :> P2ETA4 :> P2ETA5 :> P2ETA6
                -> P2SER6
XIBUF3          -> P2SER6 :> P2ETA7 :> P2ETA8
                          :> P2XI0 :> P2XI1 :> P2XI2
                          :> P2XI3 :> P2XI4 :> P2XI5
                          :> P2XI6 :> P2XI7 :> P2XI8
                -> P2SER7
PBUF  (1 REG)   -> P2SER7 :> P2XI9
                -> P2SER8
LUTC1 (1 REG)   -> P2SER8 :> P2VALID
                -> P2VALID
OUTRG1 (2 REG)  -> P2VALID :> P2_SIG :> P2_TEST
                -> P2_TEST

TOTAL LENGHT: 10 + 15 + 15 + 16 + 7 + 11 + 1 + 1 + 2 = 78 REG
(-> Delay Reg = 2)
```

Pipe 2	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/TEST	/SIG	/VALID	XI9	XI8	XI7	XI6	XI5
shift 2	XI4	XI3	XI2	XI1	XI0	ETA8	ETA7	ETA6
shift 3	ETA5	ETA4	ETA3	ETA2	ETA1	ETA0	BX7	BX6
shift 4	BX5	BX4	BX3	BX2	BX1	BX0	DDXI7	DDXI6
shift 5	DDXI5	DDXI4	DDXI3	DDXI2	DDXI1	DDXI0	DXIMAX7	DXIMAX6
shift 6	DXIMAX5	DXIMAX4	DXIMAX3	DXIMAX2	DXIMAX1	DXIMAX0	XI1MIN11	XI1MIN10
shift 7	XI1MIN9	XI1MIN8	XI1MIN7	XI1MIN6	XI1MIN5	XI1MIN4	XI1MIN3	XI1MIN2
shift 8	XI1MIN1	XI1MIN0	XIMIN9	XIMIN8	XIMIN7	XIMIN6	XIMIN5	XIMIN4
Shift 9	XIMIN3	XIMIN2	XIMIN1	XIMIN0	XIMID9	XIMID8	XIMID7	XIMID6
Shift 10	XIMID5	XIMID4	XIMID3	XIMID2	XIMID1	XIMID0	X	X

/TEST:	Indicator for test data (from random generator)
/SIG:	Indicator for signature register to load this message. This signal is used for message readout (input + output message) during TPU RUN.
/VALID:	Indicator for valid data in pipeline
XI0..9:	ξ; x coordinte at TC2
ETA0..8:	η; y coordinate at TC2
BX0..BX6:	BX (bunch-crossing) number
DDXI0..7:	ddξ; size of track window
DXIMAX0..7:	dξmax; from LUT6 dξmax $\approx$ dξ + ddξ
XI1MIN0..11:	ξ_{min1}; from LUT 4 + 5: $\xi_{min1} = \xi-(z_{TC2} - z_m)*\tan(\alpha - d\xi)$
XIMIN0..9:	ξ_{minTC2}; from LUT 2 + 3: $\xi_{minTC2} = \xi - \omega$
XIMID0..9:	ξ_{midTC2}; from LUT 1 + 2: $\xi_{midTC2} = \xi - \omega/2$
X:	Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P3

```
                -> P3SER1
XIADD1 (11 REG) -> P3SER1 :> P3XIM0 :> P3XIM1 :> P3XIM2
                          :> P3XIM3 :> P3XIM4 :> P3XIM5
                          :> P3G0 :> P3G1 :> P3G2 :> P3P1 :> P3P2
                -> P3P2
XIADD2 (13 REG) -> P3P2   :> P3XIM6 :> P3XIM7 :> P3XIM8
                          :> P3XIM9 :> P3XIM10 :> P3XIM11
                          :> P3G3 :> P3G4 :> P3G5 :> P3P3 :> P3P4 :> P3P5
                          :> P3SER3
BXREG1 (8 REG)  -> PSER3  :> P3BX0 :> P3BX1 :> P3BX2 :> P3BX3
                          :> P3BX4 :> P3BX5 :> P3BX6 :> P3BX7
                -> P3SER4
LUTC2  (1 REG)  -> P3SER4 :> P3VALID
                -> P3VALID
TOTAL LENGHT: 11 + 13 + 8 + 1 = 33 REG  (-> Delay Reg = 7)
```

Pipe 3	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/VALID	BX7	BX6	BX5	BX4	BX3	BX2	BX1
shift 2	BX0	P3SER3	P5	P4	P3	G5	G4	G3
shift 3	XIM11	XIM10	XIM9	XIM8	XIM7	XIM6	P2	P1
shift 4	G2	G1	G0	XIM5	XIM4	XIM3	XIM2	XIM1
shift 5	XIM0	X	X	X	X	X	X	X

/VALID:	Indicator for valid data in pipeline
BX0..BX6:	BX (bunch-crossing) number
P1..P5:	Intern carry bits from adder “carry propagate”
G0 .. G5:	Intern carry bits from adder “carry generate”
XIM0..11:	Partial sum of XI1MIN + XI1MAX (without carry consideration)
P3SER3:	Delay register for correct pipeline shift operation (CLKDD/CLK problem)
X:	Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P4

```
                -> P3SER1
XIADD1  (6 REG) -> P3SER1 :> P4XIM0 :> P4XIM1 :> P4XIM2
                          :> P4XIM3 :> P4XIM4 :> P4XIM5
                -> P4SER2
XIADD2  (6 REG) -> P4SER2 :> P4XIM6 :> P4XIM7 :> P4XIM8
                          :> P4XIM9 :> P4XIM10 :> P4XIM11
                -> P4SER3
ETABF3  (7 REG) -> P4SER3 :> P4ETA0 :> P4ETA1 :> P4ETA2 :> P4ETA3
                          :> P4ETA4 :> P4ETA5 :> P4ETA6
                -> P4SER4
XIBUF3 (11 REG) -> P4SER4 :> P4ETA7 :> P4ETA8
                          :> P4XI0 :> P4XI1 :> P4XI2 :> P4XI3 :> P4XI4
                          :> P4XI5 :> P4XI6 :> P4XI7 :> P4XI8
                -> P4XI8
PBUF   (1 REG)  -> P4XI8  :> P4XI9
                -> P4XI9
LUTC2  (1 REG)  -> P4XI9  :> P4VALID
                   -> P4VALID
TOTAL LENGHT: 6 + 6 + 7 + 11 + 1 + 1 = 32 REG  (-> Delay Reg = 0)
```

Pipe 4	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/VALID	XI9	XI8	XI7	XI6	XI5	XI4	XI3
shift 2	XI2	XI1	XI0	ETA8	ETA7	ETA6	ETA5	ETA4
shift 3	ETA3	ETA2	ETA1	ETA0	XIM11	XIM10	XIM9	XIM8
shift 4	XIM7	XIM6	XIM5	XIM4	XIM3	XIM2	XIM1	XIM0

/VALID: Indicator for valid data in pipeline

XI0..9: $\xi 1$ from LUT 2 + 7: $\xi 1 = \xi - \omega$

ETA0..8: η; y coordinate at TC2

XIM0..11: Sum of XI1MIN + XI1MAX (with carry consideration)

SERIAL SHIFT PIPELINE STAGE P5

```
                 -> P5SER1
XIBUF2 (10 REG) -> P5SER1   :> P5XIMID0 :> P5XIMID1 :> P5XIMID2
                            :> P5XIMID3 :> P5XIMID4
                            :> P5XIMID5 :> P5XIMID6 :> P5XIMID7
                            :> P5XIMID8 :> P5XIMID9
                -> P5SER2
XIMBF1 (10 REG) -> P5SER2 :> P5XIM0 :> P5XIM1 :> P5XIM2 :> P5XIM3 :> P5XIM4
                          :> P5XIM5 :> P5XIM6 :> P5XIM7 :> P5XIM8 :> P5XIM9
                -> P5SER3
PIPELIN1 P5SER3 (18 REG)  :> P5ID0 :> P5ID1 :> P5ALL
                          :> P5P0 :> P5P1 :> P5P2 :> P5P3 :> P5P4
                          :> P5P5 :> P5P6
                          :> P5DDXI0 :> P5DDXI1 :> P5DDXI2 :> P5DDXI3
                          :> P5DDXI4 :> P5DDXI5 :> P5DDXI6 :> P5DDXI7
                -> P5SER4
RBF1  (13 REG)  -> P5SER4 :> P5R10 :> P5R11 :> P5R12 :> P5R13 :> P5R14
                          :> P5R15 :> P5R16 :> P5R17 :> P5R18
                          :> P5R20 :> P5R21 :> P5R22 :> P5R23
                -> P5SER5
RBF2  (12 REG)  -> P5SER5 :> P5R24 :> P5R25 :> P5R26 :> P5R27
                          :> P5R30 :> P5R31 :> P5R32 :> P5R33
                          :> P5R34 :> P5R35 :> P5R36
                          :> P5XI9
                -> P5SER6
LUTC3  (1 REG)  -> P5SER6 :> P5VALID
                -> P5VALID
TOTAL LENGHT: 10 + 10 + 18 + 13 + 12 + 1 = 64 REG (-> delay register = 0)
```

Pipe 5	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/VALID	XI9	R36	R35	R34	R33	R32	R31
shift 2	R30	R27	R26	R25	R24	R23	R22	R21
shift 3	R20	R18	R17	R16	R15	R14	R13	R12
shift 4	R11	R10	DDXI7	DDXI6	DDXI5	DDXI4	DDXI3	DDXI2
shift 5	DDXI1	DDXI0	P6	P5	P4	P3	P2	P1
shift 6	P0	ALL	ID1	ID0	XIM9	XIM8	XIM7	XIM6
shift 7	XIM5	XIM4	XIM3	XIM2	XIM1	XIM0	XIMID9	XIMID8
shift 8	XIMID7	XIMID6	XIMID5	XIMID4	XIMID3	XIMID2	XIMID1	XIMID0

/VALID:	Indicator for valid data in pipeline
XI9:	ξ sign bit
R30..R36:	R3 from LUT 11: $R3 = z_{TC2} / \sqrt{(\xi^2 + \eta^2 + z^2_{TC2})}$
R20..R27:	R2 from LUT 12: $R2 = \eta / \sqrt{(\xi^2 + \eta^2 + z^2_{TC2})}$
R10..R18:	R1 from LUT 10 + 11: $R1 = \xi / \sqrt{(\xi^2 + \eta^2 + z^2_{TC2})}$
DDXI0..7:	ddξ; size of track window
P0..6:	Momentum
ALL:	Hit in every TFU
ID0, ID1:	Pretrigger identifier
XIM0..9:	ξ_m from LUT 13 + 14: $\xi_m = \xi_m^1$ converted from 12 to 10 bits:
XIMID0..9:	ξ_{midTC2}; from LUT 1 + 2: $\xi_{midTC2} = \xi - \omega/2$

SERIAL SHIFT PIPELINE STAGE P6

```
PZBF1 (7 REG)  -> P5SER1 :> P6PZ_0 :> P6PZ_1 :> P6PZ_2 :> P6PZ_3
               :> P6PZ_4 :> P6PZ_5 :> P6PZ_6
               -> P6SER2
XIMBF2 (14 REG)-> P6SER2 :> P6XIM0 :> P6XIM1 :> P6XIM2 :> P6XIM3 :> P6XIM4
               :> P6XIM5 :> P6XIM6 :> P6XIM7 :> P6XIM8 :> P6XIM9
               :> P6ID0 :> P6ID1 :> P6ALL :> P6Q
               -> P6SER3
DDXIB1 (15 REG)-> P6SER3 :> P6DDXI0 :> P6DDXI1 :> P6DDXI2 :> P6DDXI3
               :> P6DDXI4 :> P6DDXI5 :> P6DDXI6 :> P6DDXI7
               :> P6EB_2 :> P6EB_3 :> P6EB_4 :> P6EB_5 :> P6EB_6
               :> P6EB0 :> P6EB1
               -> P6SER4
XPTBF1 (14 REG)-> P6SER4 :> P6EB2 :> P6EB3 :> P6EB4 :> P6EB5 :> P6EB6
               :> P6XIPT0 :> P6XIPT1 :> P6XIPT2
               :> P6XIPT3 :> P6XIPT4
               :> P6XIPT5 :> P6XIPT6 :> P6XIPT7 :> P6XIPT8
               -> P6SER5
PBUF   (7 REG) -> P6SER5 :> P6P0 :> P6P1 :> P6P2 :> P6P3 :> P6P4
               :> P6P5 :> P6P6
               -> P6SER6
ETABF3 (7 REG) -> P6SER6 :> P6ETA0 :> P6ETA1 :> P6ETA2 :> P6ETA3
               :> P6ETA4 :> P6ETA5 :> P6ETA6
               -> P6SER7
XIBUF3 (2 REG) -> P6SER7 :> P6ETA7 :> P6ETA8
               -> P6ETA8
RBF1  (12 REG) -> P6ETA8 :> P6R10 :> P6R11 :> P6R12 :> P6R13 :> P6R14
               :> P6R15 :> P6R16 :> P6R17 :> P6R18
               :> P6R20 :> P6R21 :> P6R22 :> P6R23
               -> P6SER9
RBF2  (12 REG) -> P6SER9 :> P6R24 :> P6R25 :> P6R26 :> P6R27
               :> P6R30 :> P6R31 :> P6R32 :> P6R33
               :> P6R34 :> P6R35 :> P6R36
               :> P6XI9
               -> P6SER10
(1 REG)        :> P6VALID
TOTAL LENGHT: 7 + 14 + 15 + 14 + 7 + 7 + 2 + 13 + 12 + 1 = 92 REG
(-> delay register = 4)
```

Pipe 6	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/VALID	XI9	R36	R35	R34	R33	R32	R31
shift 2	R30	R27	R26	R25	R24	R23	R22	R21
shift 3	R20	R18	R17	R16	R15	R14	R13	R12
shift 4	R11	R10	ETA8	ETA7	ETA6	ETA5	ETA4	ETA3
shift 5	ETA2	ETA1	ETA0	P6	P5	P4	P3	P2
shift 6	P1	P0	XIPT8	XIPT7	XIPT6	XIPT5	XIPT4	XIPT3
shift 7	XIPT2	XIPT1	XIPT0	EB6	EB5	EB4	EB3	EB2
shift 8	EB1	EB0	EB_6	EB_5	EB_4	EVB_3	EB_2	DDXI7
Shift 9	DDXI6	DDXI5	DDXI4	DDXI3	DDXI2	DDXI1	DDXI0	Q
Shift 10	ALL	ID1	ID0	XIM9	XIM8	XIM7	XIM6	XIM5
Shift 11	XIM4	XIM3	XIM2	XIM1	XIM0	PZ_6	PZ_5	PZ_4
Shift 12	PZ_3	PZ_2	PZ_1	PZ_0	X	X	X	X

/VALID:	Indicator for valid data in pipeline
XI9:	ξ sign bit
R30..R36:	R3 from LUT 11: $R3 = z_{TC2} / \sqrt{(\xi^2 + \eta^2 + z^2_{TC2})}$
R20..R27:	R2 from LUT 12: $R2 = \eta / \sqrt{(\xi^2 + \eta^2 + z^2_{TC2})}$
R10..R18:	R1 from LUT 10 + 11: $R1 = \xi / \sqrt{(\xi^2 + \eta^2 + z^2_{TC2})}$
ETA0..8:	η; y coordinate at TC2
P0..6:	Momentum
XIPT0..8:	$\xi_{m,PT}$ from LUT 20 + 22 + 24: ?
EB0..6:	E+: ECAL energy
EB_2..6:	E- : ECAL energy
DDXI0..7:	ddξ; size of track window
Q:	Charge from LUT 16 + 18: $q = \mathrm{sign}\,(d\xi_m)$
ALL:	Hit in every TFU
ID0, ID1:	Pretrigger identifier
XIM0..9:	ξ_m from LUT 13 + 14: $\xi_m = \xi_m{}^1$ converted from 12 to 10 bits
PZ_0..6:	P_z from LUT 16 + 18: $P_z = 0.003 * B * l * (1/\tan\varphi - \tan\varphi/4) * (1 - \xi_m{}^2/2z^2{}_m)$
X:	Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P7

```
PYBF2 (12 REG)  -> P7SER1 :> P7PY_0 :> P7PY_1 :> P7PY_2 :> P7PY_3
                          :> P7PY_4 :> P7PY_5 :> P7PY_6 :> P7PY_7 :> P7PY_8
                          :> P7PYZ_0 :> P7PYZ_1 :> P7PYZ_2 :> P7PYZ_3
                -> P7SER2
PXBF2 (12 REG)  -> P7SER2 :> P7PYZ_4 :> P7PYZ_5 :> P7PYZ_6 :> P7PYZ_7
                          :> P7PX_0 :> P7PX_1 :> P7PX_2 :> P7PX_3
                          :> P7PX_4 :> P7PX_5 :> P7PX_6 :> P7PX_7 :> P7PX_8
                -> P7SER3
ACCBF1 (7 REG)  -> P7SER3 :> P7ACCEA0 :> P7ACCEA1 :> P7ACCEB0 :> P7ACCEB1
                          :> P7DDXI0 :> P7DDXI1
                          :> P7Q
                -> P7Q
PIPELIN1 (8 REG)->   P7Q  :> P7EB0 :> P7EB1 :> P7EB2 :> P7EB3
                          :> P7EB4 :> P7EB5 :> P7EB6
                          :> P7ETA8
PXYB1 (13 REG)  -> P7ETA8 :> P7PX0 :> P7PX1 :> P7PX2 :> P7PX3 :> P7PX4
                          :> P7PX5 :> P7PX6 :> P7PX7 :> P7PX8
                          :> P7PY0 :> P7PY1 :> P7PY2 :> P7PY3
                -> P7SER6
PZYB1 (12 REG)  -> P7SER6 :> P7PY4 :> P7PY5 :> P7PY6 :> P7PY7
                          :> P7PZ0 :> P7PZ1 :> P7PZ2 :> P7PZ3
                          :> P7PZ4 :> P7PZ5 :> P7PZ6
                          :> P7XI9
                -> P7SER7
LUTC4  (1 REG)            :> P6VALID
TOTAL LENGHT: 13 + 13 + 7 + 8 + 13 + 12 + 1 = 67 REG
(-> Delay Reg = 5)
```

Pipe 7	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/VALID	XI9	PZ6	PZ5	PZ4	PZ3	PZ2	PZ1
shift 2	PZ0	PY7	PY6	PY5	PY4	PY3	PY2	PY1
shift 3	PY0	PX8	PX7	PX6	PX5	PX4	PX3	PX2
shift 4	PX1	PX0	ETA8	EB6	EB5	EB4	EB3	EB2
Shift 5	EB1	EB0	Q	DDXI1	DDXI0	ACCEB1	ACCEB0	ACCEA1
Shift 6	ACCEA0	PX_8	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2
Shift 7	PX_1	PX_0	PYZ_7	PYZ_6	PYZ_5	PYZ_4	PYZ_3	PYZ_2
Shift 8	PYZ_1	PYZ_0	PY_8	PY_7	PY_6	PY_5	PY_4	PY_3
Shift 9	PY_2	PY_1	PY_0	X	X	X	X	X

/VALID:	Indicator for valid data in pipeline
XI9:	ξ sign bit
PZ0..6:	P_z from LUT 31: $P_z = R3 * P$
PY0..7:	P_y from LUT 30: $P_y = R2 * P$
PX0..8:	P_x from LUT 28 + 29: $P_x = R1 * P$
ETA8:	η; y coordinate at TC2 (sign bit)
EB0..EB6	E_b: $E_b = (q > 0.?\ E^+_b : E^-_b)$
Q:	Charge from LUT 16 + 18: $q = \text{sign}\ (d\xi_m)$
DDXI0..1:	$dd\xi$ from LUT 37
ACCEB0,1, ACCEA0,1:	Accept outputs from LUT 37 -> LUT 39 -> CUT3
PX_0..8:	P_x from LUT 35 + 36: $P_x = P_z * \xi_m/z_m$
PYZ_0..7:	P_{yz} from LUT 34: $P_{yz} = P_z * \sqrt{}\ (1 + \eta^2)$
PY_0..8:	P_y from LUT 32 + 33: $P_y = P_z * \eta$
X:	Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P8

```
PZBF1 (7 REG)   -> P7SER1 :> P8PZ_0 :> P8PZ_1 :> P8PZ_2 :> P8PZ_3
                          :> P8PZ_4 :> P8PZ_5 :> P8PZ_6
                -> P8SER2
PYBF2 (13 REG)  -> P8SER2 :> P8PY_0 :> P8PY_1 :> P8PY_2 :> P8PY_3 :> P8PY_4
                          :> P8PY_5 :> P8PY_6 :> P8PY_7 :> P8PY_8
                          :> P8PYZ_0 :> P8PYZ_1 :> P8PYZ_2 :> P8PYZ_3
                -> P8SER3
PXBF2 (13 REG)  -> P8SER3 :> P8PYZ_4 :> P8PYZ_5 :> P8PYZ_6 :> P8PYZ_7
                          :> P8PX_0 :> P8PX_1 :> P8PX_2 :> P8PX_3 :> P8PX_4
                          :> P8PX_5 :> P8PX_6 :> P8PX_7 :> P8PX_8
                -> P8PX_8
ACCBF1 (6 REG)  -> P8PX_8 :> P8ACCE30 :> P8ACCE31 :> P8ACCEB1
                          :> P8DDXI0 :> P8DDXI1
                          :> P8Q
                -> P8Q
PIPELIN1(10 REG)-> P8Q    :> P8EB0 :> P8EB1 :> P8EB2 :> P8EB3
                          :> P8EB4 :> P8EB5 :> P8EB6
                          :> P8ETA8
                          :> P8ID0 :> P8ID1
                -> P8SER6
PXYB1 (13 REG)  -> P8SER6 :> P8PX0 :> P8PX1 :> P8PX2 :> P8PX3 :> P8PX4
                          :> P8PX5 :> P8PX6 :> P8PX7 :> P8PX8
                          :> P8PY0 :> P8PY1 :> P8PY2 :> P8PY3
                -> P8SER7
PZYB1 (12 REG)  -> P8SER7 :> P8PY4 :> P8PY5 :> P8PY6 :> P8PY7
                          :> P8PZ0 :> P8PZ1 :> P8PZ2 :> P8PZ3
                          :> P8PZ4 :> P8PZ5 :> P8PZ6
                          :> P8XI9
                -> P8SER8
LUTC7 (1 REG)   -> P8SER8 :> P8_VALID
                -> P8_VALID

TOTAL LENGHT: 7 + 13 + 13 + 6 + 10 + 13 + 12 + 1  = 75 REG
(-> Delay Reg = 5)
```

Pipe 8	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/VALID	XI9	PZ6	PZ5	PZ4	PZ3	PZ2	PZ1
shift 2	PZ0	PY7	PY6	PY5	PY4	PY3	PY2	PY1
shift 3	PY0	PX8	PX7	PX6	PX5	PX4	PX3	PX2
shift 4	PX1	PX0	ID1	ID0	ETA8	EB6	EB5	EB4
Shift 5	EB3	EB2	EB1	EB0	Q	DDXI1	DDXI0	ACCEB1
Shift 6	ACCE31	ACCE30	PX_8	PX_7	PX_6	PX_5	PX_4	PX_3
Shift 7	PX_2	PX_1	PX_0	PYZ_7	PYZ_6	PYZ_5	PYZ_4	PYZ_3
Shift 8	PYZ_2	PYZ_1	PYZ_0	PY_8	PY_7	PY_6	PY_5	PY_4
Shift 9	PY_3	PY_2	PY_1	PY_0	PZ_6	PZ_5	PZ_4	PZ_3
Shift 10	PZ_2	PZ_1	PZ_0	X	X	X	X	X

/VALID:	Indicator for valid data in pipeline
XI9:	ξ sign bit
PZ0..6:	P_z from LUT 31: $P_z = R3 * P$
PY0..7:	P_y from LUT 30: $P_y = R2 * P$
PX0..8:	P_x from LUT 28 + 29: $P_x = R1 * P$
ID0, ID1:	Pretrigger identifier
ETA8:	η; y coordinate at TC2 (sign bit)
EB0..EB6	E_b: $E_b = (q > 0.?\ E^+{}_b : E^-{}_b)$
Q:	Charge from LUT 16 + 18: $q = \text{sign}(d\xi_m)$
DDXI0..1:	$dd\xi$ from LUT 37
ACCEB1, ACCE31..32:	Accept outputs from LUT 38 + LUT 39 -> CUT3
PX_0..8:	P_x from LUT 35 + 36: $P_x = P_z * \xi_m/z_m$
PYZ_0..7:	P_{yz} from LUT 34: $P_{yz} = P_z * \sqrt{(1 + \eta^2)}$
PY_0..8:	P_y from LUT 32 + 33: $P_y = P_z * \eta$
PZ_0..6:	P_z from LUT 16 + 18: $P_z = 0.003 * B * l * (1/\tan\varphi - \tan\varphi/4) * (1 - \xi_m^2/2z^2_m)$
X:	Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P9

```
PTBF1 (12 REG)  -> P9SER1 :> P9PT0 :> P9PT1 :> P9PT2 :> P9PT3
                          :> P9PT4 :> P9PT5 :> P9PT6 :> P9PT7
                          :> P9ID0 :> P9ID1
                          :> P9P1_0 :> P1P1_1
                -> P9SER2
P1BF1 (12 REG)  -> P9SER2 :> P9P1_2 :> P9P1_3 :> P9P1_4 :> P9P1_5 :> P9P1_6
                          :> P9Q
                          :> P9PY_0 :> P9PY_1 :> P9PY_2 :> P9PY_3
                          :> P9PY_4 :> P9PY_5
                -> P9SER3
PYBF3 (13 REG)  -> P9SER3 :> P9PY_6 :> P9PY_7 :> P9PY_8
                          :> P9PX_0 :> P9PX_1 :> P9PX_2 :> P9PX_3 :> P9PX_4
                          :> P9PX_5 :> P9PX_6 :> P9PX_7 :> P9PX_8 :> P9PX_9
                -> P9SER4
PZEBF1 (14 REG) -> P9SER4 :> P9PZ_0 :> P9PZ_1 :> P9PZ_2 :> P9PZ_3
                          :> P9PZ_4 :> P9PZ_5 :> P9PZ_6
                          :> P9EB0 :> P9EB1 :> P9EB2 :> P9EB3
                          :> P9EB4 :> P9EB5 :> P9EB6
                -> P9SER5
PIPELIN1 (1 REG)-> P9SER5 :> P9MUX
                -> P9SER6
PBUF1 (7 REG)   -> P9SER6 :> P9P0 :> P9P1 :> P9P2 :> P9P2
                          :> P9P4 :> P9P5 :> P9P6
                -> P9SER7
BXREG1 (8 REG)  -> P9SER7 :> P9BX0 :> P9BX1 :> P9BX2 :> P9BX3
                          :> P9BX4 :> P9BX5 :> P9BX6 :> P9BX7
                -> P9SER8
COMP3  (9 REG)  -> P9SER8 :> P9PZ0 :> P9PZ1 :> P9PZ2 :> P9PZ3
                          :> P9PZ4 :> P9PZ5 :> P9PZ6
                          :> P9DDXI0 :> P9DDXI1
                -> P9DDXI1
PY_BF1 (16 REG) -> P9DDXI1:> P9PY0 :> P9PY1 :> P9PY2 :> P9PY3 :> P9PY4
                          :> P9PY5 :> P9PY6 :> P9PY7 :> P9PY8
                          :> P9ACCEPT
                          :> P9PX0 :> P9PX1 :> P9PX2
                          :> P9PX3 :> P9PX4 :> P9PX5
                -> P9SER10
PX_BF1 (5 REG)  -> P9SER10:> P9PX6 :> P9PX7 :> P9PX8 :> P9PX9
                          :> P9CUT3
                -> P9CUT3
CACHE1 (11 REG) -> P9CUT3 :> P9A0 :> P9A1 :> P9A2 :> P9A3 :> P9A4 :> P9A5
                          :> P9A6 :> P9A7 :> P9A8 :> P9A9 :> P9A10
                -> P9A10
CACHE2 (11 REG) -> P9A10  :> P9A11 :> P9A12 :> P9A13
                          :> P9A14 :> P9A15 :> P9A16
                          :> P9A17 :> P9A18 :> P9A19 :> P9A20 :> P9A21
                -> P9A21
LUTC6  (1 REG)  -> P9A21  :> P9_VALID
                -> P9_VALID
TOTAL LENGHT: 12 + 12 + 13 + 14 + 1 + 7 + 8 + 9 + 16 + 5 + 11 + 11 + 1 = 120 REG  (-> Delay
Reg = 0)
```

Pipe 9	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/VALID	A21	A20	A19	A18	A17	A16	A15
shift 2	A14	A13	A12	A11	A10	A9	A8	A7
shift 3	A6	A5	A4	A3	A2	A1	A0	CUT3
shift 4	PX9	PX8	PX7	PX6	PX5	PX4	PX3	PX2
shift 5	PX1	PX0	ACCEPT	PY8	PY7	PY6	PY5	PY4
shift 6	PY3	PY2	PY1	PY0	DDXI1	DDXI0	PZ6	PZ5
shift 7	PZ4	PZ3	PZ2	PZ1	PZ0	BX7	BX6	BX5
shift 8	BX4	BX3	BX2	BX1	BX0	P6	P5	P4
shift 9	P3	P2	P1	P0	MUX	EB6	EB5	EB4
shift 10	EB3	EB2	EB1	EB0	PZ_6	PZ_5	PZ_4	PZ_3
shift 11	PZ_2	PZ_1	PZ_0	PX_9	PX_8	PX_7	PX_6	PX_5
shift 12	PX_4	PX_3	PX_2	PX_1	PX_0	PY_8	PY_7	PY_6
shift 13	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0	Q	P1_6
shift 14	P1_5	P1_4	P1_3	P1_2	P1_1	P1_0	ID1	ID0
shift 15	PT7	PT6	PT5	PT4	PT3	PT2	PT1	PT0

/VALID:	Indicator for valid data in pipeline
A0 .. A7:	Serial test bits for histogram cache1: BX0..BX7
A8:	Serial test bit for histogram cache1: /VALID
A9:	Serial test bit for histogram cache1: CACHERESET
A10:	Serial test bit for histogram cache1: CACHE ENTRY CLEAR
A11 .. A18:	Serial test bits for histogram cache2: BX0..BX7
A8:	Serial test bit for histogram cache2: /VALID
A9:	Serial test bit for histogram cache2: CACHERESET
A10:	Serial test bit for histogram cache2: CACHE ENTRY CLEAR
CUT3:	CUT3 from LUT39
PX0..9:	P_X/P_Z from LUT 44 + 45
ACCEPT	cut on P_T from LUT46
PY0..8:	P_Y/P_Z from LUT 42 + η (sign bit)
DDXI0..1:	ddξ from LUT 37
PZ0..6:	P_z from LUT 31: P_z = R3 * P
BX0..BX7:	BX (bunch-crossing) number
P0..6:	Momentum
MUX:	Signal for ACCEP1-PAL: MUX = ALL & FLAG
EB0..EB6	E_b: $E_b = (q > 0.?\ E^+_b : E^-_b)$
PZ_0..6:	P_z from LUT 16 + 18: $P_z = 0.003 * B * l * (1/\tan\varphi - \tan\varphi/4) * (1 - \xi_m{}^2/2z^2{}_m)$
PX_0..9:	P_x from LUT 44 + 45: $P_x = P_x / P_z$
PY_0..8:	P_y from LUT 42 + 43: $P_y = P_y / P_z$
Q:	Charge from LUT 16 + 18: q = sign (dξ_m)
P1_0..6:	P_1 from LUT 41: $P_1 = \sqrt{(\ p^2{}_{yz} + p^2{}_x\ )}$
ID0, ID1:	Pretrigger identifier
PT0..7:	P_T from LUT 40: $P_T = \sqrt{(\ p^2{}_x + p^2{}_y\ )}$

SERIAL SHIFT PIPELINE STAGE P10

```
PTBF1 (12 REG)  -> P9SER1 :> P10PT0 :> P10PT1 :> P10PT2 :> P10PT3
                          :> P10PT4 :> P10PT5 :> P10PT6 :> P10PT7
                          :> P10ID0 :> P10ID1
                          :> P10P1_0 :> P10P1_1
                -> P10P1_1
P1BF1 ( 6 REG)  -> P10P1_1:> P10P1_2 :> P10P1_3 :> P10P1_4 :> P10P1_5
                          :> P10P1_6
                          :> P10Q
                -> P10Q
COMP1 (51 REG)  -> P10Q   :> P10PX0 :> P10PX1 :> P10PX2 :> P10PX3 :> P10PX4
                          :> P10PX5 :> P10PX6 :> P10PX7 :> P10PX8 :> P10PX9
                          :> P10EQ0 :> P10EQ1 :> P10EQ2 :> P10EQ3
                          :> P11EQ0 :> P11EQ1 :> P11EQ2 :> P11EQ3
                          :> P12EQ0 :> P12EQ1 :> P12EQ2 :> P12EQ3
                          :> P13EQ0 :> P13EQ1 :> P13EQ2 :> P13EQ3
                          :> P14EQ0 :> P14EQ1 :> P14EQ2 :> P14EQ3
                          :> P15EQ0 :> P15EQ1 :> P15EQ2 :> P15EQ3
                          :> P16EQ0 :> P16EQ1 :> P16EQ2 :> P16EQ3
                          :> P17EQ0 :> P17EQ1 :> P17EQ2 :> P17EQ3
                          :> P18EQ0 :> P18EQ1 :> P18EQ2 :> P18EQ3
                          :> P19EQ0 :> P19EQ1 :> P19EQ2 :> P19EQ3
                          :> P10ACCEPT
               -> P10SER4
COMP2 (50 REG) -> P10SER4 :> P10PY0 :> P10PY1 :> P10PY2 :> P10PY3 :> P10PY4
                          :> P10PY5 :> P10PY6 :> P10PY7 :> P10PY8
                          :> P10EQ0 :> P10EQ1 :> P10EQ2 :> P10EQ3
                          :> P11EQ0 :> P11EQ1 :> P11EQ2 :> P11EQ3
                          :> P12EQ0 :> P12EQ1 :> P12EQ2 :> P12EQ3
                          :> P13EQ0 :> P13EQ1 :> P13EQ2 :> P13EQ3
                          :> P14EQ0 :> P14EQ1 :> P14EQ2 :> P14EQ3
                          :> P15EQ0 :> P15EQ1 :> P15EQ2 :> P15EQ3
                          :> P16EQ0 :> P16EQ1 :> P16EQ2 :> P16EQ3
                          :> P17EQ0 :> P17EQ1 :> P17EQ2 :> P17EQ3
                          :> P18EQ0 :> P18EQ1 :> P18EQ2 :> P18EQ3
                          :> P19EQ0 :> P19EQ1 :> P19EQ2 :> P19EQ3
                          :> P10MUX
               -> P10SER5
PZEBF1 (7 REG) -> P10SER5 :> P10PZ_0 :> P10PZ_1 :> P10PZ_2 :> P10PZ_3
                          :> P10PZ_4 :> P10PZ_5 :> P10PZ_6
               -> P10PZ_6
PZCOR1 (5 REG) -> P10PZ_6 :> P10CORR0 :> P10CORR1 :> P10CORR2
                          :> P10CORR3 :> P10CORR4
                   -> P10SER7
COMP3 (41 REG) -> P10SER7 :> P10BX0 :> P10BX1 :> P10BX2 :> P10BX3
                          :> P10BX4 :> P10BX5 :> P10BX6 :> P10BX7
                          :> P10ID0 :> P10ID1
                          :> P10EQ0 :> P10EQ1 :> P10EQ2
                          :> P11EQ0 :> P11EQ1 :> P11EQ2
                          :> P12EQ0 :> P12EQ1 :> P12EQ2
                          :> P13EQ0 :> P13EQ1 :> P13EQ2
                          :> P14EQ0 :> P14EQ1 :> P14EQ2
                          :> P15EQ0 :> P15EQ1 :> P15EQ2
                          :> P16EQ0 :> P16EQ1 :> P16EQ2
                          :> P17EQ0 :> P17EQ1 :> P17EQ2
                          :> P18EQ0 :> P18EQ1 :> P18EQ2
                          :> P19EQ0 :> P19EQ1 :> P19EQ2
                          :> P10MUX
             -> P10MUX
PX_BF1(8 REG)-> P10ACCEPT :> P10CUT3
                          :> P10P0 :> P10P1 :> P10P2 :> P10P3
                          :> P10P4 :> P10P5 :> P10P6
             -> P10P6
CACHE1 (23 REG)-> P10P6   :> P10BX0 :> P10BX1 :> P10BX2
                          :> P10Q0 :> P10Q1 :> P10Q2 :> P10Q3 :> P10Q4
                          :> P10R0 :> P10R1 :> P10R2 :> P10R3 :> P10R4
                          :> P10S0 :> P10S1 :> P10S2 :> P10S3 :> P10S4
                          :> P10T0 :> P10T1 :> P10T2 :> P10T3 :> P10T4
               -> P10T4
CACHE2 (23 REG)-> P10T4   :> P10BX0 :> P10BX1 :> P10BX2
                          :> P10Q0 :> P10Q1 :> P10Q2 :> P10Q3 :> P10Q4
                          :> P10R0 :> P10R1 :> P10R2 :> P10R3 :> P10R4
                          :> P10S0 :> P10S1 :> P10S2 :> P10S3 :> P10S4
                          :> P10T0 :> P10T1 :> P10T2 :> P10T3 :> P10T4
               -> P10T4OUT
COMP4 (44 REG) -> P10T4OUT :> P20EQ0 :> P20EQ1 :> P20EQ2 :> P20EQ3
                           :> P21EQ0 :> P21EQ1 :> P21EQ2 :> P21EQ3
                           :> P22EQ0 :> P22EQ1 :> P22EQ2 :> P22EQ3
                           :> P23EQ0 :> P23EQ1 :> P23EQ2 :> P23EQ3
                           :> P24EQ0 :> P24EQ1 :> P24EQ2 :> P24EQ3
                           :> P25EQ0 :> P25EQ1 :> P25EQ2 :> P25EQ3
                           :> P26EQ0 :> P26EQ1 :> P26EQ2 :> P26EQ3
                           :> P27EQ0 :> P27EQ1 :> P27EQ2 :> P27EQ3
```

```
                          :> P28EQ0 :> P28EQ1 :> P28EQ2 :> P28EQ3
                          :> P29EQ0 :> P29EQ1 :> P29EQ2 :> P29EQ3
                          :> P30EQ0 :> P30EQ1 :> P30EQ2 :> P30EQ3
                -> P30EQ3IN
COMP5 (44 REG)  -> P30EQ3IN :> P20EQ0 :> P20EQ1 :> P20EQ2 :> P20EQ3
                          :> P21EQ0 :> P21EQ1 :> P21EQ2 :> P21EQ3
                          :> P22EQ0 :> P22EQ1 :> P22EQ2 :> P22EQ3
                          :> P23EQ0 :> P23EQ1 :> P23EQ2 :> P23EQ3
                          :> P24EQ0 :> P24EQ1 :> P24EQ2 :> P24EQ3
                          :> P25EQ0 :> P25EQ1 :> P25EQ2 :> P25EQ3
                          :> P26EQ0 :> P26EQ1 :> P26EQ2 :> P26EQ3
                          :> P27EQ0 :> P27EQ1 :> P27EQ2 :> P27EQ3
                          :> P28EQ0 :> P28EQ1 :> P28EQ2 :> P28EQ3
                          :> P29EQ0 :> P29EQ1 :> P29EQ2 :> P29EQ3
                          :> P30EQ0 :> P30EQ1 :> P30EQ2 :> P30EQ3
                -> P30EQ3
COMP6 (33 REG)  -> P30EQ3 :> P20EQ0 :> P20EQ1 :> P20EQ2
                          :> P21EQ0 :> P21EQ1 :> P21EQ2
                          :> P22EQ0 :> P22EQ1 :> P22EQ2
                          :> P23EQ0 :> P23EQ1 :> P23EQ2
                          :> P24EQ0 :> P24EQ1 :> P24EQ2
                          :> P25EQ0 :> P25EQ1 :> P25EQ2
                          :> P26EQ0 :> P26EQ1 :> P26EQ2
                          :> P27EQ0 :> P27EQ1 :> P27EQ2
                          :> P28EQ0 :> P28EQ1 :> P28EQ2
                          :> P29EQ0 :> P29EQ1 :> P29EQ2
                          :> P30EQ0 :> P30EQ1 :> P30EQ2
                 -> P30EQ2
LUTC6 (1 REG)    -> P30EQ2 :> P10_VALID
                 -> P10_VALID
OUTRG1 (1 REG)   -> P10_VALID :> P10_TEST
                 -> P10_TEST
TOTAL LENGHT: 12 + 6 + 51 + 50 + 7 + 5 + 41 + 8 + 23 + 23 + 44 + 44 + 33 + 1 + 1 = 349 REG (->
Delay Reg = 3)
```

Pipe 10	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/TEST	/VALID	P30EQ2	P30EQ1	P30EQ0	P29EQ2	P29EQ1	P29EQ0
shift 2	P28EQ2	P28EQ1	P28EQ0	P27EQ2	P27EQ1	P27EQ0	P26EQ2	P26EQ1
shift 3	P26EQ0	P25EQ2	P25EQ1	P25EQ0	P24EQ2	P24EQ1	P24EQ0	P23EQ2
shift 4	P23EQ1	P23EQ0	P22EQ2	P22EQ1	P22EQ0	P21EQ2	P21EQ1	P21EQ0
shift 5	P20EQ2	P20EQ1	P20EQ0	P30EQ3	P30EQ2	P30EQ1	P30EQ0	P29EQ3
shift 6	P29EQ2	P29EQ1	P29EQ0	P28EQ3	P28EQ2	P28EQ1	P28EQ0	P27EQ3
shift 7	P27EQ2	P27EQ1	P27EQ0	P26EQ3	P26EQ2	P26EQ1	P26EQ0	P25EQ3
shift 8	P25EQ2	P25EQ1	P25EQ0	P24EQ3	P24EQ2	P24EQ1	P24EQ0	P23EQ3
shift 9	P23EQ2	P23EQ1	P23EQ0	P22EQ3	P22EQ2	P22EQ1	P22EQ0	P21EQ3
shift 10	P21EQ2	P21EQ1	P21EQ0	P20EQ3	P20EQ2	P20EQ1	P20EQ0	P30EQ3
shift 11	P30EQ1	P30EQ2	P30EQ0	P29EQ3	P29EQ2	P29EQ1	P29EQ0	P28EQ3
shift 12	P28EQ2	P28EQ1	P28EQ0	P27EQ3	P27EQ2	P27EQ1	P27EQ0	P26EQ3
shift 13	P26EQ2	P26EQ1	P26EQ0	P25EQ3	P25EQ2	P25EQ1	P25EQ0	P24EQ3
shift 14	P24EQ2	P24EQ1	P24EQ0	P23EQ3	P23EQ2	P23EQ1	P23EQ0	P22EQ3
shift 15	P22EQ2	P22EQ1	P22EQ0	P21EQ3	P21EQ2	P21EQ1	P21EQ0	P20EQ3
shift 16	P20EQ2	P20EQ1	P20EQ0	T4	T3	T2	T1	T0
Shift 17	S4	S3	S2	S1	S0	R4	R3	R2
Shift 18	R1	R0	Q4	Q3	Q2	Q1	Q0	BX2
Shift 19	BX1	BX0	T4	T3	T2	T1	T0	S4
Shift 20	S3	S2	S1	S0	R4	R3	R2	R1
Shift 21	R0	Q4	Q3	Q2	Q1	Q0	BX2	BX1
shift 22	BX0	P6	P5	P4	P3	P2	P1	P0
shift 23	CUT3	MUX	P19EQ2	P19EQ1	P19EQ0	P18EQ2	P18EQ1	P18EQ0
shift 24	P17EQ2	P17EQ1	P17EQ0	P16EQ2	P16EQ1	P16EQ0	P15EQ2	P15EQ1
shift 25	P15EQ0	P14EQ2	P14EQ1	P14EQ0	P13EQ2	P13EQ1	P13EQ0	P12EQ2
shift 26	P12EQ1	P12EQ0	P11EQ2	P11EQ1	P11EQ0	P10EQ2	P10EQ1	P10EQ0
shift 27	ID1	ID0	BX7	BX6	BX5	BX4	BX3	BX2
shift 28	BX1	BX0	CORR4	CORR3	CORR2	CORR1	CORR0	PZ_6
shift 29	PZ_5	PZ_4	PZ_3	PZ_2	PZ_1	PZ_0	MUX	P19EQ3
shift 30	P19EQ2	P19EQ1	P19EQ0	P18EQ3	P18EQ2	P18EQ1	P18EQ0	P17EQ3
shift 31	P17EQ2	P17EQ1	P17EQ0	P16EQ3	P16EQ2	P16EQ1	P16EQ0	P15EQ3
shift 32	P15EQ2	P15EQ1	P15EQ0	P14EQ3	P14EQ2	P14EQ1	P14EQ0	P13EQ3
shift 33	P13EQ2	P13EQ1	P13EQ0	P12EQ3	P12EQ2	P12EQ1	P12EQ0	P11EQ3
shift 34	P11EQ2	P11EQ1	P11EQ0	P10EQ3	P10EQ2	P10EQ1	P10EQ0	PY8
shift 35	PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0
shift 36	ACCEPT	P19EQ3	P19EQ2	P19EQ1	P19EQ0	P18EQ3	P18EQ2	P18EQ1
shift 37	P18EQ0	P17EQ3	P17EQ2	P17EQ1	P17EQ0	P16EQ3	P16EQ2	P16EQ1
shift 38	P16EQ0	P15EQ3	P15EQ2	P15EQ1	P15EQ0	P14EQ3	P14EQ2	P14EQ1
shift 39	P14EQ0	P13EQ3	P13EQ2	P13EQ1	P13EQ0	P12EQ3	P12EQ2	P12EQ1
shift 40	P12EQ0	P11EQ3	P11EQ2	P11EQ1	P11EQ0	P10EQ3	P10EQ2	P10EQ1
shift 41	P10EQ0	PX9	PX8	PX7	PX6	PX5	PX4	PX3
shift 42	PX2	PX1	PX0	Q	P1_6	P1_5	P1_4	P1_3
shift 43	P1_2	P1_1	P1_0	ID1	ID0	PT7	PT6	PT5
shift 44	PT4	PT3	PT2	PT1	PT0	X	X	X

/TEST: Indicator for test data (from random generator)
/VALID: Indicator for valid data in pipeline

P30EQ2:	Comparator 6 equal bit: P30EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 30
P30EQ1:	Comparator 6 equal bit: P30EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 30
P30EQ0:	Comparator 6 equal bit: P30EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline30
P29EQ2:	Comparator 6 equal bit: P29EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 29
P29EQ1:	Comparator 6 equal bit: P29EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 29
P29EQ0:	Comparator 6 equal bit: P29EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline29
P28EQ2:	Comparator 6 equal bit: P28EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 28
P28EQ1:	Comparator 6 equal bit: P28EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 28
P28EQ0:	Comparator 6 equal bit: P28EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline28
P27EQ2:	Comparator 6 equal bit: P27EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 27
P27EQ1:	Comparator 6 equal bit: P27EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 27
P27EQ0:	Comparator 6 equal bit: P27EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline27
P26EQ2:	Comparator 6 equal bit: P26EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 26
P26EQ1:	Comparator 6 equal bit: P26EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 26
P26EQ0:	Comparator 6 equal bit: P26EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline26
P25EQ2:	Comparator 6 equal bit: P25EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 25
P25EQ1:	Comparator 6 equal bit: P25EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 25
P25EQ0:	Comparator 6 equal bit: P25EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline25
P24EQ2:	Comparator 6 equal bit: P24EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 24
P24EQ1:	Comparator 6 equal bit: P24EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 24
P24EQ0:	Comparator 6 equal bit: P24EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline24
P23EQ2:	Comparator 6 equal bit: P23EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 23
P23EQ1:	Comparator 6 equal bit: P23EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 23
P23EQ0:	Comparator 6 equal bit: P23EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline23
P22EQ2:	Comparator 6 equal bit: P22EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 22
P22EQ1:	Comparator 6 equal bit: P22EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 22
P22EQ0:	Comparator 6 equal bit: P22EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline22
P21EQ2:	Comparator 6 equal bit: P21EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 21
P21EQ1:	Comparator 6 equal bit: P21EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 21
P21EQ0:	Comparator 6 equal bit: P21EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline21
P20EQ2:	Comparator 6 equal bit: P20EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 20
P20EQ1:	Comparator 6 equal bit: P20EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 20
P20EQ0:	Comparator 6 equal bit: P20EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline20

P30EQ3:	Comparator 5 equal bit: P30EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 30
P30EQ2:	Comparator 5 equal bit: P30EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 30
P30EQ1:	Comparator 5 equal bit: P30EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 30
P30EQ0:	Comparator 5 equal bit: P30EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline30
P29EQ3:	Comparator 5 equal bit: P29EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 29
P29EQ2:	Comparator 5 equal bit: P29EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 29
P29EQ1:	Comparator 5 equal bit: P29EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 29
P29EQ0:	Comparator 5 equal bit: P29EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline29
P28EQ3:	Comparator 5 equal bit: P28EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 28
P28EQ2:	Comparator 5 equal bit: P28EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 28
P28EQ1:	Comparator 5 equal bit: P28EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 28
P28EQ0:	Comparator 5 equal bit: P28EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline28
P27EQ3:	Comparator 5 equal bit: P27EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 27
P27EQ2:	Comparator 5 equal bit: P27EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 27
P27EQ1:	Comparator 5 equal bit: P27EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 27
P27EQ0:	Comparator 5 equal bit: P27EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline27
P26EQ3:	Comparator 5 equal bit: P26EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 26
P26EQ2:	Comparator 5 equal bit: P26EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 26
P26EQ1:	Comparator 5 equal bit: P26EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 26
P26EQ0:	Comparator 5 equal bit: P26EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline26
P25EQ3:	Comparator 5 equal bit: P25EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 25
P25EQ2:	Comparator 5 equal bit: P25EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 25
P25EQ1:	Comparator 5 equal bit: P25EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 25
P25EQ0:	Comparator 5 equal bit: P25EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline25
P24EQ3:	Comparator 5 equal bit: P24EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 24
P24EQ2:	Comparator 5 equal bit: P24EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 24
P24EQ1:	Comparator 5 equal bit: P24EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 24
P24EQ0:	Comparator 5 equal bit: P24EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline24
P23EQ3:	Comparator 5 equal bit: P23EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 23
P23EQ2:	Comparator 5 equal bit: P23EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 23
P23EQ1:	Comparator 5 equal bit: P23EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 23
P23EQ0:	Comparator 5 equal bit: P23EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline23
P22EQ3:	Comparator 5 equal bit: P22EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 22
P22EQ2:	Comparator 5 equal bit: P22EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 22
P22EQ1:	Comparator 5 equal bit: P22EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 22
P22EQ0:	Comparator 5 equal bit: P22EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline22
P21EQ3:	Comparator 5 equal bit: P21EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 21
P21EQ2:	Comparator 5 equal bit: P21EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 21
P21EQ1:	Comparator 5 equal bit: P21EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 21
P21EQ0:	Comparator 5 equal bit: P21EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline21
P20EQ3:	Comparator 5 equal bit: P20EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 20
P20EQ2:	Comparator 5 equal bit: P20EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 20
P20EQ1:	Comparator 5 equal bit: P20EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 20
P20EQ0:	Comparator 5 equal bit: P20EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline20

P30EQ3:	Comparator 4 equal bit: P30EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 30
P30EQ2:	Comparator 4 equal bit: P30EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 30
P30EQ1:	Comparator 4 equal bit: P30EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 30
P30EQ0:	Comparator 4 equal bit: P30EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline30
P29EQ3:	Comparator 4 equal bit: P29EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 29
P29EQ2:	Comparator 4 equal bit: P29EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 29
P29EQ1:	Comparator 4 equal bit: P29EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 29
P29EQ0:	Comparator 4 equal bit: P29EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline29
P28EQ3:	Comparator 4 equal bit: P28EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 28
P28EQ2:	Comparator 4 equal bit: P28EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 28
P28EQ1:	Comparator 4 equal bit: P28EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 28
P28EQ0:	Comparator 4 equal bit: P28EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline28
P27EQ3:	Comparator 4 equal bit: P27EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 27
P27EQ2:	Comparator 4 equal bit: P27EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 27
P27EQ1:	Comparator 4 equal bit: P27EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 27
P27EQ0:	Comparator 4 equal bit: P27EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline27
P26EQ3:	Comparator 4 equal bit: P26EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 26
P26EQ2:	Comparator 4 equal bit: P26EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 26
P26EQ1:	Comparator 4 equal bit: P26EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 26
P26EQ0:	Comparator 4 equal bit: P26EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline26
P25EQ3:	Comparator 4 equal bit: P25EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 25
P25EQ2:	Comparator 4 equal bit: P25EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 25
P25EQ1:	Comparator 4 equal bit: P25EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 25
P25EQ0:	Comparator 4 equal bit: P25EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline25
P24EQ3:	Comparator 4 equal bit: P24EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 24
P24EQ2:	Comparator 4 equal bit: P24EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 24
P24EQ1:	Comparator 4 equal bit: P24EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 24
P24EQ0:	Comparator 4 equal bit: P24EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline24
P23EQ3:	Comparator 4 equal bit: P23EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 23
P23EQ2:	Comparator 4 equal bit: P23EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 23
P23EQ1:	Comparator 4 equal bit: P23EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 23
P23EQ0:	Comparator 4 equal bit: P23EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline23
P22EQ3:	Comparator 4 equal bit: P22EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 22
P22EQ2:	Comparator 4 equal bit: P22EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 22
P22EQ1:	Comparator 4 equal bit: P22EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 22
P22EQ0:	Comparator 4 equal bit: P22EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline22
P21EQ3:	Comparator 4 equal bit: P21EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 21
P21EQ2:	Comparator 4 equal bit: P21EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 21
P21EQ1:	Comparator 4 equal bit: P21EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 21
P21EQ0:	Comparator 4 equal bit: P21EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline21
P20EQ3:	Comparator 4 equal bit: P20EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 20
P20EQ2:	Comparator 4 equal bit: P20EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 20
P20EQ1:	Comparator 4 equal bit: P20EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 20
P20EQ0:	Comparator 4 equal bit: P20EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline20

T0..4	Histogram cache 2 entry counter 6/7 output = [Q4,Q3,Q2,Q1,Q0]; [BX2..0] = 7
S0..4	Histogram cache 2 entry counter 4/5 output = [Q4,Q3,Q2,Q1,Q0]; [BX2..0] = 6
R0..4	Histogram cache 2 entry counter 2/3 output = [Q4,Q3,Q2,Q1,Q0]; [BX2..0] = 5
Q0..4	Histogram cache 2 entry counter 0/1 output = [Q4,Q3,Q2,Q1,Q0]; [BX2..0] = 4
BX0..BX2:	BX (bunch-crossing) number histogram cache 2 internal

T0..4	Histogram cache 1 entry counter 6/7 output = [Q4,Q3,Q2,Q1,Q0]; [BX2..0] = 3
S0..4	Histogram cache 1 entry counter 4/5 output = [Q4,Q3,Q2,Q1,Q0]; [BX2..0] = 2
R0..4	Histogram cache 1 entry counter 2/3 output = [Q4,Q3,Q2,Q1,Q0]; [BX2..0] = 1
Q0..4	Histogram cache 1 entry counter 0/1 output = [Q4,Q3,Q2,Q1,Q0]; [BX2..0] = 0
BX0..BX2:	BX (bunch-crossing) number histogram cache 2 internal

P0..6:	Momentum
CUT3:	CUT3 from LUT39
MUX:	Signal for ACCEP1-PAL: MUX = ALL & FLAG

P19EQ2:	Comparator 3 equal bit: P19EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 19
P19EQ1:	Comparator 3 equal bit: P19EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 19
P19EQ0:	Comparator 3 equal bit: P19EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline19
P18EQ2:	Comparator 3 equal bit: P18EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 18
P18EQ1:	Comparator 3 equal bit: P18EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 18
P18EQ0:	Comparator 3 equal bit: P18EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline18
P17EQ2:	Comparator 3 equal bit: P17EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 17
P17EQ1:	Comparator 3 equal bit: P17EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 17
P17EQ0:	Comparator 3 equal bit: P17EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline17
P16EQ2:	Comparator 3 equal bit: P16EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 16
P16EQ1:	Comparator 3 equal bit: P16EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 16
P16EQ0:	Comparator 3 equal bit: P16EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline16
P15EQ2:	Comparator 3 equal bit: P15EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 15
P15EQ1:	Comparator 3 equal bit: P15EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 15
P15EQ0:	Comparator 3 equal bit: P15EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline15
P14EQ2:	Comparator 3 equal bit: P14EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 14
P14EQ1:	Comparator 3 equal bit: P14EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 14
P14EQ0:	Comparator 3 equal bit: P14EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline14
P13EQ2:	Comparator 3 equal bit: P13EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 13
P13EQ1:	Comparator 3 equal bit: P13EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 13
P13EQ0:	Comparator 3 equal bit: P13EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline13
P12EQ2:	Comparator 3 equal bit: P12EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 12
P12EQ1:	Comparator 3 equal bit: P12EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 12
P12EQ0:	Comparator 3 equal bit: P12EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline12
P11EQ2:	Comparator 3 equal bit: P11EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 11
P11EQ1:	Comparator 3 equal bit: P11EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 11
P11EQ0:	Comparator 3 equal bit: P11EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline11
P10EQ2:	Comparator 3 equal bit: P10EQ2 = [ID1,ID0,BX7].pipeline 9 == [ID1,ID0,BX7].pipeline 10
P10EQ1:	Comparator 3 equal bit: P10EQ1 = [BX6,BX5,BX4].pipeline 9 == [BX6,BX5,BX4].pipeline 10
P10EQ0:	Comparator 3 equal bit: P10EQ0 = [BX3,BX2,BX1,BX0].pipeline 9 == [BX3,BX2,BX1,BX0].pipeline10

ID0, ID1:	Pretrigger identifier
BX0..BX6:	BX (bunch-crossing) number
CORR0..4:	corr from LUT 49: corr = $1 + E_b/p_1$
PZ_0..6:	P_z from LUT 16 + 18: $P_z = 0.003 * B * l * (1/\tan\varphi - \tan\varphi/4) * (1 - \xi_m^2/2z^2_m)$
MUX:	Signal for ACCEP1-PAL: MUX = ALL & FLAG

P19EQ3:	Comparator 2 equal bit: P19EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 19
P19EQ2:	Comparator 2 equal bit: P19EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 19
P19EQ1:	Comparator 2 equal bit: P19EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 19
P19EQ0:	Comparator 2 equal bit: P19EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline19
P18EQ3:	Comparator 2 equal bit: P18EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 18
P18EQ2:	Comparator 2 equal bit: P18EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 18
P18EQ1:	Comparator 2 equal bit: P18EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 18
P18EQ0:	Comparator 2 equal bit: P18EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline18
P17EQ3:	Comparator 2 equal bit: P17EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 17
P17EQ2:	Comparator 2 equal bit: P17EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 17
P17EQ1:	Comparator 2 equal bit: P17EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 17
P17EQ0:	Comparator 2 equal bit: P17EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline17
P16EQ3:	Comparator 2 equal bit: P16EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 16
P16EQ2:	Comparator 2 equal bit: P16EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 16
P16EQ1:	Comparator 2 equal bit: P16EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 16
P16EQ0:	Comparator 2 equal bit: P16EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline16
P15EQ3:	Comparator 2 equal bit: P15EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 15
P15EQ2:	Comparator 2 equal bit: P15EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 15
P15EQ1:	Comparator 2 equal bit: P15EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 15
P15EQ0:	Comparator 2 equal bit: P15EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline15
P14EQ3:	Comparator 2 equal bit: P14EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 14
P14EQ2:	Comparator 2 equal bit: P14EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 14
P14EQ1:	Comparator 2 equal bit: P14EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 14
P14EQ0:	Comparator 2 equal bit: P14EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline14
P13EQ3:	Comparator 2 equal bit: P13EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 13
P13EQ2:	Comparator 2 equal bit: P13EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 13
P13EQ1:	Comparator 2 equal bit: P13EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 13
P13EQ0:	Comparator 2 equal bit: P13EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline13
P12EQ3:	Comparator 2 equal bit: P12EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 12
P12EQ2:	Comparator 2 equal bit: P12EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 12
P12EQ1:	Comparator 2 equal bit: P12EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 12
P12EQ0:	Comparator 2 equal bit: P12EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline12
P11EQ3:	Comparator 2 equal bit: P11EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 11
P11EQ2:	Comparator 2 equal bit: P11EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 11
P11EQ1:	Comparator 2 equal bit: P11EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 11
P11EQ0:	Comparator 2 equal bit: P11EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline11
P10EQ3:	Comparator 2 equal bit: P10EQ3 = [PY1,PY0].pipeline 9 ≈ [PY1,PY0].pipeline 10
P10EQ2:	Comparator 2 equal bit: P10EQ2 = [PY4,PY3,PY2].pipeline 9 == [PY4,PY3,PY2].pipeline 10
P10EQ1:	Comparator 2 equal bit: P10EQ1 = [PY6,PY5].pipeline 9 == [PY6,PY5].pipeline 10
P10EQ0:	Comparator 2 equal bit: P10EQ0 = [PY8,PY7].pipeline 9 == [PY8,PY7].pipeline10

PY0..8:	P_y from LUT 42 + 43 (!MUX) or from P_Y/P_Z from LUT 42 + η (sign bit) (MUX): $P_y = P_y / P_z$
ACCEPT	cut on P_T from LUT46

P19EQ3:	Comparator 1 equal bit: P19EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 19
P19EQ2:	Comparator 1 equal bit: P19EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 19
P19EQ1:	Comparator 1 equal bit: P19EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 19
P19EQ0:	Comparator 1 equal bit: P19EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline19
P18EQ3:	Comparator 1 equal bit: P18EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 18
P18EQ2:	Comparator 1 equal bit: P18EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 18
P18EQ1:	Comparator 1 equal bit: P18EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 18
P18EQ0:	Comparator 1 equal bit: P18EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline18
P17EQ3:	Comparator 1 equal bit: P17EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 17
P17EQ2:	Comparator 1 equal bit: P17EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 17
P17EQ1:	Comparator 1 equal bit: P17EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 17
P17EQ0:	Comparator 1 equal bit: P17EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline17
P16EQ3:	Comparator 1 equal bit: P16EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 16
P16EQ2:	Comparator 1 equal bit: P16EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 16
P16EQ1:	Comparator 1 equal bit: P16EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 16
P16EQ0:	Comparator 1 equal bit: P16EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline16
P15EQ3:	Comparator 1 equal bit: P15EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 15
P15EQ2:	Comparator 1 equal bit: P15EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 15
P15EQ1:	Comparator 1 equal bit: P15EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 15
P15EQ0:	Comparator 1 equal bit: P15EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline15
P14EQ3:	Comparator 1 equal bit: P14EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 14
P14EQ2:	Comparator 1 equal bit: P14EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 14
P14EQ1:	Comparator 1 equal bit: P14EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 14
P14EQ0:	Comparator 1 equal bit: P14EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline14
P13EQ3:	Comparator 1 equal bit: P13EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 13
P13EQ2:	Comparator 1 equal bit: P13EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 13
P13EQ1:	Comparator 1 equal bit: P13EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 13
P13EQ0:	Comparator 1 equal bit: P13EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline13
P12EQ3:	Comparator 1 equal bit: P12EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 12
P12EQ2:	Comparator 1 equal bit: P12EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 12
P12EQ1:	Comparator 1 equal bit: P12EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 12
P12EQ0:	Comparator 1 equal bit: P12EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline12
P11EQ3:	Comparator 1 equal bit: P11EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 11
P11EQ2:	Comparator 1 equal bit: P11EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 11
P11EQ1:	Comparator 1 equal bit: P11EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 11
P11EQ0:	Comparator 1 equal bit: P11EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline11
P10EQ3:	Comparator 1 equal bit: P10EQ3 = [PX1,PX0].pipeline 9 ≈ [PX1,PX0].pipeline 10
P10EQ2:	Comparator 1 equal bit: P10EQ2 = [PX4,PX3,PX2].pipeline 9 == [PX4,PX3,PX2].pipeline 10
P10EQ1:	Comparator 1 equal bit: P10EQ1 = [PX6,PX5].pipeline 9 == [PX6,PX5].pipeline 10
P10EQ0:	Comparator 1 equal bit: P10EQ0 = [PX9,PX8,PX7].pipeline 9 == [PX9,PX8,PX7].pipeline10

PX0..9:	P_X/P_Z from LUT 44 + 45 (!MUX) or from LUT 28 + 29 (MUX): $P_x = P_x / P_z$
Q:	Charge from LUT 16 + 18: $q = \text{sign}(d\xi_m)$
P1_0..6:	P_1 from LUT 41: $P_1 = \sqrt{(p^2_{yz} + p^2_x)}$
ID0, ID1:	Pretrigger identifier
PT0..7:	P_T from LUT 40: $P_T = \sqrt{(p^2_x + p^2_y)}$
X:	Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P11

```
CUTBF1 (12 REG) -> P11SER1 :> P11ID0 :> P11ID1
                           :> P11PTCUT0 :> P11PTCUT1 :> P11PTCUT2
                           :> P11P_CORR0 :> P11P_CORR1 :> P11P_CORR2
                           :> P11P_CORR3
                           :> P11P_CORR4 :> P11P_CORR5 :> P11P_CORR6
                -> P11SER2
COMP1 (11 REG)  -> P11SER2 :> _EQUA1 :> _EQUA2 :> _EQUA3 :> _EQUA4
                           :> _EQUA5
                           :> _EQUA6 :> _EQUA7 :> _EQUA8 :> _EQUA9
                           :> _EQUA10
                           :> P11ACCEPT
                -> P11SER3
COMP2 (11 REG)  -> P11SER3 :> _EQUA1 :> _EQUA2 :> _EQUA3 :> _EQUA4
                           :> _EQUA5
                           :> _EQUA6 :> _EQUA7 :> _EQUA8 :> _EQUA9
                           :> _EQUA10
                           :> P11MUX
                -> P11SER4
PZCOR1 (7 REG)  -> P11SER4 :> PZCORR0 :> PZCORR1 :> PZCORR2 :> PZCORR3
                           :> PZCORR4 :> PZCORR5 :> PZCORR6
                -> PZCORR6
COMP3 (12 REG)  -> PZCORR6 :> _EQUB1 :> _EQUB2 :> _EQUB3 :> EQUB4 :> EQUB5
                           :> _EQUB6 :> _EQUB7 :> _EQUB8 :> EQUB9 :> EQUB10
                           :> P11MUX
                           :> P11Q
                -> P11Q
PX_BF1 (8 REG)  -> P11Q    :> P11CUT3
                           :> P11P0 :> P11P1 :> P11P2 :> P11P3
                           :> P11P4 :> P11P5 :> P11P6
                -> P11SER7
CACHE3 (3 REG)  -> P11SER7 :> P11COMP1 :> P11EQU1 :> P11COMP2
                -> P11SER8
COMP4 (11 REG)  -> P11SER8:> _EQUA11 :> _EQUA12 :> _EQUA13:> _EQUA14
                          :> _EQUA15 :> _EQUA16 :> _EQUA17:> _EQUA18
                          :> _EQUA19 :> _EQUA20 :> _EQUA21
                -> _EQUA21
COMP5 (11 REG)  -> _EQUA21:> _EQUB11 :> _EQUB12 :> _EQUB13:> _EQUB14
                          :> _EQUB15 :> _EQUB16 :> _EQUB17:> _EQUB18
                          :> _EQUB19 :> _EQUB20 :> _EQUB21
                -> _EQUB21
COMP6 (11 REG)  -> _EQUB21:> _EQUC11 :> _EQUC12 :> _EQUC13:> _EQUC14
                          :> _EQUC15 :> _EQUC16 :> _EQUC17:> _EQUC18
                          :> _EQUC19 :> _EQUC20 :> _EQUC21
                -> _EQUC21
LUTC6 (2 REG)   -> _EQUC21 :> P11_VALIDOUT :> P11_VALID
                -> P11_VALID
OUTRG1 (1 REG)  -> P11_VALID :> SIG_LOAD
                -> SIG_LOAD
TOTAL LENGHT: 12 + 11 + 11 + 7 + 12 + 8 + 3 + 11 + 11 + 11 + 2 + 1 = 100 REG (-> Delay Reg =
4)
```

Pipe 11	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/SIGLOAD	/VALID	/VALIDOU	/EQUC21	/EQUC20	/EQUC19	/EQUC18	/EQUC17
shift 2	/EQUC16	/EQUC15	/EQUC14	/EQUC13	/EQUC12	/EQUC11	/EQUB21	/EQUB20
shift 3	/EQUB19	/EQUB18	/EQUB17	/EQUB16	/EQUB15	/EQUB14	/EQUB13	/EQUB12
shift 4	/EQUB11	/EQUA21	/EQUA20	/EQUA19	/EQUA18	/EQUA17	/EQUA16	/EQUA15
shift 5	/EQUA14	/EQUA13	/EQUA12	/EQUA11	COMP2	EQU1	COMP1	P6
shift 6	P5	P4	P3	P2	P1	P0	CUT3	Q
shift 7	MUX	/EQUB10	/EQUB9	/EQUB8	/EQUB7	/EQUB6	/EQUB5	/EQUB4
shift 8	/EQUB3	/EQUB2	/EQUB1	CORR6	CORR5	CORR4	CORR3	CORR2
shift 9	CORR1	CORR0	MUX	/EQUA10	/EQUA9	/EQUA8	/EQUA7	/EQUA6
shift 10	/EQUA5	/EQUA4	/EQUA3	/EQUA2	/EQUA1	ACCEPT	/EQUA10	/EQUA9
shift 11	/EQUA8	/EQUA7	/EQUA6	/EQUA5	/EQUA4	/EQUA3	/EQUA2	/EQUA1
shift 12	P_CORR6	P_CORR5	P_CORR4	P_CORR3	P_CORR2	P_CORR1	P_CORR0	PTCUT2
shift 13	PTCUT1	PTCUT0	ID1	ID0	X	X	X	X

/SIGLOAD: Signatur register load signal: /SIGLOAD = /SIG # /TEST;
/VALID: Indicator for valid data in pipeline
/VALIDOU: Indicator for valid data in pipeline: /VALIDOU = /VALID&TEST;

/EQUC21: Comparator 6 equal bit: /EQUC21 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 30)
/EQUC20: Comparator 6 equal bit: /EQUC20 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 29)
/EQUC19: Comparator 6 equal bit: /EQUC19 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 28)
/EQUC18: Comparator 6 equal bit: /EQUC18 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 27)
/EQUC17: Comparator 6 equal bit: /EQUC17 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 26)
/EQUC16: Comparator 6 equal bit: /EQUC16 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 25)
/EQUC15: Comparator 6 equal bit: /EQUC15 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 24)
/EQUC14: Comparator 6 equal bit: /EQUC14 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 23)
/EQUC13: Comparator 6 equal bit: /EQUC13 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 22)
/EQUC12: Comparator 6 equal bit: /EQUC12 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 21)
/EQUC11: Comparator 6 equal bit: /EQUC11 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 20)

/EQUB21: Comparator 5 equal bit: /EQUB21 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 30
/EQUB20: Comparator 5 equal bit: /EQUB20 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 29
/EQUB19: Comparator 5 equal bit: /EQUB19 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 28
/EQUB18: Comparator 5 equal bit: /EQUB18 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 27
/EQUB17: Comparator 5 equal bit: /EQUB17 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 26
/EQUB16: Comparator 5 equal bit: /EQUB16 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 25
/EQUB15: Comparator 5 equal bit: /EQUB15 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 24
/EQUB14: Comparator 5 equal bit: /EQUB14 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 23
/EQUB13: Comparator 5 equal bit: /EQUB13 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 22
/EQUB12: Comparator 5 equal bit: /EQUB12 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 21
/EQUB11: Comparator 5 equal bit: /EQUB11 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 20

/EQUA21: Comparator 4 equal bit: /EQUA21 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 30
/EQUA20: Comparator 4 equal bit: /EQUA20 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 29
/EQUA19: Comparator 4 equal bit: /EQUA19 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 28
/EQUA18: Comparator 4 equal bit: /EQUA18 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 27
/EQUA17: Comparator 4 equal bit: /EQUA17 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 26
/EQUA16: Comparator 4 equal bit: /EQUA16 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 25
/EQUA15: Comparator 4 equal bit: /EQUA15 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 24
/EQUA14: Comparator 4 equal bit: /EQUA14 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 23
/EQUA13: Comparator 4 equal bit: /EQUA13 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 22
/EQUA12: Comparator 4 equal bit: /EQUA12 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 21
/EQUA11: Comparator 4 equal bit: /EQUA11 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 20

COMP1,2,EQU: Histogram cache control logic.
P0..6: Momentum
CUT3: CUT3 from LUT39
Q: Charge from LUT 16 + 18: q = sign ($d\xi_m$);
MUX: Signal for ACCEP1-PAL: MUX = ALL & FLAG;

/EQUB10: Comparator 3 equal bit: /EQUB10 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 19)
/EQUB9: Comparator 3 equal bit: /EQUB9 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 18)
/EQUB8: Comparator 3 equal bit: /EQUB8 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 17)
/EQUB7: Comparator 3 equal bit: /EQUB7 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 16)
/EQUB6: Comparator 3 equal bit: /EQUB6 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 15)
/EQUB5: Comparator 3 equal bit: /EQUB5 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 14)
/EQUB4: Comparator 3 equal bit: /EQUB4 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 13)
/EQUB3: Comparator 3 equal bit: /EQUB3 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 12)
/EQUB2: Comparator 3 equal bit: /EQUB2 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 11)
/EQUB1: Comparator 3 equal bit: /EQUB1 = ([ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 9
== [ID1,ID0,BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0].pipeline 10)

CORR0..6: P_{zcorr} from LUT 53: $P_{zcorr} = P_z * corr$;
MUX: Signal for ACCEP1-PAL: MUX = ALL & FLAG;

/EQUA10: Comparator 2 equal bit: /EQUA10 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 19

/EQUA9: Comparator 2 equal bit: /EQUA9 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 18

/EQUA8: Comparator 2 equal bit: /EQUA8 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 17

/EQUA7: Comparator 2 equal bit: /EQUA7 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 16

/EQUA6: Comparator 2 equal bit: /EQUA6 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 15

/EQUA5: Comparator 2 equal bit: /EQUA5 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 14

/EQUA4: Comparator 2 equal bit: /EQUA4 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 13

/EQUA3: Comparator 2 equal bit: /EQUA3 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 12

/EQUA2: Comparator 2 equal bit: /EQUA2 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 11

/EQUA1: Comparator 2 equal bit: /EQUA1 = [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 9
≈ [PY8,PY7,PY6,PY5,PY4,PY3,PY2,PY1,PY0].pipeline 10

ACCEPT cut on P_T from LUT46

/EQUA10: Comparator 1 equal bit: /EQUA10 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 19

/EQUA9: Comparator 1 equal bit: /EQUA9 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 18

/EQUA8: Comparator 1 equal bit: /EQUA8 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 17

/EQUA7: Comparator 1 equal bit: /EQUA7 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 16

/EQUA6: Comparator 1 equal bit: /EQUA6 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 15

/EQUA5: Comparator 1 equal bit: /EQUA5 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 14

/EQUA4: Comparator 1 equal bit: /EQUA4 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 13

/EQUA3: Comparator 1 equal bit: /EQUA3 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 12

/EQUA2: Comparator 1 equal bit: /EQUA2 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 11

/EQUA1: Comparator 1 equal bit: /EQUA1 = [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 9
≈ [PX9,PX8,PX7,PX6,PX5,PX4,PX3,PX2,PX1,PX0].pipeline 10

P_CORR0..6: P_{corr} from LUT 52: $P_{1corr} = P_1 * corr$;

PTCUT0..2: cut on P_t from LUT 51

ID0, ID1: Pretrigger identifier

X: Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P12

```
ACCEP1 (5 REG)  -> P12SER1 :> FIFOWR
                           :> P12_WEN
                           :> _TPUCOUNT
                           :> _CACHECOUNT
                           :> _COMPCOUNT
                -> P12SER2
COMP1 (10REG)   -> P12SER2 :> P12PX0 :> P12PX1 :> P12PX2
                           :> P12PX3 :> P12PX4
                           :> P12PX5 :> P12PX6 :> P12PX7
                           :> P12PX8 :> P12PX9
                -> P12SER3
COMP2 (9 REG)   -> P12SER3 :> P12PY0 :> P12PY1 :> P12PY2
                           :> P12PY3 :> P12PY4
                           :> P12PY5 :> P12PY6 :> P12PY7 :> P12PY8
                -> P12SER4
COMP3 (20 REG)  -> P12SER4 :> P12BX0 :> P12BX1 :> P12BX2 :> P12BX3
                           :> P12BX4 :> P12BX5 :> P12BX6 :> P12BX7
                           :> P12ID0 :> P12ID1
                           :> P12PZ0 :> P12PZ1 :> P12PZ2 :> P12PZ3
                           :> P12PZ4 :> P12PZ5 :> P12PZ6
                           :> P12DDXI0 :> P12DDXI1
                           :> P12Q
                -> P12SER5
OUTRG1 (2 REG)  -> P12SER5 :> P12_VALID :> M_DRENI
                -> M_DRENI
TOTAL LENGHT: 5 + 10 + 9 + 20 + 2 = 46 REG (-> Delay Reg = 2)
```

Pipe 12	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	M_DRENI	/VALID	Q	DDXI1	DDXI0	PZ6	PZ5	PZ4
shift 2	PZ3	PZ2	PZ1	PZ0	ID1	ID0	BX7	BX6
shift 3	BX5	BX4	BX3	BX2	BX1	BX0	PY8	PY7
shift 4	PY6	PY5	PY4	PY3	PY2	PY1	PY0	PX9
shift 5	PX8	PX7	PX6	PX5	PX4	PX3	PX2	PX1
shift 6	PX0	/COMPCOU	/CACHECO	/TPUCOUN	/WEN	FIFOWR	X	X

M_DRENI: Read enable message buffer (for the delay fifos on the message board).
/VALID: Indicator for valid data in pipeline
Q: Charge from LUT 16 + 18: q = sign ($d\xi_m$);
DDXI0..1: $dd\xi$ from LUT 37
PZ0..6: P_z from LUT 31: P_z = R3 * P
ID0, ID1: Pretrigger identifier
BX0..BX7: BX (bunch-crossing) number
PY0..8: P_y from LUT 42 + 43 (!MUX) or from P_Y/P_Z from LUT 42 + η (sign bit) (MUX): $P_y = P_y / P_z$
PX0..9: P_X/P_Z from LUT 44 + 45 (!MUX) or from LUT 28 + 29 (MUX): $P_x = P_x / P_z$
/COMPCOUNT: Indicator for tracks suppressed from double track comparator
/CACHECOUNT: Indicator for tracks suppressed from histogram cache
/TPUCOUNT: Indicator for tracks accepted from TPU processor
/WEN: Write indicator for histogram cache control
FIFOWR Message transmitter fifo write signal
X: Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P19

(virtual pipeline stage: only used for test)

```
COMP1 (10 REG)->   P19SER1 :> P19PX0 :> P19PX1 :> P19PX2
                           :> P19PX3 :> P19PX4
                           :> P19PX5 :< P19PX6 :> P19PX7
                           :> P19PX8 :< P19PX9
              ->   P19SER2
COMP2 (10 REG)->   P19SER2 :> P19PY0 :> P19PY1 :> P19PY2 :> P19PY3
                           :> P19PY4
                           :> P19PY5 :< P19PY6 :> P19PY7 :> P19PY8
                           :> P19VAL
              ->   P19SER3
COMP3 (10 REG)->   P19SER3 :> P19BX0 :> P19BX1 :> P19BX2 :> P19BX3
                           :> P19BX4 :> P19BX5 :> P19BX6 :> P19BX7
                           :> P19ID0 :> P10ID1
                  ->   P19ID1
TOTAL LENGHT: 10 + 10 + 10 = 30 REG (-> Delay Reg = 2)
```

Pipe 19	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	ID1	ID0	BX7	BX6	BX5	BX4	BX3	BX2
shift 2	BX1	BX0	VAL	PY8	PY7	PY6	PY5	PY4
shift 3	PY3	PY2	PY1	PY0	PX9	PX8	PX7	PX6
shift 4	PX5	PX4	PX3	PX2	PX1	PX0	X	X

Comparators 1..3 outputs to comparators 4 .. 6:

ID0, ID1: Pretrigger identifier

BX0..BX7: BX (bunch-crossing) number

VAL: Comparator register 19 data valid

PY0..8: P_y from LUT 42 + 43 (!MUX) or from P_Y/P_Z from LUT 42 + η (sign bit) (MUX): $P_y = P_y / P_z$

PX0..9: P_X/P_Z from LUT 44 + 45 (!MUX) or from LUT 28 + 29 (MUX): $P_x = P_x / P_z$

X: Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT PIPELINE STAGE P13/P20

(VIRTUAL PIPELINE STAGE: ONLY USED FOR TEST)

```
COMP4 (10 REG)->   P20SER1 :> P20PX0 :> P20PX1 :> P20PX2
                           :> P20PX3 :> P20PX4
                           :> P20PX5 :< P20PX6 :> P20PX7
                           :> P20PX8 :< P20PX9
              ->   P20SER2
COMP5 (10 REG)->   P20SER2 :> P20PY0 :> P20PY1 :> P20PY2
                           :> P20PY3 :> P20PY4
                           :> P20PY5 :< P20PY6 :> P20PY7 :> P20PY8
                           :> P20VAL
              ->   P20SER3
COMP6 (10 REG)->   P20SER3 :> P20BX0 :> P20BX1 :> P20BX2 :> P20BX3
                           :> P20BX4 :> P20BX5 :> P20BX6 :> P20BX7
                           :> P20ID0 :> P20ID1
              ->   P20ID1
CACHE1 (9 REG)->   P20ID1  :> P13BX0 :> P13BX1 :> P13BX2 :> P13BX3
                           :> P13BX4 :> P13BX5 :> P13BX6 :> P13BX7
                           :> P13FWEN
              ->   P13FWEN
CACHE2 (9 REG)->   P20ID1  :> P13BX0 :> P13BX1 :> P13BX2 :> P13BX3
                           :> P13BX4 :> P13BX5 :> P13BX6 :> P13BX7
                           :> P13FWEN
              ->   P13FWEN
TOTAL LENGHT: 10 + 10 + 10 + 9 + 9 = 48 REG (-> Delay Reg = 0)
```

Pipe 19	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	/WEN	BX7	BX6	BX5	BX4	BX3	BX2	BX1
shift 2	BX0	/WEN	BX7	BX6	BX5	BX4	BX3	BX2
shift 3	BX1	BX0	ID1	ID0	BX7	BX6	BX5	BX4
shift 4	BX3	BX2	BX1	BX0	VAL	PY8	PY7	PY6
shift 4	PY5	PY4	PY3	PY2	PY1	PY0	PX9	PX8
shift 4	PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0

/WEN: Write indicator for histogram cache control (cache 2 input)
BX0..BX7: BX (bunch-crossing) number (cache 2 input)
/WEN: Write indicator for histogram cache control (cache 1 input)
BX0..BX7: BX (bunch-crossing) number (cache 1 input)

Comparator 4 .. 6 inputs to comparator1 1 .. 3:
ID0, ID1: Pretrigger identifier
BX0..BX7: BX (bunch-crossing) number
VAL: Comparator register 19 data valid
PY0..8: P_y from LUT 42 + 43 (!MUX) or from P_Y/P_Z from LUT 42 + η (sign bit) (MUX): $P_y = P_y / P_z$
PX0..9: P_X/P_Z from LUT 44 + 45 (!MUX) or from LUT 28 + 29 (MUX): $P_x = P_x / P_z$

SERIAL SHIFT SIGNATUR REG

```
COMP1 (10 REG)  -> SIGSERIN :> SREG1 :> SREG2 :> SREG3 :> SREG4 :> SREG5
                            :> SREG6 :> SREG7 :> SREG8 :> SREG9 :> SREG10
                -> SREG10
COMP2 (9 REG)   -> SREG10 :> SREG11 :> SREG12 :> SREG13 :> SREG14 :> SRER15
                          :> SREG16 :> SREG17 :> SREG18 :> SREG19
                -> SREG19
COMP3 (10 REG)  -> SREG19 :> SREG20 :> SREG21 :> SREG22 :> SREG23 :> SREG24
                          :> SREG25 :> SREG26 :> SREG27 :> SREG28 :> SREG29
                -> SREG29
OUTRG1 (5 REG)  -> SREG29 :> REG0 :> REG1 :> REG2 :> REG3 :> REG4 :>
                -> REG4
TOTAL LENGHT: 10 + 9 + 10 + 5 =  34 REG (-> Delay Reg = 6)
```

SIG REG	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	REG4	REG3	REG2	REG1	REG0	SREG29	SREG28	SREG27
shift 2	SREG26	SREG25	SREG24	SREG23	SREG22	SREG21	SREG20	SREG19
shift 3	SREG18	SREG17	SREG16	SREG15	SREG14	SREG13	SREG12	SREG11
shift 4	SREG10	SREG9	SREG8	SREG7	SREG6	SREG5	SREG4	SREG3
shift 5	SREG2	SREG1	X	X	X	X	X	X

REG0:	Test register for accept status: REG0 = P12_VALID
REG1:	Test register for accept status: REG1 = WENFIFO
REG2:	Test register for accept status: REG2 = TPUCOUNT
REG3:	Test register for accept status: REG3 = CACHECOUNT
REG4:	Test register for accept status: REG4 = COMPCOUNT
SREG1..10	Signature registers: SREG1..10 = PX0 .. 9
SREG11..19	Signature registers: SREG11..19 = PY0 .. 8
SREG20..26	Signature registers: SREG20..26 = PZ0 .. 6
SREG27..28	Signature registers: SREG27..28 = DDXI0.. 1
SREG29	Signature registers: SREG29 = Q
X:	Delay registers to extend the serial pipeline length to byte boundary

SERIAL SHIFT RANDOM REG

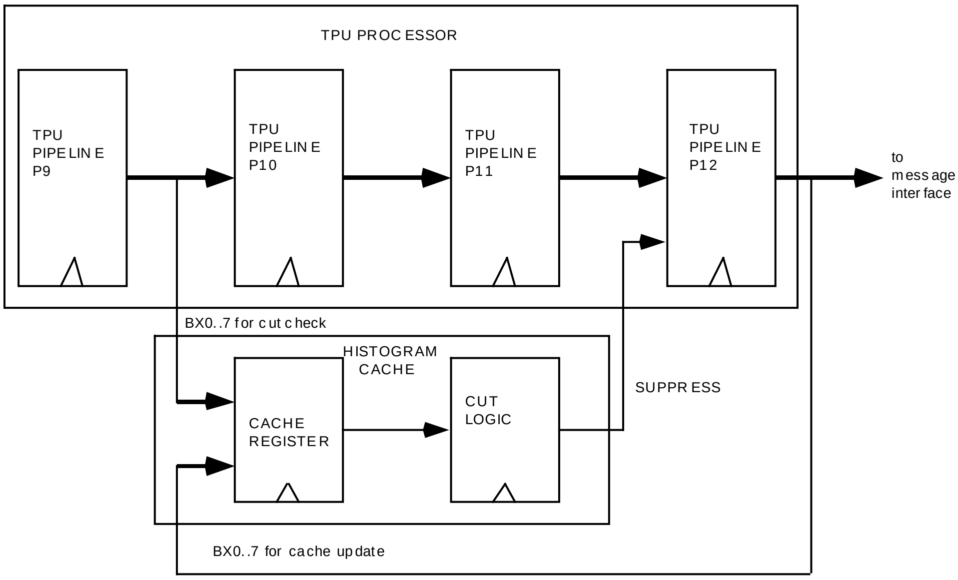
```
XIBUF1 (12 REG) -> SIGSERIN :> RND0 :> RND1 :> RND2 :> RND3 :> RND4 :> RND5
                            :> RND6 :> RND7 :> RND8 :> RND9 :> RND10
                            :> RND11
                -> RND11
DDXI1  (12 REG) -> RND11    :> RND12 :> RND13 :> RND14 :> RND15 :> RND16
                            :> RND17 :> RND18 :> RND19 :> RND20 :> RND21
                            :> RND22 :> RND23
                -> RND23
ETA1   (13 REG) -> RND23    :> RND24 :> RND25 :> RND26 :> RND27 :> RND28
                            :> RND29 :> RND30 :> RND31 :> RND32 :> RND33
                            :> RND34 :> RND35 :> RND36
PIPE1  (33 REG) -> RND36    :> RND37 :> RND38 :> RND39 :> RND40 :> RND41
                            :> RND42 :> RND43 :> RND44 :> RND45 :> RND46
                            :> RND47 :> RND48 :> RND49 :> RND50 :> RND51
                            :> RND52 :> RND53 :> RND54 :> RND55 :> RND56
                            :> RND57 :> RND58 :> RND59 :> RND60 :> RND61
                            :> RND62 :> RND63 :> RND64 :> RND65 :> RND66
                            :> RND67 :> RND68 :> RND69
                -> RNDSEROUT
TOTAL LENGHT: 12 + 12 + 13 + 33 = 70 REG (-> Delay Reg = 2)
```

RND REG	D7	D6	D5	D4	D3	D2	D1	D0
shift 1	RND69	RND68	RND67	RND66	RND65	RND64	RND63	RND62
shift 2	RND61	RND60	RND59	RND58	RND57	RND56	RND55	RND54
shift 3	RND53	RND52	RND51	RND50	RND49	RND48	RND47	RND46
shift 4	RND45	RND44	RND43	RND42	RND41	RND40	RND39	RND38
shift 5	RND37	RND36	RND35	RND34	RND33	RND32	RND31	RND30
shift 6	RND29	RND28	RND27	RND26	RND25	RND24	RND23	RND22
shift 7	RND21	RND20	RND19	RND18	RND17	RND16	RND15	RND14
shift 8	RND13	RND12	RND11	RND10	RND9	RND8	RND7	RND6
shift 9	RND5	RND4	RND3	RND2	RND1	RND0	X	X

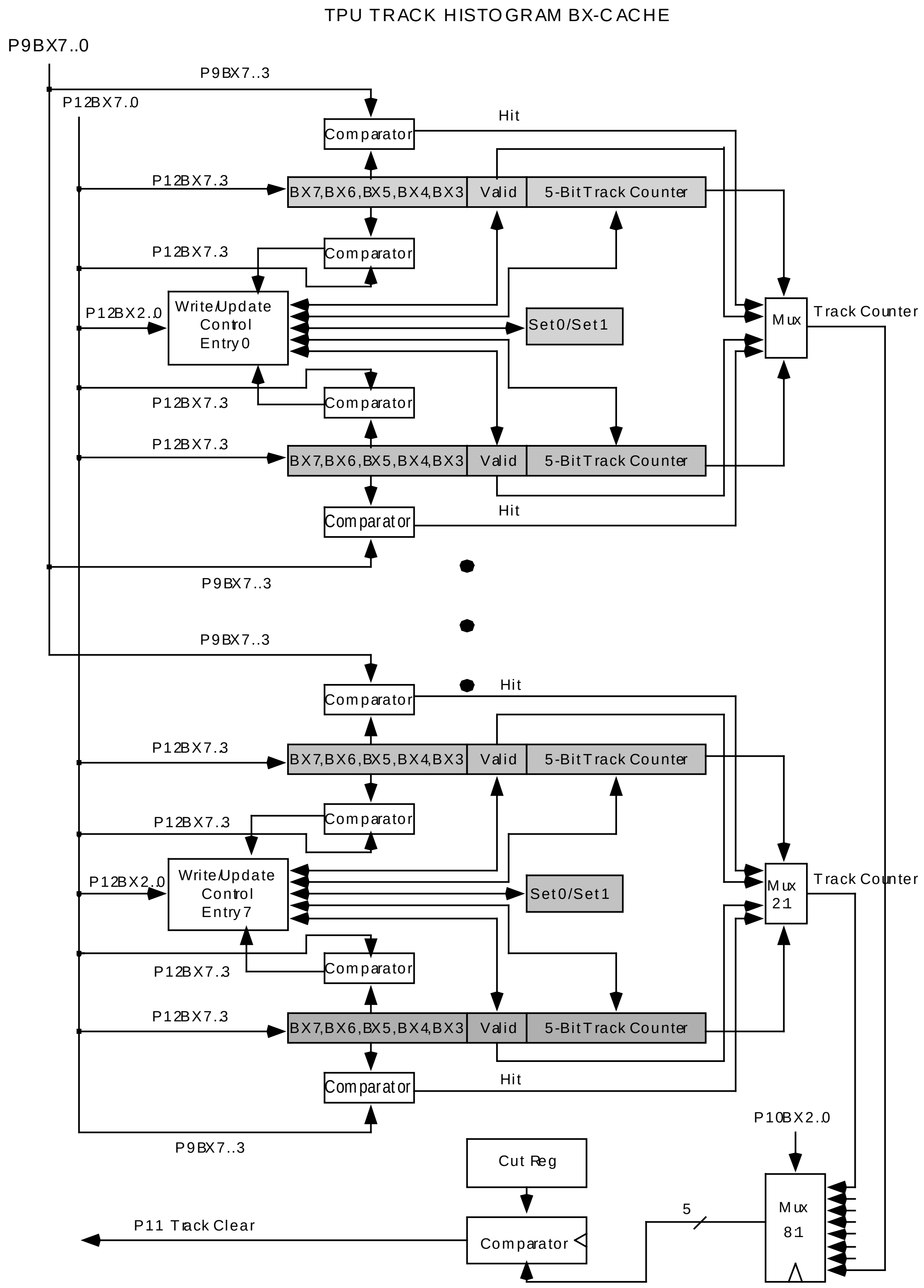
RND0..9:	Random generator registers: RND0 .. 9 = XI0 .. 9
RND10..11:	Random generator registers: RND10 .. 11 = OM0 .. 1
RND12..19:	Random generator registers: RND12 .. 19 = DDXI0 .. 7
RND20..27:	Random generator registers: RND20 .. 27 = DXI0 .. 7
RND28..36:	Random generator registers: RND28 .. 36 = ETA0 .. 8
RND37..43:	Random generator registers: RND37 .. 43 = EB0 .. 6
RND44..50:	Random generator registers: RND44 .. 50 = EB_0 .. 6
RND51..52:	Random generator registers: RND51 .. 52 = ID0 .. 1
RND53:	Random generator registers: RND53 = ALL
RND54..60:	Random generator registers: RND54 .. 60 = P0 .. 6
RND61..68:	Random generator registers: RND61 .. 68 = BX0 .. 7
RND69:	Random generator registers: RND69 = FLAG
X:	Delay registers to extend the serial pipeline length to byte boundary

Histogram Cache Operation

Additionally to the normal TPU pipeline processor a histogram cache is implemented on the TPU. This histogram cache is used for track suppression, if the number of the tracks belonging to a BX number exceeds a programmable limit. Suppression of insignificant tracks increases the overall performance of the FLT. To reduce the expense for the implementation, the histogram logic was implemented using a cache memory architecture.
Realization: The cache consists of 16 entries with two way set associative organisation. BX2 to BX0 is used to select one of eight cache entry pairs. For the replacement of old entries with new entries, the oldest entry of the pair is replaced (last-recently-used algorithm). Each entry contains three fields: The data field contains the upper five bits of the BX-number. The valid bit is used as indicator, if the entry contains valid data or not. The track counter field contains a 5-bit histogram counter. Additional each entry pair contains a SET0/SET1 indicator, for the replacement algorithm.
Operation: During normal TPU pipeline operation the histogram cache compares the BX-number of accepted tracks from the TPU pipeline processor with the cache entries. Two cases are possible: Entry found or entry not found. If the entry was found, the histogram counter is incremented. If the entry was not found, the cache logic generates a new entry by using the replacement algorithm: Copy the upper five bits of the BX-number to the data field, set valid true and set the histogram counter to one.
Additional, if the entry was found, the value of the track counter is compared with the value of the cut register to generate the track suppression signal. Because of the pipelined operation of the TPU processor and the histogram cache, there is a delay between entry write and generation of the track suppression signal. Therefore the cut limit depends on the order of the tracks.



Pipeline operation of histogram cache

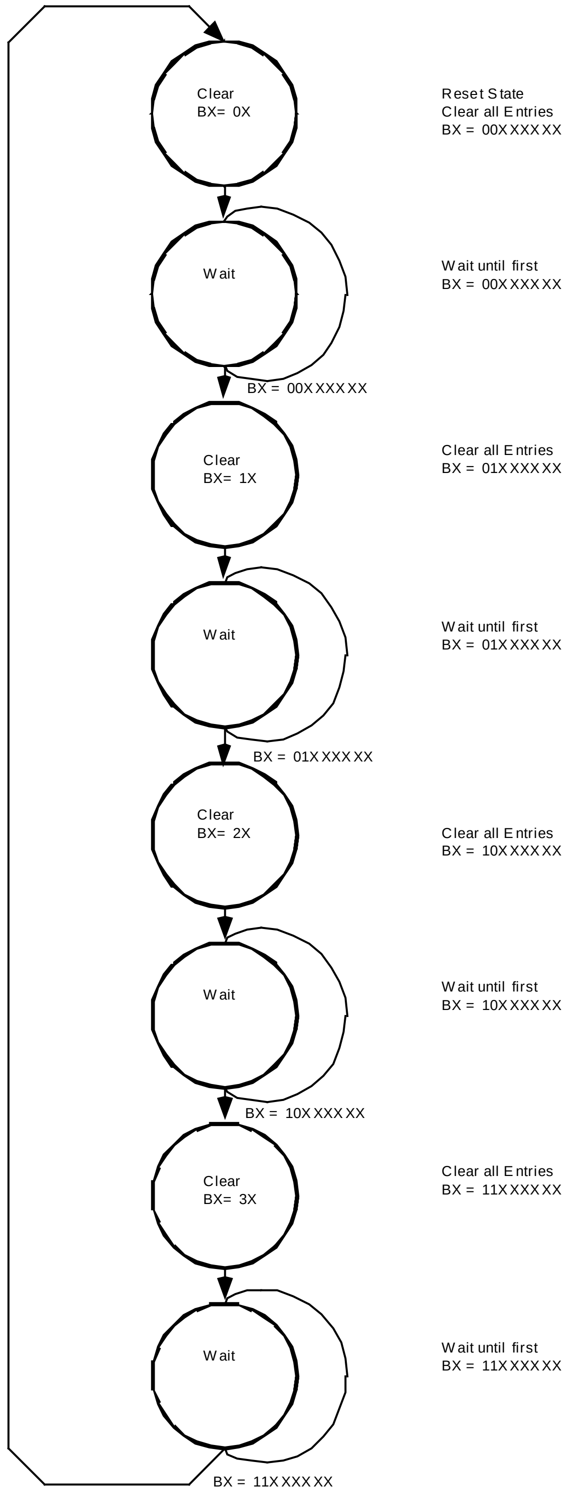

TPU TRACK HISTOGRAM BX-CACHE
P9BX7..0
P9BX7..3
P12BX7..0
Comparator
Hit
P12BX7..3
BX7,BX6,BX5,BX4,BX3
Valid
5-Bit Track Counter
P12BX2..0
Write/Update
Control
Entry 0
Set0/Set1
Mux
Track Counter
Write/Update
Control
Entry 7
Mux
2:1
P10BX2..0
Cut Reg
P11 Track Clear
5
Mux
8:1

Cache entry clear operation
To ensure the clear operation of old cache entries, the cache control logic contains some functions:
Since the BX number behaves like a count up counter, the actual BX number gives the information about actual and old BX entries (BX = [BX7,BX6,BX5,BX4,BX3,BX2,BX1,BX0]):

Time:	actual BX number	Cache clear operation
0 - 3 µs:	-> BX = [0,0,X,X,X,X,X,X]	clear all entries BX = [0,1,X,X,X,X,X,X]
3 - 6 µs:	-> BX = [0,1,X,X,X,X,X,X]	clear all entries BX = [1,0,X,X,X,X,X,X]
6 - 9 µs:	-> BX = [1,0,X,X,X,X,X,X]	clear all entries BX = [1,1,X,X,X,X,X,X]
9 - 12 µs:	-> BX = [1,1,X,X,X,X,X,X]	clear all entries BX = [0,0,X,X,X,X,X,X]
12 - 15 µs:	-> BX = [0,0,X,X,X,X,X,X]	clear all entries BX = [0,1,X,X,X,X,X,X]
15 - 18 µs:	-> BX = [0,1,X,X,X,X,X,X]	clear all entries BX = [1,0,X,X,X,X,X,X]
18 - 21 µs:	-> BX = [1,0,X,X,X,X,X,X]	clear all entries BX = [1,1,X,X,X,X,X,X]
21 - 24 µs:	-> BX = [1,1,X,X,X,X,X,X]	clear all entries BX = [0,0,X,X,X,X,X,X]
24 - 27 µs:	-> BX = [0,0,X,X,X,X,X,X]	clear all entries BX = [0,1,X,X,X,X,X,X]
...		

Example:

Time	Actual BX number	Cache clear operation
0 µs	[0,0,X,X,X,X,X,X]	clear all entries BX = [0,1,X,X,X,X,X,X]
	[0,0,X,X,X,X,X,X]	no clear operation
	[1,1,X,X,X,X,X,X]	no clear operation
	[0,0,X,X,X,X,X,X]	no clear operation
	...	
	[0,0,X,X,X,X,X,X]	no clear operation
≈ 3 µs	[0,1,X,X,X,X,X,X]	clear all entries BX = [1,0,X,X,X,X,X,X]
	[0,0,X,X,X,X,X,X]	no clear operation
	[0,0,X,X,X,X,X,X]	no clear operation
	[0,1,X,X,X,X,X,X]	no clear operation
	[0,0,X,X,X,X,X,X]	no clear operation
	[0,1,X,X,X,X,X,X]	no clear operation
	...	
	[0,1,X,X,X,X,X,X]	no clear operation
≈ 6 µs	[1,0,X,X,X,X,X,X]	clear all entries BX = [1,1,X,X,X,X,X,X]
	[0,1,X,X,X,X,X,X]	no clear operation
	[0,1,X,X,X,X,X,X]	no clear operation
	[1,0,X,X,X,X,X,X]	no clear operation
	[0,1,X,X,X,X,X,X]	no clear operation
	[1,0,X,X,X,X,X,X]	no clear operation
	...	
	[1,0,X,X,X,X,X,X]	no clear operation
≈ 9 µs	[1,1,X,X,X,X,X,X]	clear all entries BX = [0,0,X,X,X,X,X,X]
	[1,0,X,X,X,X,X,X]	no clear operation
	[1,0,X,X,X,X,X,X]	no clear operation
	[1,1,X,X,X,X,X,X]	no clear operation
	[1,0,X,X,X,X,X,X]	no clear operation
	[1,1,X,X,X,X,X,X]	no clear operation
	[1,1,X,X,X,X,X,X]	no clear operation
	[1,1,X,X,X,X,X,X]	no clear operation
	...	
	[1,1,X,X,X,X,X,X]	no clear operation
≈ 12 µs	[0,0,X,X,X,X,X,X]	clear all entries BX = [0,1,X,X,X,X,X,X]
	[1,1,X,X,X,X,X,X]	no clear operation
	...	



State Diagram of entry clear control

Additional four timeout counters are implemented, to ensure the clear operation of old entries:
Counter0 for entries with BX = 0XXXX0XX.
Counter1 for entries with BX = 0XXXX1XX.
Counter2 for entries with BX = 1XXXX0XX.
Counter3 for entries with BX = 1XXXX1XX.
Functional description for counter0 (counters1 -3 have the analog function):
If a track with `BX=0XXXX0XX` is processed by the TPU, the counter0 is cleared. After about 5 µs of the last track with `BX=0XXXX0XX` (timeout condition) a cache entry clear signal for all entries with `BX=0XXXX0XX` is activated for one clock cycle. This function secures clear operation, if there are only a low number of tracks.
To enable or disable the timeout clear function, the TIMEOUTEN bit in the COMPARATOR + HISTOGRAM CACHE CONTROL register is used (`$0040A0001`, D2):
Write transfer with D2 = 1: Old entries are cleared after timeout
Write transfer with D2 = 0: Timeout clear disabled

Realisation: Counter cycle time = 320 ns, counter size = 4 Bits, timeout time = 16 * 320 ns = 5120 ns

Example:

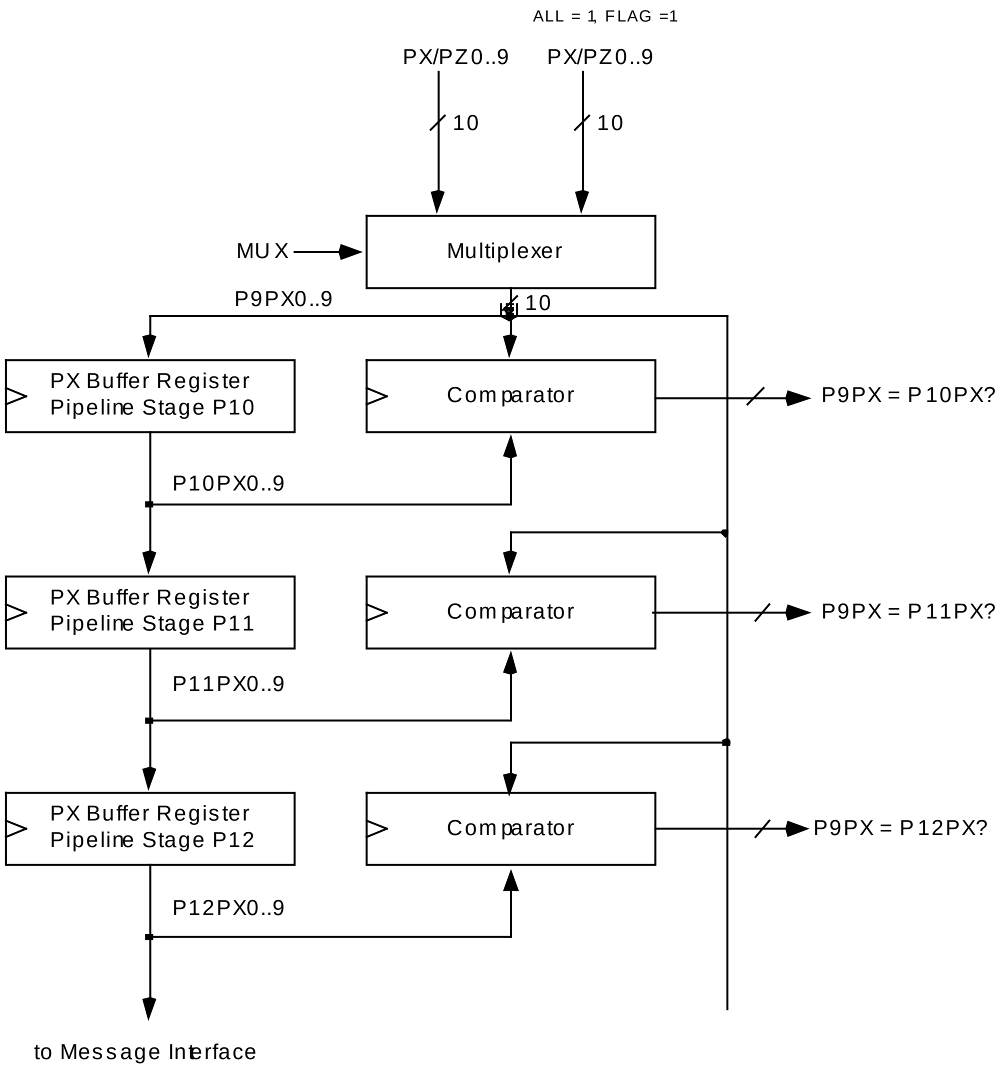
Time	Actual BX number	Timeout counter0	Cache clear operation
0 µs	[0,X,X,X,X,0,X,X]	0000	no clear operation
0.320 µs	[1,X,X,X,X,0,X,X]	0001	no clear operation
0.640 µs	no track	0010	no clear operation
≈ 1 µs	no track	0011	no clear operation
≈ 2 µs	no track	0110	no clear operation
≈ 3 µs	[0,X,X,X,X,0,X,X]	0000	no clear operation
≈ 4 µs	no track	0011	no clear operation
≈ 5 µs	no track	0110	no clear operation
≈ 6 µs	no track	1001	no clear operation
≈ 7 µs	no track	1100	no clear operation
≈ 8 µs	no track	1111	clear all entries [0,X,X,X,X,0,X,X]

Example: Cache activation
```
CACHE_CONTROL  EQU $0040A001
      MOVE.B      CACHE_CONTROL,D0  ;read cache control reg
      ORI.B       #$0C,D0           ;set CACHERES + TIMEOUTEN
      MOVE.B      D0,CACHE_CONTROL  ;write cache control reg
```

`0040A001`: COMPARATOR + HISTOGRAM CACHE CONTROL
```
[ X  , X  , X  , X  ,CACHERES,TIMEOUTEN,COMPRES,COMPEN]
[ D7 , D6 , D5 , D4 ,   D3   ,    D2    ,   D1   ,  D0  ]
```

Double Track Comparator Operation (under test)

The comparator is used to recognize and to suppress double tracks. The comparator compares each incoming track in pipeline stage 9 with the tracks in the following pipeline stages. Since the TPU processor pipeline ends with the pipeline stage P12, a buffer register set was implemented additionally. The buffer register stores the last 18 track parameters (PX, PY , BX + ID) of the accepted messages. For each pipeline or buffer register stage the comparator consistst of four individual comparators: PX-comparator, PY-comparator, BX-comparator and ID-comparator. Each comparator is used to compare the input from the pipeline stage P9 with ist stored TPU pipeline- or buffer register.



Double Track Comparator for PX/PZ Pipelien Stage P10..P12

The comparator for PX/PZ and for PY/PZ does not only check the exact identity, but also tolerates +/-1 deviation in the most cases. This feature is convenient, because the coordinates PX, PY and PZ have a tolerance of +/-1. The comparator for BX and ID check the exact identity.

Logical equation for PX-comparator of the pipeline stage P10 (simplified):

```
PX9..PX2:
EQUAL0 := CLKEN&([P9PX9,P9PX8,P9PX7,P9PX6,P9PX5,P9PX4,P9PX3,P9PX2]==
         [P10PX9,P10PX8,P10PX7,P10PX6,P10PX5,P10PX4,P10PX3,P10PX2])
          !CLKEN & EQUAL0;
PX1+PX0:
EQUAL1 := CLKEN&(
         [P9PX1,P9PX0] == 0)  &([P10PX1,P10PX0] == 0) ;pipeline operation
       # [P9PX1,P9PX0] == 0)  &([P10PX1,P10PX0] == 1)
       # [P9PX1,P9PX0] == 1)  &([P10PX1,P10PX0] == 0)
       # [P9PX1,P9PX0] == 1)  &([P10PX1,P10PX0] == 1)
       # [P9PX1,P9PX0] == 1)  &([P10PX1,P10PX0] == 2)
       # [P9PX1,P9PX0] == 2)  &([P10PX1,P10PX0] == 1)
       # [P9PX1,P9PX0] == 2)  &([P10PX1,P10PX0] == 2)
       # [P9PX1,P9PX0] == 2)  &([P10PX1,P10PX0] == 3)
       # [P9PX1,P9PX0] == 3)  &([P10PX1,P10PX0] == 2)
       # [P9PX1,P9PX0] == 3)  &([P10PX1,P10PX0] == 3)
         )
       # !CLKEN & EQUAL1;                             ;stall operation

EQUAL_P9PX_P10PX = EQUAL0 & EQUAL1 & DATA_VALID;
```

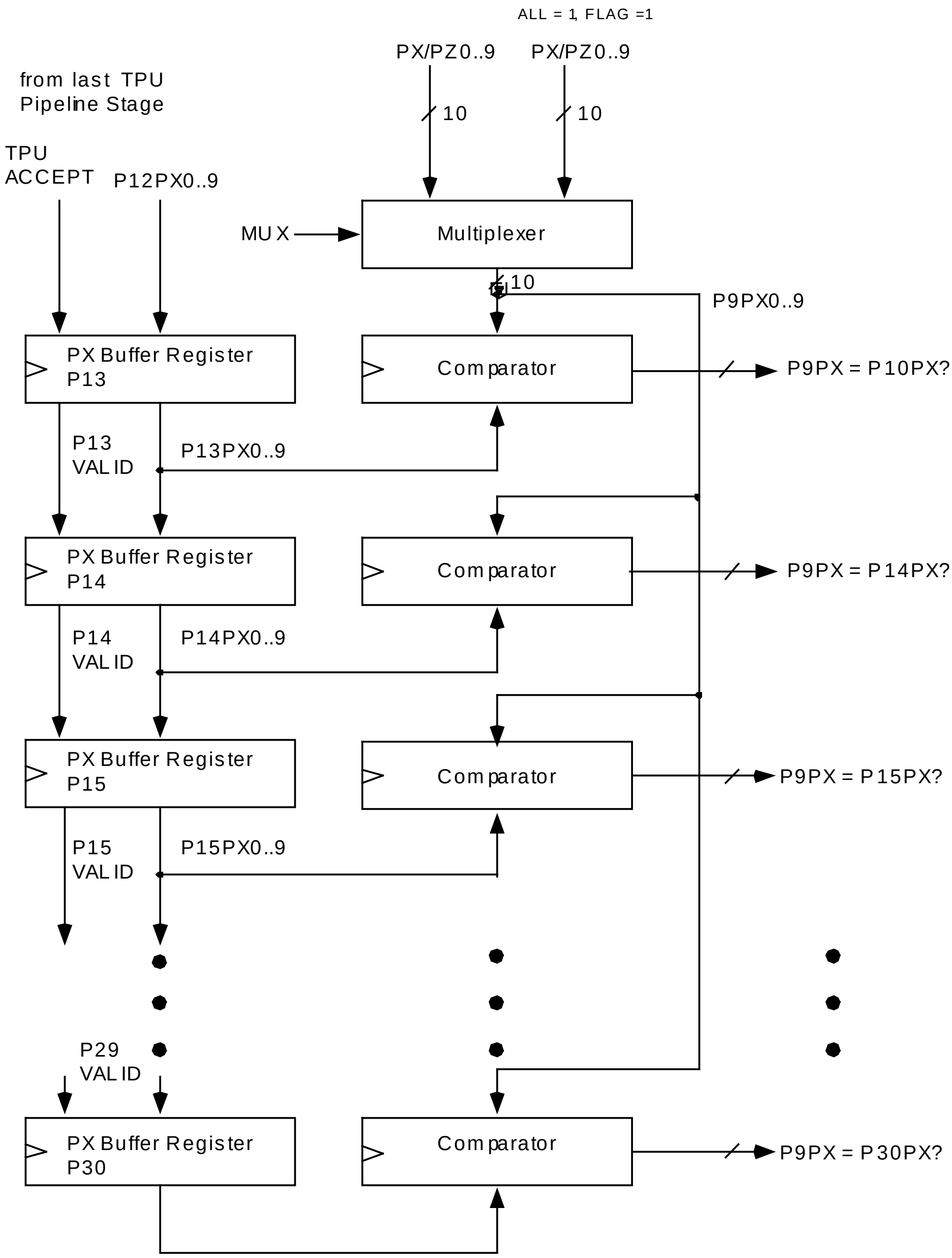
Example: `P9PX/PZ  = ^B1111111101`

```
      EQUAL1 if P10PX/PZ = ^B1111111100, ^B1111111101, ^B1111111110
```

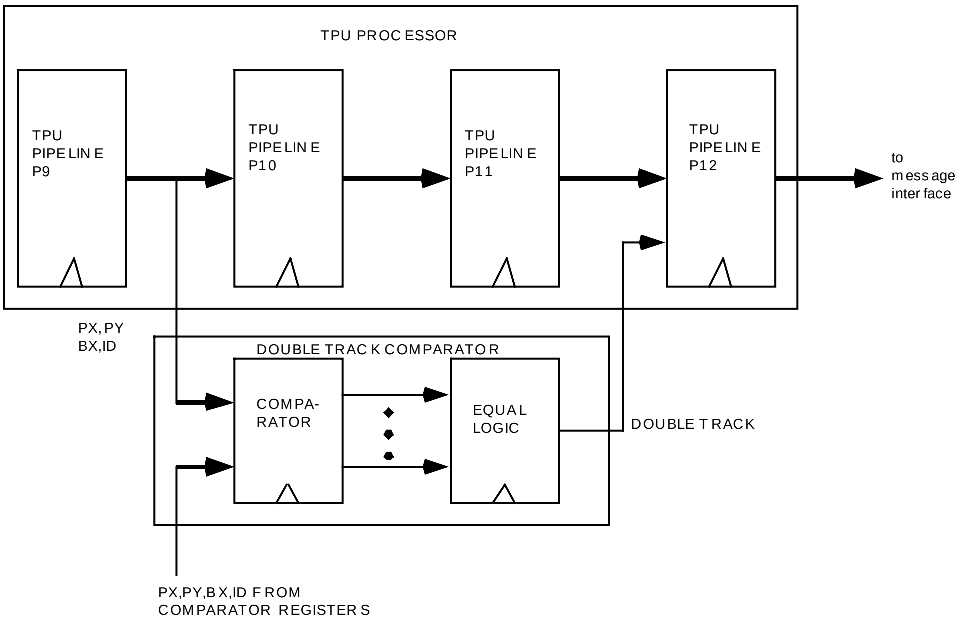
Logical equation for PX buffer register set for the pipeline stage P13 - P30 (simplified):

```
P13PX := CLKEN  &  FIFOWR &    P12PX.FB          „buffer register update
            # (!CLKEN # !FIFOWR) &   P13PX.FB;         „buffer register
hold

P13VALID := !COMPRES & CLKEN  &  FIFOWR    „set data valid for register P13
                     # !COMPRES & (!CLKEN # !FIFOWR); „hold valid
```

`CLKEN`: TPU clock enable (active high)
`FIFOWR`: TPU message output fifo write (send message to TDU, active high)
`COMPRES`: Comparator buffer register reset (clear all entries, active high)



Comparator for buffer registers set P13 - P30



Comparator pipeline operation

Example: Comparator initialization + activation

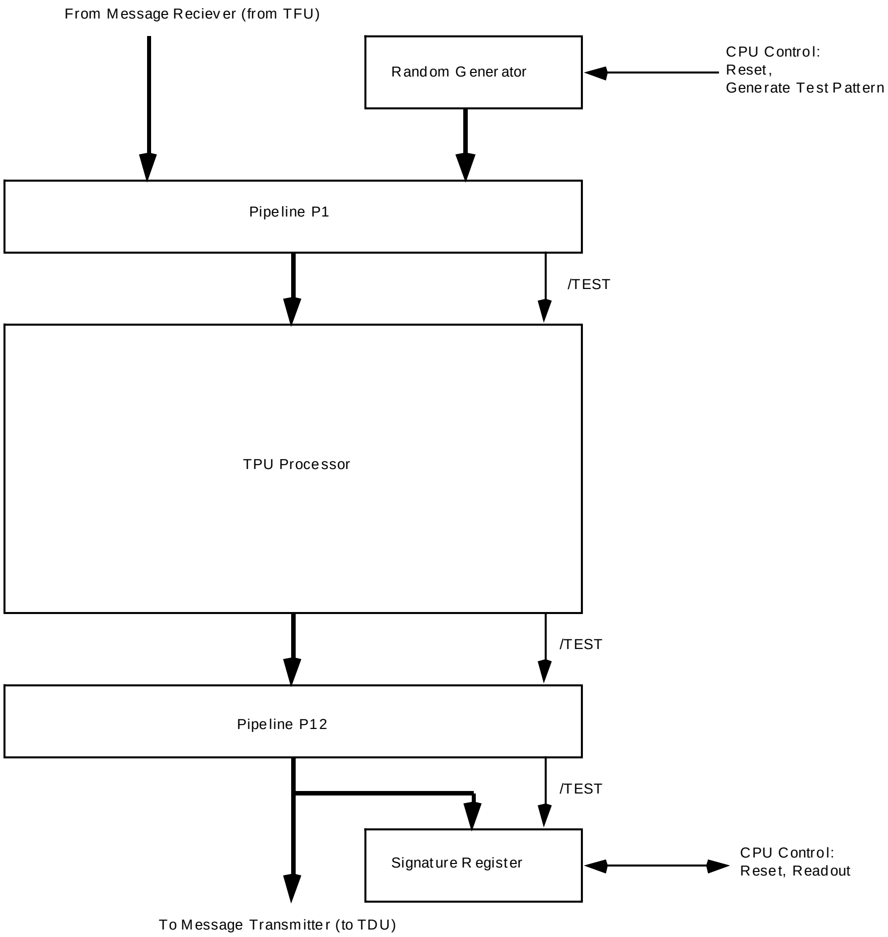
```
COMPARATOR_CONTROL  EQU $0040A001
      MOVE.B COMPARATOR_CONTROL,D0  ;read comparator control reg
      ORI.B  #$02,D0                ;set COMPRES
      MOVE.B D0,COMPARATOR_CONTROL  ;write comparator control reg
      ANDI.B #$FD,D0                ;disable COMPRES
      ORI.B  #$01,D0                ;set COMPEN
      MOVE.B D0,COMPARATOR_CONTROL  ;write comparator control reg
```

`0040A001:` COMPARATOR + HISTOGRAM CACHE CONTROL

```
[ X , X , X , X ,CACHERES,TIMEOUTEN,COMPRES,COMPEN]
[ D7, D6, D5, D4,   D3   ,   D2    ,   D1  ,  D0  ]
```

Signature Register + Random Generator Operation

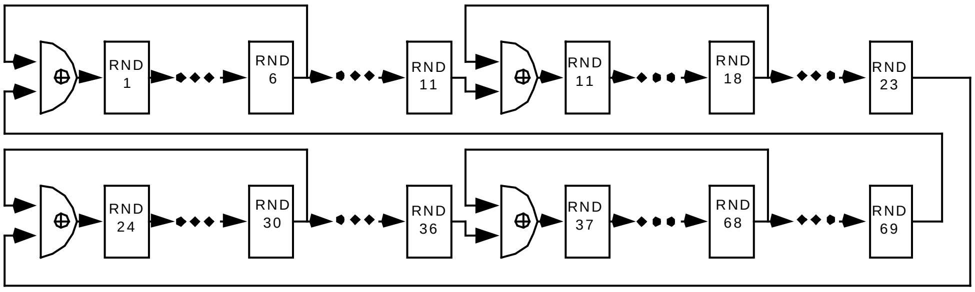
Signature Analysis is a fast and accurate troubleshooting method for the verification of the correct TPU operation during (!) the normal TPU operation ('TPU RUN'). Fault finding is reduced to readout the signature register and comparing measured signature to the known good signature recorded on a file.
A signature is a cyclic redundancy code (CRC) used as an error detection check on blocks of data. Test patterns (test messages) are generated from a pseudo random generator in the first pipeline stage P1 of the TPU. The signature register calculates the CRC code from the parameters of the last pipeline stage (P12) of the TPU.



Test data flow for Signature register analysis

CPU operation for the Signature register check to verify correct TPU operation:
1. Reset the Random generator + Signature register
Repeat this operation N times (N > 10000!):
2. Trigger random generator to generate test message
   TPU processor operation: 1. Stall message readout for one clock cycle.
   2. Insert test message with random parameters instead of the message from the TFU.
   3. Calculate new random parameters.
   4. Calculate TPU output from test message.
   5. Extract the test message in pipeline stage P12 (to avoid transfer to TDU) and store the test message in the Signature Register.
   6. Calculate new CRC in the signature register.
3. Serial readout of the Signature register value
4. Compare measured Signature value with the known good signature for N test messages.

Realization of the Random generator:



Data flow for random number calculation

Operations of the Random generator:
Reset operation (RND INIT) : This operation is used for the initialization of the random generator. RND0..RND69 == 1.
Read operation (RND READ): This operation is used for to send a test message through the TPU processor to the signature register and to calculate a new random number.
Shift operation (RNDSHFT) : During this operation, the random generator is used as a buffer register for a TPU input message. The shift operation is used for message readout during run.

Logical Equations (simplified):
Random register with EXOR input

```
RND0 :=   RND_INIT                                      "INITIALISATION"
        # /RND_INIT&  RND_SHFT & SERIN                  "SERIAL SHIFT"
        # /RND_INIT& /RND_SHFT & RND_READ               "CALCULATE NEW RANDOM"
        & (RNDIN&!RND6.FB # !RNDIN&RND6.FB)             "NUMBER(EXOR)"
        # /RND_INIT& /RND_SHFT & /RND_READ              "REGISTER HOLD"
        & RND0;
```

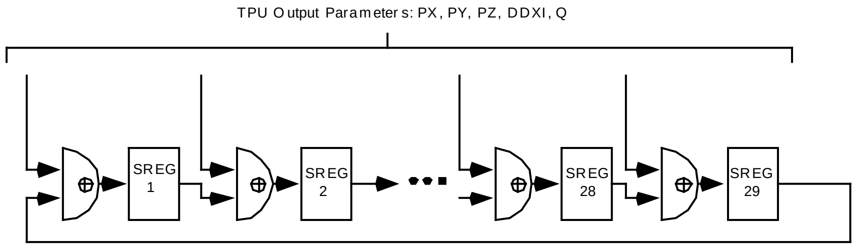
Random register without EXOR input

```
RND1 :=   RND_INIT                                      "INITIALISATION"
        # /RND_INIT&  RND_SHFT & RND0                   "SERIAL SHIFT"
        # /RND_INIT& /RND_SHFT & RND_READ               "CALCULATE NEW RANDOM"
        & (RND1)                                        "NUMBER(EXOR)"
        # /RND_INIT& /RND_SHFT & /RND_READ              "REGISTER HOLD"
        & RND1;
```

Function of the random generator registers:

Random generator registers:	TPU parameter
RND0 .. RND9	XI0 .. XI9
RND10 .. RND11	OM0 .. OM1
RND12 .. RND19	DDXI0 .. DXI7
RND20 .. RND27	DXI0 .. DXI7
RND28 .. RND36	ETA0 .. ETA8
RND37 .. RND43	EB0 .. EB6
RND44 .. RND50	EB_0 .. EB_6
RND51 .. RND52	ID0 .. ID1
RND53	ALL
RND54 .. RND60	P0 .. P6
RND61 .. RND68	BX0 .. BX7
RND69	FLAG

Realization of the Signature register :



Data flow for Signature calculation

Operations of the Signature register:
Reset operation (SIG RES) : This operation is used for the initialization of the Signature register. SREG1 .. SREG29 == 0.
Load operation (SIG LOAD): This operation is used for CRC calculation.
Shift operation (SIG SHFT) : During this operation, the signature register is used as a buffer register for a TPU output message. The shift operation is used for message readout during run.

```
SREG2 :=   SIG_RES                                  "INITIALISATION"
        # /SIG_RES&  SIG_SHFT & SREG1               "SERIAL SHIFT"
        # /SIG_RES& /SIG_SHFT & SIG_LOAD            "CALCULATE NEW SIGNATURE"
        & (SREG1 != P12PX1)                         "NUMBER(EXOR)"
        # /SIG_RES& /SIG_SHFT & /SIG_LOAD           "REGISTER HOLD"
        & SREG2
```

Function of the Signature registers:

Signature registers:	TPU parameter input
SREG1 .. SREG10	PX0 .. PX9
SREG11 .. SREG19	PY0 .. PY8
SREG20 .. SREG26	PZ0 .. PZ6
SREG27 .. SREG28	DDXI0 .. DDXI1
SREG29	Q

TPU Input + Output Message Control (not tested)

For control operations and statistical analysis it is possible, to transfer input messages and the corresponding output messages to the CPU during TPU run.

The necessary TPU registers are:
Random generator [RND0..RND69] to store the input message.
Signature register [SREG1.. SREG29] to store the corresponding output message.

CPU operation to get TPU input and output messages:
1. Set RNDREAD bit (write transfer with D2 = 1 to TPU DATA FLOW CONTROL REG)
TPU processor operation: 1. Wait until the next input message appears in the TPU pipeline stage P1
2. Store this input message in the Random register.
3. Calculate TPU output from this message.
4. Store this message output in the Signature register.
5. Set indicator SIG_VALID in the TPU DATA FLOW CONTROL REG to zero (message valid for readout).
2. Check SIG_VALID: wait until SIG_VALID = 0
3. Serial readout of the Random register (PIPESEL = $1E): Set RNDSHIFT = 1 (write transfer with D3 = 1)
4. Serial readout of the Signature register (PIPESEL = $1F): Set SIG_RES/SIG_SHIFT = 01 (write transfer with D6 = 0; D5 = 1)
Function of the random generator registers:

Random generator registers:	TPU input message parameter
RND0 .. RND9	XI0 .. XI9
RND10 .. RND11	OM0 .. OM1
RND12 .. RND19	DDXI0 .. DXI7
RND20 .. RND27	DXI0 .. DXI7
RND28 .. RND36	ETA0 .. ETA8
RND37 .. RND43	EB0 .. EB6
RND44 .. RND50	EB_0 .. EB_6
RND51 .. RND52	ID0 .. ID1
RND53	ALL
RND54 .. RND60	P0 .. P6
RND61 .. RND68	BX0 .. BX7
RND69	FLAG

Function of the Signature registers:

Signature registers:	TPU message parameter output
SREG1 .. SREG10	PX0 .. PX9
SREG11 .. SREG19	PY0 .. PY8
SREG20 .. SREG26	PZ0 .. PZ6
SREG27 .. SREG28	DDXI0 .. DDXI1
SREG29	Q

TPU Histogram Cache Modifications for TDU Replacement

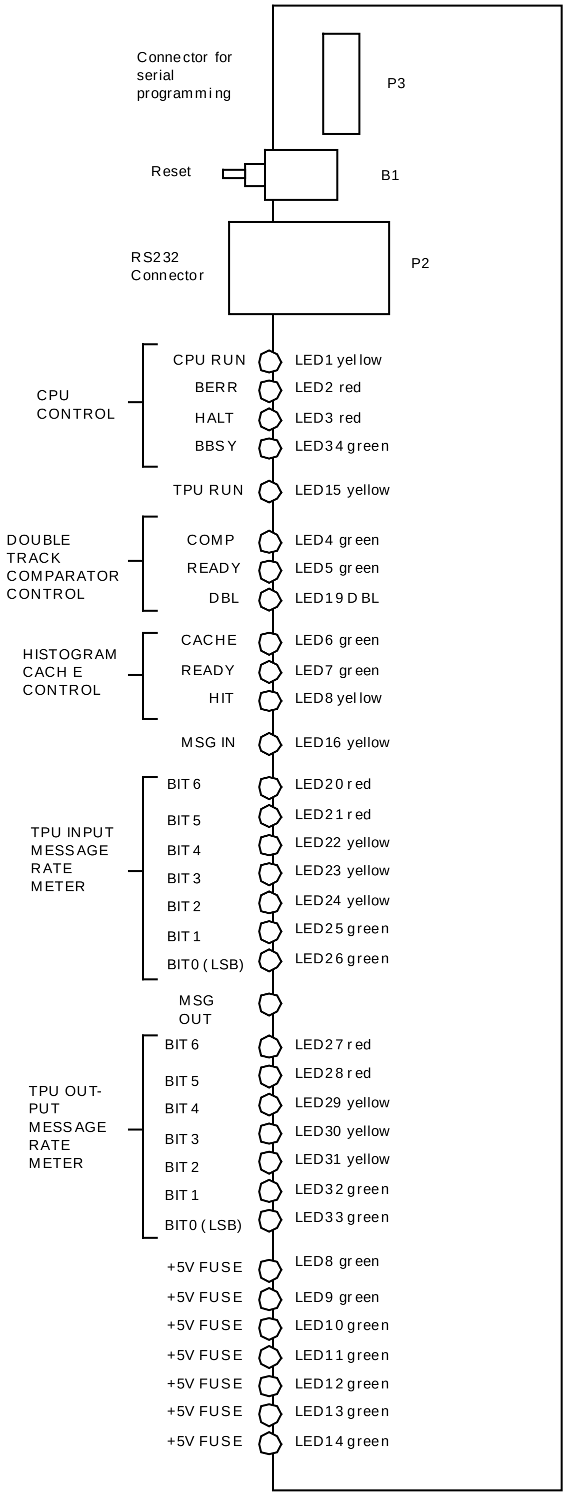
Instead of the normal histogram cache operation of track suppression, if the number of the tracks belonging to BX number exceeds a programmable limit, employs this modification a track suppression, if the number of tracks belonging to BX number reaches not a programmable limit.
To perform this modification replace the following parts:
1. Replace Accept1 with Accept2 (CACHEKILL -> /CACHEKILL)
2. Replace CACHE3 with CACHE4 (/CACHEKILL IF CUTREG < CACHE ENTRY)
3. Load CACHE1 + CACHE2 with new JEDEC-files TRIG1.JED for CACHE1 and TRG2.JED for CACHE2.

Operation: During normal TPU pipeline operation the histogram cache compares the BX-number of the tracks in pipeline stage P9 with the cache entries. Two cases are possible: Entry found or entry not found. If the entry was found, the histogram counter is incremented. If the entry was not found, the cache logic generates a new entry by using the replacement algorithm: Copy the upper five Bits of the BX-number to the data field of the cache entry, set valid true and set the histogram counter to one. Additional, if the entry was found, the value of the track counter is compared with the value of the cut register to generate the track suppression signal. If the value of the track counter equal or greater as the value of the cut register , the track supression is disabled.
(If the cut register value is zero, the track suppression is disabled, independent from histogram cache entries.

Example for histogram cache operation(CUTREG = 3):
1. Track BX = 0: Track suppression
2. Track BX = 1: Track suppression
3. Track BX = 0: Track suppression
4. Track BX = 0: Track suppression
5. Track BX = 1: Track suppression
6. Track BX = 0: No track suppression
7. Track BX = 1: Track suppression
8. Track BX = 0: No track suppression
9. Track BX = 1: No track suppression
10. Track BX = 0: No track suppression

Because of the pipeline operation of the histogram cache, there is one CLK-cycle delay between entry write/update and generation of the track suppression signal. Therefore the cut limit depends from the order of tracks.

TPU FRONT PLANE CONTROL ELEMENTS DESCRIPTION



Connector for serial programming: This connector is used to control the programming of the TPU in-circuit-programmable devices: MSCNT1, PIPE1, CACHE1, CACHE2, COMP1, COMP2, COMP3, COMP4, COMP5 + COMP6.

RESET: This switch is used to reset the TPU on-board CPU MC68020.

RS232 Connector: This connector is used for the communication between the COM2 serial interface(RS232) of the host computer (PC) and the monitor program of the TPU CPU.

CPU CONTROL:
CPU RUN LED: Indicator for the CPU run or halt operation. LED shining indicates CPU bus operation. No LED shining indicates CPU crash.
CPU BERR LED: Indicator for the CPU bus error trap. LED shining indicates that an invalid CPU bus operations is being attempted, or, when the HALT LED is also shining, that the CPU performs a retry bus operation. Invalid CPU bus operation means, if the CPU bus transfer was terminated by the TPU on-board timeout control logic. If the CPU bus cycle is not finished during a time of about 16 µs, a timeout control logic terminates the bus cycle with bus error termination. The CPU retry operation is used to avoid deadlocks during VME transfers: If the CPU tries to perform a VME transfer and another VME master has granted the VME bus and tries to perform a transfer to the internal TPU memory or control logic. The CPU cannot perform the VME transfer until the other VME bus master has finished his VME transfer and the other VME master cannot get the internal CPU bus to perform the transfer to the TPU memory until the CPU has finished the transfer. The arbitration logic of the TPU internal CPU bus avoids this deadlock situation by activation of the CPU rerun (retry) operation: The CPU terminates the bus cycle and retries the bus operation after the external VME master has finished his operation.
HALT LED: Indicator for CPU halt status. LED shining is used to indicate that the CPU has halted due to a double bus fault.
BBSY LED: Indicator for VME bus busy. LED shining indicates that the TPU CPU MC68020 is the VME master.

TPU RUN LED: Indicator for TPU pipeline processor RUN/HALT mode. LED shining indicates that the TPU processor is in the operating mode: Message readout - track parameter calculation - message transfer to TDU. No LED shining indicates TPU pipeline processor halt mode: This mode is used for LUT programming and test operations.

Double Track Comparator Control:
COMP LED: Indicator for double track comparator enable. LED shining indicates that the double track comparator is disabled. No LED shining indicates that the double track comparator is enabled.
READY LED: Indicator for double track comparator reset. LED shining indicates that the double track comparator is in reset mode (clear of all entries). No LED shining indicates that the comparator is in the ready mode.
DBL LED: Indicator for double track. LED shining indicates that the double track comparator has found a double track.

Histogram cache control:
CACHE LED: Indicator for histogram cache timeout enable: LED shining indicates that the cache timeout entry clear logic is enabled. No LED shining indicates, that the clear logic is disabled.
READY LED: Indicator for histogram cache enable/disable status. LED shining indicates that the cache is in the disable mode: No cache operation. All entries cleared. No LED shining indicates that the cache is enabled.
HIT LED: Indicator for cache hit: LED shining indicates that the actual message do match in the histogram cache.

MSG IN LED: Indicator for message inputs from the TFUs. LED shining indicates that the TPU processor has read a track parameter message from the receiver fifos.

TPU INPUT MESSAGE RATE METER: Indicator for the input message rate during 1024 clock cycles.
BIT0 LED: Indicator for 1 - 1024 messages during 1024 clock cycles (valid in every 64th cycle)
BIT1 LED: Indicator for 32 - 1024 messages during 1024 clock cycles (valid in every 32th cycle)
BIT2 LED: Indicator for 64 - 1024 messages during 1024 clock cycles (valid in every 16th cycle)
BIT3 LED: Indicator for 128 - 1024 messages during 1024 clock cycles (valid in every 8th cycle)
BIT4 LED: Indicator for 256 - 1024 messages during 1024 clock cycles (valid in every 4th cycle)
BIT5 LED: Indicator for 512 - 1024 messages during 1024 clock cycles (valid in every 2th cycle)
BIT6 LED: Indicator for 1024 messages during 1024 clock cycles (valid in every cycle)

MSG OUT LED: Indicator for message outputs from the TPU. LED shining indicates that the TPU processor has accepted a track parameter message and written in the transmitter fifos.

TPU OUTPUT MESSAGE RATE METER: Indicator for the output message rate during 1024 clock cycles.
BIT0: Indicator for 1 - 1024 messages during 1024 clock cycles (valid in every 64th cycle)
BIT1 LED: Indicator for 32 - 1024 messages during 1024 clock cycles (valid in every 32th cycle)
BIT2 LED: Indicator for 64 - 1024 messages during 1024 clock cycles (valid in every 16th cycle)
BIT3 LED: Indicator for 128 - 1024 messages during 1024 clock cycles (valid in every 8th cycle)
BIT4 LED: Indicator for 256 - 1024 messages during 1024 clock cycles (valid in every 4th cycle)
BIT5 LED: Indicator for 512 - 1024 messages during 1024 clock cycles (valid in every 2th cycle)
BIT6 LED: Indicator for 1024 messages during 1024 clock cycles (valid in every cycle)

+5V Fuse LEDs: +5V power indicator. The TPU has seven separate power areas. LED shining indicates power on. LED no shining indicates power off or destroyed fuses.