

CMS HCAL Data Concentrator Card

Operational Specification

1 June 2005

E. Hazen – Boston University

This document describes the operation of the HCAL Data Concentrator Card (DCC) for 2004 and beyond in the CMS environment.

Major Firmware Revisions

2005 Version

- Add '1' to TTC event number per CMS request
- Add comprehensive error bit handling (counters, TTS state table)
- Update to handle modified HTR data format

Updates are marked with [2005].

2004 Version

- Implement the TTS outputs and corresponding state machine
- Implement a front-panel LED status display
- Reduce the number of active HTR inputs to 15 (omit mip1 site)
- Renumber the HTR inputs in a more human-friendly fashion
- Respond to additional status bits in the HTR data
- Implement some changes to the CDF output format

2003 Version

Here is a brief review of the features implemented for the 2003 testbeam program:

- Operate in "trigger-driven mode", with event processing started by receipt of L1A (level 1 accept) from the TTCRx
- Timeout after 100 us (programmable) if one or more HTRs sends no data
- Temporarily buffer HTR data for each event before sending to SDRAM
- Transmit header containing various information extracted from HTR data blocks
- Compare EVN (event number) from TTCRx with EVN from each HTR and flag an error if they mis-match. Attempt to correct mis-matches by inserting or dropping event numbers and flagging the corresponding error.
- Transmit the data via SLink64 in the CMS CDF (Common Data Format)

Table of Contents

Major Firmware Revisions.....	1
2005 Version.....	1
2004 Version.....	1
2003 Version.....	1
TTCRx Interface.....	3
Triggers, Event Count and Bunch Count.....	3
TTC Broadcast Commands [2005].....	3
Calibration Triggers [2005].....	4
Trigger Rules.....	4
LED Status Display.....	5
State Machine and TTS Outputs.....	5
HTR/LRB Interface.....	7
Output Data Format.....	11
Appendix A – CRC VHDL Code.....	15

TTCRx Interface

Triggers, Event Count and Bunch Count

The primary function of the TTCRx interface[TTCRx] is to receive and process L1A triggers. The EVN is a 24-bit number provided by the TTCRx at each L1A. The TTCRx has a 12-bit bus BCnt [11 : 0] which can be programmed to emit in sequence the *bunch count*, *event counter low byte* and *event counter high byte* on three successive clocks beginning with the L1A clock. The TTC should be operated in the default trigger mode '11'. These values should be captured at each L1A and inserted in the data stream.

[2005] The bunch count (BcN) emitted by the TTCRx is incorrect and should be ignored. A local counter for BcN should be used. See details in the next section.

[2005] The event number (EvN) provided by the TTCRx is off by one. It may be used, but '1' must be added before inserting the number in the data stream. The event number will reset properly within the TTCrx in response to broadcast commands.

TTC Broadcast Commands [2005]

[2005] The DCC must respond to the following broadcast commands received by the TTCrx.

<i>Command</i>	<i>Brcst<7:0></i>	<i>Meaning</i>
ResetOrbitCounter	001X1XXX	Reset orbit count to zero
ReSync	010X1XXX	Reset the DAQ path
HardReset	011X1XXX	Reset the DAQ path
Start	100X1XXX	Start accepting L1As
StatReq	101X0XXX	Set LRB status request bit
Stop	101X1XXX	Stop accepting L1As
CalibTrig	110X1XXX	Generate Calibration Trigger
EvtCntReset	XXXXXX1X	Reset EvN (handled by TTCrx)
BC0	XXXXXX1	Reset BcN

ResetOrbitCounter: Reset orbit counter to a fixed value near zero. This should probably be a programmable constant with ~4 bits.

ReSync, HardReset: Clear the DAQ path and reset all data-containing buffers in response to any of these. L1A will be suspended well in advance of ReSync. VME (PCI) programmable registers should not be affected. TTS output should go to 'BUSY' when ReSync is received, and return to 'READY' when appropriate.

Start, Stop: Enable and Disable accepting L1A. Should be an 'or' with corresponding VME control bits for testing.

StatReq: Indicates that the LRB status request header bit was set for this event. LRB monitoring data should be captured.

CalibTrig: Generate a calibration trigger. See section below.

EvtCntReset (aka ECR): Reset the event number to 1. The TTCrx will reset it's internal event number to 0, so if the TTCrx event number is used, a constant 1 must be added for each event.

BC0 (aka BCR): Reset BcN (BX_id) to zero after a (programmable) pipeline delay of a few clocks. When reset occurs, check that BcN = 3563. If not, generate error condition.

Calibration Triggers [2005]

In response to a TTCrx CalibTrig broadcast command, a special trigger is generated. It is handled just like an L1A from the TTCrx except for the following:

- Event data goes only to VME monitor buffer (not to S-Link even if enabled)
- No prescaling (every CalibTrig causes one event to be stored in monitor buffer)
- A *Separate event number* is maintained for calibration triggers. This event number should be checked against the HTR event number just like the regular one. It may be reset to '1' in response to a VME command.
- L1A and CalibTrig may never occur at the same time.

Trigger Rules

The estimated L1A rate is 100~kHz. There are ``trigger rules" which limit the L1A rate further. From the CMS Trigger TDR[TDR] (see section 16.4). These are provided only for reference... the DCC does not need to check for violations of these.

- i. No more than 1 L1A per 3 BX (75ns)
- ii. No more than 2 L1A per 25 BX (625ns)
- iii. No more than 3 L1A per 100 BX (2.5µs)
- iv. No more than 4 L1A per 240 BX (6µs)

The L1A with bunch count and event count should be stored in a FIFO. An overflow of this FIFO is a fatal error and should be reported as a loss-of-sync.

The TTCrx is programmed with an **I2C Interface** which should be controllable from the DCC logic board.

LED Status Display

The DCC front panel contains a status display (programmed over a simple serial interface) which provides 24 programmable LEDs.

Group 1 – Below SLink64		off	on	blink
VME	VME Activity	no activity	activity	-
TTC	TTC Status	not ready	ready	error
L1A	Level 1 Accepts	no L1A	L1A present	-
DAQ	SLink64 Status	not enabled	sending	error
DCC	DCC Status	not enabled	running	errors
Group 2 – Below TTS				
RDY	Ready	These LEDS represent the real-time status of the corresponding TTS outputs. Stretch 'on' condition to 100ms?		
BSY	Busy			
OFW	Overflow Warning			
SYN	Out of Sync			
Group 3 – Bottom (15 LEDs)		off	on	blink
HTR 0	HTR input status	Disabled / no data	data present	errors
HTR 1				
HTR 2				
...				
HTR 14				

LED Display

Error Handling [2005]

There are many possible sources of error or warning conditions which are of potential interest to CMS operators. All should be handled in a consistent, flexible way.

Error Sources (not an exhaustive list):

- HTR Error bits (EVT_STATUS field) – 15 bits of word 3 of each event from HTR. Since there are up to 15 HTR inputs, this corresponds to 15*15 or 225 error sources.
- DCC Error conditions: EvN mis-match, BcN mis-match, overflow threshold up/down, BcN mis-match at BCZero

For each error, the following actions should be performed:

- Increment a dedicated counter, accessible from VME (4-8 bits is sufficient with overflow protection)
- Cause a transition on the TTS state machine to a particular state (optional)
- Set a particular bit in DCC output header word 1 (details *t.b.d.*).

Each individual error condition has a control register with the following fields:

changeTTSstate	enable TTS state change on '1'
newTTSstate[3:0]	new TTS state

Each HTR may be treated identically, so 15 registers are required to handle the error processing for all enabled HTRs. However, each individual HTR requires its own set of 15 error counters. The table below shows the layout of the error counters. Each of the error conditions 0-31 which may occur set the corresponding bit in the CDF output DCC header word#1 when the counter is non-zero.

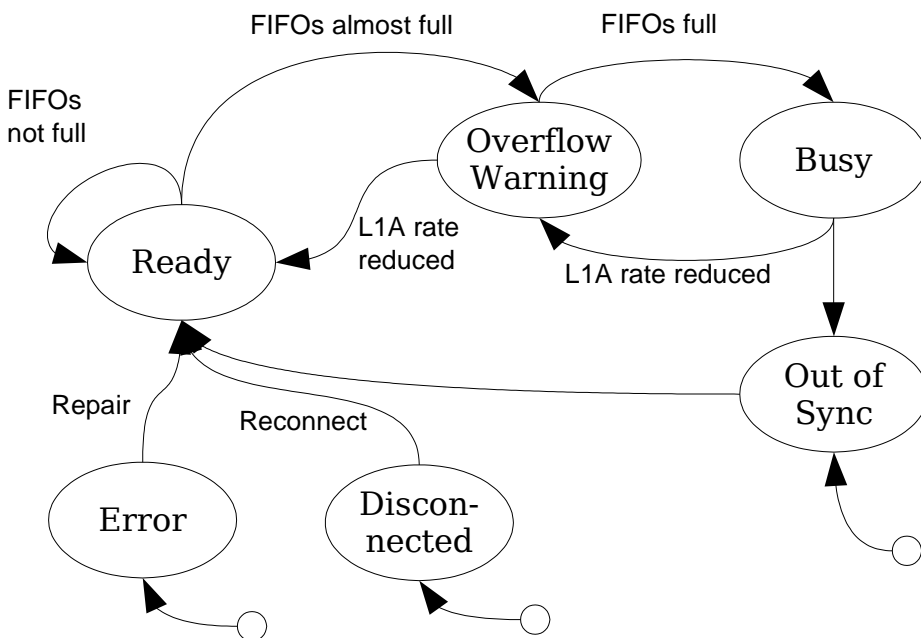
All error counters may be reset at once by writing a '1' to a VME control bit.

Offset			
0	HTR 0	Counter 0	OW
1		Counter 1	BZ
2		Counter 2	EE
3		Counter 3	RL
4		Counter 4	LE
5		Counter 5	LW
6		Counter 6	OD
7		Counter 7	CK
8		Counter 8	Spare
9		Counter 9	Spare
0a		Counter 10	Spare
0b		Counter 11	Spare
0c		Counter 12	CT
0d		Counter 13	HM
0e		Counter 14	TM
0f	HTR 1	Counter 0	
10		Counter 1	
11		Counter 2	
...		...	
1d		Counter 14	
1e	HTR 2		
...			
df	HTR 14	Counter 14	
c0	DCC	Counter 0	L1A FIFO overflow warning on
c1		Counter 1	L1A FIFO overflow warning off
c2		Counter 2	L1A FIFO busy threshold on
c3		Counter 3	L1A FIFO busy threshold off
c4		Counter 4	L1A FIFO full (lost sync) on
c5		Counter 5	L1A FIFO full (lost sync) off
c6		Counter 6	HTR EvN mis-match L1A event
c7		Counter 7	HTR BcN mis-match L1A event
c8		Counter 8	HTR EvN mis-match calibration event
c9		Counter 9	HTR BcN mis-match calibration event
ca		Counter 10	BcN != 3563 on BCR
cb		Counter 11	... more DCC counters ...

DCC Error Counters

State Machine and TTS Outputs

During data taking (run mode) the DCC must obey the state transition diagram (from the CMS DAQ/FE Interfacing Guide [DAQ]) shown below.



A description of each state, and the corresponding values of the TTS output signals is given in the table below. States not shown are disallowed.

<i>Value</i> (RDY+BSY+SYN+OFW)	<i>Name</i>	<i>Comment</i>
'0000' or '1111'	Disconnected	Hardware failure or broken cable
'0001'	Overflow Warning	Imminent buffer overflow
'0010'	Out of Sync	FED is not synchronized with TTC event number
'0100'	Busy	Cannot accept triggers
'1000'	Ready	Ready for triggers
'1100'	Error	Any other state that prevents correct functioning

TTS State Values

The TTS output connector must provide a real-time (updated once per L1A) copy of the state bits. The signals are LVDS on an RJ-45 connector, with pin assignments given in the table below.

Pin 1	- BSY	Pin 5	- OFW
Pin 2	+ BSY	Pin 6	+ RDY
Pin 3	- RDY	Pin 7	- SYN
Pin 4	+ OFW	Pin 8	+ SYN

TTS Output Connector

HTR/LRB Interface

In response to an L1A, each HTR sends one block of data. The format (as of Feb 2005) is given in the table below. See the online documentation[HTR] for updated information. The shaded cells in the table indicate values which the DCC must access when building the header for output. The HTR can send three types of events: Normal, Histogramming and Empty. The DCC can treat all three types identically.

The HTR inputs are numbered 0-14, beginning with the top input of the 2nd LRB (mip2 site). The mip1 site is not used. The data format and event builder structure must reflect this.

Word Type	Byte 1								Byte 0										
	S1	S0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
HEADER	1	1	chan_id								EvN[7:0]								
Ext. Hdr.2	1	0	EvN[23:8]																
Ext. Hdr.3	1	0	1	EVT_Status															
Ext. Hdr.4	1	0	OrN[5:0]						HTR_sub_module_Number[9:0]										
Ext. Hdr.5	1	0	FmtVers[3:0]				BcN[11:0]												
Ext. Hdr.6	1	0	NumTPWords[7:0]							-	-	-	-	DLL_lock				TTC	
Ext. Hdr.7	1	0	PipeLength[7:0]							HTR_Firmware_Version[7:0]									
Ext. Hdr.8	1	0	Reserved																
TP-DATA1	1	0																	
...	1	0																	
TP-DATAm	1	0																	
DAQ-DATA1	1	0																	
...	1	0																	
DAQ-DATAn	1	0																	
Parity Word	1	0	Insert 'FFFF' if (m+n) is odd																
Extra-Info1	1	0																	Fiber1_Latency[11:0]
	1	0																	Fiber2_Latency[11:0]
	1	0																	Fiber3_Latency[11:0]
	1	0																	Fiber4_Latency[11:0]
	1	0																	Fiber5_Latency[11:0]
	1	0																	Fiber6_Latency[11:0]
	1	0																	Fiber7_Latency[11:0]
	1	0																	Fiber8_Latency[11:0]
Extra-Info8	1	0	NumDaqSamples						NumDAQWords[10:0]										
	1	0	-	-	-	-	WordCount[11:0] (16-bit words)												
Pre-Trailer	1	0	WordCount[15:0] (32-bit words)																
TRAILER	0	1	EvN[7:0]								LRB_Errors								

HTR to DCC Data Format – Normal Event [2005]

Word Type	Byte 1								Byte 0									
	S1	S0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HEADER	1	1	chan_id								EvN[7:0]							
Ext. Hdr.2	1	0	EvN[23:8]															
Ext. Hdr.3	1	0	1	htr_errors														
Ext. Hdr.4	1	0	OrN[5:0]						HTR_sub_module_Number[9:0]									
Ext. Hdr.5	1	0	FmtVers[3:0]				BcN[11:0]											
Pre-Trailer	1	0	-	-	-	-	WordCount[11:0] (16-bit words)											
	1	0	WordCount[15:0] (32-bit words)															
TRAILER	0	1	EvN[7:0]								LRB_Errors							

HTR to DCC Data Format – Empty Event [2005]

Missing Table

HTR to DCC Data Format – Histogramming Mode

<i>Name</i>	<i>Bits</i>	<i>Function</i>
chan_id	bits 7-5	unused
	bit 4	PCI bus number (0/1) of DCC
	bits 3-2	LRB number (0-2) on PCI bus
	bits 1-0	LRB channel (0-2)
EvN[24:0]		Event (L1A) number
EVT_Status	Bit 15	Always '1' to designate new format
	Bit 14 (TM)	Test Mode – counter data or pattern data
	Bit 13 (HM)	Histogramming Mode enabled
	Bit 12 (CT)	Calibration Trigger
	Bits 11-8	Reserved
	Bit 7 (CK)	Clock Problem on HTR
	Bit 6 (OD)	Optical Data error
	Bit 5 (LW)	Latency Warning
	Bit 4 (LE)	Latency Error
	Bit 3 (RL)	Rejected L1A due to violation of trigger rules
	Bit 2 (EE)	Empty Event
	Bit 1 (BZ)	Internal Buffers are Busy
	Bit 0 (OW)	Overflow Warning
LRB_Errors	bit 7	Odd 16-bit word count
	bit 6	Block structure error (missing header/trailer)
	bit 5	Overflow; data truncated
	bit 4	FIFO empty while reading block
	bit 3	EvN header/trailer mis-match
	bit 2	Block size overflow (not implemented?)
	bit 1	Uncorrected link error
	bit 0	Corrected link error

HTR Input Data Field Details [2005]

Output Data Format

The data should be output to SLink64 or VME in the format shown below. This format is intended to be compatible with the CMS TriDAS Common Data Format.

CMS Common Data Format (overview)

K	BOE_1	Evt_ty	LV1_ID (EvN)	BX_id	Source_id(10+2)	FOV	H	x	\$	\$
K	BOE_2								\$	\$
D	SubDetector Payload word 1				SubDetector Payload word 0					
D	SubDetector Payload word 3				SubDetector Payload word 2					
K	EOE_1	xxxx	Evt_lgth(24)	CRC (16)	xxxx	Stat	TTS(4)	T	x	\$
	63	60 59	56 55	32 31	20 19	16 15	12	8 7	4 3	2 1 0

The CDF fields are defined as follows (as of Dec 2003):

- BOE_n Identifier for beginning of event BOE_1 = 0x05
- Evt_ty Event type (currently undefined)
- LV1_id Level 1 event number
- BX_id Bunch crossing number
- Source_id Upper 10 bits are programmable
Low 2 bits are reserved for FED
- FOV FED data format version identifier
- H Set to '1' if another header word follows
- EOE_n Identifier for end of event EOE_1 = 0x0a
- Evt_lgth Length of event in 64-bit words
including headers and trailers
- CRC CRC of event including header and trailer
- Evt_stat Event status (currently undefined)
- TTS Current values of the TTS outputs
- T Set to '1' if another trailer word follows
- x Reserved bit

\$ Bit used by S-Link64 hardware

The CDF is specified in terms of 64 bit words only. It defines a header with various fixed fields, plus the **EvN** (event number) and **BX_id** (bunch crossing ID). These come from the TTCRx event count and bunch count, respectively. The **Evt_ty** and **Evt_stat** fields should be programmable (via PCI registers) but should not change from event to event. The **TTS** field should be set from the current state of the TTS output signals.

There may be a second header word (tbd) Following the header is a subdetector-defined payload. We arbitrarily choose the little-endian convention for consistency with the rest of our system, so the low 32 bits of 64 contain the first word of our payload. The payload may need to be padded to an even number of 32-bit words. Following the payload is a trailer, which must contain the total length (including header and trailer) of 64-bit words, plus a CRC (see Appendix A for VHDL code).

The format of the subdetector payload is given in the table below. There are two sections. The first is a fixed-length header inserted by the DCC. The second section contains all data exactly as received from the HTRs.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DCC Event Header 0																															
DCC Event Header 1																															
HTR Errors								LRB Errors								E	P	V	x	x	x	HTR#1 Word Count									
HTR Errors								LRB Errors								E	P	V	x	x	x	HTR#2 Word Count									
HTR Errors								LRB Errors								E	P	V	x	x	x									
HTR Errors								LRB Errors								E	P	V	x	x	x	HTR#15 Word Count									
zeroes																															
zeroes																															
zeroes																															
HTR a Data																HTR a Data															
...																...															
HTR a Data																HTR a Data															
HTR b Data																HTR b Data															
...																...															
HTR b Data																HTR b Data															
...																...															

The DCC header begins with a single 64-bit word which summarizes the event. In the 2003 firmware, all bits were zeroes. In the 2004 firmware, the following format will be used:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	HTR Status															Undefined					Format Version								
DCC Error Summary																															

Appendix A – CRC VHDL Code

```
-----  
-- File:   PCK_CRC16_D64.vhd  
-- Date:   Tue Dec 10 10:29:35 2002  
--  
-- Copyright (C) 1999 Easics NV.  
-- This source file may be used and distributed without restriction  
-- provided that this copyright statement is not removed from the file  
-- and that any derivative work contains the original copyright notice  
-- and the associated disclaimer.  
--  
-- THIS SOURCE FILE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS  
-- OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED  
-- WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  
--  
-- Purpose: VHDL package containing a synthesizable CRC function  
-- * polynomial: (0 2 15 16)  
-- * data width: 64  
--  
-- Info:   jand@easics.be (Jan Decaluwe)  
--         http://www.easics.com  
-----
```

```
library IEEE;  
use IEEE.std_logic_1164.all;
```

```
package PCK_CRC16_D64 is
```

```
    -- polynomial: (0 2 15 16)  
    -- data width: 64  
    -- convention: the first serial data bit is D(63)  
    function nextCRC16_D64  
    ( Data:  std_logic_vector(63 downto 0);  
      CRC:   std_logic_vector(15 downto 0) )  
    return std_logic_vector;
```

```
end PCK_CRC16_D64;
```

```
library IEEE;  
use IEEE.std_logic_1164.all;
```

```
package body PCK_CRC16_D64 is
```

```
    -- polynomial: (0 2 15 16)  
    -- data width: 64  
    -- convention: the first serial data bit is D(63)  
    function nextCRC16_D64  
    ( Data:  std_logic_vector(63 downto 0);  
      CRC:   std_logic_vector(15 downto 0) )  
    return std_logic_vector is
```

```
        variable D: std_logic_vector(63 downto 0);  
        variable C: std_logic_vector(15 downto 0);  
        variable NewCRC: std_logic_vector(15 downto 0);
```

```
begin
```

```
    D := Data;  
    C := CRC;
```

```
    NewCRC(0) := D(63) xor D(62) xor D(61) xor D(60) xor D(55) xor D(54) xor  
                D(53) xor D(52) xor D(51) xor D(50) xor D(49) xor D(48) xor  
                D(47) xor D(46) xor D(45) xor D(43) xor D(41) xor D(40) xor  
                D(39) xor D(38) xor D(37) xor D(36) xor D(35) xor D(34) xor  
                D(33) xor D(32) xor D(31) xor D(30) xor D(27) xor D(26) xor  
                D(25) xor D(24) xor D(23) xor D(22) xor D(21) xor D(20) xor  
                D(19) xor D(18) xor D(17) xor D(16) xor D(15) xor D(13) xor  
                D(12) xor D(11) xor D(10) xor D(9) xor D(8) xor D(7) xor  
                D(6) xor D(5) xor D(4) xor D(3) xor D(2) xor D(1) xor  
                D(0) xor C(0) xor C(1) xor C(2) xor C(3) xor C(4) xor  
                C(5) xor C(6) xor C(7) xor C(12) xor C(13) xor C(14) xor
```

```

    C(15);
NewCRC(1) := D(63) xor D(62) xor D(61) xor D(56) xor D(55) xor D(54) xor
D(53) xor D(52) xor D(51) xor D(50) xor D(49) xor D(48) xor
D(47) xor D(46) xor D(44) xor D(42) xor D(41) xor D(40) xor
D(39) xor D(38) xor D(37) xor D(36) xor D(35) xor D(34) xor
D(33) xor D(32) xor D(31) xor D(28) xor D(27) xor D(26) xor
D(25) xor D(24) xor D(23) xor D(22) xor D(21) xor D(20) xor
D(19) xor D(18) xor D(17) xor D(16) xor D(14) xor D(13) xor
D(12) xor D(11) xor D(10) xor D(9) xor D(8) xor D(7) xor
D(6) xor D(5) xor D(4) xor D(3) xor D(2) xor D(1) xor
C(0) xor C(1) xor C(2) xor C(3) xor C(4) xor C(5) xor
C(6) xor C(7) xor C(8) xor C(13) xor C(14) xor C(15);
NewCRC(2) := D(61) xor D(60) xor D(57) xor D(56) xor D(46) xor D(42) xor
D(31) xor D(30) xor D(29) xor D(28) xor D(16) xor D(14) xor
D(1) xor D(0) xor C(8) xor C(9) xor C(12) xor C(13);
NewCRC(3) := D(62) xor D(61) xor D(58) xor D(57) xor D(47) xor D(43) xor
D(32) xor D(31) xor D(30) xor D(29) xor D(17) xor D(15) xor
D(2) xor D(1) xor C(9) xor C(10) xor C(13) xor C(14);
NewCRC(4) := D(63) xor D(62) xor D(59) xor D(58) xor D(48) xor D(44) xor
D(33) xor D(32) xor D(31) xor D(30) xor D(18) xor D(16) xor
D(3) xor D(2) xor C(0) xor C(10) xor C(11) xor C(14) xor
C(15);
NewCRC(5) := D(63) xor D(60) xor D(59) xor D(49) xor D(45) xor D(34) xor
D(33) xor D(32) xor D(31) xor D(19) xor D(17) xor D(4) xor
D(3) xor C(1) xor C(11) xor C(12) xor C(15);
NewCRC(6) := D(61) xor D(60) xor D(50) xor D(46) xor D(35) xor D(34) xor
D(33) xor D(32) xor D(20) xor D(18) xor D(5) xor D(4) xor
C(2) xor C(12) xor C(13);
NewCRC(7) := D(62) xor D(61) xor D(51) xor D(47) xor D(36) xor D(35) xor
D(34) xor D(33) xor D(21) xor D(19) xor D(6) xor D(5) xor
C(3) xor C(13) xor C(14);
NewCRC(8) := D(63) xor D(62) xor D(52) xor D(48) xor D(37) xor D(36) xor
D(35) xor D(34) xor D(22) xor D(20) xor D(7) xor D(6) xor
C(0) xor C(4) xor C(14) xor C(15);
NewCRC(9) := D(63) xor D(53) xor D(49) xor D(38) xor D(37) xor D(36) xor
D(35) xor D(23) xor D(21) xor D(8) xor D(7) xor C(1) xor
C(5) xor C(15);
NewCRC(10) := D(54) xor D(50) xor D(39) xor D(38) xor D(37) xor D(36) xor
D(24) xor D(22) xor D(9) xor D(8) xor C(2) xor C(6);
NewCRC(11) := D(55) xor D(51) xor D(40) xor D(39) xor D(38) xor D(37) xor
D(25) xor D(23) xor D(10) xor D(9) xor C(3) xor C(7);
NewCRC(12) := D(56) xor D(52) xor D(41) xor D(40) xor D(39) xor D(38) xor
D(26) xor D(24) xor D(11) xor D(10) xor C(4) xor C(8);
NewCRC(13) := D(57) xor D(53) xor D(42) xor D(41) xor D(40) xor D(39) xor
D(27) xor D(25) xor D(12) xor D(11) xor C(5) xor C(9);
NewCRC(14) := D(58) xor D(54) xor D(43) xor D(42) xor D(41) xor D(40) xor
D(28) xor D(26) xor D(13) xor D(12) xor C(6) xor C(10);
NewCRC(15) := D(63) xor D(62) xor D(61) xor D(60) xor D(59) xor D(54) xor
D(53) xor D(52) xor D(51) xor D(50) xor D(49) xor D(48) xor
D(47) xor D(46) xor D(45) xor D(44) xor D(42) xor D(40) xor
D(39) xor D(38) xor D(37) xor D(36) xor D(35) xor D(34) xor
D(33) xor D(32) xor D(31) xor D(30) xor D(29) xor D(26) xor
D(25) xor D(24) xor D(23) xor D(22) xor D(21) xor D(20) xor
D(19) xor D(18) xor D(17) xor D(16) xor D(15) xor D(14) xor
D(12) xor D(11) xor D(10) xor D(9) xor D(8) xor D(7) xor
D(6) xor D(5) xor D(4) xor D(3) xor D(2) xor D(1) xor
D(0) xor C(0) xor C(1) xor C(2) xor C(3) xor C(4) xor
C(5) xor C(6) xor C(11) xor C(12) xor C(13) xor C(14) xor
C(15);

return NewCRC;

end nextCRC16_D64;

end PCK_CRC16_D64;

```