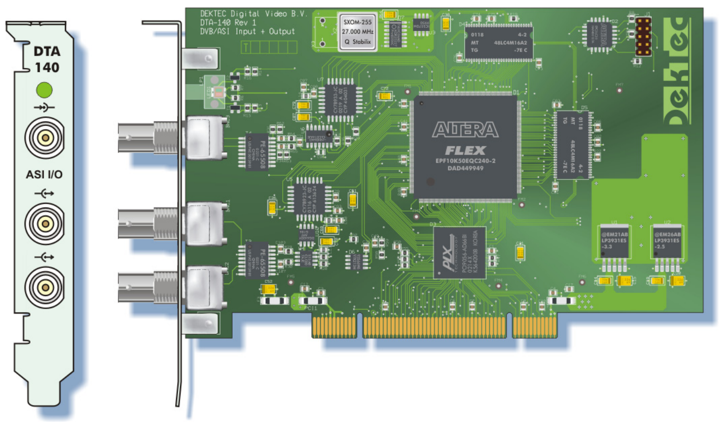
DTA-140

DVB/ASI Input+Output Adapter for PCI Bus

- Input and Output at Full 214 Mbps
- 8-MB Input Buffer, 8-MB Output Buffer
- Time Stamping

FEATURES

- High-speed input and output, compliant to DVB/ASI-C as defined in DVB document A010 rev 1 and EN50083
- Parallel operation of input and output at the full DVB/ASI bit-rate range from 0 to 214 Mbit/s
- Adaptive cable equalisation
- Optional time stamp per packet
- Synchronised and raw receive modes
- Automatic recognition and adjustment of inverted DVB/ASI input signals
- Counters for bit-rate measurement, 8B/10B code violations and synchronisation errors
- LED indicator shows lock and synchronisation status of input circuitry
- 8-Mbytes input buffer for large jitter tolerance; independent 8-Mbytes output buffer
- Double-buffering: DVB/ASI output available on two BNC connectors
- Inverted DVB/ASI for special tests
- Continuous mode or burst mode
- PCI rev 2.2, 32 bit, **66 MHz**



- Comes with free:
  - Windows-2000/XP device driver
  - Linux Software Development Kit
  - API for developing custom applications
  - Example streamer and grabber code
  - *DtGrabber*: Windows grabber program

APPLICATIONS

- Universal DVB/ASI input/output adapter for PC-based applications that record, play and/or process MPEG-2 transport streams
- iTV server
- IP-to-DVB and DVB-to-IP gateway

KEY ATTRIBUTES

Parameter	Value
Physical Layer	DVB/ASI-C
DVB/ASI Connector	75 Ω BNC (3x*)
Transmit Bit Rate	0…214 Mbit/s
Transmit Bit Rate Resolution	<1 bit/s
Transmit Bit Rate Stability	±10 ppm
Maximum Transmit Jitter**	70 ns p-p
Input Bit Rate	0…214 Mbit/s
Input Return Loss	17 dB
Error Free Cable Length	300m max

* 1x input, 2x doubly-buffered output
** In continuous mode

PCI-BUS CHARACTERISTICS

PCI rev 2.2, 32 bit, **66 MHz**, Master/Target, Universal 3.3V/5V

RELATED PRODUCTS

Type	Description
DTA-100	DVB/ASI **Output** Adapter for PCI Bus
DTA-120	DVB/ASI **Input** Adapter for PCI Bus
DTC-300	Transport-Stream Player **Software**
DTC-320	Real-Time Analyser **Software**
DTC-7X3	"Ray" **Stream Station**

Table of Contents

1. General Description 3
1.1. Typical Application 3
1.2. Free Software 3
1.3. Commercial Applications 3
1.4. Block Diagram 4
1.4.1. PCI-9056 Bus Interface 4
1.4.2. Master Control 4
1.4.3. Transmit FIFO 4
1.4.4. Clock Synthesizer 4
1.4.5. ASI Transmit Logic 4
1.4.6. Equaliser 5
1.4.7. ASI Receive Logic 5
1.4.8. Receive FIFO 5
1.5. References 5
1.6. Document Overview 5
2. External Interfaces 7
2.1. Overview 7
2.2. DVB/ASI Input 7
2.2.1. Applicable Standard 7
2.2.2. LED Status Indicator 7
2.2.3. LED Power-Up Sequence 8
2.3. DVB/ASI Output 8
2.3.1. Applicable Standard 8
2.3.2. Transmission Jitter 8
3. Transmitting Data 10
3.1. Buffer Model 10
3.1.1. Single-Buffer Model 10
3.1.2. Double-Buffer Model 10
3.1.3. Multiple Buffers 11
3.1.4. Transmit FIFO 11
3.2. Synchronisation 11
3.2.1. DMA-Driven 11
3.2.2. Transmission-Driven 12
3.3. Buffer Management 12
3.3.1. Ping-Pong, DMA-Driven 12
3.3.2. DMA-Driven, Start Up 13
3.3.3. Ping-Pong, Transmission-Driven 13
3.3.4. Transmission-Driven Start Up 14
4. Receiving Data 15
4.1. Buffer Model 15
4.1.1. Single-Buffer Model 15
4.1.2. Double-Buffer Model 15
4.1.3. Multiple Buffers 16
4.2. Synchronisation 16
4.3. Ping-Pong Buffering 16
4.3.1. Start Up 17
4.4. Packet Alignment 17
4.4.1. Fixed Buffer Structure 17
4.4.2. Switching Receive Mode 17
4.5. Time Stamping 18
4.5.1. Block Diagram 18
4.5.2. Time-Stamp Format 18
5. Configuration Space 19
6. Target Address Space 21
6.1. General Registers 25
6.1.1. General Control (00h) 25
6.1.2. General Status (04h) 26
6.1.3. Programming (08h) 26
6.1.4. Ref. Clock Count (0Ch) 27
6.2. Transmit Registers 27
6.2.1. Transmit Control (04h) 27
6.2.2. Transmit Status (08h) 30
6.2.3. Transmit Clock (0Ch) 32
6.2.4. Transmit FIFO Size (14h) 32
6.2.5. Transmit FIFO Load (18h) 33
6.2.6. Transmit Diagnostics (1Ch) 33
6.2.7. Transmit Loop-Back Data (20h) 34
6.2.8. Transmit-FIFO Data (40h…7Ch) 34
6.3. Receive Registers 35
6.3.1. Receive Control (04h) 35
6.3.2. Receive Status (08h) 37
6.3.3. Receive FIFO Load (18h) 39
6.3.4. Receive Diagnostics (1Ch) 39
6.3.5. Receive Loop-Back Data (20h) 40
6.3.6. Receive Valid Count (30h) 40
6.3.7. Receive Violation Count (34h) 41
6.3.8. Receive-FIFO Data (40h…7Ch) 41
7. Vital Product Data 42
7.1. Serial EEPROM Lay-Out 42
7.2. VPD ID String 42
7.3. VPD Read-Only Resources 42
7.4. VPD Read-Write Resources 43
7.5. Reading VPD Data 44
7.6. Writing VPD Data 44
8. Hardware Details 46
8.1. Aspects of DMA 46
8.1.1. DMA vs. Direct PCI Access 46
8.1.2. DMA Buffers 46
8.2. Transmit FIFO 47
8.2.1. Coarse-Grain Model 47
8.2.2. Fine-Grain Model 47
8.2.3. Minimum Delay 49
8.3. Latency 50
8.3.1. PCI-Bus Latency 50
8.3.2. Interrupt Latency 50
8.3.3. Scheduling Latency 50
8.3.4. Compensating Latencies 50
8.4. Reading the Entire Rx FIFO 51

1. General Description

The DTA-140 is a PCI adapter that combines a doubly-buffered DVB/ASI output and a flexible DVB/ASI input into a single-slot unit.

The input and output channel run independently from each other, both support the full DVB/ASI range[1] from 0 to 214 Mbit/s.

1.1. Typical Application

The DTA-140 is typically deployed as a DVB/ASI Input/Output unit in an industrial PC[2]. The low cost and high performance of the PC are leveraged to create cost-effective digital-video solutions.

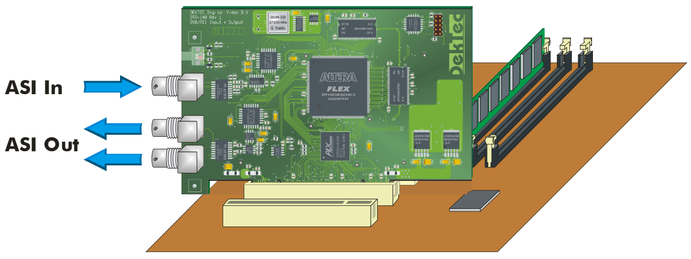


Figure 1. The DTA-140 enables the receiving and transmitting of DVB/ASI signals from a single PCI slot in a PC.

The DTA-140 is well-suited for applications that record and play-out DVB/ASI streams, and for real-time processing of MPEG-2 Transport Streams.

Examples of digital-video applications with the DTA-140 include:
- iTV inserter;
- DVB/SI or PSIP inserter;
- Bit-rate transcoder;
- Logo inserter;
- Advertisement inserter.

1.2. Free Software

The DTA-140 comes with the following freely downloadable software:
- Windows-2000/XP WDM driver;
- Linux device driver;
- *DTAPI* library, available both for Windows and Linux.

Note
- The Linux device driver and DTAPI are part of the open-source Linux SDK.

The device driver implements "low-level" operations that require direct access to the DTA-140 hardware, such as initiation and co-ordination of DMA transfers, handling interrupts and reading and writing of Vital Product Data (VPD, §7).

The DTAPI is a thin layer of user-mode software that packages the driver functions into an easy to use API.

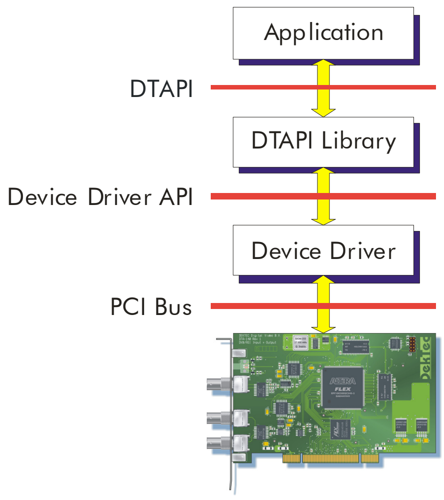


Figure 2. Software stack for the DTA-140.

1.3. Commercial Applications

DEKTEC offers a number of standard applications running on top of (amongst others) the DTA-140:
- DTC-300 streaming software;
- DTC-320 real-time analyser software.

Please refer to www.dektec.com for further details.

DEKTEC's commercial applications use a licensing system that ties the software to a specific DTA-140: the software will only run if an appropriate license code has been programmed into the on-board PROM.

Licenses can be purchased upfront, or the board can be upgraded in the field with the license code.

[1] The carrier frequency of a DVB/ASI signal is 270 MHz. is. The maximum effective bit rate is 214 Mbit/s.
[2] The DTA-140 can also be used in desktop or mobile PC, or any other platform that supports the PCI bus.

1.4. Block Diagram

Figure 3 shows a conceptual block diagram of the DTA-140. Transport-stream data (and all other control data) enters the board via the *PCI-9056 Bus Interface*. The *Master-Control* block relays the packets to the *Transmit FIFO*. From there, an *ASI Adapter* translates the signals to the required DVB/ASI format.

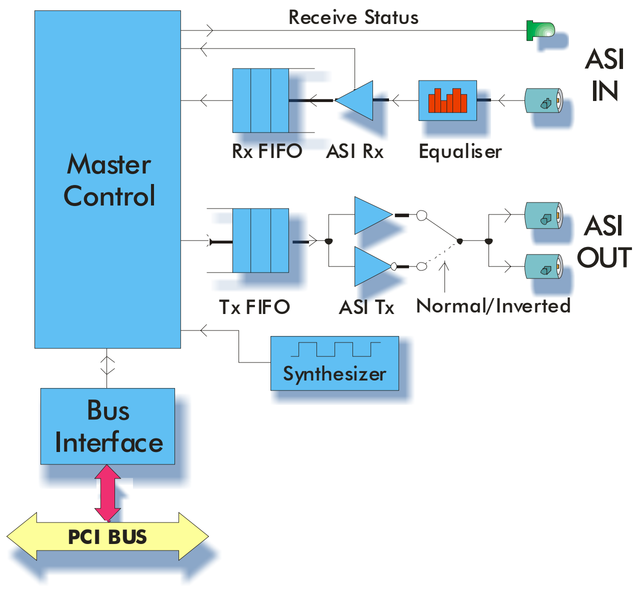


Figure 3. Conceptual block diagram of the DTA-140.

The other functional blocks in the diagram support the various special functions offered by the DTA-140. The major functional units are discussed below in separate subsections.

1.4.1. PCI-9056 Bus Interface

The DTA-140 uses the PCI-9056 IC made by PLX for interfacing to the PCI Bus. The PCI-9056 also implements the DMA functions required for high-speed streaming of transport packets, with minimal host interaction.

The PCI-9056 supports large host buffers with *scatter/gather* DMA mode. Such buffers may get fragmented through allocation / de-allocation of memory by OS-components. A scatter/gather list glues the buffer together without requiring software intervention.

Whenever appropriate, this specification provides information on the way the PCI-9056's registers should be used on the DTA-140. To obtain more details on the operation of the PCI-9056, please refer to the *PCI 9056 Data Book*.

1.4.2. Master Control

Master Control is the block that supervises the DTA-140. It contains the control registers for configuring and operating the DTA-140. These registers are accessible from the PCI Bus. A description of the registers is given in §6.

1.4.3. Transmit FIFO

The Transmit FIFO is a large buffer (8-MBytes) for transport-stream data. The DTA-140 uses a standard SDRAM to implement this buffer economically.

The function of the Transmit FIFO is to enable the sustained generation of a low-jitter DVB/ASI stream. The Transmit FIFO compensates for latencies on the PCI Bus and for scheduling jitter of the software that generates the Transport Stream.

The DTA-140 supports a mode to automatically insert null packets when the Transmit FIFO underflows. In "raw" mode the DVB/ASI output signal is invalidated by stuffing with K28.5 characters.

1.4.4. Clock Synthesizer

The transport-stream clock for the DVB/ASI output channel is generated by a clock synthesizer. This circuit can be programmed to any bit rate between 0 and 214 Mbit/s, with a resolution of better then 1 bit/s.

1.4.5. ASI Transmit Logic

The DVB/ASI Transmit Logic converts the bytes coming from the Transmit FIFO into a DVB-compliant ASI signal which is buffered on two BNC connectors.

A special feature of the DTA-140 is a "Normal" / "Inverted" switch in the output circuitry. This enables the generation of a DVB/ASI signal with inverted polarity, for testing inputs for their ability to properly decode normal and inverted-polarity ASI signals.

1.4.6. Equaliser

The receive circuitry starts with a cable equaliser, which automatically adapts to equalise any cable length from zero to three hundred meters (Belden 8281 or equivalent).

1.4.7. ASI Receive Logic

The equalised signal is processed by a DVB/ASI receiver, which de-serialises the signal, synchronises to the MPEG-2 packet structure and writes the data into the Receive FIFO.

Special features of the ASI Receiver include:

- Time stamping: a sample of a reference clock is attached to each packet, so that the time of entrance of the packet can be derived.
- ASI polarity detection. Inverted DVB/ASI signals are automatically detected and corrected.
- Packet-size conversion, e.g. storage of 188-byte packets, irrespective of the input packet size (188 or 204).

The status of the ASI Receiver is made visible to the outside world through the Receive Status LED.

1.4.8. Receive FIFO

The Receive FIFO is the counter part of the Transmit FIFO, but then for the receive channel. It has the same size (8-MBytes) uses standard SDRAM for its implementation.

The Receive FIFO provides a great deal of freedom for the receiving software. While the application processes data, new transport-stream data is buffered in the Receive FIFO.

1.5. References

- *Interfaces for CATV / SMATV Headends and Similar Professional Equipment, DVB DOCUMENT A010 rev.1, May 1997* – This is the original DVB document that specifies physical interfaces for the interconnection of signal processing devices for professional digital-television equipment. One of the interfaces described is DVB/ASI.
  DVB document A010 document has also been issued as *CENELEC EN50083-9*.
- *ISO/IEC 13818-1, Information technology – Generic coding of moving pictures and associated audio information: Systems, April 27th, 1995*, also known as "MPEG-2 Systems" – Specification of the structure of a MPEG-2 Transport Stream.
- *DTAPI: C++ API for DTA-series of Digital-Video PCI-Bus Cards, DEKTEC Digital Video B.V., 2001* – Specification of **DTAPI**: the C++ interface to access the DTA-140 functions at a higher level of abstraction than would be possible using direct device-driver calls.
- *PCI 9056 Data Book, PLX Technology, V0.91b, February 2002* – Specification of the PCI 9056, the chip used on the DTA-140 to interface with the PCI bus. Use this document if you need to program the PCI-9056 directly, e.g. when writing a custom device driver.
  The latest version of this document is available on line at http://www.plxtech.com.
- *PCI Local Bus Specification, Revision 2.2, December 18, 1998* – Formal specification of the protocol, electrical, and mechanical features of the PCI bus.

1.6. Document Overview

This specification describes the details relevant for operating the DTA-140. The information herein is primarily intended for device driver writers and for software developers that have to access the DTA-140 directly from a real-time operating system.

The WDM device driver and DTAPI library encapsulate many programming details of the DTA-140. Users of DTAPI may find this document useful for providing background information, but do not need to master each and every detail.

- *Section 1* introduces the main features of the DTA-140.
- *Section 2* describes the physical interfaces of the DTA-140. For the DVB/ASI output, this section explains the two DVB/ASI output modes: *burst*- and *continuous*- mode.

- *Section 3* provides a detailed description of synchronisation and buffer-management issues to stream data with the DTA-140 in an efficient and reliable way.
- *Section 4* does the same for synchronisation and buffer management in the Receive channel.
- *Section 5* lists the PCI Configuration-Space registers supported by the DTA-140.
- *Section 6* describes the *operational registers* on the DTA-140. These registers can be used to control and monitor the streaming of digital-video data.
- *Section 7* defines the structure of *Vital Product Data (VPD)* as supported by the DTA-140 and other DEKTEC PCI cards.
- Finally, *Section 8* examines a number of hardware details that are irrelevant for most applications, yet important to applications that require the ultimate in performance

2. External Interfaces

2.1. Overview

The lay-out of the DTA-140's PCI bracket is shown in Figure 4 below.

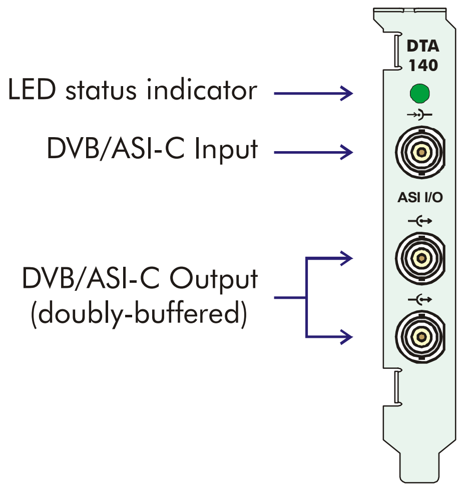


Figure 4. DTA-140 physical interfaces.

The LED indicator provides the status of the DVB/ASI input beneath it.

The DVB/ASI output is doubly buffered. The same signal is available on two BNC connectors.

2.2. DVB/ASI Input

The upper BNC connector on the DTA-140 bracket is the DVB/ASI input.

2.2.1. *Applicable Standard*

The DVB/ASI input is compliant to the *Asynchronous Serial Interface on coaxial cable (ASI-C)*, as defined in DVB Document A010.

The input connector is 75-Ω BNC.

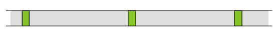
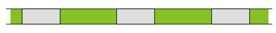
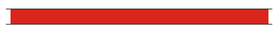
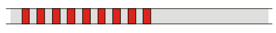
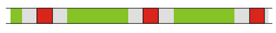
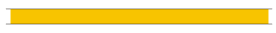
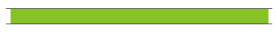
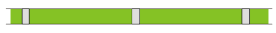
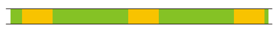
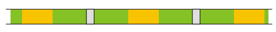
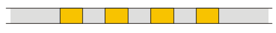
2.2.2. *LED Status Indicator*

The bi-colour (red/green) LED above the input connector displays the status of the DVB/ASI-C input signal.

The status is indicated using a combination of colour-encoding and a flashing pattern. Table 1 shows the various indications.

The flash-pattern pictures show the colour of the LED over time. Grey represents "LED off".

Table 1. LED Status Indications

LED Status	Meaning
No data	
	No signal
	ASI carrier without TS, or data rate too low (less than about 1 kbps)
Error conditions	
	No lock
	Code violation(s)
	Receive-FIFO overflow
Valid data	
	Valid data but unknown packet size
	Valid 188-byte packets
	Valid 188-byte packets, inverted ASI
	Valid 204-byte packets
	Valid 204-byte packets, inverted ASI
Special	
	Receive-channel reset

Note

- The LED indication may also be *overruled* by software (refer to §6.1.1.7, §6.3.1.7).

The *error conditions* "No lock" and "Code violation" may be caused by a non-DVB/ASI signal (e.g. a SMPTE signal), a badly attached BNC connector, a coax cable of poor quality or a cable that's too long.

Connecting a cable to the DTA-140 almost certainly leads to a temporary "No lock" (red) indication.

The *valid-data* indications show packet size and inverted-ASI status. If the DVB/ASI input signal is recognised as inverted DVB/ASI, then a very short flash, only just noticeable, interrupts the normal packet-size indication.

2.2.3. *LED Power-Up Sequence*

Just after power-up, the LED flashes a few times to indicate that the board is initialising. During this short period, the usual meaning of the LED indications does not apply. Instead, the LED pattern shows the board type and the firmware revision. An example start-up pattern is shown in Figure 5 below.

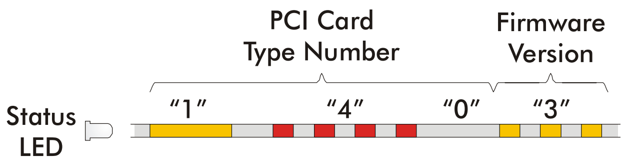


Figure 5. Power-up pattern for DTA-140 with (as example) firmware version 3.

2.3. DVB/ASI Output

The bottom two BNC connectors on the DTA-140 bracket is the DVB/ASI input.

2.3.1. *Applicable Standard*

The DVB/ASI outputs are compliant to the *Asynchronous Serial Interface on coaxial cable (ASI-C)*, as defined in DVB Document A010.

The output connectors is 75-Ω BNC.

2.3.2. *Transmission Jitter*

DVB/ASI is characterised by a fixed symbol rate (27 MSymbol/s), but a variable transport rate. Special stuff characters (K28.5) are inserted to fill up the difference.

The 'real' data (8→10 encoded bytes) and K28.5 stuff characters can be distributed over time in different ways. The DTA-140 supports two modes:

- *Continuous Mode*. MPEG-2 transport packets are spread evenly over time.
- *Burst Mode*. Packets are sent in bursts.

Figure 6 below illustrates these two modes. Short 4-byte packets are used in the figure to make the point. In reality, packet bursts consist of 188 or 204 consecutive bytes.

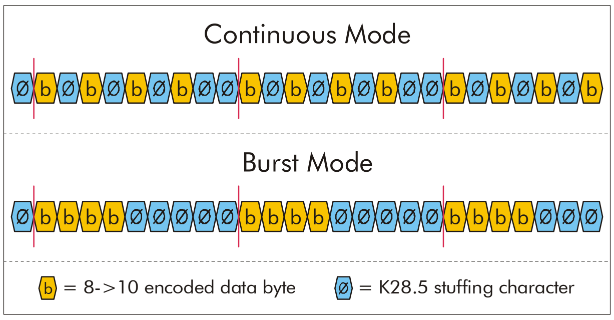


Figure 6. Continuous- and Burst- Mode for hypothetical 4-byte packets.

2.3.2.1. Continuous Mode

Continuous Mode is the recommended mode for normal operation. The DTA-140 limits the amount of transmission jitter to the theoretical minimum of ±35 ns.

In DVB/ASI some jitter is necessarily introduced because:

- DVB/ASI characters align to a 27-MHz grid (transmission time is "quantised"), and
- The DVB/ASI specification requires that each MPEG-2 Sync character has to be preceded by at least two K28.5 stuffing characters.

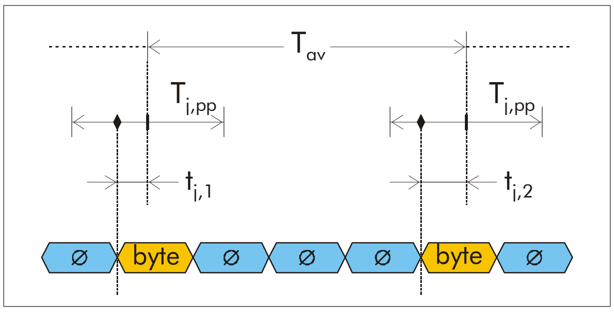


Figure 7. Jitter in the DVB/ASI output stream. ∅ = K28.5 stuffing character

T_{av}	Average time between two characters. T_{av} is the reciprocal of the transport rate: $8/R_{TS}$ (factor 8 to get time between *bytes*).
$T_{j,pp}$	Peak-to-peak jitter interval = 70 ns
t_j	Jitter of a single byte: time difference between average and actual position of an encoded DVB/ASI output byte.

The DTA-140 hardware ensures that the DVB/ASI characters are always output within the 70-ns time interval labelled $T_{j,pp}$, corresponding to a maximum jitter of ±35 ns.

2.3.2.2. Burst Mode

The DTA-140 supports Burst Mode for special test applications, e.g. to check whether certain equipment with a DVB/ASI input accepts packets transmitted in bursts.

In Burst Mode, all 188 or 204 bytes in a packet are sent in one continuous data burst, without interspersed stuff bytes. The period between the end of a packet and the start of the next packet is stuffed with K28.5 characters.

Note

- In Burst Mode, the data bytes *within* a packet are sent in a burst, but the packets *as a whole* are distributed smoothly over time (no bursts of packets).
- If the transmission mode is **Raw**, the DTA-140 is ignorant of the packet size, and Burst Mode is not supported.

3. Transmitting Data

This section discusses *transmission* of MPEG-2 transport packets using the DTA-140, with the focus on *real-time* operation, this is the avoidance of "gaps" in the transmitted stream. Refer to Section 4 for a similar treatise on *reception* of MPEG-2 data.

3.1. Buffer Model

Transport-stream data to be transmitted by the DTA-140 is buffered in a cascade of two buffers:

1. The *DMA Buffer*, located in host memory. The host processor writes transport packets into this buffer.
2. The *Transmit FIFO*, which is an 8-Mbytes hardware buffer located on the DTA-140.

The DMA Controller on-board of the DTA-140 transfers the data from DMA Buffer to the Transmit FIFO. This data transfer incurs hardly any overhead for the host processor.

Notes

- Data can also be transferred from DMA Buffer to Transmit FIFO by direct PCI-read cycles, but this is far less efficient. Refer to §8.1.1 for a discussion.
- Details of the structure of DMA buffers (like scatter-gather), are described in §8.1.2.

From the viewpoint of the DMA Buffer, one process is writing in the buffer (host processor) and one process is reading from the buffer (DMA controller). Obviously, for correct data transfer, these processes must be synchronised to each other.

3.1.1. Single-Buffer Model

The first (and simplest) buffer model is to use a single DMA Buffer, and to alternate DMA- and host access to this buffer:

- While the host processor is writing data to the DMA Buffer, the DMA Controller remains idle.
- While the DMA Controller reads data from the DMA Buffer, the host processor remains idle.

This option is sub-optimal, but not as bad as it may seem. The DMA phase is relatively short, because DMA is very fast (>100 MByte/s).

Example

- The output transport rate is 40 Mbit/s (5 MByte/s) and the data-transfer size is 1 MByte. Then the length of the DMA phase is 10 ms, while the length of the processing phase is as long as 190 ms. Just 5% of time is "lost" with transferring data.

The single-buffer model may break down in the following cases:

- At very high output rates, e.g. 200 Mbps.
- When the PCI Bus is heavily loaded, e.g. by data transfers for other DEKTEC cards. This may significantly increase the data-transfer time.

3.1.2. Double-Buffer Model

In the double-buffer model, the DMA Buffer is segmented into two sub-buffers: DMA Buffer 1 and DMA Buffer 2. The DMA Controller writes data to one sub-buffer, and in parallel the host processes packets from the other sub-buffer.

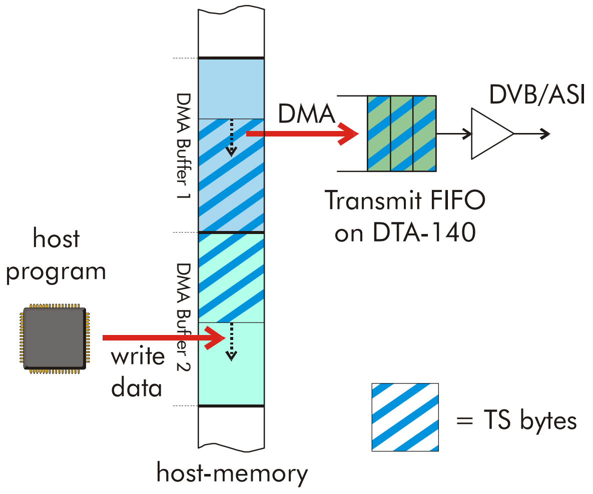


Figure 8. Tx double-buffer model. The DMA Controller reads data from DMA Buffer 1, while the host writes new data into DMA Buffer 2.

Figure 8 shows a snapshot of the double-buffer model in action. The buffer is split in DMA Buffer 1 and DMA Buffer 2, together forming one circular buffer. The blue-hatched area represents data that has been written to the buffer earlier.

MPEG-2 data in DMA Buffer 1 is transferred to the Transmit FIFO by the DMA Controller on the DTA-140. At the same time, new packets are written in DMA Buffer 2 by the program running on the host.

If all of the data in DMA Buffer 1 has been read, and DMA Buffer 2 has been filled completely, the function of both DMA Buffers is swapped.

3.1.3. Multiple Buffers

The double-buffer model can be generalised to a scheme with N DMA buffers. Each process handles one DMA Buffer. When a process is done with one buffer it continues with the next buffer, unless the other process is still busy with that buffer.

This specification will not dig deeper into transmission schemes with more than two DMA Buffers.

3.1.4. Transmit FIFO

The contents of the DMA Buffer are transferred to the *Transmit FIFO* on-board of the DTA-140. The Transmit FIFO converts the write bursts to a smooth output stream: The hardware reads bytes at the output side of the Transmit FIFO reads bytes at a constant rate – the transport-stream rate – and converts the data bytes to DVB/ASI format.

For most applications, the Transmit FIFO can be considered a large conventional FIFO that can buffer 8-Mbytes of data. The maximum load of the Transmit FIFO can be reduced artificially through the Transmit-FIFO-Size register (§6.2.4).

Notes

- The Transmit-FIFO-Size register does not determine the 'real' Transmit-FIFO size: The actual size of the Transmit FIFO is always 8 Mbytes (dependent on SDRAM size; §6.2.4). The Transmit-FIFO-Size register is the high-water mark that controls demand for DMA. Whenever the Transmit-FIFO load becomes greater or equal than the value in the Transmit-FIFO-Size register, the DTA-140 disables DMA until the Transmit-FIFO load drops below Transmit-FIFO Size again.
- It is hard to envisage an application that benefits from a reduced Transmit-FIFO Size. Therefore, most applications can simply program Transmit-FIFO Size to its maximum of 8 Mbytes.
- The hardware implementation of the Transmit FIFO is more complex than suggested by the model described above; Refer to §8.2 for particulars. The implementation details may be relevant for special applications, e.g. if one wants to achieve a low end-to-end delay at a low bit rate. Most regular applications need not bother about the finer points of the Transmit-FIFO implementation.

3.2. Synchronisation

The host processor generates transport packets into a DMA Buffer, while the DTA-140 reads the packets from the buffer and transmits them. Obviously, for a continuous Transport Stream, packet generation must be synchronised to packet transmission.

The DTA-140 transmission hardware supports two synchronisation methods:
- DMA-driven: §3.2.1;
- Transmission-driven: §3.2.2.

3.2.1. DMA-Driven

With *DMA-Driven* synchronisation, the host software locks generation of new packets to the completion of the last DMA transfer:

1. Host writes packets in DMA Buffer;
2. DMA Controller transfers data to DTA-140;
3. Host waits for DMA completion;
4. Go to Step 1.

When the host generates packets faster than the DTA-140 can transmit them, which must be the case for sustained operation without discontinuities, it seems that the Transmit FIFO on the DTA-140 would overflow.

However, DMA-driven synchronisation still works reliably because the DTA-140 hardware implements *demand-mode* DMA: DMA transfers are requested on the PCI Bus only as long as the DTA-140's Transmit FIFO has empty buffer space left. When the FIFO becomes *fully loaded*[3], the DMA process stalls. When FIFO space is available again, DMA resumes.

In other words: The Transmit FIFO cannot overflow in DMA-driven operation. The handshaking hardware prevents this from happening.

3.2.2. Transmission-Driven

An alternative method of synchronisation is *Transmission Driven*: the generation of new packets is locked directly to the transmission of (previously-computed) packets.

1. Initial load of packets (>*x*) is written to the Transmit FIFO;
2. Host starts transmission;
3. Host computes *x* new packets;
4. (In parallel to step 3) Host tracks number of transmitted packets and waits until *x* packets have been sent.
5. The *x* new packets are transferred to Transmit FIFO.
6. Go to Step 3.

The number of transmitted packets can be tracked using the so-called *Periodic Interrupt*, which is generated precisely every 2^{21} cycles of the 27-MHz reference clock (this is: about once every 77.7 ms). The number of packets transmitted in this time interval is easily computed by multiplying the packet rate by $\frac{2^{21}}{27 * 10^6}$.

Example

- Suppose 188-byte packets are transmitted at 40 Mbps, corresponding to a packet rate of 26596 transport packets per second. Multiplying by 77.7 ms yields the number of packets transmitted between two periodic interrupts: 2066.
  So, after each periodic interrupt the program may write 2066 new packets to the DTA-140.

[3] "Fully loaded" in the sense that the load of the Transmit FIFO has become greater or equal than the value programmed in the FIFO-Size register.

The example above uses approximations. In practice, the synchronisation method will work correctly in the long run only if the *full fractional* part ("bits behind the comma") is taken into account. Otherwise, slight timing errors may accumulate and cause underflow or overflow in time.

Taking into account "all bits" involves using in the computations the full 32-bits value programmed in the Transmit-Clock (§6.2.3) register. As the periodic interrupt and the transmit clock are derived from the same master 27-MHz clock oscillator, the number of packets transmitted can be tracked exactly.

3.3. Buffer Management

This section discusses how to manage DMA Buffers such that synchronisation of packet generation by the host and packet transmission by the DTA-140 is achieved.

3.3.1. Ping-Pong, DMA-Driven

As explained in §3.1, efficient streaming of data to the DTA-140 requires at least two DMA Buffers. The host program computes new packets and writes them to one buffer. At the same time, the DMA Controller on the DTA-140 reads data from the other buffer. When both DMA is done and a new buffer with packet data has been filled, the DMA Buffers swap function. This process continues ad infinitum.

The use of two buffers that swap function after each cycle – also known as *Ping-Pong* buffering – is illustrated in Figure 9.

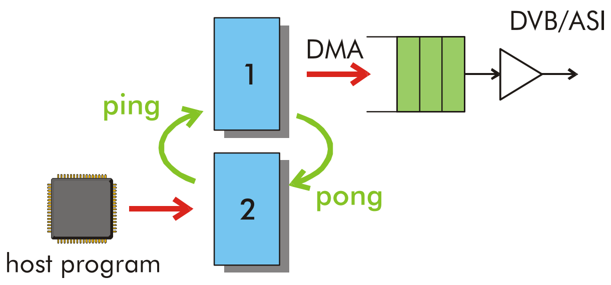


Figure 9. Ping-Pong buffering. The host writes packets in one buffer, while the DMA Controller reads packets from the other buffer. When both are finished, the "Ping-Pong" swap is executed.

DMA-Driven flow control in a double-buffering scheme is illustrated in the message-sequence chart shown in Figure 10.

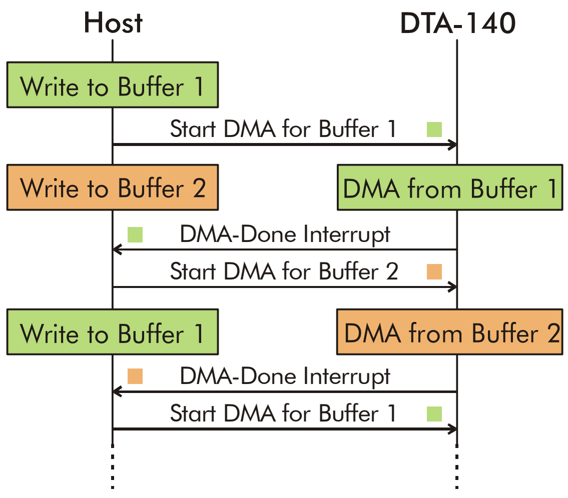


Figure 10. Ping-Pong buffer management using DMA-Driven flow control. After the host has filled a buffer and DMA on the other buffer is done, the buffers swap function.

The DMA-Done Interrupt is the handshake signal. It triggers the host to compute new packets and to initiate a new DMA transfer.

Note

- While waiting for the DMA-Done Interrupt, it is opportune for a device driver to sleep the process and give another process a chance to run.

It is instructive to ponder on the limiting factor in the Ping-Pong process: Host or DMA?

In the beginning, when the Transmit FIFO is not filled yet, the host will be the bottle neck. DMA transfers complete quickly, because DMA is only constrained by the PCI Bus.

After some time, when the Transmit FIFO is filled, DMA will become the limiting factor. The effective average rate of DMA will drop to the transmit rate, because of demand-mode DMA (Refer to §3.2.1). The host has to wait for the DMA-Done Interrupt before it can execute the next *Ping-Pong* swap.

3.3.2. DMA-Driven, Start Up

The robustness of the start-up phase of the Ping-Pong process can be improved by pre-loading the Transmit FIFO with data before starting the Ping-Pong process.

This can be accomplished with the Transmit-Control field (§6.2.1.4) in the Transmit-Control register. Preloading is enabled by setting Transmit Control to **Hold** and issuing DMA transfers until the Transmit FIFO is (almost) filled. Then, Transmit Control is set to **Send**. The DTA-140 starts packet transmission and the Ping-Pong process may begin.

The controlled start-up method prevents potential underflow of the Transmit-FIFO in the start-up phase[4].

3.3.3. Ping-Pong, Transmission-Driven

The Ping-Pong buffering scheme can also be used in combination with Transmission-Driven flow control. Figure 11 shows two Ping-Pong cycles using Transmission-Driven synchronisation, assuming that the process has already stabilised.

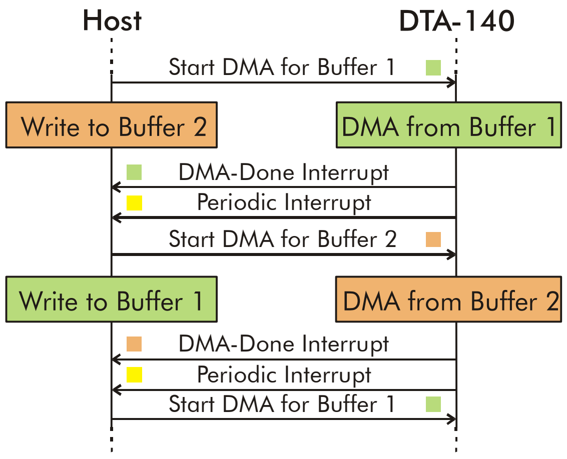


Figure 11. Ping-Pong buffer management using Transmission-Driven flow control. The buffers swap function when both the DMA-Done Interrupt and the Periodic Interrupt have occurred.

The most obvious difference with DMA-Driven flow control is that the host software now has to wait for <u>both</u> DMA done <u>and</u> the periodic

[4] If the process is started with Transmit Control set to **Send**, the Transmit FIFO may underflow early in the process (e.g. due to PCI-Bus latency), because no buffer load has been accumulated yet for compensation.

interrupt before proceeding with the next Ping-Pong cycle[5].

A minor complication specific to Transmission-Driven flow control is that, in general, the number of packets transmitted between two Periodic Interrupts is not integer. The number of packets computed in cycle N must be rounded to the nearest integer. The length of the DMA transfer in cycle N+1 must be adapted to this size.

Note

- In Transmission-Driven flow control it makes no sense to artificially reduce the FIFO size. The FIFO-Size register should be programmed to its maximum value.

3.3.4. Transmission-Driven Start Up

The start-up procedure for Transmission-Driven flow control is somewhat more complicated then the procedure for DMA-Driven flow control.

In the start-up phase, the Transmit FIFO has to get an initial load. Figure 11 shows an example start-up procedure in which two DMA-Buffer loads are transferred to the Transmit FIFO.

Note

- This example uses initial loading in the "Ping-Pong way". A method with a single buffer could have been used too.

The start-up scenario proceeds through the following steps, assuming that the board and DMA Buffers have been properly initialised.

1. Host sets `TxCtrl` to **Hold**.
   This enables DMA, but no packets will be transmitted yet.
2. Host fills DMA Buffer 1 with packets, and starts DMA.
3. Host fills DMA Buffer 2, while the DTA-140 reads DMA Buffer 1 in parallel.
4. Host waits for DMA-Done interrupt for DMA Buffer 1, then starts DMA for DMA Buffer 2.

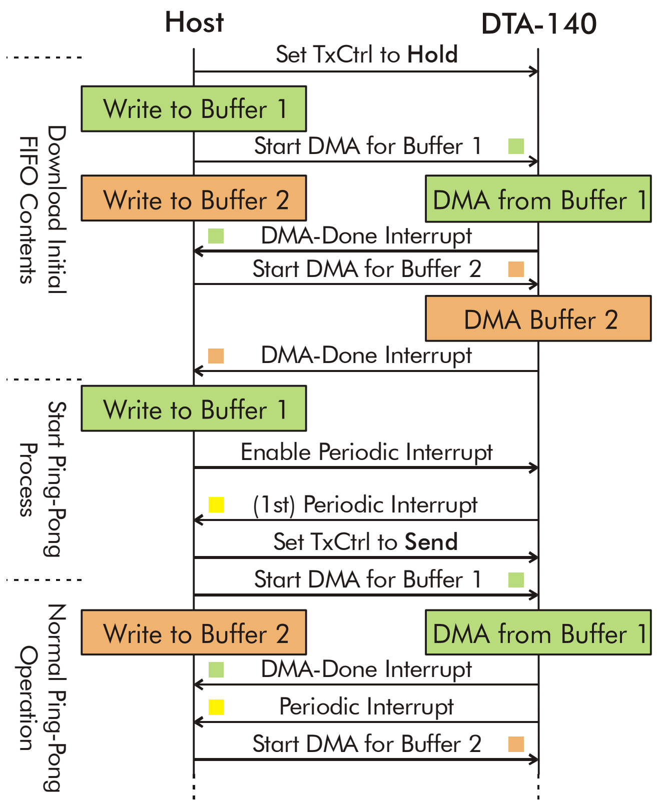


Figure 12. Start-up phase of Ping-Pong buffer management with Transmission-Driven flow control. In this scenario, the initial load of the Transmit FIFO is set to 2 times the size of the DMA Buffer.

5. Host waits for DMA-Done interrupt for DMA Buffer 2.
   The Transmit FIFO now has an initial load.
6. Host fills DMA Buffer 1 with packet data. This will be the first buffer that is transferred in "Normal Ping-Pong Operation".
7. Host enables Periodic Interrupts and waits for the occurrence of the first Periodic Interrupt.
8. Host sets `TxCtrl` to **Send**.
9. Normal Ping-Pong Operation is entered.

[5] Under normal circumstances, DMA is much faster than packet transmission. This means that the DMA interrupt will almost always occur long before the periodic interrupt arises. However, for robustness the software should also wait for the DMA interrupt.

4. Receiving Data

This section describes several aspects of *receiving* MPEG-2 transport packets with the DTA-140.

The discussion will be quite similar to Section 3's treatise on transmission, only with the processes working in the opposite direction.

4.1. Buffer Model

When receiving an MPEG-2 Transport-Stream with the DTA-140, the incoming data is buffered in a cascade of two buffers:

1. The *Receive FIFO*, which is an 8-Mbytes hardware buffer located on the DTA-140.
2. The *DMA Buffer*, which is a "software" buffer located in host memory. Data in the DMA Buffer is available for further processing by the host processor.

Just like data transfers for transmission, the DMA Controller on the DTA-140 also transferred data from Receive FIFO to DMA Buffer, without much overhead for the host processor.

Notes

- Data can also be transferred from Receive FIFO to DMA Buffer by direct PCI-read cycles, but this is far less efficient. Refer to §8.1.1 for a discussion.
- Details of the structure of DMA buffers (like scatter-gather), are described in §8.1.2.

From the viewpoint of the DMA Buffer, one process is writing in the buffer (DMA controller) and one process is reading from the buffer (host processor). These processes must be synchronised to each other, otherwise DMA may overwrite data that is just being processed.

4.1.1. Single-Buffer Model

As with transmission, the simplest buffer model is to use a single DMA Buffer, and to alternate DMA- and host access to this buffer:

- While the DMA Controller is writing data to the DMA Buffer, the host application remains idle.
- While the host processes the data from the DMA Buffer, the DMA controller remains idle.

The single-buffer model is sub-optimal, but not as bad as it may seem. The DMA phase is relatively short, because DMA is very fast (>100 MByte/s). In the "processing phase", new incoming data can be easily accommodated in the Receive FIFO.

Example

- Data is entering the system at 40 Mbit/s (5 MByte/s) and the data-transfer size is 1 MByte. Then the length of the DMA phase is 10 ms, but the length of the processing phase is as long as 190 ms. Just 5% of time is "lost" with transferring data.

The single-buffer model may break down in the following cases:

- At very high input rates, e.g. 200 Mbps.
- When the PCI Bus is heavily loaded.

4.1.2. Double-Buffer Model

In the double-buffer model, the DMA Buffer is segmented into two sub-buffers: DMA Buffer 1 and DMA Buffer 2. The DMA Controller writes data to one sub-buffer, and in parallel the host processes packets from the other sub-buffer.

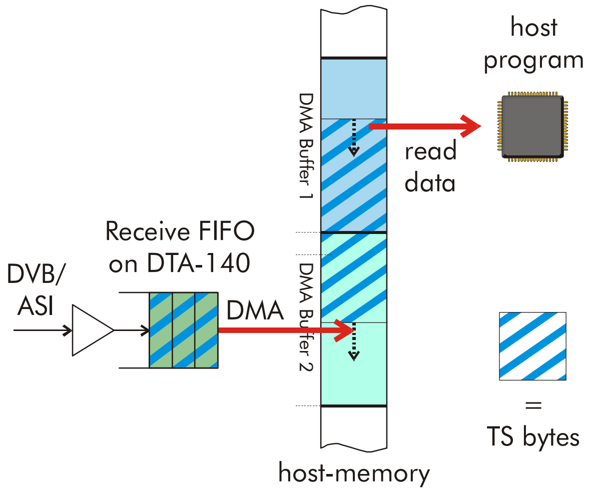


Figure 13. Rx double-buffer model. The host reads data from DMA Buffer 1, while the DMA Controller writes new data into DMA Buffer 2.

Figure 13 shows a snapshot of the double-buffer model in action. DMA Buffer 1 and DMA Buffer 2 together form one circular buffer. The blue-hatched area represents Transport-Stream data that still has to be processed by the host program.

MPEG-2 data in DMA Buffer 1 is processed by the program running on the host. At the same time, the DMA Controller transfers new packets from the Receive FIFO on the DTA-140 to DMA Buffer 2.

If all of the data in DMA Buffer 1 has been read, and DMA Buffer 2 has been filled completely, the function of the DMA Buffers is swapped.

4.1.3. Multiple Buffers

The double-buffer model can be generalised to a scheme with N DMA buffers. Each process handles one DMA Buffer. When a process is done with one buffer it continues with the next buffer, unless the other process is still busy with that buffer.

This specification will not dig deeper into reception methods that use more than two DMA Buffers.

4.2. Synchronisation

The DTA-140 reads transport packets, and the host processor processes these packets. Obviously, packet reception must be synchronised to packet processing.

The DTA-140 reception hardware is tailored for *DMA-Driven* synchronisation: The host software locks packet processing to the completion of DMA transfers.

DMA transfers from Receive FIFO to DMA buffer can be initiated before the Receive FIFO contains sufficient data to complete the DMA transfer. The DMA-done interrupt signals to the host software that another buffer load is available for processing by the application.

DMA-driven synchronisation works reliably because the DTA-140 hardware implements *demand-mode* DMA: DMA transfers are requested on the PCI Bus only as long as the DTA-140's Receive FIFO has data available. When the FIFO becomes empty, the DMA process stalls. When new data enters the Receive FIFO again, DMA process restarts.

In other words: The Receive FIFO cannot underflow in DMA-driven operation. The handshaking hardware prevents this from happening.

4.3. Ping-Pong Buffering

This section discusses the combination of double buffering and DMA-driven synchronisation.

The use of two buffers that swap function after each cycle – also known as *Ping-Pong* buffering – is illustrated in Figure 14.

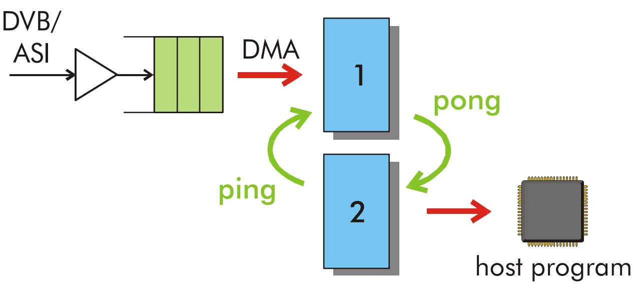


Figure 14. Ping-Pong buffering. The DMA Controller writes packets from the Receive FIFO into one buffer, while the host reads packets from the other buffer. When both are finished, the "Ping-Pong" swap is executed.

DMA-Driven flow control in a double-buffering scheme is illustrated in the message-sequence chart shown in Figure 15.

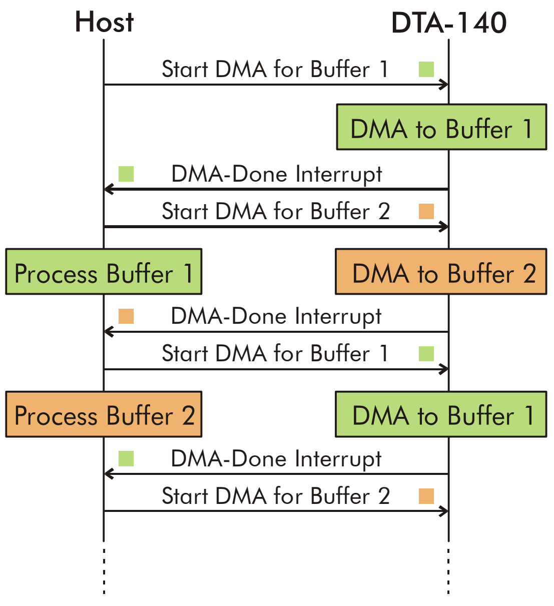


Figure 15. Ping-Pong buffer management using DMA-Driven flow control. After the host has read a buffer and the DMA write to the other buffer is done, the buffers swap function.

The DMA-Done Interrupt acts as handshake signal. It triggers the host to read and process new packets, and to initiate a new DMA transfer.

Note

- While waiting for the DMA-Done Interrupt, the device driver should sleep the process and give another process a chance to run.

4.3.1. Start Up

The following procedure is recommended to start the Ping-Pong process:

1. Allocate DMA Buffers and associated scatter/gather DMA descriptors.
2. Clear the Receive FIFO on the DTA-140 by setting `RxClfFifo` in the Receive-Control register (§6.3.1.10). This will also reset the Receive Control (§6.3.1.3) to **Idle**.
3. Initiate the first DMA transfer from Receive FIFO to DMA Buffer.
   As the Receive FIFO is still in idle mode, no data will be transferred yet.
4. Set Receive-Control to **Rcv**.
   At this time, transport packets are allowed to enter the Receive FIFO. Shortly thereafter, the first data will be transferred to the DMA Buffer.
5. Normal Ping-Pong Operation has started!

4.4. Packet Alignment

The DTA-140 supports alignment of transport packets in the DMA buffer:

- MPEG-2 sync bytes (0x47) are stored at 32-bit aligned byte positions;
- After setting Receive-Control to **Rcv**, the first data transferred to the DMA Buffer will start at a packet boundary.

Packet alignment is provided for in Receive Modes **St188**, **St204** and **StMp2**, but not in **StRaw**. In raw mode, the DTA-140 does not take the packet structure into account, and sync bytes may appear at any relative address.

4.4.1. Fixed Buffer Structure

Packet alignment opens up the possibility of a fixed packet lay-out in the DMA Buffer(s). Hereto, the Receive Mode must be set to **St188** or **St204**, and the buffer size must be chosen a multiple of the packet size.

Figure 16 shows an example of packet alignment in an 1880-byte DMA-Buffer (10 packets), with Receive Mode set to **St188**.

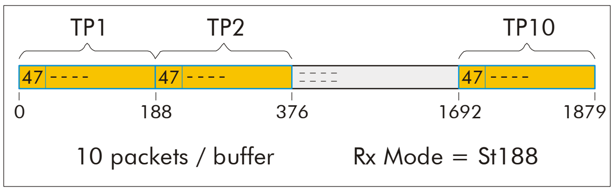


Figure 16. Transport packets stored at fixed locations. The buffer size must be a multiple of the packet size.

Notes

- Receive Mode **StMp2** cannot be used reliably for packet alignment in a DMA Buffer: When the size of the incoming transport packets changes dynamically, alignment will likely break.
- It is good software-engineering practice to test for packet alignment, even if packets "should always be aligned". If the test would fail, the input channel should be reset and input processing restarted.

4.4.2. Switching Receive Mode

The DTA-140 supports *dynamic* switching of Receive Mode, this is going from one mode to another while, at the same time, data is being stored in the Receive FIFO. However, such dynamic switching may <u>break</u> packet alignment.

Note

- Dynamic switching may even break 32-bit alignment of MPEG-2 sync bytes.

If you want to switch Receive Mode, while reliably maintaining packet alignment (and 32-bit aligment), a channel reset is required:

1. Set Receive Control to **Idle**;
2. Reset receive channel;
2. Set Receive Mode to the desired value;
3. Set Receive Control to **Rcv**.

An inevitable side effect of the use of a channel reset is the "loss" of a number of incoming packets.

4.5. Time Stamping

This section discusses the use of time stamping, as a method to measure the time of entrance of incoming transport packets. The measurement is made available to the application in the form of *time stamps*, attached to the packets.

Applications of time-stamping include:

- Real-time processing of transport streams, e.g. for PCR correction.
- Synchronisation of multiple incoming transport streams in (re-)multiplexers.
- Measurement of real-time packet-delivery jitter.

4.5.1. Block Diagram

Time stamps are derived from a *reference clock counter*, running at 40.5 Mhz. Every time a packet enters the DTA-140, a sample of the counter is taken and the value is prepended to the packet.

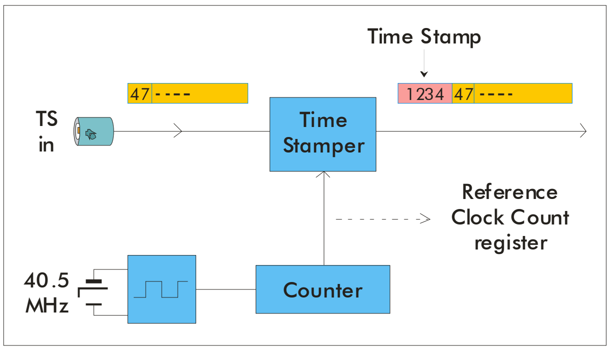


Figure 17. Schematic block diagram of the time-stamping hardware.

The value of the reference-clock counter can also be observed from the PCI bus, by reading the Reference-Clock-Count register (§6.1.4).

4.5.2. Time-Stamp Format

Time stamping can be enabled by setting the `RxTimeStamp` the (§6.3.1.4) in the Receive-Control register. Receive Mode must be one of **St188**, **St204** or **StMp2**.

Note

- Time stamping is not supported in raw mode (**StRaw**).

Time stamps are stored in Little-Endian format into the 4 bytes that are placed before the packet, as shown in Table 2.

Table 2. Time Stamps – Packet Format

Addr[6]	Value	Description
00h	bit 7..0	Time stamp in 4 bytes
01h	bit 15..8	
02h	bit 23..16	
03h	bit 31..24	
04h	47h	MPEG-2 Sync
:	:	Transport-packet header and payload
BFh	XX	Last byte of packet

The application program can find the location of the time stamps by synchronising to the MPEG-2 Sync bytes.

Another option is to choose a buffer size that is a multiple of the packet + time-stamp size (e.g. a multiple of 192 for **St188**). The first four bytes of the buffer will then always contain a time stamp. See also §4.4.1.

[6] Relative address in packet.

5. Configuration Space

The DTA-140 acts as a single logical PCI Bus device. It implements the configuration registers required for identifying the device, control PCI Bus functions, and provide PCI Bus status.

Table 3 displays the address map of registers defined in configuration space:

- Black fields indicate configuration registers supported by the DTA-140.
- Red-text cells represent registers supported by the PCI bridge chip, but not used for operating the DTA-140.
- Grey-text cells represent registers defined in the PCI Local Bus Revision 2.2 specification, but not supported on the DTA-140.

Table 3. Configuration Space – Address Map

Address Offset	Byte 3	Byte 2	Byte 1	Byte 0
00h	Device ID		Vendor ID	
04h	Status Register		Command Register	
08h	Class Code			Revision ID
0Ch	BIST	Header Type	Latency Timer	Cache Line Size
10h	PCI Base Address 0; used for memory-mapped configuration registers (PCI 9056)			
14h	PCI Base Address 1; not used			
18h	PCI Base Address 2; used for memory-mapped operational registers[7]			
1Ch	PCI Base Address 3; not used			
20h	PCI Base Address 4; not used			
24h	PCI Base Address 5; not used			
28h	*Cardbus CIS Pointer; not supported*			
2Ch	Subsystem ID		Subsystem Vendor ID	
30h	Expansion ROM Base Address Register; not used			
34h	*Reserved*			Next_Cap = 40h
38h	*Reserved*			
3Ch	Maximum Latency	Minimum Grant	Interrupt Pin	Interrupt Line
40h	Power Management Capabilities; not used		Next_Cap = 48h	Capability ID = 01h
44h	Power Management Register; not used			
48h	Hot Swap; not used		Next_Cap = 4Ch	Capability ID = 06h
4Ch	VPD; used for manufacturing / service		Next_Cap = 00h	Capability ID = 03h
50h	VPD; used for manufacturing / service			

Table 4 shows a short description of the registers in configuration space.

[7] Operational registers are mapped in "Local Address Space 0" of the PCI 9056.

Table 4. Configuration Space – Register Overview

Register	Bits	RW*	Value	Short Description
Vendor ID	16	R	10B5h	Identifies PLX as manufacturer of the PCI interface chip.
Device ID	16	R	9056h	Identifies the PCI interface chip (PCI 9056).
Command Register	16	RW	-	Provides coarse control on the ability to generate and respond to PCI cycles.
Status Register	16	RWC	-	Status of PCI-Bus relevant events.
Revision ID	8	R	0	Revision number of your DTA-140.
Class Code	24	R	FF0000h	Generic function of the DTA-140.
Cache Line Size	8	R	16	System cache line size in units of 32-bit words.
Latency Timer	8	RW	-	Amount of time in PCI-Bus-clock units that the DTA-140 may retain ownership of the PCI Bus.
Header Type	8	R	0	Specifies layout of configuration addresses 10h through 3Fh and single / multiple functions.
BIST	8	R	0	PCI Built-In Self Test (BIST).
PCI Base Address 0	32	RW	-	Memory attributes and base memory address for memory accesses to PCI-9056 registers
PCI Base Address 2	32	RW	-	Memory attributes and base memory address for memory accesses to *Local Address Space 0*, which is used to access the DTA-140's operational registers (Refer to Table 5).
Subsystem Vendor ID	16	R	14B4h	Identifies the manufacturer of the DTA-140. Subsystem Vendor ID and Subsystem Device ID are leased from Philips BE.
Subsystem Device ID	16	R	D140h	Identifies the PCI card as a DTA-140.
Interrupt Line	8	RW	-	Interrupt line routing information.
Interrupt Pin	8	R	01h	Interrupt pin used by the DTA-140.
Minimum Grant	8	R	10h	Length of time (in 250-ns units) DTA-140 would like to retain master ship of the PCI Bus.
Maximum Latency	8	R	1Ah	Frequency in which the DTA-140 would like to gain access to the PCI Bus.

6. Target Address Space

The DTA-140's operational registers are mapped in Local-Address Space 0 of the PCI 9056. The PCI Base address of these registers is specified in BAR2. All accesses to the operational registers shall be 32-bit transfers.

Table 5. Operational Registers – Memory Map

Address Offset	Byte 3	Byte 2	Byte 1	Byte 0
General				
00h	General Control			
04h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	General Status
08h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 Programming
0Ch	Reference-Clock Count			
10h … 7Ch	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
Transmit (Tx)				
80h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
84h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 Transmit Control	
88h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 Transmit Status	
8Ch	Transmit Clock			
90h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
94h	0 0 0 0 0 0 0 0	Transmit FIFO Size		
98h	0 0 0 0 0 0 0 0	Transmit FIFO Load		
9Ch	Transmit Diagnostics			
A0h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Tx Loop-Back Data
A4h … BCh	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
C0h … FCh	Transmit FIFO Data			
Receive (Rx)				
100h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
104h	0 0 0 0 0 0 0 0	0 0 Receive Control		
108h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 Receive Status		
10Ch … 114h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
118h	0 0 0 0 0 0 0 0	Receive FIFO Load		
11Ch	Receive Diagnostics			
120h	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Rx Loop-Back Data
124h … 12Ch	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
130h	Receive Valid Count			
134h	Receive Violation Count			
138h … 13Ch	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
140h … 17Ch	Receive FIFO Data			

Table 6. Operational Registers – General Registers

Register	Field	Bit Pos	#	RWC*	Short Description
General Control	`PE`	0	1	RW	Serial EEPROM Program Enable
	`PRE`	1	1	RW	Serial EEPROM Protect Register Enable
	`Reset`	2	1	W	Reset DTA-140 circuitry
	`reserved`	3	1	R	Not used
	`PerIntEn`	4	1	RW	Periodic-Interrupt Enable
	`reserved`	7…5	3	R	Not used
	`FirmwRev`	15…8	8	R	Firmware Revision
	`TypeNum`	23…16	8	R	Type Number: 140 for DTA-140
	`LedControl`	24	1	RW	Take over LED Control
	`LedGreen`	25	1	RW	State of Green LED
	`LedRed`	26	1	RW	State of Red LED
General Status	*`reserved`*	3…0	4	R	Not used
	`PerInt`	4	1	RC	Periodic Interrupt
Programming	`TRST`	0	1	RW	Control of TRST# pin
	`TDO`	1	1	R	Status of EPC2 TDO pin
	`TMS`	2	1	W	Control of EPC2 TMS pin
	`TCK`	3	1	RW	Control of EPC2 TCK pin
	`TDI`	4	1	RW	Control of EPC2 TDI pin
	`ProgramEpc`	5	1	RW	Program EPC2
Reference Clock	`RefClkCnt`	31…0	32	R	Reference-Clock Count

Table 7. Operational Registers – Transmit Registers

Register	Field	Bit Pos	#	RWC*	Short Description
Transmit Control	`TxMode`	1…0	2	RW	Transmit Mode: **188**/**204**/**Add16**/**Raw**
	`TxBurstMode`	2	1	R	DVB/ASI Burst Mode
	`reserved`	3	1	R	Reserved for expansion of `TxMode`
	`TxPckStuff`	4	1	RW	Null Packet Stuffing On/Off
	`TxCtrl`	6…5	2	RW	Transmit Control: **TxIdle**/**TxHold**/**TxSend**
	`reserved`	8…7	2	R	Reserved for expansion of `TxCtrl`
	`TxUflIntEn`	9	1	RW	Transmit-FIFO-Underflow Interrupt Enable
	`TxSyncIntEn`	10	1	RW	Synchronisation-Error Interrupt Enable
	`reserved`	21..11	11	R	Not used
	`TxClrFifo`	22	1	W	Clear Transmit FIFO
	`TxRst`	23	1	RW	Reset Transmit Channel
	`TxInvAsi`	24	1	W	Invert DVB/ASI signal

Transmit Status	TxFifoFilled	0	1	R	Transmit-FIFO-Filled flag
	reserved	3…1	3	R	Not used
	TxSdramSize	7…4	4	R	SDRAM Size = Maximum Tx-FIFO size
	reserved	8	1	R	Not used
	TxUflInt	9	1	RC	Transmit-FIFO-Underflow Interrupt
	TxSyncInt	10	1	RC	Transmit-Synchronisation-Error Interrupt
Transmit Clock	TxClock	31…0	32	RW	Bit rate of transmitted DVB/ASI signal
Transmit FIFO Size	TxFifoSize	23…0	24**	RW	Size of Transmit FIFO in #bytes
Transmit FIFO Load	TxFifoLoad	23…0	24**	R	Current Load of Transmit FIFO in #bytes
Transmit Diagnostics	TxSfData	7…0	8	R	Data at output of Smoothing FIFO
	TxLoopBack	8	1	RW	Loop-Back mode
	reserved	9	1	R	Not used
	TxBfFull	10	1	R	Burst-FIFO Full flag
Tx Loop-Back Data	TxSfDataNxt	7…0	8	R***	Data at output of Smoothing FIFO
Transmit FIFO Data	TxFifoData	31…0	4x8	RW	Transport-stream data: 4 bytes at a time

Table 8. Operational Registers – Receive Registers

Register	Field	Bit Pos	#	RWC*	Short Description
Receive Control	RxMode	1…0	2	RW	Receive Mode: **St188**/**St204**/…
	RxAsiInv	3…2	2	RW	Invert ASI input: **Auto**/**Normal**/**Invert**
	reserved	4	1	R	Not used
	RxCtrl	5	1	RW	Receive Control: **Idle**/**Rcv**
	reserved	6	2	R	Reserved for expansion of RxCtrl
	RxTimeStamp	7	1	RW	Insert Time Stamps before packets
	reserved	8	1	R	Not used
	RxOvfIntEn	9	1	RW	Receive-FIFO Overflow Interrupt Enable
	RxSyncIntEn	10	1	RW	Synchronisation-error Interrupt Enable
	reserved	18…11	8	R	Not used
	RxLedControl	19	1	RW	Take over Rx-LED Control
	RxLedGreen	20	1	RW	State of Green Rx LED
	RxLedRed	21	1	RW	State of Red Rx LED
	RxClrFifo	22	1	RW	Clear Receive FIFO
	RxRst	23	1	RW	Reset Receive Channel

Table 8. Operational Registers – Receive Registers

Register	Field	Bit Pos	#	RWC*	Short Description
Receive Status	`RxPckSize`	1…0	2	R	Packet size: **Rx188**/**Rx204**/**RxInv**
	`reserved`	3…2	2	R	Not used
	`RxSdramSize`	7…4	4	R	SDRAM Size = Size of Receive FIFO
	`reserved`	8	1	R	Not used
	`RxOvfInt`	9	1	RC	Receive-FIFO Overflow Interrupt
	`SyncInt`	10	1	RC	Synchronisation-error Interrupt
	`reserved`	13…11	3	R	Not used
	`RxAsiCD`	14	1	R	DVB/ASI Carrier Detect
	`RxAsiLock`	15	1	R	Locked to DVB/ASI input signal
	`RxRateOk`	16	1	R	Input Rate "Ok" (not too low)
	`RxAsiInv`	17	1	R	Invert DVB/ASI input signal – status
Receive FIFO Load	`RxFifoLoad`	23…0	24**	R	Current Load of Receive FIFO in #bytes
Receive Diagnostics	*`reserved`*	7…0	8	R	Not used
	`RxLoopBack`	8	1	RW	Loop-back mode
	`reserved`	9	1	R	Not used
	`RxRfRateOvf`	10	1	R	Receive-FIFO Rate Overflow
	`RxSdFull`	11	1	R	SDRAM Full
Rx Loop-Back Data	`RxLbData`	7..0	8	W	Write data to Receive FIFO
Rx Valid Count	`RxValidCnt`	31…0	32	R	Sample of DVB/ASI Valid-byte Counter
Rx Violation Count	`RxViolCnt`	31…0	32	R	Sample of Violation Counter
Receive-FIFO Data	`RxFifoData`	31…0	4x8	R	Transport-Stream data: 4 bytes at a time

* R=Readable, W=Writeable, C=Clearable (clear when a '1' is written to bit position).
** Number of bits depends on `TxSdramSize`. Shown size (24-bits) is valid for 8-Mbyte SDRAM.
*** Reading from the Loop-Back Data register causes next byte to be read from smoothing FIFO.

6.1. General Registers

The General Registers contain control and status bits that are not directly related to the DVB/ASI input- and output- channel on the DTA-140.

The lay-out of the General Registers is similar to that of other DTA-1xx PCI adapter cards, to simplify the writing of generic device drivers that can handle multiple DEKTEC cards.

6.1.1. General Control (00h)

The lay-out of the General-Control register is shown in Table 9.

Table 9. General-Control Register – Format

Bit	Mnem	Description
0	`PE`	Serial EEPROM Program Enable
1	`PRE`	Serial EEPROM Protect Register Enable
2	`Reset`	Soft Reset
3	*`reserved`*	
4	`PerIntEn`	Periodic-Interrupt Enable
7…5	*`reserved`*	
15…8	`FirmwRev`	Firmware Revision
23…16	`TypeNum`	Value=140
24	`LedControl`	Take over LED Control
25	`LedGreen`	State of Green LED
26	`LedRed`	State of Red LED

Bits 31…27 of this register are tied to '0'.

6.1.1.1. PE – Program Enable

The Program-Enable field directly controls the PE signal of the serial EEPROM. The EEPROM can only be programmed when this bit is set to '1'. During normal operation, PE should remain '0'.

6.1.1.2. PRE – Protect Register Enable

The Protect-Register-Enable field directly controls the PRE signal of the serial EEPROM. It enables the write-protection mechanism in the EEPROM. During normal operation, this field should remain '0'.

Warning

- Issuing a write-protection command to the serial EEPROM is an irreversible operation. Incautious use of the PRE bit may destroy the Vital-Product Data read/write capability!

6.1.1.3. Reset – Software Reset

Writing a '1' to the Reset bit issues a "soft" reset to the DTA-140. The following fields and logic circuitry are affected:

- `RxCtrl` in the Receive-Control register is reset to **TxIdle**.
- `TxCtrl` in the Transmit-Control register is reset to **TxIdle**.
- The entire contents of the Input- and Transmit FIFO are cleared, including data in the Burst-, Bulk- and Smoothing FIFO (§8).
- The Receive-FIFO- and Transmit-FIFO Load registers are reset to zero.
- The Interrupt-Status flags in the Receive-Status and Transmit-Status register are cleared.
- A number of internal state machines are reset.

Other fields in the operational registers are not affected, notably:

- Receive Mode (`RxMode`) in the Receive-Control register.
- Loop-Back Mode in the Diagnostics register.
- Interrupt-Enable bits in the Transmit-Control register: interrupts that were enabled remain enabled.
- `PE` and `PRE` in the General-Control register.

Other fields in the operational registers are not affected, notably:

- Receive Mode (`RxMode`) and Transmit Mode (`TxMode`).
- Transmit clock (`TxClock`).
- Transmit-FIFO Size (`TxFifoSize`).
- Receive- and Transmit Loop-Back Mode in the Diagnostics registers.
- Interrupt-Enable bits in the Transmit-Control register: interrupts that were enabled remain enabled.
- `PE` and `PRE` in the General-Control register.

This behaviour is by design, so that the data pipelines in the DTA-140 can be reset without compromising other processes running on the PCI card.

The Reset bit is write-only. The write operation triggers the reset action: it is not required to reset the bit to '0' again. The next time a '1' is written to the Reset bit, the board will be reset again.

6.1.1.4. PerIntEn – Periodic Interrupt Enable

Writing a '1' to this bit enables[8] the *Periodic Interrupt* (§).

6.1.1.5. FirmwRev – Firmware Revision

This read-only field identifies the current revision level of the DTA-140's firmware.

Note

- The Firmware Revision level is independent of the DTA-140 *board* revision (which can be read from VPD).

6.1.1.6. TypeNum – Type Number

The Type-Number field identifies the board in a straightforward way. For the DTA-140, the field's value is fixed to **140** (decimal).

Note

- Apart from this field, the board's type number is also encoded in the Vital Product Data (VPD), which is the primary source of descriptive data about a board. The purpose of the Type-Number field is to provide a convenient way for device drivers to distinguish between different kinds of DTA-1xx boards at start-up.

6.1.1.7. LedControl – Take over LED Control

When this field is set to '0', the state of the bi-colour LED indicator on the PCI bracket is determined by the hardware, as described in §XXX.

When this field is set to '1', the receive hardware is "disconnected" from the LED indicator and the LED is controlled directly by fields `LedGreen` and `LedRed`.

[8] To actually enable the Periodic Interrupt on the PCI Bus, the Local-Interrupt-Input-Enable bit in the PCI9054's Interrupt Control/Status register must also be set to '1'.

This bit field is reset to '0' upon a hardware- or software reset.

6.1.1.8. LedGreen – State of Green LED

When `LedControl` is '1', this field controls the **green** colour of the bi-colour LED next to the connector on the PCI bracket.

6.1.1.9. LedRed – State of Red LED

When `LedControl` is '1', this field controls the **red** colour of the bi-colour LED next to the connector on the PCI bracket.

6.1.2. General Status (04h)

The lay-out of the General-Status register is shown in Table 10.

Table 10. General-Status Register – Format

Bit	Mnem	Description
3…0	*reserved*	
4	`PerInt`	Periodic Interrupt
Bits 31…5 of this register are tied to '0'.		

6.1.2.1. PerInt – Periodic Interrupt

When set to '1', this bit indicates that the *Periodic Interrupt* is pending. The Periodic Interrupt is generated automatically every 2^{21} clock cycles of the on-board 40.5-MHz reference clock. This corresponds to approximately once every 51.8 ms, or 19.31 times per second.

6.1.3. Programming (08h)

The Programming Register can be used to update the firmware of the DTA-140 ("flashing"). The fields are connected to the JTAG programming lines of the EPC2 EEPROM

Table 11. Programming Register – Format

Bit	Mnem	Description
0	`TRST`	Control of TRST# pin
1	`TDO`	Status of TDO pin
2	`TMS`	Control of TMS pin
3	`TCK`	Control of TCK pin
4	`TDI`	Control of TDI pin
5	`PerInt`	Periodic Interrupt
Bits 31…6 of this register are tied to '0'.		

6.1.4. *Ref. Clock Count (0Ch)*

The Reference-Clock-Count register provides access to the DTA-140's *reference-clock*, which is a free-running counter at a rate of 40.5 MHz.

Table 12. General-Status Register – Format

Bit	Mnem	Description
31…0	`RefClkCnt`	Reference Clock Count

The hardware uses the reference clock for the following purposes:

- Generation of time stamps for incoming transport packets.
- The carrier frequency of the DVB/ASI output is the 40.5-MHz clock divided by 1.5.
- The transmit rate is also directly derived from the 40.5-MHz clock, again with the division factor of 1.5.

The Reference-Clock-Count register enables tracking of the 40.5-MHz clock in software, e.g. to correlate the input time stamps of multiple receive boards.

6.2. Transmit Registers

The transmit registers control the operation of the DVB/ASI output channel on the DTA-140.

The lay-out of the Transmit Registers is the same as, or similar to, the lay-out of such registers on other DTA-1xx PCI adapter cards, to simplify the writing of generic device drivers that can handle multiple DEKTEC cards.

6.2.1. *Transmit Control (04h)*

The Transmit-Control register contains a number of fields that allow the device driver to control transmission-specific functions of the DTA-140.

Table 13. Transmit-Control Register – Format

Bit	Mnem	Description
1…0	`TxMode`	Transmit Mode
2	`TxBurstMode`	DVB/ASI Burst Mode
3	*`reserved`*	
4	`TxPckStuff`	Null-Packet Stuffing
6…5	`TxCtrl`	Transmit Control
8…7	*`reserved`*	
9	`TxUflIntEn`	Tx-FIFO Underflow Interrupt Enable
10	`TxSyncIntEn`	Synchronisation-Error-Interrupt Enable
21…11	*`reserved`*	
22	`TxClrFifo`	Clear Transmit FIFO
23	`TxRst`	Reset Transmit Channel
24	`TxInvAsi`	Invert DVB/ASI signal
Bits 31…25 of this register are tied to '0'.		

6.2.1.1. TxMode – Transmit Mode

Transmit Mode is a 2-bit field in the Transmit-Control register that controls:

- The size of transmitted packets;
- The automatic insertion of extra bytes at the end of transport packets.

Table 14. Transmit Mode – Values

Value	Mode	Definition
`00`	**188**	188-byte packets in FIFO, 188-byte packets transmitted.
`01`	**204**	204-byte packets in FIFO, 204-byte packets transmitted.
`10`	**Add16**	188 byte packets in FIFO, 204-byte packets transmitted; 16 extra bytes appended to each packet.
`11`	**Raw**	No notion of packets. Data bytes are transmitted "as is". Null-packet stuffing is not supported.

The default Transmit Mode is **188**[9]. The DTA-140 assumes that the Transmit FIFO contains transport packets of 188 bytes. Packets are sent unmodified as plain 188-byte packets. The transport rate of the output stream (as defined in the MPEG-2 Systems specification) is equal to the transmit clock.

In Transmit Mode **204**, the DTA-140 expects that transport packets in the Transmit FIFO consist of 204 bytes: an MPEG-2 compliant packet of 188 bytes, followed by 16 "extra" bytes. The software has full control over the contents of these 16 bytes[10].

In mode **204** (and also in the next mode: **Add16**), the transmit-clock rate (Tx-Rate) is greater than the transport-stream rate (TS-Rate). Additional clock cycles are required for outputting the 16 extra bytes. These clock cycles are included in the transmit clock but excluded from the transport-stream rate. The relation between Tx-Rate and TS-Rate can be expressed in a formula (see also Table 20):

$$\text{Mode } \mathbf{204}, \mathbf{Add16}: \quad \text{TxRate} = \frac{204}{188} * \text{TSRate}$$

[9] Mode 188 also exists in the DVB Professional-Interfaces specification. This is the only mode marked 'mandatory' for implementation on equipment producing a DVB-compliant ASI stream.

[10] For example, a fast processor (fed with a smart algorithm), may fill the extra bytes on the fly with a RS(204,188) forward-error-correction code, or any other packet attachment.

In mode **Add16**, the DTA-140 expects that the user supplies 188-byte packets (like in mode **188**), but now appends 16 null bytes to each packet. These inserted bytes may act as a placeholder for external equipment that inserts e.g. a FEC code. The transmit-clock rate is 188/204 times the MPEG-2 transport rate, the same as in mode **204**.

Note

- In transmit modes **188**, **204** and **Add16**, the DTA-140 expects that the user writes *correctly formatted* transport packets of the specified length to the Transmit FIFO. If this assumption fails, the DTA-140 will drop packet data, until synchronisation is achieved again.
- The DTA-140 does not implement "correctly formatted" in a very sophisticated way: The only requirement is that sync bytes appear at the right position…

In **Raw** mode, the DTA-140 has no notion of transport packets. The bytes in the Transmit FIFO are output without any processing. Packet stuffing is not possible (by definition, as in this mode the DTA-140 knows nothing about packets). When the FIFO underflows the output stream is stuffed with K28.5 characters.

Raw mode can be used to simulate special transport-stream faults, e.g. an occasional packet with 187 or 189 bytes for testing synchronisation behaviour of receiving equipment.

Note

- In **Raw** mode, the DTA-140 will *never* drop a byte from the Transmit FIFO.

6.2.1.2. TxBurstMode – DVB/ASI Burst Mode

The Burst-Mode flag controls whether the individual bytes in each transmitted transport packet are spread evenly spread over time (Burst Mode is '0'), or are output in one burst (Burst Mode is '1'). The characteristics of these two modes are explained in §2.3.2.

6.2.1.3. TxPckStuff – Packet Stuffing

The Packet Stuffing bit controls the behaviour of the output stream when no packet data is available in the Transmit FIFO.

- If Packet Stuffing is '0' (Off), the output stream is not stuffed with null packets[11].
- If Packet Stuffing is '1' (On), the output stream is stuffed with null packets. The size of inserted null packets is matched to Transmit Mode.

In Transmit mode **Raw**, Packet Stuffing is not supported: Both packet size and packet boundaries are unknown. Consequently, setting Packet Stuffing to '1' in **Raw** mode has no effect. When the Transmit FIFO underflows in **Raw** mode, transmission of data stalls until data is written to the Transmit FIFO again.

6.2.1.4. TxCtrl – Transmit Control

The Transmit-Control field controls the packet-transmission process. After a power-up condition, Transmit Control is initialised to **TxIdle**.

Table 15. Transmit Control – Values

Value	Mnem	Definition
00	**TxIdle**	No DMA requests, no packet transmission.
01	**TxHold**	Request DMA until Transmit FIFO is filled, no packet transmission.
10	–	Reserved.
11	**TxSend**	Read and transmit packets.

Whenever Transmit Control is set to **TxIdle**, the transmission process becomes inactive. A pending DMA transfer, if any, is suspended. No new DMA transfers are initiated. The DTA-140 stuffs the output stream with null packets if transmit mode is **188**, **204** or **Add16**, and Packet Stuffing is '1'. Otherwise, the DVB/ASI output stream is 'empty' (a constant stream of K28.5 stuffing characters).

In **TxHold** mode, packet transmission is stalled in the same way as in **TxIdle** mode (null-packet stuffing or empty output). However, unlike in **TxIdle** mode, DMA is enabled. If a DMA transfer is set up in the PCI 9056 – and the corresponding Scatter/Gather descriptor(s) has been created – DMA transfers will be initiated so that packet data is written to the Transmit FIFO. DMA continues until the DMA transfer terminates or the FIFO load reaches the high-level watermark specified in the FIFO-Size register.

When Transmit Control is set to **TxSend**, actual transmission of transport packets begins. New DMA data is requested through the PCI 9056, but only when sufficient space is available in the Transmit FIFO.

For a clean start-up of continuous packet transmission, it is required to sequence control operations in the following order:

- Allocate the DMA Buffer(s) in host memory and create the corresponding scatter/gather descriptor list[12];
- Write **TxIdle** to Transmit Control;
- Set-up DMA-transfer in PCI 9056;
- Write **TxHold** to Transmit Control;
- Wait until DMA transfer is completed. Initiate additional DMA transfers in the PCI 9056 until the Transmit FIFO contains the desired initial load;
- Write **TxSend** to Transmit Control.

For data-only applications, the transition through **TxHold** might be omitted. In that case, the DTA-140 may insert null packets due to Transmit-FIFO underflow, but this is irrelevant for data-only applications.

6.2.1.5. TxUflIntEn – Underflow Interrupt Enable

Writing a '1' to this bit enables the Transmit-FIFO-*Underflow Interrupt* (§6.2.2.3). Refer to Table 17 for a description of the interrupt status flags.

6.2.1.6. TxSyncIntEn – Synchronisation-Error Interrupt Enable

Writing a '1' to this bit enables the *Synchronisation-Error Interrupt* (§6.2.2.4). Refer to Table 17 for a description of the interrupt status flags.

[11] The output stream is stuffed with K28.5 characters. This is an inherent characteristic of DVB/ASI.

[12] Only required if the DMA Buffer is scattered over physical memory.

6.2.1.7. TxClrFifo – Clear Transmit FIFO

Writing a '1' to this field[13] clears the contents of the Transmit FIFO and thereby resets the FIFO-Load register to zero. Furthermore, `TxCtrl` in the Transmit-Control register is reset to **Idle**.

In all Transmit Modes except **Raw**, the Transmit FIFO can be cleared without introducing a truncated packet. The packet framing structure (distance between sync bytes) is not broken.

If the Transmit FIFO is transmitting data while the clear operation occurs, then the contents of the last packet sent may get corrupted. Nevertheless, the packet's length will be preserved.

On the other hand, if no data is being transmitted (Transmit Control is **Idle** and/or Transmit FIFO empty), no corrupted data will be sent at all.

To avoid generation of a packet with corrupted contents, DEKTEC recommends the following procedure for clearing the Transmit FIFO:
- Set Transmit Control to **Idle**.
- Wait a few packet periods so that the DTA-140's internal pipeline is flushed.
- Write '1' to `ClrFifo`.

6.2.1.8. TxRst – Reset Transmit Channel

Writing a '1' to this field[14] issues a full reset of the DVB/ASI Transmit Channel. This operation includes the actions brought about by clearing the Transmit FIFO using `TxClrFifo`.

The following fields and logic circuitry are affected by a Reset Transmit Channel action:
- `TxCtrl` in the Transmit-Control register is reset to **TxIdle**.
- The entire contents of the Input- and Transmit FIFO are cleared, including data in the Burst-, Bulk- and Smoothing FIFO.
- The Transmit-FIFO Load register is reset to zero.
- The transmit interrupt conditions and the corresponding status flags in the Transmit-Status register are cleared.
- The state machines used in the Transmit-Channel hardware are reset.

Other fields in the operational registers are <u>not</u> affected, notably:
- Transmit Mode (`TxMode`).
- Transmit clock (`TxClock`).
- Transmit-FIFO Size (`TxFifoSize`).
- Transmit Loop-Back Mode in the Diagnostics register.
- Interrupt-Enable bits in the Transmit-Control register; any interrupt that was enabled before the reset operation remains enabled.

Resetting the transmit channel may lead to the generation of a truncated packet. Therefore, `TxRst` should only be used to recover from a lock-up situation. In all other cases, the use of `TxClrFifo` is preferred.

6.2.1.9. TxInvAsi – Invert DVB/ASI Signal

This field controls the polarity of the DVB/ASI output signal.

Table 16. Invert DVB/ASI Signal – Values

Value	Mnem	Definition
0	**Normal**	No inversion.
1	**InvAsi**	Generate an inverted DVB/ASI signal.

Inverted DVB/ASI may be used as a validation tool to check whether a DVB/ASI input is resilient to an inverted DVB/ASI signal.

6.2.2. Transmit Status (08h)

The Transmit-Status register contains a number of fields that allow the device driver to read status information from the DTA-140's transmit channel.

[13] The Tx`ClrFifo` bit is write-only. The write operation triggers the clear action: it is not required to reset the bit to '0' again.

[14] The Tx`Rst` bit is write-only. The write operation triggers the clear action: it is not required to reset the bit to '0' again.

Table 17. Transmit-Status Register – Format

Bit	Mnem	Description
0	`TxFifoFilled`	Transmit-FIFO Filled
3…1	*reserved*	
7…4	`TxSdramSize`	SDRAM Size
8	*reserved*	
9	`TxUflInt`	Transmit-FIFO Underflow Interrupt
10	`TxSyncInt`	Synchronisation Error
Bits 31…11 of this register are tied to '0'.		

The interrupt-status flags (bit 9 and 10) in this register share common behaviour:

- An interrupt-status flag is set when the corresponding condition occurs. The flag remains set until it is explicitly cleared.
- Writing a '1' to the flag clears the interrupt-status flag, and also clears the PCI interrupt[15] (unless another interrupt condition is pending).
- The interrupt-status flag only leads to an interrupt if the corresponding interrupt-enable bit in register `TxControl` is set, and interrupts in the PCI 9056 have been enabled.
- The operation of the interrupt-status bits is independent from the state of the interrupt-enable bit: If the interrupt-enable bit is '0', the interrupt-status flag still latches the corresponding condition.

6.2.2.1. TxFifoFilled – Transmit-FIFO Filled

When set to '1', this read-only bit indicates that the DTA-140's Transmit FIFO is filled up to the level specified in the FIFO-Size register. The FIFO-Filled flag may be used in a controlled initialisation procedure to signal to the device driver that the Transmit FIFO is filled completely.

Notes

- The buffer-management examples provided in this specification do not use the Transmit-FIFO Filled flag.
- In DMA-driven operation (§3.2.1), the DTA-140 attempts to keep the Transmit FIFO filled by requesting DMA data whenever empty space is available in the Transmit FIFO. The Transmit-FIFO Filled flag will occasionally be set to '1'. This does not indicate an error condition.

6.2.2.2. TxSdramSize – SDRAM Size

Transmit-SDRAM Size is a static read-only field that indicates the size of the SDRAM-memory chip used for the transmit channel on DTA-140. The Transmit-SDRAM size determines the maximum size of the Transmit FIFO.

Table 18. SDRAM Size – Values (Mbytes)

Value	Size	Comment
0000	8 MB	Standard size.
0001	16 MB	May be supported in future revisions of the DTA-140.
0010	32 MB	May be supported in future revisions of the DTA-140.
other	*reserved*	

6.2.2.3. TxUflInt – Underflow Interrupt

When set to '1', this bit indicates that an underflow condition has occurred for the Transmit FIFO: The host could not supply enough transport-stream data to sustain a continuous output stream.

6.2.2.4. TxSyncInt – Synchronisation-Error Int.

When set to '1', this bit indicates that a synchronisation error has occurred in the packet-processing logic in the transmit channel. The distance between two MPEG-2 sync bytes (0x47) in the Transmit FIFO appeared to be not 188 (Packet Mode **188** or **Add16**) or 204 (Packet Mode **204**.)

The DTA-140 only checks for synchronisation errors if Transmit Control is **TxSend**. When a synchronisation error occurs, the DTA-140 will try to resynchronise to the data from the Transmit FIFO. In this process, a number of bytes will be skipped (not transmitted), until resynchronisation has been established again.

[15] No write action to a PCI-9056 register is required.

In Packet Mode **Raw**, no notion of packet size exists and therefore synchronisation-error checking is disabled.

6.2.3. *Transmit Clock (0Ch)*

The Transmit-Clock register defines the frequency (rate) of the transmit clock. The transmit-clock rate (Mbit/s) is 8 times the number of DVB/ASI symbols transmitted per second, excluding K28.5 stuffing characters[16].

Table 19. Transmit-Clock Register – Format

Bit	Mnem	Definition
31…0	`TxClock`	32 bits of the phase-increment value used to generate the transmit clock.

The relation between the Transmit-Clock register and the resulting transmit-clock rate is defined as follows:

$$R_{Tx}(\text{Mbit/s}) = \frac{216 * \texttt{TxClock}}{2^{32}}$$

To compute the value of the Transmit-Clock register for a given desired transmit-clock rate, the reverse formula shall be used:

$$\texttt{TxClock} = \frac{2^{32} * R_{Tx}(\text{Mbit/s})}{216}$$

The maximum transmit-clock rate that can be handled properly by the DTA-140 is 150 Mbit/s. This is equivalent to a `TxClock` value of 2,98,2616,178 (B1C71C71h). The behaviour of the board for transmit-clock rates that exceed 150 Mbit/s is unspecified.

Note

- The DTA-140 allows programming of bit rates higher than 150 Mbit/s: No interrupt or other error-trap mechanism is invoked.

The Transmit-Clock bit rate is not necessarily equal to the Transport-Stream bit rate (as defined in the MPEG-2 Systems specification). For 188-byte packets the two rates are equivalent, but for 204-byte packets, the Transport-Stream bit rate is 188/204 * Transmit-Clock bit rate. Table 20 below summarises the relationship between the different kinds of rates.

Table 20. Relation between Transmit-Clock rate (Tx-Rate) and Transport-Stream rate (TS-Rate).

Mode	TxRate	TSRate
188	$TxRate = TSRate$	$TSRate = TxRate$
204 Add16	$TxRate = \frac{204}{188} * TSRate$	$TSRate = \frac{188}{204} * TxRate$

6.2.4. *Transmit FIFO Size (14h)*

The Transmit FIFO-Size register defines the maximum load of the DTA-140's Transmit FIFO when data is transferred using DMA. When the Tx-FIFO load reaches the value in Tx-FIFO Size, the DTA-140 inhibits DMA requests.

The maximum load is expressed in number of bytes. The number of significant bits in the Tx-FIFO-Size register depends on the type of the SDRAM on-board of the DTA-140. Table 21 below is valid for an SDRAM size of 8 Mbytes.

Table 21. Tx-FIFO-Size Register – Format[17]

Bit	Mnem	Definition
23…0	`TxFifoSize`	Maximum Tx-FIFO load. May not be 0.
Bits 31…24 of this register are tied to '0'.		

The Tx-FIFO-Size register can be programmed to any value between 0 and 2^{24}-1 bytes[17], but not all values make sense:

- The absolute minimum meaningful value for Tx-FIFO Size is the number of bytes in a transport packet (188 or 204). Setting Tx-FIFO Size to a lower value may block demand-mode DMA entirely.
  The minimum *practical* value for Tx-FIFO Size is about 500 bytes (refer to §8.2).

[16] The *line*-clock rate for DVB/ASI is 27 MHz by definition and cannot be programmed. K28.5 stuffing characters are used to construct a variable transmit-clock rate, independent from the line-clock.

[17] Assuming an SDRAM-size of 8 MB. If the SDRAM is larger, more significant bits are included.

- The maximum meaningful value for Tx-FIFO Size is 2^{23} = 8.388.592 bytes[17]. Values between $2^{23}+1$ and 2^{24} are invalid: These values can be programmed in the Tx-FIFO-Size register without warning, but they would render Tx-FIFO-overflow protection inoperative.

The register name Tx-FIFO "Size" is somewhat misleading: the absolute size of the Transmit FIFO is always 8 Mbytes[17, 18]. In fact, the Tx-FIFO-Size register specifies a *virtual* FIFO size that is used for flow control of DMA transfers: When the FIFO load becomes greater or equal than Tx-FIFO Size, the Tx-FIFO-Filled flag is asserted and DMA transfers are temporarily disabled. DMA is re-enabled when the Tx-FIFO load drops below Tx-FIFO Size. In other words, DMA is *demanded* whenever the Tx-FIFO load is less than Tx-FIFO Size.

When the DTA-140 is operated under DMA, the Tx-FIFO load can become slightly higher than Tx-FIFO Size. This is due to the fact that, when demand for DMA stops, a few words may be in the pipeline and still need to be stored in the Transmit FIFO. Even so it is safe to program the Tx-FIFO-Size register to precisely 2^{23} (= 8 Mbytes[17]), because the Tx-FIFO can accommodate for a few hundred additional bytes on top of the SDRAM size.

Note

- When data is written into the Tx-FIFO *directly* through the Tx-FIFO-Data register (without using DMA), the Tx-FIFO-Size register has no effect.

6.2.5. Transmit FIFO Load (18h)

The Transmit-FIFO-Load register contains the current load of the DTA-140's Transmit FIFO, expressed in number of bytes. Table 22 below is valid for an SDRAM size of 8 Mbytes.

Table 22. Tx-FIFO-Load Register – Format[17]

Bit	Mnem	Definition
23…0	`TxFifoLoad`	Current Tx-FIFO load.
Bits 31…24 of this register are tied to '0'.		

While the DTA-140 is streaming data, the value read from this register is volatile. The value may change with every transmitted byte and with every DMA transfer.

Note

- The actual number of bytes buffered on the PCI card may be slightly higher than FIFO Load due to words residing in the PCI-9056 FIFO (not accounted for in Tx-FIFO Load) and/or in pipeline registers.

The absolute maximum value of Tx-FIFO Load on a DTA-140 is the SDRAM size plus 256 bytes. When Tx-FIFO Load has reached this value, the hardware blocks write operations to the Transmit FIFO. Write attempts to the Tx-FIFO-Data register will seem to succeed, but the data disappears and Tx-FIFO Load is not incremented.

The use of the Tx-FIFO-Load register in flow-control algorithms is optional. It can be used for enhancing robustness by checking at specific moments in time whether the Tx-FIFO Load is contained within a certain expected range.

6.2.6. Transmit Diagnostics (1Ch)

The Transmit-Diagnostics register contains a number of special fields that can be used for validation and testing of the DTA-140. In normal operation, this register should not be touched. It is recommended to clear the Tx-Diagnostics register to all zeros in the device-driver's initialisation routine[19].

Note

- The Tx-Diagnostic register in the DTA-140 contains fewer fields then in the DTA-100 and DTA-102: fields that existed for debugging purposes only have been removed.

[18] Plus a few hundred bytes (refer to §8.2).

[19] Issuing a soft reset through the General-Control register will also clear the Diagnostics register.

Table 23. Tx-Diagnostics Register – Format

Bit	Mnem	Definition
7…0	`TxSfData`	Data at output of Smoothing FIFO
8	`TxLoopBack`	Transmit Loop-Back mode
9	*reserved*	
10	`TxBfFull`	Transmit Burst-FIFO Full flag
Bits 31…11 of this register are tied to '0'.		

6.2.6.1. TxSfData – Tx-Smoothing-FIFO Data

Transmit-Smoothing-FIFO Data is a read-only 8-bit field that represents the output data of the Smoothing-FIFO (§8.2.2.5). In normal operation, the value may change with every transmitted byte.

The Smoothing-FIFO-Data register is not suited for a data-path or memory test, because reading this register will only take a snapshot of the value at the Tx-FIFO's output. The data byte is not popped from the Smoothing FIFO.

Note

- The Transmit-Loop-Back-Data register can be used for simultaneous reading and popping of data bytes from the Smoothing FIFO.

6.2.6.2. TxLoopBack – Tx-Loop-Back Mode

Writing a '1' to this bit disconnects the DVB/ASI transmit circuitry from the Smoothing FIFO. This enables diagnostic software to read back bytes at the end of the FIFO pipeline through the Loop-Back Data register.

In normal operation, Loop-Back Mode should be set to '0'. Loop-Back Mode can be set to '1' for a manufacturing-test set-up in which data-path and memory integrity must be tested.

The Loop-Back Mode field is not cleared to '0' by a software reset.

6.2.6.3. TxBfFull – Tx-Burst-FIFO Full

Diagnostic read-only bit that is set if the Burst FIFO (§8.2.2.2) is full. The DTA-140 will only assert `TxBfFull` in exceptional circumstances.

This status flag has no purpose in normal operation.

6.2.7. Transmit Loop-Back Data (20h)

The Transmit-Loop-Back-Data register is an extension to the Tx-Diagnostics register. `TxSfDataNxt`[7..0] in the Tx-Loop-Back-Data register holds the same data as `TxSfData`[7…0] in the Tx-Diagnostics register. However, reading `TxSfDataNxt` will also cause a data byte to be popped from the Smoothing FIFO.

Table 24. Loop-Back Data Register – Format

Bit	Mnem	Definition
7…0	`TxSfDataNxt`	Read data at output of Smoothing FIFO *and issue a read pulse*
Bits 31…8 of this register are tied to '0'.		

The purpose of this register is to enable diagnostics data-path-integrity and memory-test software.

Note

- Tx-Loop-Back Mode must be '1' for meaningful use of Tx-Loop-Back Data.

6.2.8. Transmit-FIFO Data (40h…7Ch)

The Transmit-FIFO-Data register is the input register of the Transmit FIFO. This is the main register for streaming data with the DTA-140.

Data written to the Tx-FIFO-Data register directly enters the Transmit FIFO, four bytes at a time. Byte 0 (bit 7…0) will be the first transmitted byte, byte 3 (bit 3…24) the last one.

Table 25. Tx-FIFO-Data Register – Format

Bit	Mnem	Definition
7…0	`TxFifoData`	First transmitted byte
15…8	`TxFifoData`	Second byte
23…16	`TxFifoData`	Third byte
31…24	`TxFifoData`	Last transmitted byte

The suggested way to operate the DTA-140 is to program (in the PCI-9056) DMA transfers with the Tx-FIFO-Data register as constant destination address[20].

Instead of using DMA, an application program may also choose to write directly to the Tx-FIFO-Data register. In certain circumstances this may be a simple, convenient way to write data, e.g. for diagnostic purposes. However, performance will suffer, mainly because the host processor is required to write each individual word. The recommended way to operate the DTA-140 is using DMA transfers.

Notes

- The Tx-FIFO-Data register appears at 16 consecutive word addresses in the Operational-Registers Memory Map. This enables write bursts on the PCI bus, so that applications that choose not to use DMA can still achieve reasonable performance.
- When the DTA-140 is operated under DMA, a mechanism is in effect to protect the Transmit FIFO against overflow (§6.2.4, §8).
  When data is written directly to the Transmit FIFO, no such mechanism exists. Direct write operations always succeed, but when the Transmit FIFO contains SDRAM size + 256 bytes, new data just vanishes without warning.

6.3. Receive Registers

6.3.1. Receive Control (04h)

The Receive-Control register contains a number of fields that allow the device driver to control receive-specific functions of the DTA-140.

[20] Local Address in the PCI-9056's gather/scatter DMA descriptor should be set to 00000040h and Local Addressing Mode to '1' (hold Local Address bus constant).

Table 26. Receive-Control Register – Format

Bit	Mnem	Description
1…0	`RxMode`	Receive Mode
3…2	`RxAsiInv`	Invert ASI-input control
4	*`reserved`*	
5	`RxCtrl`	Receive Control
6	*`reserved`*	
7	`RxTimeStamp`	Insert Time Stamps
8	*`reserved`*	
9	`RxOvfIntEn`	Receive-FIFO-Overflow-Interrupt Enable
10	`RxSyncIntEn`	Synchronisation-error-Interrupt Enable
18…11	*`reserved`*	
19	`RxLedControl`	Take over LED Control
20	`RxLedGreen`	State of Green LED
21	`RxLedRed`	State of Red LED
22	`RxClrFifo`	Clear Receive FIFO
23	`RxRst`	Reset Receive Channel
Bits 31…24 of this register are tied to '0'.		

6.3.1.1. RxMode – Receive Mode

Receive Mode is a 2-bit field that controls the processing applied to incoming packets.

Table 27. Receive Mode – Values

Value	Mode	Definition
00	**St188**	Store 188-byte packets.
01	**St204**	Store 204-byte packets.
10	**StMp2**	Store 188- or 204-byte packets.
11	**StRaw**	No notion of packets. All incoming data is stored in the Receive FIFO.

The default Receive Mode is **St188**. In this mode the DTA-140 accepts 188- and 204-byte packets, but always stores 188 bytes per packet. If the input contains 204-byte packets, the 16 trailing bytes are dropped, irrespective of their content. Incoming data without MPEG-2 packet structure is dropped entirely.

Receive Mode **St204** is similar to **St188**, but now 204 bytes are stored per packet. If the input contains 188-byte packets, 16 zero bytes are appended.

In Receive Mode **StMp2**, MPEG-2 packets are stored as they enter the system. However, if the DTA-140 cannot synchronise to a packet structure in the incoming data, input data is dropped.

Finally, in Receive Mode **StRaw** all data bytes are stored in the Receive FIFO, without MPEG-2 synchronisation.

Notes

- Receive Mode can be switched to another mode *dynamically*, this is while data is being stored in the Receive FIFO. A few packets may get lost in the resynchronisation process.
- Dynamically switching Receive Mode in modes **St188**, **St204** and **StMp2** may break 32-bit alignment and packet alignment. Refer to §4.4.2 for additional information.

6.3.1.2. RxAsiInv – Invert DVB/ASI-Input Control

Invert-DVB/ASI-Input Control is a 2-bit field that controls whether the DTA-140 attempts to automatically detect the polarity of the DVB/ASI input signal, or forces non-inverted or inverted usage of the input signal.

Table 28. Invert DVB/ASI-Input Control – Values

Value	Mnem	Definition
00	**Auto**	Auto-detect polarity of DVB/ASI input signal.
10	**Normal**	Do not invert DVB/ASI input
11	**Invert**	Invert DVB/ASI input signal

The DVB/ASI signal is sensitive to signal polarity. Without corrective measures, an inverted DVB/ASI signal (which may be caused by e.g. an inverting distribution amplifier) will be decoded incorrectly by a standard DVB/ASI receiver.

The DTA-140 contains circuitry to automatically detect whether a signal is inverted. Hereto, the card first tries to lock on 188/204-byte packets with a non-inverted signal. If synchronisation cannot be achieved within about 10 packets, the DTA-140 tries again with the input signal inverted. This cycle continues until synchronisation is achieved.

In certain circumstances, e.g. if it is known that the incoming signal does not contain 188-byte or 204-byte MPEG-2 packets, automatic detection of signal polarity is undesirable. In such cases, Manual-Invert Control should be set to **Normal** or **Invert**, so that the DTA-140 will not arbitrarily switch between inverted and non-inverted modes.

6.3.1.3. RxCtrl – Receive Control

The Receive-Control field controls storage of data into the Receive FIFO.

Table 29. Receive Control – Values

Value	Mnem	Definition
0	**Idle**	No new data is stored in Receive FIFO.
1	**Rcv**	Store incoming data into Receive FIFO.

After a power-up condition, Receive Control is initialised to **Idle**. Whenever Receive Control is set to **Idle**, the input circuitry is “disconnected” from the Receive FIFO and no new data can be stored in the Receive FIFO.

When Receive Control is set to **Rcv**, actual storage of transport packets in the Receive FIFO begins.

6.3.1.4. RxTimeStamp – Insert Time Stamps

The Time-Stamp field controls whether incoming packet are tagged with a 32-bit time stamp.

Table 30. Insert Time Stamps – Values

Value	Mnem	Definition
0	**NoTs**	Do not insert time stamps.
1	**Stamp**	Insert time stamps.

Time stamps are derived from the 40.5-MHz reference clock counter. The value of this counter is stamped at the moment that the

MPEG-2 sync byte enters the DTA-140 (with a delay of less than 1 μs).

Please refer to §4.5.2 for a description of the format of time stamp.

Insertion of time stamps is not available in Receive Mode **StRaw**.

6.3.1.5. RxOvfIntEn – Receive-FIFO Overflow Interrupt Enable

Writing a '1' to this bit enables the Receive-FIFO-*Overflow Interrupt* (§6.2.2.3).

6.3.1.6. RxSyncIntEn – Synchronisation-Error Interrupt Enable

Writing a '1' to this bit enables the *Synchronisation-Error Interrupt* (§6.3.2.4).

6.3.1.7. RxLedControl – Take over LED Control

When this field is '0', the state of the bi-colour LED indicator on the PCI bracket is determined by the hardware, as described in §2.2.

When this field is '1', the hardware is disconnected from the LED indicator. Instead, the LED is controlled directly by fields `LedGreen` and `LedRed`.

This bit is reset to '0' upon a hardware- or software reset.

6.3.1.8. RxLedGreen – State of Green LED

If `LedControl` is '1', this field controls the **green** colour of the bi-colour LED next to the connector on the PCI bracket.

6.3.1.9. RxLedRed – State of Red LED

If `LedControl` is '1', this field controls the **red** colour of the bi-colour LED next to the connector on the PCI bracket.

6.3.1.10. RxClrFifo – Clear Receive FIFO

Writing a '1' to this field[21] clears the contents of the Receive FIFO and resets a number of related control fields:
- The FIFO-Load register is cleared to zero.
- The Receive-FIFO Overflow Interrupt flag is cleared.
- `RxCtrl` in the Receive-Control register is reset to **Idle** (to avoid that new data is immediate written in the Receive FIFO again).

6.3.1.11. RxRst – Reset Receive Channel

Writing a '1' to this field[21] resets the DVB/ASI Receive Channel.

The following fields and logic circuitry are affected by a Reset Transmit Channel action:
- The actions brought about by `RxClrFifo`: clearing the Receive FIFO and the Receive - FIFO Overflow flag, setting Receive Control to **Idle**.
- The Synchronisation-Error Interrupt flag is cleared.
- The state machines used in the Receive-Channel hardware are reset.

Other fields in the operational registers are <u>not</u> affected, notably:
- Receive Mode (`RxMode`).
- Invert DVB/ASI-Input Control (`RxAsiInv`).
- Insert Time Stamps (`RxTimeStamp`).
- Receive Loop-Back Mode in the Diagnostics register.
- Interrupt-Enable bits in the Receive-Control register: interrupts that were enabled remain enabled.

6.3.2. *Receive Status (08h)*

The Receive-Status register contains a number of fields that allow the device driver to read status information from the DTA-140.

[21] The Rx`ClrFifo` and RxRst bits are write-only. The write operation triggers the clear action: it is not required to reset the bit to '0' again.

Table 31. Receive-Status Register – Format

Bit	Mnem	Description
1…0	`PckSize`	Size of incoming packets
3…2	*reserved*	
7…4	`RxSdramSize`	SDRAM Size
8	*reserved*	
Interrupt status flags		
9	`OvfInt`	Receive-FIFO Overflow
10	`RxSyncInt`	Synchronisation error
13…11	*reserved*	
Other status flags		
14	`RxAsiCD`	DVB/ASI Carrier Detect
15	`RxAsiLock`	Locked to DVB/ASI signal
16	`RxRateOk`	Input Rate "Ok"
17	`RxAsiInv`	Invert ASI signal – status
Bits 31…18 of this register are tied to '0'.		

The interrupt-status flags (bits 9 and 10) in this register share common behaviour:

- An interrupt-status flag is set when the corresponding condition occurs. The flag remains set until it is explicitly cleared.
- Writing a '1' to the flag clears the interrupt-status flag, and also clears the PCI interrupt[22] (unless another interrupt condition is pending).
- The interrupt-status flag only leads to an interrupt if the corresponding interrupt-enable bit in the Receive-Control register is set, and interrupts in the PCI 9054 have been enabled.
- The operation of the interrupt-status bits is independent from the state of the interrupt-enable bit: If the interrupt-enable bit is '0', the interrupt-status flag still latches the corresponding condition.

6.3.2.1. RxPckSize – Packet Size

Packet Size is a 2-bit field that indicates the length of the packets currently being received on the Transport-Stream input.

[22] No write action to a PCI-9054 register is required.

Table 32. Packet Size – Values

Value	Mode	Definition
`00`	**RxInv**	Invalid packet size.
01	-	*reserved*
10	**Rx188**	Receiving 188-byte packets.
11	**Rx204**	Receiving 204-byte packets.

Packet-Size values **Rx188** and **Rx204** indicate that the DTA-140 receives correctly formatted packets of 188 or 204 bytes respectively.

Packet-Size value **RxInv** indicates that the DTA-140 is currently out of synchronisation with the input Transport Stream.

6.3.2.2. RxSdramSize – SDRAM Size

SDRAM Size is a static read-only field that indicates the size of the SDRAM on-board of the DTA-140. The SDRAM size determines the maximum size of the Receive FIFO.

Table 33. SDRAM Size – Values

Value	Size	Comment
`0000`	8 MB	Minimum supported size.
`0001`	16 MB	May be supported in future revisions of the DTA-140.
`0010`	32 MB	May be supported in future revisions of the DTA-140.
other	*reserved*	

6.3.2.3. RxOvfInt – Receive-FIFO Overflow Interrupt

When set to '1', this bit indicates that an overflow condition has occurred for the Receive FIFO: The data in the Receive FIFO could not be transferred fast enough to a system buffer in host memory (or to another PCI agent).

6.3.2.4. RxSyncInt – Synchronisation-Error Interrupt

When set to '1', this bit indicates that a synchronisation error has been detected in the packet-synchronising logic on the DTA-140.

6.3.2.5. RxAsiCD – DVB/ASI Carrier Detect

When set to '1', this bit indicates that a carrier signal has been detected on the DVB/ASI Transport-Stream input, and that the PLL (Phase-Locked Loop) monitoring the incoming signal is in lock.

If field `AsiCD` has value '1', this does not necessarily mean that the input signal is DVB/ASI compliant. In particular, connecting an SMPTE SDI source to the DTA-140 will also set `AsiCD` to '1'.

6.3.2.6. RxAsiLock – Locked to DVB/ASI Input

This 1-bit status field indicates whether the PLL (Phase-Locked Loop) monitoring the incoming DVB/ASI signal is in lock.

Table 34. `AsiLock` – Values

Value	Mode	Definition
0	**NoLock**	PLL cannot lock to DVB/ASI input
1	**InLock**	PLL is locked to DVB/ASI input

6.3.2.7. RxRateOk – Input Rate Ok

When set to '1', this bit indicates that the input rate of the incoming DVB/ASI signal is sufficiently high for further processing on the DTA-140. This bit becomes '0' if the input rate falls below approximately 900 bits per second.

6.3.2.8. RxAsiInv – Invert DVB/ASI Signal – Status

This 1-bit status field indicates whether the DVB/ASI input signal is currently being inverted.

Table 35. `AsiInv` – Values

Value	Mode	Definition
0	**Normal**	DVB/ASI signal is <u>not</u> inverted
1	**Invert**	DVB/ASI signal is being inverted

When the Invert-DVB/ASI-Input Control field in the Receive-Control register is set to **Auto**, then the `AsiInv` status bit shows the output of the automatic DVB/ASI polarity detector on the DTA-140.

When the Invert-DVB/ASI-Input Control field in the Receive-Control register is set to **Normal** or **Invert**, then the `AsiInv` status bit is a direct copy of the value of the control field.

6.3.3. Receive FIFO Load (18h)

The FIFO-Load register contains the current load of the DTA-140's Receive FIFO, expressed in number of bytes. Table 22 below is valid for an SDRAM size of 8 Mbytes.

Table 36. FIFO-Load Register – Format[23]

Bit	Mnem	Definition
23…0	`FifoLoad`	Current FIFO load.
Bits 31…24 of this register are tied to '0'.		

While the DTA-140 is streaming data, the value read from this register is volatile. The value may change with every transmitted byte and with every DMA transfer.

Note

- The actual number of bytes buffered on the PCI card may be slightly higher than FIFO Load due to words residing in pipeline registers.

The maximum value of the FIFO-Load register is (approximately) the size of the Receive FIFO, which is the SDRAM size plus 960 bytes.

The use of the FIFO-Load register in flow-control algorithms is optional. It can be used for enhancing robustness by checking at specific moments in time whether the FIFO Load is contained within a certain expected range.

6.3.4. Receive Diagnostics (1Ch)

The Receive Diagnostics register contains a number of special fields that can be used for validation and testing of the DTA-140. In normal operation, this register should not be touched. It is recommended to clear the Diag-

[23] Assuming an SDRAM-size of 8 MB. If the SDRAM is larger, more significant bits are included.

nostics register to all zeros in the device-driver's initialisation routine[24].

Table 37. Diagnostics Register – Format

Bit	Mnem	Definition
7…0	`reserved`	
8	`LoopBack`	Loop-back mode
9	`reserved`	
10	`RfRateOvf`	Receive FIFO Rate Overflow
11	`SdFull`	SDRAM Full
31…12	`reserved`	

6.3.4.1. LoopBack – Loop-Back Mode

Writing a '1' to this bit disconnects the DVB/ASI input circuitry from the Receive FIFO. This enables software to write a test pattern to the Receive FIFO through the Loop-Back-Data register (`LbData`).

In normal operation, this field should be set to '0'. Loop-Back Mode can be used in the manufacturing test to check data-path and memory integrity.

The Loop-Back-Mode field is not cleared by a software reset.

6.3.4.2. RfRateOvf – Receive-FIFO Rate Overflow

The `RfRateOvf` flag indicates whether the rate at which data bytes enter the system exceeds the maximum write rate for the SDRAM.

In principle this overflow condition cannot occur on the DTA-140, because the maximum write rate to the SDRAM is sufficiently high for a DVB/ASI stream of any rate.

6.3.4.3. SdFull – SDRAM Full

The `SdFull` flag indicates whether the SDRAM on-board the DTA-140 is full. If this flag is set, further writing to the Receive FIFO is inhibited and DVB/ASI input data will be dropped.

[24] Issuing a soft reset through the General-Control register will also clear the Diagnostics register.

6.3.5. Receive Loop-Back Data (20h)

The Loop-Back Data register can be used to write 8-bit test data to the Receive FIFO in loop-back mode.

Table 38. Loop-Back Data Register – Format

Bit	Mnem	Definition
7…0	`SfDataNxt`	Write data to Receive FIFO
Bits 31…8 of this register are tied to '0'.		

The purpose of this register is to enable diagnostics data-path-integrity and memory-test software.

Note

- Loop-Back Mode (in the Diagnostics Register) must be '1' for meaningful use of Loop-Back Data.
  If Loop-Back Mode is '0', writes to the Loop-Back-Data register have no effect.

6.3.6. Receive Valid Count (30h)

The Valid-Count register contains a sample of a free-running counter that is incremented for every "valid" (non-stuffing) byte received at the DVB/ASI input.

Table 39. Valid Count – Format

Bit	Mnem	Definition
31…0	`ValidCnt`	Sample of valid-byte counter.

The Valid-Count register can be used to estimate the transport rate of the incoming DVB/ASI Transport Stream.

The Valid Counter is sampled halfway between periodic interrupts (§6.1.2.1): the time interval between two samples is 2^{21} clock cycles of the on-board 40-MHz reference clock (52.4 ms). The register is designed to be read just after each periodic interrupt, e.g. in an interrupt-service routine.

Note

- The absolute value of the Valid-Count register has no significance. Just the difference between two successive samples is relevant.

6.3.7. Receive Violation Count (34h)

The Violation-Count register contains a sample of a free-running counter that is incremented for every code-violation detected in the DVB/ASI input signal.

Table 40. Violation Count – Format

Bit	Mnem	Definition
31…0	`ViolCnt`	Sample of code-violation counter.

A code violation is a bit error that leads to an illegal 8B/10B code (the line code used by DVB/ASI). Bit errors may be caused by poor cable quality, or if the input cable is too long.

Note

- Connecting or disconnecting the cable causes a massive amount of code violations. This is "normal" behaviour, caused by the locking process of the DVB/ASI input chip on the DTA-140.

6.3.8. Receive-FIFO Data (40h…7Ch)

The FIFO-Data register is connected to the output side of the DTA-140's Receive FIFO. This is the main register for receiving MPEG-2 data with the DTA-140.

7. Vital Product Data

Vital Product Data (VPD) is information stored in a PCI device to uniquely identify the hardware and, potentially, software elements of the device. *PCI Local Bus Specification Rev2.2* defines both a standard storage structure and access method for VPD.

The DTA-140 uses VPD to store the serial number, revision level, etc. The sections below list all supported fields. The VPD is stored in the serial EEPROM on-board of the DTA-140. The VPD can be accessed through the VPD-function support built in the PCI-9056.

The DEKTEC DTA-series of PCI cards share the same layout of the serial EEPROM, so that the VPD data can be accessed in a uniform way for each board.

7.1. Serial EEPROM Lay-Out

Figure 18 below shows the memory map of the serial EEPROM and the positioning of VPD elements within the EEPROM memory. Note that addresses are *byte* addresses, whereas the PCI-9056 specification sometimes uses word (16-bit) or long word (32-bit) addresses.

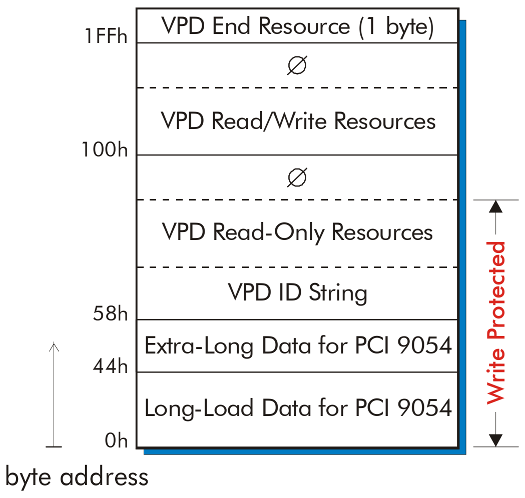


Figure 18. Memory Map of the Serial EEPROM.

The 88 bytes at address 0h…57h are loaded into the PCI-9056 upon power-on reset, to define the initial value of the register set. This part of the EEPROM data is not related to VPD.

The other 424 bytes in the serial EEPROM, starting at address 58h, are dedicated to VPD.

First, the VPD Identification String and the VPD Read-Only Resources are stored sequentially, followed by zero-byte stuffing. The VPD Read/Write Resources are located at address 140h, also followed by zero-byte stuffing. The EEPROM is “closed” by a VPD End resource at address 1FFh.

The VPD Read-Only and Read/Write Resources each consist of a two-character key (e.g. “PN” for Part Number) and a length-prefixed string defining the actual resource.

The first part of the serial EEPROM (register-data for PCI-9056, VPD ID String and VPD Read-Only Resources) is programmed in the factory and write-protected to avoid accidental modification. The VPD Read/Write Resources are not write-protected and may be modified by the end user, e.g. for storing the customer’s system-asset identifier. DEKTEC utilises VPD Read/Write Resources to store software licenses.

7.2. VPD ID String

The VPD ID-String Resource contains the name of the board in ASCII characters. The content of this resource is fixed for all incarnations of the DTA-140.

Table 41. VPD ID String – Syntax

Addr[25]	Value	Item
58h	82h	ID-String tag.
59h	24h	Length LSB.
5Ah	00h	Length MSB.
5Bh	“DTA-140 DVB/ASI Input+Output”	ID-String data[26].

7.3. VPD Read-Only Resources

The VPD Read-Only Resources describe permanent hardware characteristics of the PCI card. This VPD section is stored in the EEPROM just behind the VPD ID-String section.

[25] Byte address in serial EEPROM.
[26] No trailing zero (‘\0’) character!

The following VPD read-only resources are supported on the DTA-140:

- PN = Part Number
  The PN Resource is fixed to "DTA-140".
- EC = Engineering Change Level
  The EC Resource identifies the hardware revision of the board, e.g. "Rev 0".
- MN = Manufacture ID
  The MN Resource is a 2-digit code[27] identifying the manufacturer of the board.
- CL = Customer ID
  The CL Resource is a 6-digit code[27] identifying the (initial) customer of the board.
- SN = Serial Number
  The SN Resource holds a unique serial number. For the DTA-140, this number begins with "4140", followed by a sequence number of 6 or more digits.
- GC = Guard Code
  The GC Resource is a coded string used in DEKTEC license management.
- PD = Production Date
  The PD Resource keeps the production date of this DTA-140 instance, e.g. "2002.12".
- XT = Crystal Accuracy
  The XT Resource lists the accuracy of the 27-MHz crystal oscillator as a string, e.g. "10ppm".

Table 42 below shows an example of the contents of the VPD Read-Only Resources section. DEKTEC reserves the right to append private descriptors in the reserved part of the read-only area.

Table 42. VPD Read-Only Resources – Syntax

Addr	Value	Item
07Fh	90h	VPD-R tag.
080h	7Eh	Length LSB.
081h	00h	Length MSB.
082h	"PN"	VPD keyword.
083h	7	Field length.
084h	"DTA-140"	Part number.
08Ch	"EC"	VPD keyword.
08Eh	5	Field length.
08Fh	"Rev 0"	Engineering-Change level.
094h	"MN"	VPD keyword.
096h	2	Field length.
097h	"02"	Manufacture ID.
099h	"SN"	VPD keyword.
09Bh	10	Field length.
09Ch	"4140127564"	Serial number.
0A6h	"CL"	VPD keyword.
0A8h	6	Field length.
0A9h	"300000"	Customer ID
0AFh	"GC"	VPD keyword.
0B1h	10	Field length.
0B2h	"/WEJ#Q0GC("	Guard Code
0BCh	"PD"	VPD keyword.
0BEh	2	Field length.
0BFh	"2002.12"	Production Date.
0C6h	"XT"	VPD keyword.
0C8h	2	Field length.
0C9h	"10ppm"	Crystal Accuracy.
0CFh	"RV"	VPD keyword.
0D1h	2Eh	Field length.
0D2h	ABh	Checksum.
0D3h	45 x 00h	Reserved.
100h		Read-Write section

The length of the VPD Read-Only Resources section is tuned such that the VPD Read/Write Resources section starts at byte address 100h.

7.4. VPD Read-Write Resources

The VPD read/write section can hold 255 data bytes that can be updated dynamically from software. Potential usage includes diverse ap-

[27] DekTec internal code.

plications such as software keying, system-asset identification and storage of fault codes for inspection by service personnel.

Every byte in the serial EEPROM can be rewritten about 10^6 times. Therefore, the VPD Read/Write Resource cannot be used for data that is updated a lot, e.g. every second. It is recommended to use the Read/Write section only for data that has a near-static nature.

The following standard tags are defined in *PCI Local Bus Specification Rev2.2*.

– Vx = Vendor Specific
  This is a DEKTEC-specific item, e.g. a software license. The second character (x) of the keyword can be 0 through 9 and A through Z.

– Yx = System Specific
  This is a system-specific item. The second character (x) of the keyword can be 0 through 9 and B through Z.

– YA = Asset Tag Identifier
  The resource contains the system-asset identifier provided by the system owner.

– RW = Remaining Read/Write Area
  This descriptor is used to identify the unused portion of the read/write space.

The data bytes are stored in the serial EEPROM at address 100h up to 1FEh inclusive. The byte at address 1FFh is used to store the VPD-End tag. Table 43 below shows an example of the syntax of the VPD-Read/Write-Resources section.

Table 43. VPD Read/Write Resources – Syntax

Addr	Value	Item
100h	91h	VPD-W tag.
101h	FCh	Length LSB.
102h	00h	Length MSB.
103h	"V3"	VPD keyword.
105h	16	Field length.
106h	"9^bK$q0p@i!(kzm&"	License string.

Table 43. VPD Read/Write Resources – Syntax

Addr	Value	Item
116h	"YA"	VPD keyword.
118h	25	Field length.
119h	"DVB/ASI Test Generator 23"	System-asset identifier.
132h	"RW"	VPD keyword.
134h	202	Field length.
135h	202 x 00h	Reserved.
1FFh	78h	VPD End tag.

7.5. Reading VPD Data

VPD Resources can be read 4 bytes at a time with the procedure described below. The hardware does not support any form of parsing VPD data, this is the job of the device driver.

1. Ensure that the read address is 32-bit aligned: the two least-significant address bits shall be zero.
2. Write the address to the 16-bit PCI-9056 register **PVPDAD** at PCI offset 4Eh in PCI-*Configuration* Space.
   Set the **VPD-Address** field to the read-address field (last two bits 0) and, in the same operation, set the **F**-flag field to '0', signalling a read operation.
3. Poll the **F**-flag in a loop until it becomes '1'. This indicates that the VPD read data is actually available.
4. Read the 32-bit PCI-9056 register **VPDDATA** at PCI offset 50h to obtain the requested 4 VPD data bytes.
5. Repeat steps 1..4 for all VPD words to be read.

7.6. Writing VPD Data

The VPD Read/Write Resources section can be written 4 bytes at a time with the procedure described below.

1. Ensure that the write address is 32-bit aligned: the two least-significant bits shall be zero.

2. Enable programming of the serial EEPROM by writing a '1' to PE in the General Control register (§6.1.1.1).
3. Change the *Serial EEPROM Write-Protected Address Boundary* register in the PCI 9056 (register PROT_AREA at PCI-offset 0Eh[28]) to a value less or equal than the write address divided by four.
   The division by four is required because PROT_AREA contains a 7-bit field that points to a 32-bit "long-word" address.
4. Write the desired data (32-bits!) to PCI-9056 register VPDDATA.
5. Write the destination address to the 16-bit PCI-9056 register PVPDAD at PCI offset 4Eh in PCI-*Configuration* Space.
   Set the VPD-Address field to the write address (last two bits 0) and, in the same operation, set the F-flag to '1', which signals a write operation.
6. Poll the F-flag until it changes to '0' to ensure that the write operation has completed.
7. Repeat steps 1..4 for all VPD words to be written.
8. For safety, change PROT_AREA back to 7Fh, and:
9. Disable programming of the serial EEPROM by writing a '0' to PE in the General Control register.

Note

- It is the responsibility of the device driver to maintain integrity of the VPD Resources.
  For example, if a VPD Resource to be rewritten does not start at a 32-bit boundary, then the host should first read the original 32-bit VPD word, AND/OR-in the new data, and write the resulting 32-bit word back.

[28] In PCI-memory space.

8. Hardware Details

This section covers a number of low-level hardware details, in order to provide a deeper insight into the operation of the DTA-140.

This is an advanced section that only needs to be studied by developers who seek to achieve the maximum from the DTA-140, or who want to write a device driver for a new platform.

You don't need to read this section for day-to-day usage of the DTA-140: the device driver and DTAPI library will hide most if not all of the details.

8.1. Aspects of DMA

The DTA-140 contains a DMA[29] Controller for transferring data from main memory to the FIFO on the DTA-140, and vice versa.

8.1.1. DMA vs. Direct PCI Access

Transport-Stream data can be transferred to or from the DTA-140 in one of two ways:

1. The host[30] reads/writes the data *directly* over the PCI Bus from/to the DTA-140.
2. An intermediate *DMA Buffer* in host memory is used. The DMA controller on-board of the DTA-140 transfers the data between DMA Buffer and DTA-140.

In the first method, the host processor executes a long series of read- or write- instructions over the PCI Bus. This has the effect of slowing down the processor speed to the PCI-Bus rate. Theoretically, PCI-access instructions could be interleaved with instructions performing other tasks, but in practice, this is awkward.

The second method (DMA) allows the processor to access to main memory, which is at least an order of a magnitude faster than directly accessing the PCI Bus. The DMA cycles to transfer data between main memory and DTA-140 – via the PCI Bus – are invisible to the host processor.

To sum up, the "direct-access" method is simple, but ties the processor to the PCI timing. The DMA method is more complex, yet much faster as a consequence of the vast difference in speed between accessing main memory and accessing the PCI Bus.

Because of its superior speed, DMA is the preferred method for transferring data. However, in one special case, described in § a complication occurs.

8.1.2. DMA Buffers

A DMA Buffer is an array of bytes allocated in the application's address space for the purpose of transferring data to or from the buffer under DMA.

The start- and end- address of a DMA Buffer must be aligned on a 4-byte boundary. The byte at relative address 0 is the first byte that leaves/has entered the DTA-140, followed by the byte at address 1, etc.

Obviously, the DMA Buffer may not be virtual memory that is swapped out to disk. Either non-paged memory should be used, or the driver should ensure that the pages are locked into physical memory whenever the DTA-140's DMA Controller may access them.

A DMA Buffer maps to a contiguous address range in *virtual*-address[31] space. The DMA Buffer needs not be contiguous in *physical*-address space: Memory pages may be *scattered* over physical memory[32]. The PCI-9056 Scatter/Gather DMA mode can be used to transfer such a scattered DMA Buffer in one go, without requiring processor intervention to glue pages together.

Scatter/Gather DMA uses a list of *Scatter/Gather DMA Descriptors* stored in (non-paged) host memory. This so-called *scat-*

[29] DMA (Direct Memory Access) is a technique for transferring data from or to the computer's main memory by hardware other than the main processor.

[30] "The host" is the application program or driver running on the host.

[31] It is not a strict requirement that the DMA Buffer is contiguous in virtual address space. Nonetheless, application programmers will find it very convenient.

[32] This means that the DMA Buffer may be allocated from a fragmented memory pool.

ter/gather list can be built at the same time as the DMA Buffer is allocated.

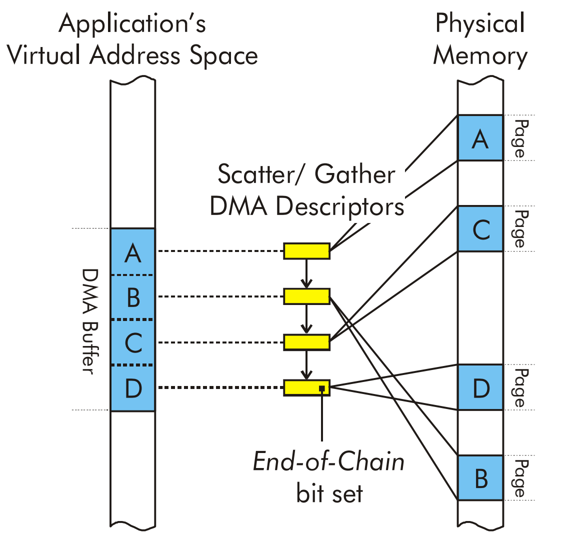

Application's Virtual Address Space
Physical Memory
Scatter/ Gather DMA Descriptors
DMA Buffer
A
B
C
D
Page
End-of-Chain bit set

Figure 19. The DMA Buffer appears contiguous to the application, while pages are scattered over physical memory. Scatter/gather DMA allows transfer of all pages in the DMA Buffer in one sweep without processor intervention.

The last descriptor in the list shall have its *End-of-Chain* bit set. The corresponding interrupt can be enabled, so that the driver is alerted when the entire DMA Buffer has been transferred.

The descriptor syntax and the way to initialise and operate scatter/gather DMA are described in the PCI-9056 data book. Note that the DTA-140 uses demand-mode DMA.

Note

- The scatter/gather mechanism incurs a little overhead per descriptor. Therefore, scatter/gather buffers should not be made too small, as this will lead to degraded performance. Buffers with the size of a memory page are fully acceptable.

8.2. Transmit FIFO

Transport-stream data may enter the DTA-140 in bursts. The Transmit FIFO temporarily buffers the data bytes, so that a smooth DVB/ASI stream can be generated. This section provides a number of details about the implementation of the Transmit FIFO.

8.2.1. Coarse-Grain Model

For most applications, the Transmit FIFO on the DTA-140 can be regarded as a conventional First-In First-Out buffer with a programmable maximum load, see Figure 20. This approximation of the Transmit FIFO is entirely adequate as long as the average FIFO Load remains well above a kilobyte.

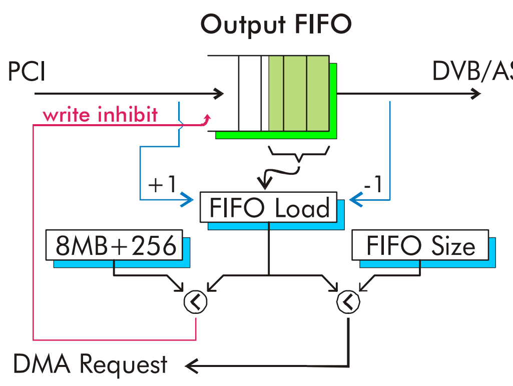


Figure 20. Coarse-grain model of the Transmit FIFO on the DTA-140.

The FIFO-Load register maintains the number of bytes loaded in the FIFO. Every time a data byte is written into the Transmit FIFO, the load is incremented. Whenever a byte is read for transmission in the DVB/ASI output stream, the load is decremented.

FIFO-overflow protection is implemented by continuously comparing the FIFO Load to the FIFO-Size register. As long as FIFO Load is less than FIFO Size, DMA transfers are requested. Otherwise, the Transmit FIFO is considered "full" and DMA is inhibited until FIFO Load drops below FIFO Size again.

The FIFO Load is also compared to the SDRAM Size + 256 bytes. If the FIFO Load is greater or equal than this load, new writes to the Transmit FIFO are inhibited, and FIFO Load is not incremented any more.

8.2.2. Fine-Grain Model

Some special applications may require a modest average FIFO load (say less than a kilobyte), e.g. when the transport rate is relatively low and/or the end-to-end delay has to be as low as possible. For these applications, knowledge of the intricacies of the Transmit FIFO may be useful for improving the performance of the system incorporating the DTA-140. The sections below provide a second-order model of the Transmit FIFO.

8.2.2.1. Block Diagram

Figure 21 shows a fine-grain block diagram of the Transmit FIFO.

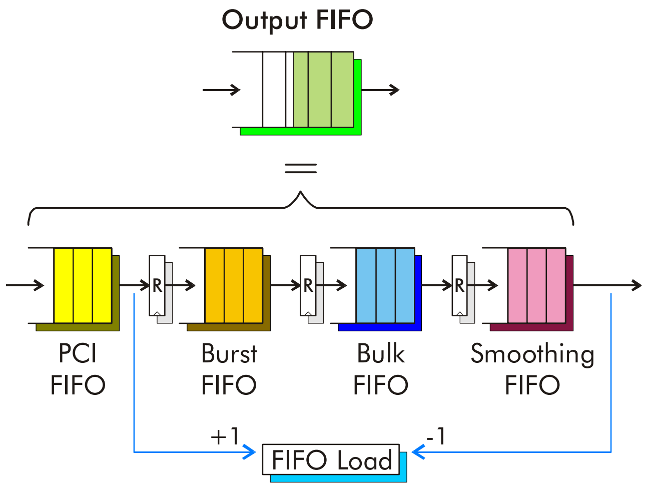


Figure 21. The Transmit FIFO can be modelled as a cascade of four FIFOs.

The Transmit FIFO can be considered as the concatenation of four FIFOs and a number of pipeline registers.

- First in line is the *PCI FIFO*, a small FIFO that buffers DMA bursts coming in from the PCI-Bus.
- The *Burst FIFO* is an extension of the PCI FIFO, to further enhance efficiency of DMA bursts on the PCI-Bus.
- The *Bulk FIFO* provides the major part of the buffer capacity, using an SDRAM chip.
- The *Smoothing FIFO* converts bursts from the SDRAM to a low-jitter DVB-ASI stream.

Why are four FIFOs required to implement a single Transmit FIFO? The core reason is the use of an SDRAM chip for the Bulk FIFO. The SDRAM does not support parallel reads and writes. Furthermore, the SDRAM can be operated efficiently only when read- and write operations are bundled in bursts. So, both at the input and the output-side of the SDRAM extra FIFOs are required to temporarily buffer data.

8.2.2.2. PCI FIFO

The *PCI FIFO* is a 32-word FIFO (128 bytes) embedded in the PCI-9056 bridge chip. This FIFO ensures that the DTA-140 can accept a 32-bit word at each PCI clock cycle, so that DMA transfers on the PCI Bus can occur in bursts.

Note

- For efficient utilisation of the PCI bus it is essential that DMA transfers occur in bursts. Every interruption of a burst incurs a penalty of many PCI cycles and a (potentially huge) degradation in net transfer speed.

The PCI FIFO cannot overflow: data words received in the PCI FIFO are always written to the Burst FIFO within a few clock cycles.

Note

- While passing data from PCI- to Burst FIFO, the DTA-140 does not check for overflow of the Burst FIFO.

8.2.2.3. Burst FIFO

The *Burst FIFO* extends the PCI FIFO with 256 words (1024 bytes), further increasing the length of DMA-transfer bursts. The Burst FIFO is implemented in the FPGA (Altera) on-board of the DTA-140.

The number of bytes loaded in the Burst FIFO controls whether DMA transfers are requested. DMA requests start when the Burst-FIFO load is less than 64 bytes[33]. The DTA-140 continues to request DMA until the Burst FIFO is half full (512 bytes). If this high-water mark is reached, the DTA-140 stops DMA until the load drops below 64 bytes again. At that moment DMA is re-enabled, unless the Transmit FIFO is full.

The DMA-Request signal can be observed through the DMA-Request field (`DmaReq`) in the Diagnostics register. The number of 32-bit words (not bytes) in the Burst FIFO can be observed in the Burst-FIFO-Load field (`BfLoad`) in the same register.

[33] And the load of the entire Transmit FIFO is less than FIFO Size.

8.2.2.4. Bulk FIFO

The *Bulk FIFO* is the real meat of the Transmit FIFO. It provides the buffer capacity for compensating large PCI-, interrupt- and software scheduling latencies. The Bulk FIFO is implemented with an 8-Mbyte 16-bit wide SDRAM.

The SDRAM alternates read bursts, write bursts and refresh cycles. The hardware supports a maximum length of read- or write- bursts of 64 cycles (128 bytes).

Data is written to the SDRAM when (1) the Burst FIFO contains data, and (2) the maximum write-burst length has not been reached.

Data is read from the SDRAM as long as (1) the SDRAM contains data, (2) the Smoothing FIFO is not full and (3) the maximum read-burst length has not been reached.

8.2.2.5. Smoothing FIFO

The *Smoothing FIFO* is a 512-byte FIFO that smoothens data from the Bulk FIFO to a jitter-free DVB/ASI stream. This FIFO is required because the Bulk FIFO produces – by the nature of SDRAMs – bursts at its output.

The DTA-140 writes data from Bulk FIFO to Smoothing FIFO as long as if (1) the Bulk FIFO can supply data and (2) the Smoothing FIFO is not full.

If not in **Raw** mode, the DTA-140 reads data from the Smoothing FIFO when (1) the DVB/ASI output stage demands data to transmit another packet and (2) the Smoothing FIFO contains at least one packet. An Transmit-FIFO-Underflow condition occurs when it's time to send a packet, but the Smoothing FIFO contains less than one packet. In this case and when Null-Packet Stuffing is enabled (§6.2.1.3), the DTA-140 output stage will insert null packets.

In **Raw** mode, the DTA-140 does not check for the presence of at least one packet in the Smoothing FIFO. Whenever the DVB/ASI output stage requires another data byte (given the current transmit rate) and the Smoothing FIFO is not empty, a data byte is inserted in the output stream. If the Smoothing FIFO is empty (Transmit-FIFO Underflow), nothing is sent and the output bit rate will become lower than the rate specified in the Transmit-Clock register.

8.2.3. *Minimum Delay*

This section considers the question: In a real-time streaming application, given the structure of the Transmit FIFO described above, what's the minimum propagation delay incurred by the DTA-140?

The analysis assumes:
- Transmit Mode **188** (188-byte packets);
- A requirement for real-time operation, such that null-packet stuffing is not acceptable, e.g. because it breaks MPEG-2 time stamping. This means that Transmit-FIFO Underflow may not occur.
- The output delay is the time passing between a transport packet being available in host memory and the packet appearing in the DVB/ASI output stream[34].
- To make the analysis more concrete, a transport rate of 40 Mbps is assumed. The time to transmit one packet is approximately 38 μs.

8.2.3.1. Absolute Minimum Delay

The absolute minimum output delay of the DTA-140 has three components.

- The time between starting a DMA transfer and the first data byte arriving in the Smoothing FIFO. If the PCI bus is not used for other purposes, this latency will be below 10 μs.
- The Smoothing FIFO shall contain at least one packet (188 bytes) at all times, or an Transmit-FIFO Underflow may occur. The corresponding delay is 38 μs.
- The propagation delay of the DVB/ASI output stage. This delay is below 1 μs and can safely be ignored.

So, the absolute minimum output delay with which the DTA-140 can be operated is about 50 μs @ 40 Mbps.

[34] To compute the application's *end-to-end* delay, a similar analysis must be made for the input- and processing stage of the application.

8.2.3.2. Practical Minimum Delay

In practice, the theoretical minimum delay cannot be met reliably. The main problem, which has nothing to do with the DTA-140, is the initiation of DMA transfers by the host. This issue has been discussed extensively in §8.3.

With respect to the Transmit FIFO the following remark can be made. For consistent, reliable operation the Smoothing FIFO should contain significantly more data than the absolute minimum of 1 packet. DEKTEC recommends trying to keep the Smoothing FIFO full, which is about 500 bytes. The corresponding minimum delay is 100 μs.

In most applications, 100 μs will be very small compared to the delay required for a reliable buffer-management and synchronisation model. So, in all but some very special applications, the fine-grain model of the Transmit FIFO is irrelevant.

8.3. Latency

A continuous Transport Stream is essential for flawless operation in many MPEG-2 applications[35]. If not properly compensated, data-transfer latencies may lead to discontinuities.

This section describes the hardware- and software- latencies that may occur when streaming MPEG-2 packets with the DTA-140.

8.3.1. PCI-Bus Latency

The DTA-140 shares the PCI Bus with other bus masters that also compete for PCI cycles. The DTA-140 DMA Controller may have to wait a certain amount of time – the *PCI-Bus latency* – before it can acquire the bus and begin a DMA transfer.

Under normal conditions, the maximum duration of PCI latencies is in the order of a few microseconds. On a heavily loaded PCI Bus, latencies can be longer, but practical experience indicates that in all but pathological cases (see note) 2 ms can be safely taken as the absolute maximum PCI-Bus latency.

Note

- Cases are known[36] in which PCI latencies become unbounded. If real-time streaming is required, it is essential to check the host system fur such adversary conditions.

8.3.2. Interrupt Latency

Interrupt latency is the time between hardware raising an interrupt and software actually servicing the interrupt.

The DTA-140 hardware/software synchronisation methods (as described below in §3.2) rely on interrupts to signal certain hardware conditions to the software. The maximum interrupt latency has to be added to the total latency.

8.3.3. Scheduling Latency

The host CPU cannot dedicate all of its time to computing packets for the DTA-140: Other threads need processor cycles as well. *Scheduling latency* is the maximum time – in the worst-case scenario – the CPU still has to spend on other jobs, when new data needs to be processed for the DTA-140. Scheduling latency has to be added to the total latency too.

Scheduling latency is hard to grasp. It depends on many factors, like operating system, other software running on the host, relative priorities of threads and other hardware to be serviced.

For critical systems, a hard real-time operating system is required to bound the maximum scheduling latency.

8.3.4. Compensating Latencies

Adding all up, the total latency is the sum of PCI-Bus-, interrupt- and scheduling- latencies.

8.3.4.1. Compensating Transmit Latency

The principal technique to compensate for transmit latency is to transfer packets to the DTA-140 well in advance of actual transmission. The Transmit FIFO on-board of the

[35] MPEG-2 applications involving video or audio will badly suffer from any uncorrected distortion of the transport stream. Even the stuffing of a single packet invalidates the entire transport steam (leading to PCR errors and potential T-STD problems).

[36] E.g. the host CPU writes continuously to a frame buffer on the PCI-Bus for a long period of time.

DTA-140 functions as a temporary buffer, so that a smooth data stream can be provided at all times.

The size of the Transmit FIFO is programmable (§6.2.4). Two conflicting factors should be taken into account when choosing a value for the Transmit-FIFO size.

- *Latency tolerance*. The larger the Transmit FIFO, the more headroom exists for compensating latencies.
- *End-to-end delay*. In applications that process real-time input data[37], the Transmit FIFO acts as a delay buffer. A larger FIFO implies a longer end-to-end delay of the application. Note that, given a certain FIFO size, the delay is inversely proportional to the transport-stream bit rate.

DEKTEC cannot provide a cut-and-dried recipe for determining the optimal size of the Transmit FIFO. As for any hard real-time system, careful system analysis is required for the configuration at hand.

Good engineering practice demands that computed values are validated by experiment. A final safety factor (say 20%) will add to the robustness of the application.

8.3.4.2. Receive Latency

If the data from the Receive FIFO is not transferred fast enough to host memory, Receive-FIFO overflow will occur. Obviously, this condition should be avoided at all costs, to avoid discontinuities in the processed MPEG-2 data.

Receive-FIFO Overflow will not occur if the worst-case total latency is less than the time it takes to fill the entire Receive FIFO.

Thanks to the large size of the Receive FIFO (8 MB), the maximum total latency that can be tolerated is relatively long. At maximum input rate (214 Mbps), the maximum latency is still 314 ms.

[37] This is: the application cannot process data ahead of time. In applications that can pre-fetch data, such as in a disk-reader application, end-to-end delay is less of an issue.

8.4. Reading the Entire Rx FIFO

In some applications, e.g. test applications, it may be required to read all bytes in the Receive FIFO, up to and including the last byte. This requires some special consideration when using DMA transfers.

Note

- The special measures described below have been implemented in the DTAPI. So, if you use DTAPI's `TsInpChannel::Read`, the entire Receive FIFO can be read without complication.

The hardware architecture of the DTA-140 is optimised for streaming data at high speed from Receive FIFO to main memory. To sustain the data stream, the DMA Controller requires a certain amount of data in the Receive FIFO. If too few bytes remain, the DMA process stalls.

The above implies that, if one tries to read all the bytes in the Receive FIFO, the last few bytes (approximately 32 bytes) of the Receive FIFO cannot be read with DMA.

The following procedure should be followed to read all bytes in the Receive FIFO:

1. Read FIFO load;
2. Transfer FIFO load minus 40 bytes using DMA;
3. Transfer the last 40 bytes by direct reads from the Receive-FIFO-Data register.

Note

- This procedure works for receive modes **St188**, **St204** and **StMp2**. In these modes, the FIFO load is always a multiple of 4.
  In receive mode **StRaw**, however, the Receive-FIFO load may not be a multiple of 4. In that case, the last few bytes cannot be read, because the hardware only supports 32-bit access to the Receive FIFO.