DOR – API Description

Rev. 3.4b

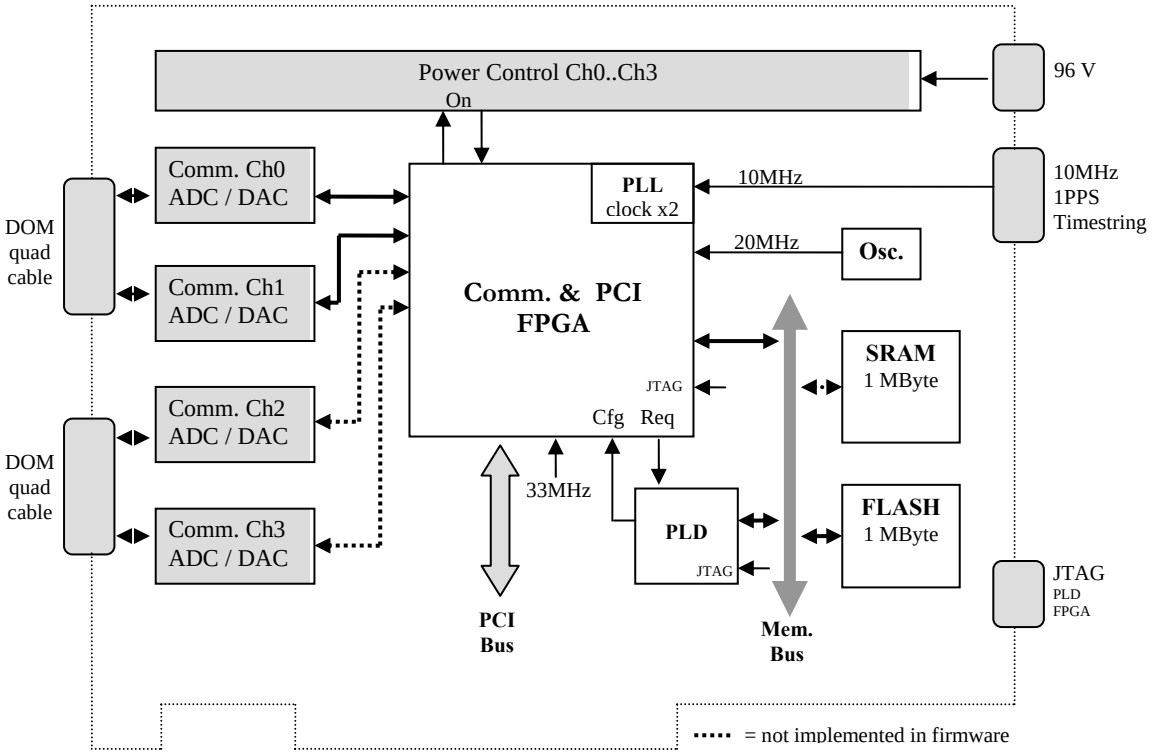
Karl-Heinz Sulanke
DESY Zeuthen

Created on 5/17/2007
com_104q or later

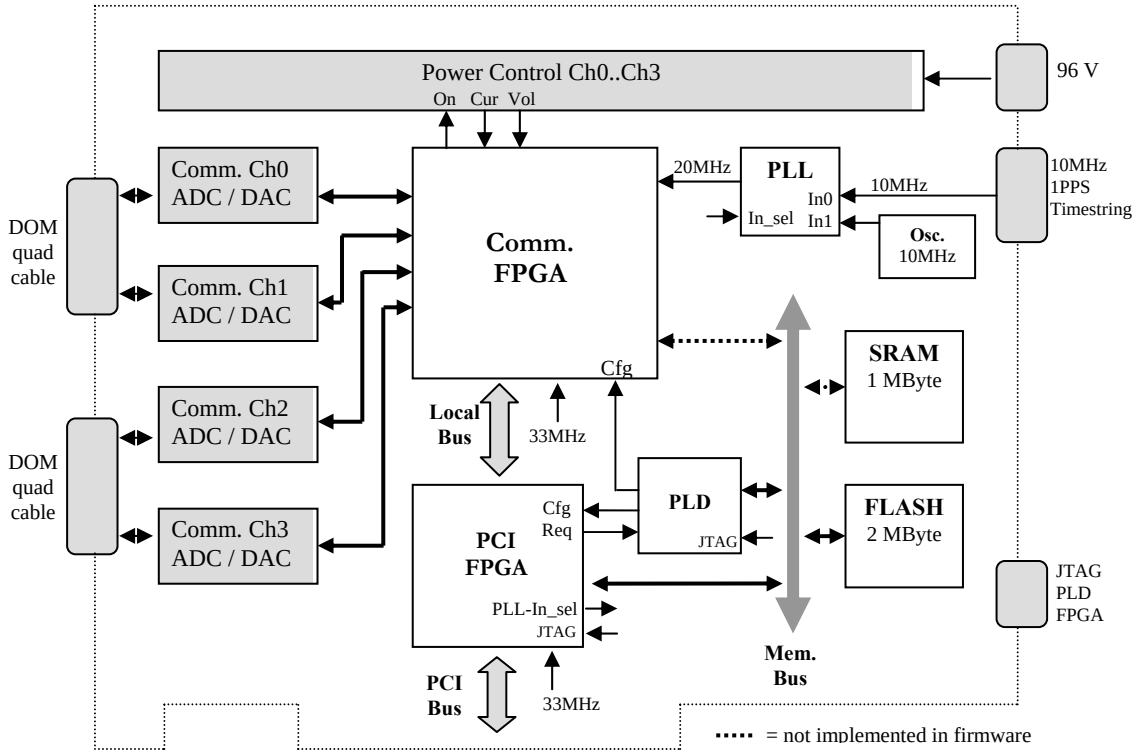
Introduction

This document is a first trial to describe the DOR (DOm Readout interface) - API. It should be seen as a basis of discussion. Comments and proposals are welcome.

DOR_rev0, Block Diagram



DOR_rev1, Block Diagram



Configuration Space Registers

Vendor ID			**1234h**
Device ID			**5678h**
Revision ID			**01h**
Class Code			**0c8000h**
Bits	**Meaning**	**Value**	**Description**
23..16	Class Code	0ch	Serial bus controller
15..8	Sub-Class Code	80h	Other network controller
7..0	Prog. I/F	00h	
Subsystem ID			**0000h**
Subsystem VendorID			**0000h**

Remark: With respect to PCI Rev. 2.2 the Class Code 0c8000h is an illegal combination ! It should get changed, if the extra software effort is within the limits.

FPGA Configuration, DOR_rev0

According to the PCI Bus conventions it is not allowed to connect more than one chip-pin to a PCI Bus signal. Therefore it is not possible to use a second, EEPROM based chip, like a PLD to control the PCI Bus while loading the FPGA.
However a PLD is useful after power-on or PC-Reset to load the FPGA by copying the data from a FLASH memory to the FPGA byte-wide configuration port. It takes about 0.2 seconds for the PLD (ALTERA EPM7064AETC44) to configure the FPGA by copying 246002 bytes from a 1 MB FLASH memory (AM29LV800B) to the FPGA-configuration port. The FLASH is divided into four 256KB pages.
After power on or PC reset flash page #0 is taken as default page always.

3 ways to initiate a FPGA configuration:

- Power on
- PC reset button
- Software controlled, see description of the CTRL register

FPGA Configuration, DOR_rev1

At DOR_rev1 we deal with a 3 chip ensemble: a PLD (Altera EPM7128BFC100), a PCI bus-controller FPGA (Altera-ACEX, EP1K50FC256) and a communication-controller FPGA (Altera-CYCLONE, EP1C20FC400).
After power on or PC reset the PLD loads the PCI controller FPGA first and the comm. controller FPGA afterwards. It takes about 0.2 seconds for the PLD to load both FPGAs.
The 2MB FLASH is divided into four 512KB pages. Pages #0..2 are preserved to be used for the comm. controller FPGA. The appropriate unkompressed .rbf file is exactly 444951 bytes in size. The compressed (enabled at the Quartus compiler: /Asignments/Device/…) .rbf file is roughly half that big. Both files can be used.
The first byte has to be written to the first byte position of the appropriate Flash page always.
Flash page #3 is preserved to accommodate the PCI bus controller design, 98023 bytes in size. Only two 64KB sectors are used.
It is up to the user what to do with the remaining six 64 KB sectors.
The additional Flash address bit19 is available at the FLASH reg. now.

Changing the FLASH and Reconfiguring the FPGA

When the FPGA is loaded, the FPGA itself can be used as an interface to alter the FLASH memory content. Once the FLASH FPGA-configuration region is completely overwritten, the PLD located state machine can be started again to reconfigure the FPGA. See the description of register #0 (CTRL) for more details.
Use register #15 (FLASH) to access the FLASH memory chip. A PCI bus read initiates a FLASH read, a PCI bus write **plus the WE-bit set**, a FLASH write cycle. Accessing the FLASH registers puts the FLASH into byte mode always. Reading is a two cycle operation, if the address value has to be written before. Writing commands to the FLASH can be achieved by a one cycle operation. To program a FLASH byte, 4 cycles are needed. A detailed knowledge of the FLASH datasheet is required. (DOR_rev0 -> Am29LV800B, DOR_rev1-> M29W160ET or AM29LV160BT) Both Flash types are 100% compatible (except the size).

Keep in mind that the PCI configuration space (a set of standard registers) is part of the FPGA as well. Some of these register like e.g. the Base_Address_0 are initialized by the BIOS while running the POST (Power-On Self Test). It is in the responsibility of the software to save these values and to restore them after the FPGA reconfiguration.

A short description of the procedure can be found here:

1. Save the PCI Configuration Space, the bus number and the device number
2. Reprogram the FLASH by copying the .rbf file bytes to the FLASH
3. When all bytes have been programmed, activate the FPGA reload (from the recently prepared FLASH) by selecting the FLASH page and setting the CTRL reg.-bit FPGA_RELOAD to one.
4. It takes about 0.2 seconds to reload the FPGA by the onboard PLD. Take the previously stored bus and - device number to verify the Configuration register
5. Check the firmware revison number (FREV, reg. #31) of the new FPGA image

If the reconfiguration failed, reset or powercycle the PC to get the protected default page loaded.

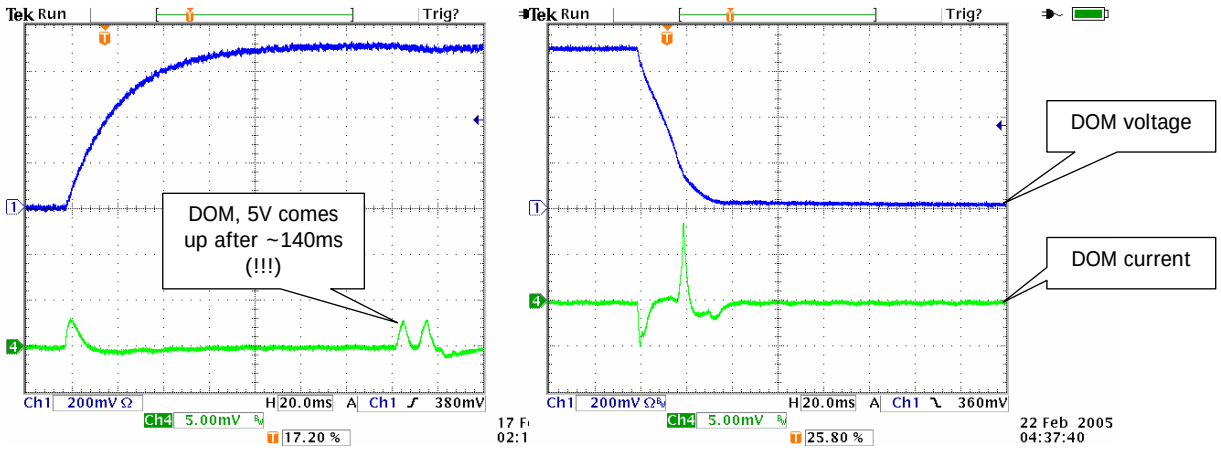
DOR Communications Clock Source Selection

The communication clock is 20Mhz. The DOR cards have an external 10MHz clock input, to be synchronized by the GPS.
At **DOR_rev0** the clock source get selected by the firmware version. dor_02--.rbf runs from the 20MHz onboard oscillator. dor_01--.rbf gets driven by the external 10MHz clock.
At **DOR_rev1** just one firmware version is needed. The clock source get selected by CTRL-bit30. The clock selection takes place at the onboard PLL circuit (QS5LV919).
IMPORTANT !!! After selecting the clock source it may take up to 10msec for the 20MHz communication clock (PLL-output) to be stable. Undefined clock transition after reselecting the clock source might bring the Comm. FPGA into an undefined state. Therefore a global reset (CTRL-bit23) should be issued afterwards always.

DOR Power Switch Control

At **DOR_rev0** the power switch is controlled by CTRL-bits 0..3 only. The wire_pair_power_on_ready bits 4..7 of DSTAT are hard-wired (faked) to the CTRL-bits 0..3. There is NO firmware based over / under current / voltage protection at all ! To protect against overcurrent the software has to read the DCUR register.

At **DOR_rev1** the power switch is controlled by a state machine, based on a continously running current and voltage measurement. The cycle time is about **9 µs**. There is full over / under current / voltage protection. The appropriate limits have to be adjusted using the CURL register. To fade out current peaks while powering on or off, the states PONING and POFFING have been introduced. There is an over voltage protection in the PONING state. The power gets switched off if the current value is bigger than four times previously set at the CURL register or the voltage is to high.
Full protection is given in the PON state only.

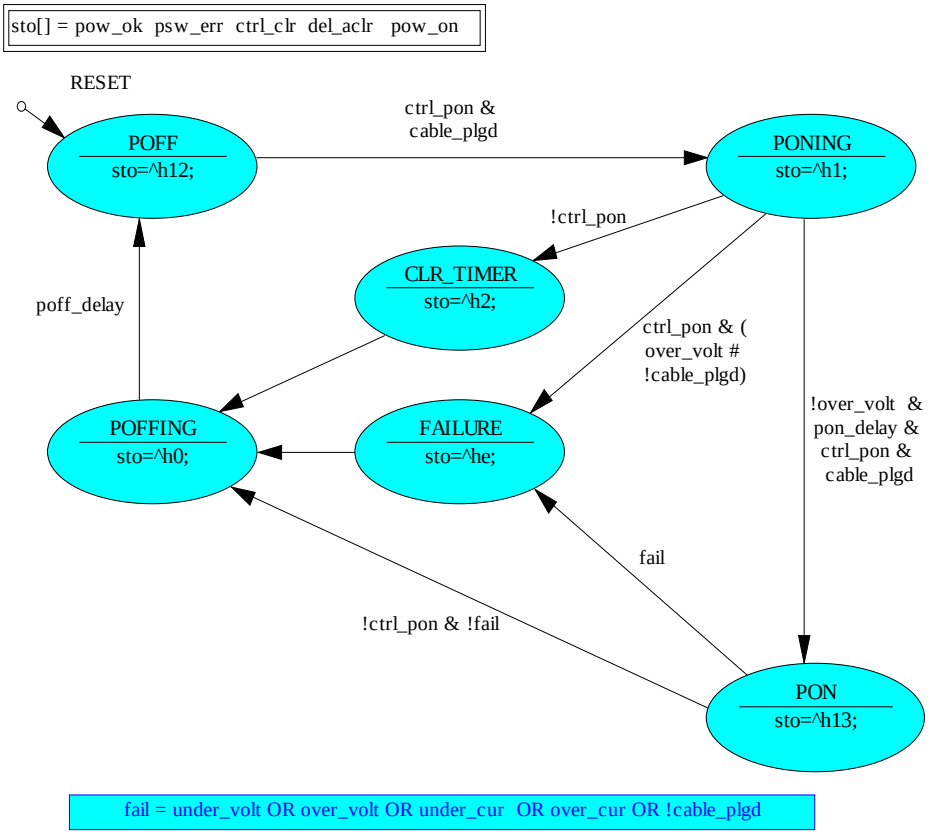


corresponding state machine – states, see below

POFF	PONING	PON	POFFING	POFF

The state machine, controlling the power switch is shown below.
All states except POFF and PON are temporarily states. The following table illustrates the meaning of signals / conditions with respect to the DOR API.

!, #, &, ^h	inversion, OR, AND, hex
ctrl_pon	the appropriate read/write CTRL-bits 0..3 is ‘1’
pow_on	sto[0], signal, connected to the power switch
del_aclr	sto[1], clears the power on/off delay timer
ctrl_clr	sto[2], clears the previously set CTRL-bit 0..3 (ctrl_pon ->’0’)
psw_err	sto[3], snapshots the error condition, see DSTAT-bits 8..11,16..31
pow_ok	sto[4], power on OR off is ready, see DSTAT-bits 4..7
pon_delay	300 ms, by power on/off delay timer
poff_delay	80 ms, by power on/off delay timer
over_volt	typical 100V, programmable, see reg.#13 CURL
under_volt	typical 70V, programmable, see reg.#13 CURL
over_cur	typical 140mA, programmable, see reg.#13 CURL
under_cur	typical 20mA, programmable, see reg.#13 CURL



A typical software power on sequence:

read DSTAT	check for power_on_off_ready bits 4..7, if ready do …
write CTRL	set appropriate bits 0..3, then
wait	at least 380 ms or poll on DSTAT bits 4..7 getting ‘1’ again, then
read CTRL	if the appropriate CTRL bit is still on, power on was successful, else read DSTAT for debugging the error condition

DOR Timer

The DOR timer is a binary 56 bit counter, clocked by the 20MHz communication clock.

The timer reset is:

Timer_reset = FPGA_reload **OR** CTRL_bit9

To get a snapshot of the DOR timer you have to toggle the CTRL_bit10 "TIMER_SNAP".
By this measure the DOR timer value gets synchronized to the PCI clock and can be taken from UTCRD0/1.
Toggling CTRL_bit9 creates a Timer_reset pulse. CTRL_bit9 is NOT a static clear and cannot be used to hold the timer in a reset state.

Time Synchronization & Calibration

The DOR timer, driven by the 20MHz communication clock, provides the time stamp needed for the time calibration.
Presently the time calibration is software controlled, simply achieved by a push-button method.. For every DOM a bit in the DOMC register is available. After this bit has been set, the software must poll on the same bit. The bit gets cleared after the completion of the TCAL procedure. A TCAL data packet is available now.
The time calibration data according to the described format can be taken from the appropriate TCAL data buffer (TCBUF).

GPS Time String Buffering

The GPS time string gets received at the TSTRG-LVDS input which is feeded by the DSB (DOMhub Service Board).
The time string format is: "(SOH)DDD:HH:MM:SSQ(CR)(LF)" .
It gets recorded every second until the time string buffer (reg.#28, TSTRG) is full (eleven time strings). (CR)(LF) does NOT get recorded !

The RS2323 protocol is

9600 Baud, one start bit, eight data bits, and one stop bit.

The protocol parameters can not be changed by software. The GPS system must be setup accordingly.

After receiving the quality indicator Q the firmware waits for the next upgoing PPS (Pulse Per Second) edge. This edge triggers the snap-shot-taking of the 64 bit DOR timer (highest byte is zero).
After recording the 8 timer snapshot bytes a complete time string (22 bytes) is available.
Status bits (GSTAT, bit12,13) are set after having TEN / ONE time strings recorded. An appropriate interrupt can get enabled (INTEN, bit12,13).
The firmware takes care that the time strings are always in a 22 byte boundary.

The 22 bytes are defined as followed:

byte	meaning	comment
0	(SOH)	Start Of Header (ASCII control character)
1..3	DDD	Julian day
4	":"	delimiter
5..6	HH	hour
7	":"	delimiter
8..9	MM	minute
10	":"	delimiter
11..12	SS	second
13	Q	Quality indicator of the 1PPS accuracy, see table below
14..21	CCCCCCCC	Binary (!!!) timer[63..0] snapshot, multiples of 50ns (20MHz), byte14 is the most -, byte 21 the least significant byte

The 1PPS quality indicator Q according to the GPS systems (ET6010 ExacTime GPS TC & FG) manual:

ASCII Character	HEX Equivalent	Definition
(space)	20	< 1 microsecond
.	2E	< 10 microsecond
*	2A	< 100 microsecond
#	23	< 1 millisecond
?	3F	> 1 millisecond

Use test connector J17 (DOR_rev0) or J2 (DOR_rev1) to verify a correct 1PPS to 20MHz alignment.
1PPS_setup time = 5.. 45ns (with respect to the 20MHz_out -LH-edge)

DOR_rev0 J17-pin	DOR_rev1 J2-pin	**Signal** (LVTTL)
1	14	GND
2	15	20MHz_out
3	13	PPS_out
4	16	GND

Data Presentation, DOR-Tx Buffer:

Data will be send in a little endian format. The example below shows the sending of 7 bytes:

PCI-Bus data to Tx Buffer:

Header	12348007
1. Dataword	44332211
2. Dataword	00776655

Data Presentation, Cable:

Data bytes, sent to the DOM A, cursive means ***added by hardware***:

Start_Of_Frame	*e3*
Packet_Length_7..0 DOM_AnotB, Packet_Type_2..0, Packet_Length_11..8 Sequence_Number_7..0 Sequence_Number_15..8	07 80 34 12
1. Dataword	11 22 33 44
2. Dataword	55 66 77 00
CRC_byte3	*xx*
CRC_byte2	*xx*
CRC_byte1	*xx*
CRC_byte0	*xx*
End_Of_Frame	*99*

Remark: bit DOM_AnotB gets overwritten by hardware, depending on the message buffer (MBFx) used

The Data Byte

0	1	2	3	4	5	6	7	8	9
Start	**Data bits**								**Stop**
1	bit0	bit1	bit2	bit 3	bit 4	bit 5	bit 6	bit 7	**1**

The Control Bytes STF and EOF

0	1	2	3	4	5	6	7	8	9
Start	**“E3” or “99”**								**Stop**
1	bit0	bit1	bit2	bit 3	bit 4	bit 5	bit 6	bit 7	**0**

Adjusting the Communication for ICECUBE and ICETOP

Currently the communications signals decoder is based on the decoding of the falling edge. If the falling edge is longer than the adjusted threshold (comm_thresh[]), a logic one gets detected.
In the final design phase the DOM got an additional capacitor of 150pF, in parallel to the receiver input,. The idea was to reduce the communications signal EMI, caught by the DOM's analog path. Unfortunately this is a built in hazard for the communication interface, because this capacitor is causing immense reflections, see the figure below.
The problem arises after the DOR has sent it's last byte and switched to the receiving mode (half duplex). The reflection gets misinterpreted as the very first communications signal edge, being equivalent to the UART start bit. The first byte, the start of frame (stf), did not get recognized, the whole packet is lost.
An adjustable DOR-receiver disable time is helpful to fade out the above mentioned reflections. The disable time called rec_delay[] (see table below) has to be chosen with care. If the time is to big, the real signal gets lost as well. The right receiver delay can be calculated according to the formula:

rec_delay[] = cable_length (m) / 5 + 10

If the cable is longer than **1225**m, the max. value of 255 should be taken.
Two parameters affect the size of the reflected signal, the length of the cable and the initial size of the signal, sent by the DOR. Therefore the amplitude (dac_max[]) of the sent by the DOR signal should be as small as possible. Another good measure would be, to have a constant DOM send delay >10us (length of one byte @ 1Mbit/s) . Unfortunately the DOM power up (configboot) firmware, featuring zero delays, is fixed already.
An "external" measure to get rid of the reflections is to use a filter box, like done for ICETOP already. Although the motivation there was to unsharp the signal edges for more TCAL waveform samples.
Finally I want to mention, that both, the DOM and the DOR PCBs allow low pass filtering in the receiver path as a mounting option. Using this option, we wouldn't need neither adjustable communication parameters nor extra ICETOP filter boxes.

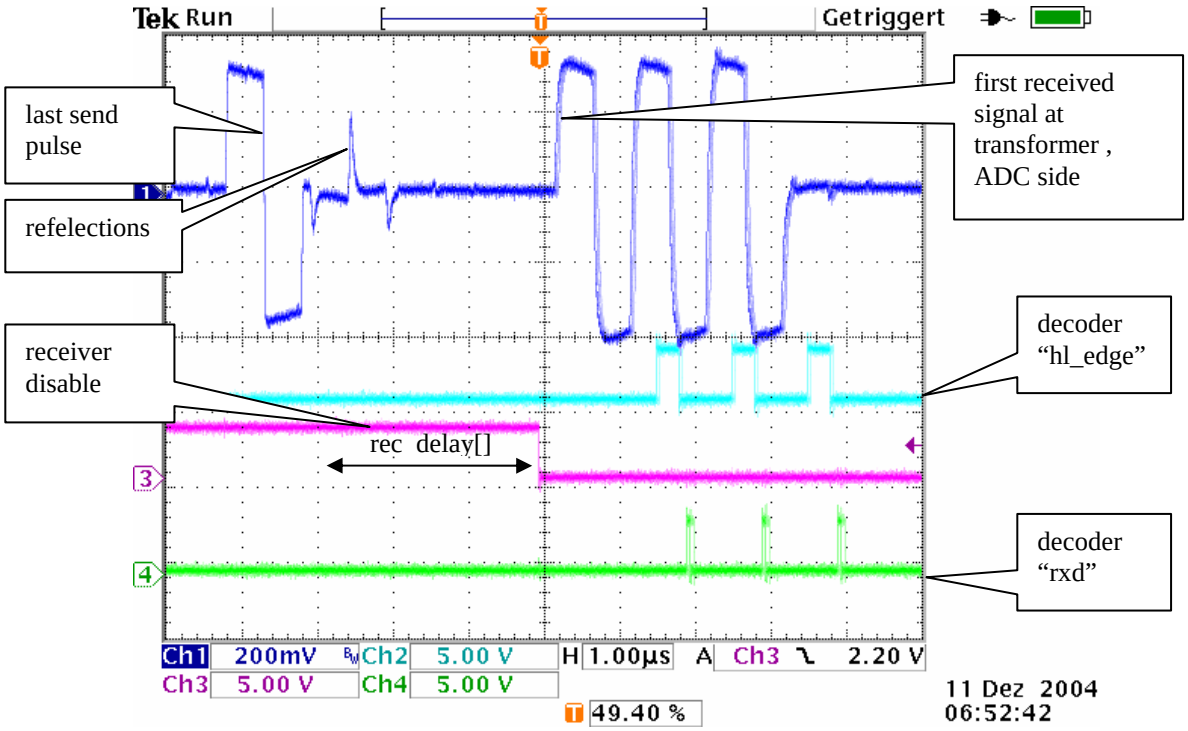


Figure **, Receiver disable (scope channel 3) to fade out reflections, 100m cable**

Use the write function of reg. **#30** (DCREV) to adjust the communication threshold, the receive-enable delay and the send-enable delay.
The following table shows recommended values for DOR_rev1 and DOR_rev0 (in()) :

bits	**name**	**range**	**ICECUBE cable > 1500m**	**ICETOP + filter box cable > 100m**	**UW filter box**	**comment**
7..0	comm_thresh[]	0..255	64	255	64	length of the falling edge of the comm. Signal after 4 clocks (200ns)
13..12	dac_max[]	0..3	2 (3)	1 (2)	2 (3)	comm_DAC_amplitude = 128 + dac_max[]*32 + 31, see in () for DOR_rev0
23..16	rec_delay[]	1..255	255	30	10	multiple of 50ns to fade out reflections from the last DOR-send, seen by the DOR receiver stage; receiver_enable_delay = shortest_cable_length (m) / 5 + 10
31..24	send_delay[]	1..255	1	255	255	multiple of 50ns to fade out reflections from the last DOM-send seen by the DOM receiver stage

Message Format (by Arthur Jones)

32 bit word aligned, data arranged in little endian format...

```
|<------------ 32 bits ------------>|
|<--- 16 bits --->|<--- 16 bits --->|
| sequence number | type and length | <- start of packet
|                 .                 |
|                 .                 |
|                 .                 |
|        length bytes of data       |
|                 .                 |
|                 .                 |
|                 .                 |
| pad2  |  pad1   | pad0  |  datan  |
|<------------ CRC 32 ------------->| <- last 32 bit word
```

total packet length in bytes = ((length+3)/4) * 4 + 8

where:

sequence number:

the (unsigned 16 bit) sequence number of the data packet sent or, if the packet is an acknowledge (ack) the sequence number of the packet that is being acknowledged. the sequence number is only used in data and ack packets, other packet types could use this field for other purposes, see the type field description below for other uses of this field by other packet types.

rationale:

in order to do error correction in software, we need to keep track of the packet sequence number. 16 bits of sequence number allows us to look forward or backward 32K packets, this is far more than we are ever going to buffer, but the next byte aligned step down (8 bits) only gives us +-128 of range which is currently less that we're using now and so would not be enough margin for a safe design. a non-byte aligned sequence number size would not allow us to use a standard "C" type to hold the value, hence all sequence number arithmetic would require masks, a process that would be error-prone and probably more complicated than could be justified by the savings in bits on the wire.

type and length:

```
|A |<-type->|<------------- length ------------->|
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- |
|15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0 |
```

where:

A (1 bit): dom A or B bit.

the "A" bit allows us to determine whether this packet was destined for the A dom (1) or the B dom(0). in the case of packets heading for the surface, this bit indicates whether packets are sourced from the A dom or the B dom.

rationale:

tagging each packet with the destined dom may allow simplifications of the hardware design at a later state. esp on the dor side, where it would allow the doms sharing a cable to also share a tx fifo -- hence, more effectively using the available fifo space...

type (3 bits): packet type.

the packet type is (base 2):

000: data packet in the beginning/middle of a sequence.
001: acknowledgement of a data packet.
010: data packet, at the end of a sequence.
011: control packet.
100: initiate connection
101: connection initiated
110: DOR control message
111: undefined

rationale:
in order to support software error correction,
we need the first 6 messages. we use these as follows:

000: data packet, no syn_fin. this is a data packet that is not yet ready to be passed to the user. it has another packet behind it with the rest of the data. the sequence number field contains the sequence number of this data packet. using this scheme we are able to translate "software" packets of any length to "hardware" packets of a fixed maximum length.

001: ack packet. acknowledge a data packet. the sequence number field describes the data packet that we are acknowledging. no payload with this packet.

010: data packet, syn_fin set. this is a data packet that is complete and ready to ship to the user. any previous packets with no syn_fin are prepended in order to this packet when shipping to the user.

011: control packet. this packet never makes it to the user but is used to communicate unreliably between the "kernel" communications agents. we are currently using this type of packet to pass comm statistics from the dom to the dor.

100: initiate connection. on startup, the dom and dor must negotiate sequence numbers. this packet type allows us to signal that we want to start the negotiation. there is no payload to this packet.

101: connection initiated. this packet type confirms to the other side of the comm link that we are alive and ready to start our tx and rx sequence numbers from zero.

110: DOR control message, see detailed description on next page.

111: undefined. left over type

110: DOR control message.

This packet type allows DOR control messages to be added to the packets. The control message data are put in the sequence number field of the packet, the only messages defined now are :

DOR commands (**bit11..8** of sequence number field):

		by DOR	by DOM	
0x1	---------			not used
0x2	---------			not used
0x3	IDREQ	x		dom id request
0x4	---------			not used
0x5	DRREQ	x		data read request
0x6	DRAND		x	data read ack by dom, dom tx has no data
0x7	DRBT		x	dom reboot
0x8	MRWB		x	message received, more Rx buffer avail
0x9	MRNB		x	message received, no more Rx buffer avail
0xa	MRWE		x	message received, but CRC error detected
0xb	COMRES	x		communication channel reset
0xc	BFSTAT	x		dom rx buffer status request
0xd	SYSRES	x		dom system Reset (softboot)
0xe	TCAL	x		start time calibration
0xf	IDLE	x	x	dor idle, answered by DOM idle

Bit 12 on means the appropriate DOM left CONFIGBOOT. This bits gets used to block TCAL or SYSRESET (softboot) commands (by software) while the DOM is still in CONFIGBOOT. Used for “up going” messages only.

The following schema is used to control the communication signals strength (**bit15..14** of sequence number field in DOR control messages) :

bit15 signal_up_request
bit14 signal_down_request

When receiving:

```
IF (signal_up_request  AND  comm_DAC_level < 255)
        THEN comm_DAC_level = comm_DAC_level +1;
IF (signal_down_request  AND  comm_DAC_level > 160)
        THEN comm_DAC_level = comm_DAC_level -1;
```

When sending:

```
IF (comm_ADC_max_level [] < CLEV_MIN)
        THEN signal_up_request =1;
IF (comm_ADC_max_level > CLEV_MAX)
        THEN signal_down_request =1;
```

Remark: CLEV_MIN, CLEV_MAX can be programmed in some firmware revisions (e.g. 010w, 011d). Typical values are CLEV_MIN=800..960, CLEV_MAX=810..970. See also reg.#31 (FREV) description.

length (12 bits):

data (payload) length in bytes
this field allows us to know how big the data portion of the packet will be. the actual packet length can be calculated from this number using the "C" code (integer math):
packet length = ((length+3)/4) * 4 + 8

rationale:

this field is first in the header as hardware crc detection and creation would require the length of the data in order to work (and nothing else). the 12 bits of packet length allow us up to 4096 byte packets (4104 bytes including header and crc). at 4096 byte packets the header/crc overhead is about 1 part in 512. this should be negligible overhead for the high speed data throughput mode. keeping the length reasonably small allows us to buffer less data on the dor and dom side when using software error correction mode.

data (((length+3)/4) * 4 bits):

this is the payload of the packet. the data are organized in
little endian. and padding is added to the end of this field
so that the data are a multiple of 4 bytes.

rationale:

4 byte alignment allows the software to be greatly simplified and also allows us to use 32 bit dual ported memory in the dom side. also, the pci bus core we're using on the dor side has a 32 bit bus, so data can be directly dropped into the packets. also, 32 bit alignment allows us to drop in the 32 bit crc more cleanly using firmware. the overhead of average loss of 12 bits is not significant compared to the common packet sizes that we expect.

CRC (32 bits):

this field is the 32 bit aligned CRC32.

rationale:

crc32 is commonly used to detect data transmission errors, 32 bits gives us quite a bit of confidence in the packet integrity, and is relatively easy to calculate in firmware.

Data Buffer Format

Two 32 bit wide data buffer per DOM exists to handle all data traffic, TXBFn and RXBFn. The overall buffer size per DOM is 2KB. All 16 DOM buffers are physically located in the FPGA. The Tx and the Rx part of the buffer are symmetrically, 1 KB for Tx and 1 KB for Rx . From the user point of view these buffers behave like fifos.
When sending, several messages can be piped into the TXBFn. One just has to ensure that enough place is available.
When receiving, first step is to know which DOM has sent data. Next step is to to copy the data from RXBFn into memory.
Incomplete or corrupted (CRC error) packets are getting rejected by the firmware. The appropriate 16 bit communication error counter (CERR, reg. #25) gets incremented.

Sending Data to the DOM

The message buffer (**TXBF**n) is 250x32 ***(1000 Bytes)*** deep. It can get accessed in two ways, either as a FIFO-port register or by initiating DMA (Direct Memory Access). Per DOM three status bits are available in the **TTSIC** register:

Status bit	*Register*	*Description*
TX_EF_n	**TTSIC**	Tx Buffer of DOM_n is empty, 250 data words (32 bit) can be written to
TX_AEF_n	**TTSIC**	Tx Buffer of DOM_n is almost empty, at least 150 data words (32 bit) can be written to
TX_AFF_n	**TTSIC**	Tx Buffer of DOM_n is (almost) full, at least 254 data words (32 bit) are in the Tx buffer

The following steps have to be undertaken by the DOR driver to access the **TXBF**n in a register mode:

1. Look to the **TTSIC** reg. to know about the available space for a certain DOM
2. Write the message to the **TXBF**n register 32 bit word-wise (!), if the packet length is not in a four-byte-boundary fill the missing positions with zeros
3. if necessary, poll on the global status or on the individual **TX_AEF_n** bit or expect an interrupt

The following steps have to be undertaken by the driver to access **TXBF**n in a DMA mode:

1. Look to the **TTSIC** reg. to know about the available space for a certain DOM
2. Load the appropriate memory pointer to the **MRAR** reg.
3. Load the appropriate memory byte count and the DOM number to the **MRTC** reg.. The byte count has to be in a four-byte-boundary (!!!). Encode the buffer number (0..7) by using MRTC bits 24..26. If the **MR_RD_EN** (**CTRL** reg.) is already set, the DMA starts immediately, skip point 4.
4. Set the **MR_RD_EN** bit in the **CTRL** reg.
5. to know about the status of the ongoing DMA poll on the **MR_RD_TC** (Master Read Transfer Complete) bit of the **GSTAT** reg. or expect an interrupt

Do not access **TXBF**n by a write access to the same DOM buffer if a DMA read is ongoing. Processor – Write and DMA-Read means the same direction of data flow ! Messages 100 bytes or less in size are probably more efficient transfered by accessing **TXBF**n in a register mode.

Receiving Data from the DOM

The message buffer (**RXBF**n) is 256x32 (***1024 Bytes***) deep. It can get accessed in two ways, either as a FIFO-port register or by initiating DMA (Direct Memory Access). Per DOM three status bits are available in the **RTSIC** register:

Status bit	*Register*	*Description*
RX_EF_n	**RTSIC**	Rx Buffer of DOM_n is empty
RX_AFF_n	**RTSIC**	Rx buffer almost full flag of DOM_n, Rx buffer of DOM_n contains more than 104 data words (32bit),
RX_MRCVD_n	**RTSIC**	At least one complete Message has been received, coming from DOM_n, if enabled these bits can cause an **interrupt**,

The following steps have to be undertaken by the driver to access **RXBF**n in a register mode:

1. look to the **RTSIC** reg. to know which DOM has sent a message
2. read the very first 32 bit word from **RXBF**n to know about the message size
3. read the remaining words and copy them to memory, if a non 4 byte boundary packet length has been found, read the last word (32 bit access) and copy the needed bytes by using an 8 bit memory write instruction

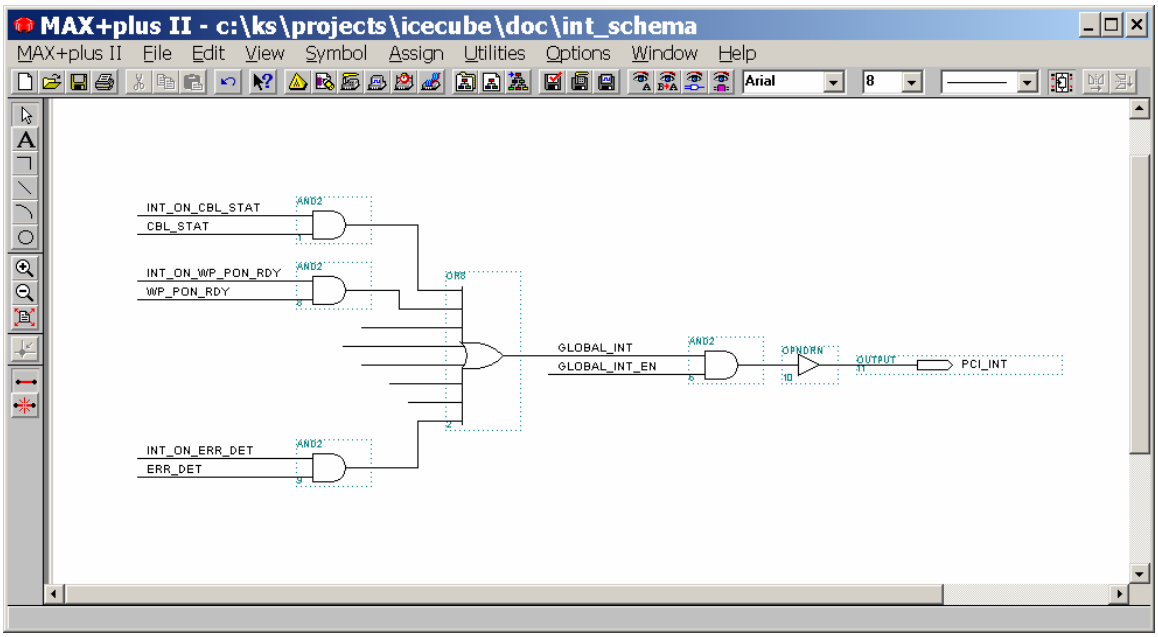
The following steps have to be undertaken by the driver to access **RXBF**n in a DMA mode:

1. Look to the **RTSIC** reg. to know which DOM has sent a complete message
2. Read the very first 32 bit word from **RXBF**n to know about the message size
3. Load the appropriate memory pointer to the **MWAR** reg.
4. Load the appropriate memory byte count and the DOM number to the **MWTC** reg.. Use a four-byte-boundary always !!! Encode the buffer number (0..7) by using MWTC bits 24..26. If the **MR_WR_EN** (**CTRL** reg.) is already set, the DMA starts immediately, skip point 5.
5. Set the **MR_WR_EN** bit in the **CTRL** reg.
6. to know about the status of the ongoing DMA poll on the **MR_WR_TC** (Master Write Transfer Complete) bit of the **GSTAT** reg. or expect an interrupt
7. if a non 4 byte boundary packet length has been found, read the last word using a 32 bit register access and copy the needed bytes by using an 8 bit memory write instruction

Do not access **RXBF**n by a read access to the same DOM buffer if a DMA write is ongoing. Processor – Read and DMA-Write means the same direction of data flow ! Messages 100 bytes or less in size are probably more efficient transfered by accessing the **RXBF**n in a register mode.

Interrupts

Similar to the ISA bus conventions there exist a shared interrupt on the PCI Bus as well. The interrupt line is not fixed but configured by the BIOS while running the POST (Power On self Test). Read the Interrupt Line Register of the DOR Configuration Space to know about the interrupt line. It should be a number in between 0 to 15 (decimal). Beside the classical style of using open drain outputs to share an interrupt line another method has been defined by PCI Bus standard Rev. 2.2. It is called Message Signaled Interrupt (MSI). The idea is to overcome the delay caused by the interrupt sharing. The question is, if the platform (CPU board) is supporting this type of interrupts. We should use the classical style I assume.
The DOR interrupt schema is shown below. Every interrupt source is enabled by an appropriate bit of the Interrupt Enable register (INTEN) performed by an 2 input AND. All these ANDs or ORed to perform the signal GLOBAL_INT (see GSTAT reg.). If the GLOBAL_INT_EN bit is on, the PCI_INT open drain output gets pull down in case of a pending interrupt.



In an interrupt service routine the first step is to find the cause of the interrupt by checking the GSTAT register bit GLOBAL_INT. If this bit is set, the other GSTAT register bits have to be analyzed. E. g. a message has been received; the appropriate DMA setup could take place.
To clear the pending interrupt three ways are possible. See the above schema: remove the interrupt causing condition. Write for instance a one to the CBL_STAT bit of the GSTAT register. Disabling the specific (e.g. INT_ON_CBL_STAT) or the global interrupt enable (GLOBAL_INT_EN) is another way to stop driving the PCI_INT line.

The Time Calibration Packet

The TCAL packet concists out of a header and two symmetrical data portions. The first half are the DOR related, the second are the DOM related data. The software should always use the packet length information (, amount of bytes, header bits 0..15) to calculate the pointer to both data records. The TCAL buffer (TCBUF) is 64x32bit in size. The presently used DOR firmware creates packets with the header 0x000100E0 and 56x32 bit words payload, which corresponds to 2 x 48 TCAL pulse waveform samples. The '1' in the header remained for historical reasons and became meaningless now.
The structure of the time calibration packet is shown below.

31 24	23 16	15 0	
0x00	0x01	Packet_Type_and_Length=0x00E0	
DOR_Tx_Time_31..0			
DOR_Tx_Time_63..32			
DOR_Rx_Time_31..0			
DOR_Rx_Time_63..32			
0x0	DOR_ADC_1_11..0	0x0	DOR_ADC_0_11..0
...			
0x0	DOR_ADC_47_11..0	0x0	DOR_ADC_46_11..0
DOM_Rx_Time_31..0			
DOM_Rx_Time_63..32			
DOM_Tx_Time_31..0			
DOM_Tx_Time_63..32			
0x0	DOM_ADC_1_11..0	0x0	DOM_ADC_0_11..0
...			
0x0	DOM_ADC_47_11..0	0x0	DOM_ADC_46_11..0

Figure 2, **TCAL Packet Format**

DOR_Tx_Time_63..0

Multiples of 20MHz clock cycles (50ns) . On board counter, driven by the local clock (for test purposes) or an external clock (standard ICECUBE mode). The time corresponds to the leading edge of the sent time calibration pulse.

DOR_Rx_Time_63..0

Multiples of 20MHz clock cycles (50ns) . On board counter, driven by the local clock (for test purposes) or an external clock (standard ICECUBE mode). The time corresponds to the last ADC sample of the received time calibration pulse.
IMPORTANT !!! The first revision of the DOR card is equipped with an ADC (AD9235) having an internal delay of **8 samples**, while the DOM uses the ADC AD9215 with an internal delay of **6 samples !!!** The latest DOR card revision is equipped with the AD9215.

DOM_Rx_Time_63..0

Multiples of **40MHz (!!!)** clock cycles (25ns) . On board counter, driven by the local clock . Bits 47..0 are used only. Bit 63..48 are zero. The time corresponds to the last ADC sample of the received time calibration pulse. **The ADC is clocked with 20MHz (!!!).** The 20 MHz clock is made from the 40 MHz clock. The leading edges are corresponding.
IMPORTANT !!! The DOM uses the ADC AD9215 with an internal delay of **6 samples !!!**

DOM_Tx_Time_63..0

Multiples of **40MHz (!!!)** clock cycles (25ns) . On board counter, driven by the local clock. Bits 47..0 are used only. Bit 63..48 are zero. The time corresponds to the leading edge of the sent time calibration pulse.
After receiving the TCAL pulse the DOM waits for 15 µs before sending it's TCAL pulse back.

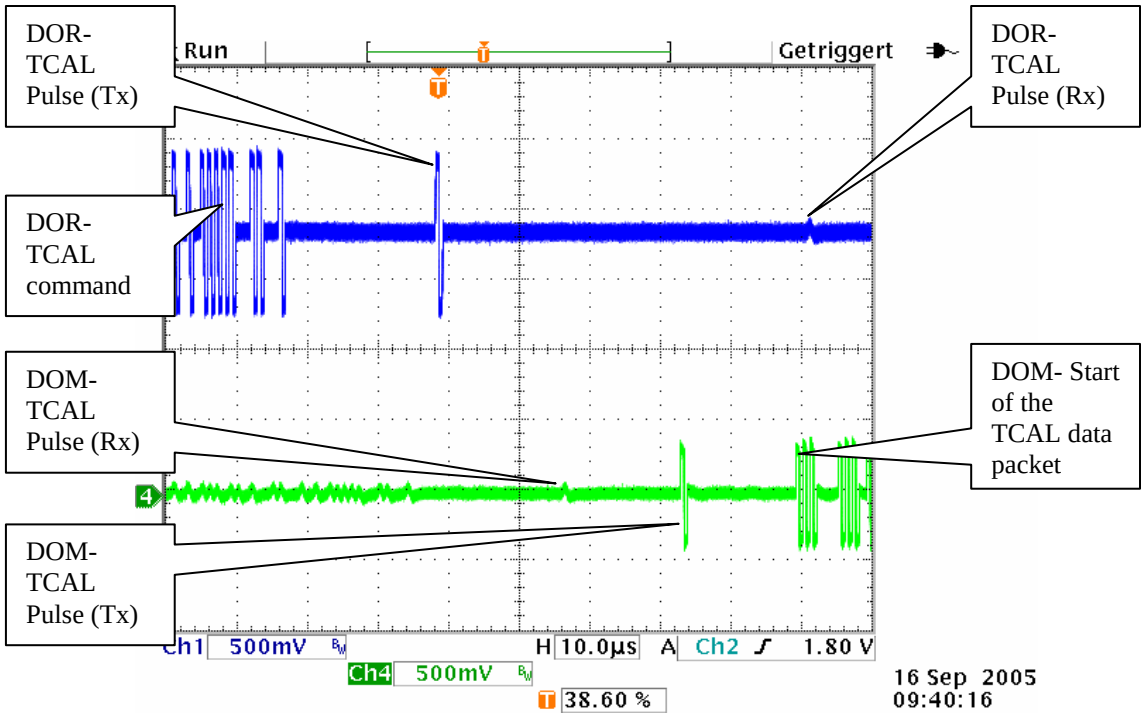


Figure 3, **TCAL pulses, 3.5 km New Ericsson Cable, 1→DOR, 4→DOM, full cycle ~ 1.3ms**

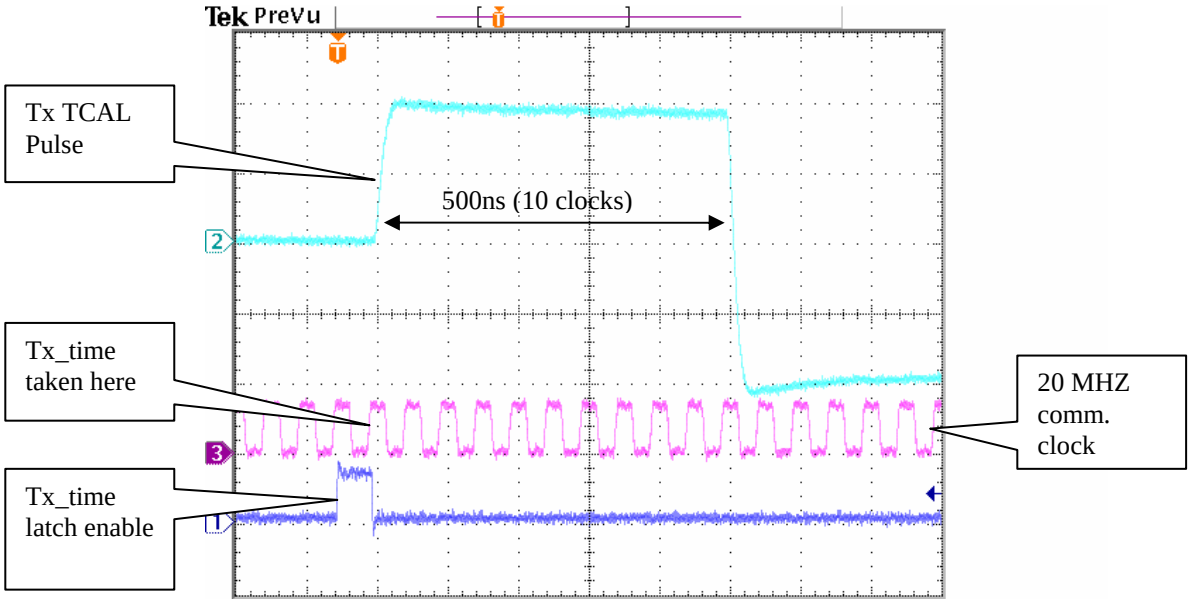


Figure 4, **TCAL, DOR / DOM Tx_time sample point**

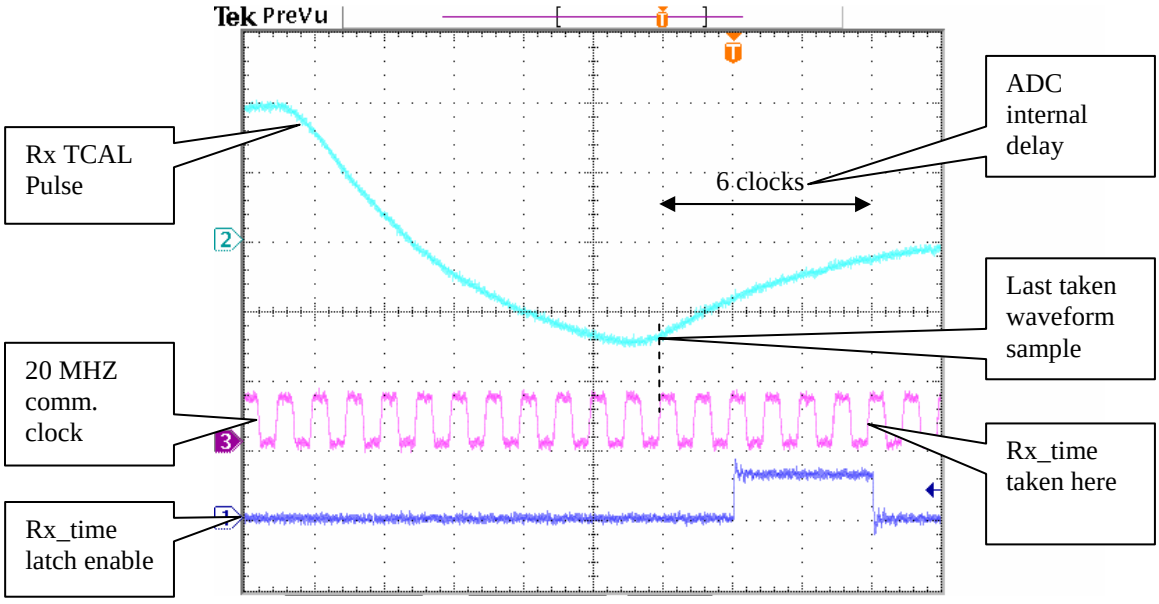


Figure 5, **TCAL, DOR / DOM low going edge, Rx_time sample point at ADC[n] > ADC[n-1]**

Recommended DOR - Global Register Init

Register	Bits	Write Operation / Comment
CTRL	31..0	CTRL = 0x00000000
		500ms delay (if DOM power was on before)
CTRL	31, 30	clock_select, timout_enable
		10ms delay
INTEN	31..0	INTEN = 0x00000000, Clear all interrupt enable bits
TTSIC	31..24	TTSIC = 0x00000000, Clear all interrupt enable
RTSIC	31..24	RTSIC = 0x00000000, Clear all interrupt enable
TCSIC	31..16	TTSIC = 0xffff0000, Clear interrupt enable and TCAL packet received bits
MRTC	31..0	MRTC = 0x00000000, Clear Master Read Transfer Count (DMA)
MWTC	31..0	MWTC = 0x00000000, Clear Master Write Transfer Count (DMA)
GSTAT	10,3,2	GSTAT = 0x000004c0, Clear previous status

Recommended DOR - Register Init for individual Wire Pairs / DOMs

Register	Bits	Comment
DSTAT	3..0	check if cable is plugged
CTRL	7..4	wire pair reset on
CTRL	3..0	wire pair power on
		…1000 ms delay (DOM related)
DSTAT	11..8, 7..4	check for wire_pair_current_off_limits, check for power_on_ready
CTRL	7..4	wire pair reset off
DOMC	15..8	DOM comm. reset on
DOMS	7..0	check for DOM_detected
CTRL	20	if no_DOM, after software time_out clear pending comm. reset
TTSIC	31..24	set interrupt enable bits
RTSIC	31..24	set interrupt enable bits
TCSIC	31..24	set interrupt enable bits
MWAR	31..2	if DMA, set master write address
MRAR	31..2	if DMA, set master read address
MRTC	26..24, 19..2	if DMA, set master read transfer count
MWTC	26..24, 19..2	if DMA, set master write transfer count
INTEN	16,13,12,10..6,2	set interrupt enable bits

DOR, FPGA – 32 Bit Register Map

Interface Control & Status Registers – Read / Write Function, Overview

Resulting **I/O or Memory_ Address = (Base_Address _0 & 0xfffffffc) + Offset**

Access: 32 bit **always**

Nr. # (dec.)	Name	Offset (hex)	Description	PCI FPGA	Comm FPGA
00	**CTRL**	00	Control Reg.	x	x
01	**GSTAT**	04	Global Status Reg.	x	x
02	**DSTAT**	08	Detailed Status Reg.	x	x
03	**TTSIC**	0c	Tx Transfer Status & Interrupt Control reg		x
04	**RTSIC**	10	Rx Transfer Status & Interrupt Control reg		x
05	**INTEN**	14	Interrupt Enable Reg.	x	x
06	**DOMS**	18	DOM Status Register		x
07	**MRAR**	1c	Master Read Address Register	x	
08	**MRTC**	20	Master Read Transfer Count	x	x
09	**MWAR**	24	Master Write Address Register	x	
10	**MWTC**	28	Master Write Transfer Count	x	x
11	**UTCRD0**	2c	UTC (Universal Time Coordinated) Read #0		x
12	**UTCRD1**	30	UTC (Universal Time Coordinated) Read #1		x
13	**CURL**	34	Current / Voltage Limits		x
14	**DCUR**	38	Dom Current		x
15	**FLASH**	3c	FLASH memory Address / Data Register	x	
16	**MBF0**	40	Message data buffer, DOM_0		x
17	**MBF1**	44	Message data buffer, DOM_1		x
18	**MBF2**	48	Message data buffer, DOM_2		x
19	**MBF3**	4c	Message data buffer, DOM_3		x
20	**MBF4**	50	Message data buffer, DOM_4, DOR_rev1-only		x
21	**MBF5**	54	Message data buffer, DOM_5, DOR_rev1-only		x
22	**MBF6**	58	Message data buffer, DOM_6, DOR_rev1-only		x
23	**MBF7**	5c	Message data buffer, DOM_7, DOR_rev1-only		x
24	**DOMC**	60	DOM Control register		x
25	**CERR**	64	Communication (Rx) Error counter		x
26	**TCSIC**	68	TCAL Status & Interrupt Control reg.		x
27	**TCBUF**	6c	TCAL data Buffer		x
28	**TSTRG**	70	Time String buffer, 11x22bytes		x
29	**DOMID**	74	DOM-ID reg.		x
30	**DCREV**	78	Supported DOM Comm. Module Revision		x
31	**FREV**	7c	Firmware Revision	x	x
Shadow Register for Test & Debugging					
03	**DBCS**	0c	Debug Status & Control Register		x
04	**CDLT**	10	Communaction DAC Look up Table		x

06	**WFRB**	18	Waveform Read Buffer		x
11	**STMF**	2c	Statemachine log FIFO, 1kx32, (wire pair #0)		x

Interface Control & Status Registers – Detailed Description

RW	Read / Write
RWC	Read / Write-Clear, writing a 1 clears the status bit
RO	Read Only
WO	Write Only
RWSC	Read / Write / Self Clear, gets 0 when ready (poll function)

#0 **CTRL** @ 00h Control register **power on state**: C000:0000h

Bit	Signal	Attribute	Description
0..3	WP_PON_0..3	RW	Wire pair power on/off bits, poll on WP_PON_RDY_0..3 (DSTAT) to get the status,
4..7	WP_RES_0..3	RW	Wire pair communication modules reset signal
8	RETRANS_EN	RW	packet retransmit by firmware, only if in 8B/10B mode
9	TIMER_CLEAR	RW	The 0-1 transition clears the 64 bit DOR timer
10	TIMER_SNAP	RW	The 0-1 transition cause a snapshot of the 64 bit DOR timer, The value can be read from UTCRD0/1. Needed because the timer is not synchronous to the PCI clock,
11	ENC_8B10B_DIS	RW	8b10b encoding-disable, default is off ! if **NOT** set, the firmware trys to run at 2 Mbaud speed, using an 8b10b enoding schema.
12..14	TCBUF_SEL_0..2	RW	To select one of the TCAL data buffers 0..7**, mirrored by TCSIC bits 8..10 (!!!)**
15	LEV_ADAPT_DIS	RW	comm. signal level adaption – disable. The comm. level adaption only effective with the DOM being in ICEBOOT.
16..19	x	RW	Unused
20	DEBUG_LEDS_EN	RW	switches the data transfer monitoring LEDs into software debug mode, see reg. #31 FREV_write
21	CPAR_RD_EN	RW	default is off ! when set, the comm. parameters can get read back from reg. #30,#31
22	DBG_REG_SEL	RW	Debug (Shadow) Register Space Select
23	GLOBAL_RES	RW	resets the comm. FPGA, Select the clock source (CTRL bit 31) first. After reloading the comm. FPGA this bit should be toggled not earlier than 20 µs after CFG_DONE (GSTAT-bit 2) went high. IMPORTANT !!! After switching off the bit the reset is still active for about 500ns. A delay should be inserted here.
24..25	FPGA_IMAGE_0..1	RW	FPGA firmware version (FLASH page) select bits, to be used together with FPGA_RELOAD
26	FPGA_RELOAD	WO	Initiates the FPGA reload. The (previously saved) PCI config space has to be restored after the reload operation !
27	MR_RD_EN	RW	Master (DMA) Read Enable Bus master view, PC_memory -> DOR_buffer Writing a 0 while running DMA Read cancels the operation
28	MR_WR_EN	RW	Master (DMA) Write Enable Bus master view, DOR_buffer-> PC_memory Writing a 0 while running DMA Write cancels the operation
29	MEM_RD_MULT	RW	Memory Read Multiple used for Master Read operations
30	COM_TOUT_EN	RW	Communication Time Out Enable, time out occurs if DOM_x is not responding after 16 sec. (256 retrials) Default (and after GLOBAL_RES) is on.
31	LOCAL_OSC_EN	RW	Local Oscillator select instead of external clock. IMPORTANT !!! Wait at least 10 ms, required by the PLL to settle down.

Rem.: bits 23..29, 31 are located in the PCI-control FPGA

#1 GSTAT @ **04h** Global Status Register **power on state:** 0000:01C4**h**

Bit	Signal	Attribute	Description
0	CFG_ERR	RO	Comm. Controller FPGA configuration (Flash page 0..2) failed due to HW problems or a corrupt Flash page. DOR_rev1 – only !!!
1	CFG_DONE	RO	Comm. Controller FPGA configuration (Flash page 0..2) ready, DOR_rev1 – only !!!
2	CBL_STAT_CHD	RWC	Cable Status Changed, at least one cable has been plugged or unplugged
3	WP_POW_FAILED	RWC	At least one wire pair has been powered on and the current / voltage is beyond the limits (see CURL reg.) or a cable has been unplugged while under power. Check DSTAT reg. for details DOR_rev1 – only !!!
4	HW_TIMEOUT	RO	Or of all eight possible DOM hardware timeouts, see DOMS
5	0	RO	Unused
6	MR_WR_TC	RO	Master (DMA) Write Transfer Complete
7	MR_RD_TC	RO	Master (DMA) Read Transfer Complete
8	TXBFn_EF	RO	TX Buffer_n Empty Flag, an OR of all DOM message buffer empty flags
9	RXBFn_MRCVD	RO	RX Buffer_n Message Received, At least one DOM message buffer RXBFn is containing a complete message
10	TCALn_RDY	RWC	The time calibration for a selected DOM (see CTRL reg.) is ready, the time calibration data can be taken from TCBUF. For status you can also use the DOMC reg., bits 24..31.
11	PPS_DET	RWC	Pulse Per Second Detected, for synchronization and test purposes. Write a “1” clears the bit.
12	TSTRG_10x	RO	Ten time strings are available for readout.
13	TSTRG_1x	RO	One time string (at least) is available for readout.
14	TSTRG_F_FF	RO	Time String Fifo Full Flag. Eleven time strings are available for readout.
15	0	RO	Unused
16	GLOBAL_INT	RO	This is an OR of all enabled interrupt sources (see INTEN reg.), gets cleared by clearing the interrupt causing status bits or by disabling the appropriated interrupt-enable bits. If GLOBAL_INT_EN (INTEN_bit16) is set an interrupt will occur.
17..23	0	RO	unused
24	PPS_MISSED	RWC	Pulse Per Second missed . Write a “1” clears the bit.
25	SOH_MISSED	RWC	SOH (GPS time string) missed . Write a “1” clears the bit.
26..31	0	RO	unused

#2 DSTAT @ 08h **Detailed Status Register** power on state: **0000:0003**h

Bit	Signal	Attribute	Description
0..3	CBL_PLGD_0..3	RO	Cable (wire pair) 0..3 has been plugged
4..7	WP_PON_RDY_0..3	RO	wire pair_0..3, power on **OR** off is ready. **AND** these bits with CTRL bits 0..3 to know about the power on / off status
8..11	WP_UNPLGD_0..3	RWC	wire pair_0..3, cable was unplugged under power
12..15	CARD_ID_0..3	RO	Read back of the CARD_ID hex switch, IMPORTANT !!! Don't do this while programming the FLASH !!!
16..19	WP_CUR_BL_0..3	RWC	wire pair_0..3, current is BELOW the limits, the current was smaller than the programmed value, see CURL reg.#13
20..23	WP_CUR_AL_0..3	RWC	wire pair_0..3, current is ABOVE the limits, the current was bigger than the programmed value, see CURL reg.#13
24..27	WP_VOLT_BL_0..3	RWC	wire pair_0..3, voltage is BELOW the limits, the voltage was smaller than the programmed value, see CURL reg.#13
28..31	WP_VOLT_AL_0..3	RWC	wire pair_0..3, voltage is ABOVE the limits, the voltage was bigger than the programmed value, see CURL reg.#13

#3 TTSIC @ 0ch **Tx Transfer Status & Interrupt Ctrl. Reg.** power on state: **0000:0F0F**h

Bit	Signal	Attribute	Description
0..7	TX_EF_0..7	RO	Tx buffer empty flag of DOM_n At least 250 32 bit words can be written to. These bits cause an **interrupt**, if the appropriate interrupt enables bits were set
8..15	TX_AEF_0..7	RO	Tx buffer almost empty flag of DOM_n, Max. 150 32 bit words can be written to,
16..23	TX_AFF_0..7	RO	Tx buffer almost full flag of DOM_n. At least 252 32bit words are in the Tx buffer, ,
24..31	TX_ INT_EN_0..7	RW	Tx interrupt enable for DOM_n to get an interrupt when TX_EF_x = 1

#4 RTSIC @ 10h **Rx Transfer Status & Interrupt Control Reg.** power on state: **0000:000F**h

Bit	Signal	Attribute	Description
0..7	RX_EF_0..7	RO	Rx buffer empty flag of DOM_n
8..15	RX_AFF_0..7	RO	Rx buffer almost full flag of DOM_n, Rx buffer of DOM_n contains 103..256 32bit words,
16..23	RX_MRCVD_0..7	RWC	At least one complete Message has been received, coming from DOM_n, these bits cause an **interrupt**, if the appropriate interrupt enables bits were set. Every Rx buffer has a message-received-counter. The incoming message does a counting up. Writing a one to RX_MRCVD_x (software acknowledge) is **needed** to count down.. As long as the counter is not zero the bit RX_MRCVD_x remains set.
24..31	RX_INT_EN_0..7	RW	Rx interrupt enable for DOM_n to get an interrupt when RX_MRCVD_x = 1

#5 INTEN @ 14h **Interrupt Enable Register** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0	X	RW	reserved
1	X	RW	reserved
2	INT_ON_CBL_STAT	RW	Cable status changed, at least one cable has been plugged or unplugged
3	INT_ON_WP_POW_FAILED	RW	at least one of the power switch error conditions is true, see DSTAT bits 8..11, 16..31, implemented for **com_102j** and later
4	INT_ON_HW_TIMEOUT	RW	Interrupt on hardware timeout
5	X	RW	reserved
6	INT_ON_MR_WR_TC	RW	Master (DMA) Write Transfer Complete
7	INT_ON_MR_RD_TC	RW	Master (DMA) Read Transfer Complete
8	INT_ON_TXBFn_EF	RW	TX Buffer_n Empty Flag, At least one DOM message buffer TXBFn is empty. While this is a global Tx-Interrupt enable bit, individual enables are available at the TTSIC reg..
9	INT_ON_RXBFn_MRCVD	RW	RX Buffer_n Message Received Flag At least one DOM message buffer RXBFn is containing a complete message. While this is a global Rx-Interrupt enable bit, individual enables are available at the RTSIC reg..
10	INT_ON_TCALn_RDY	RW	The time calibration for a selected DOM (see TCSIC reg.) is ready, the time calibration data can be taken from TCBUF, addressed by CTRL_bit14..12
11	x	RW	reserved
12	INT_ON_TSTRG_10x	RW	Ten time strings can get read out now.
13	INT_ON_TSTRG_1x	RW	One time string can get read out now.
14..15	x	RW	Reserved
16	GLOBAL_INT_EN	RW	If set and GLOBAL_INT (GSTAT_bit16) is on an interrupt will occur.
17..31	x	RW	reserved

#6 DOMS @ 18h **DOM Status register** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..7	DOM_DET_0..7	RO	DOM 0..7 detected
8..15	DOM_TOUT_0..7	RWC	Hardware timeout detected, DOM 0..7 were not responding within 4.1 sec, gets cleared as well by DOMC-bits 7..0 or 15..8
16..23	DOM_BFULL_0..7	RO	DOM 0..7 Rx Buffer has filled up,
24..31	DOM_NOT_CFG_BOOT_0..7	RO	DOM 0..7, DOM are not in CONFIGBOOT.

#7 MRAR @ 1Ch **Master Read Address Register** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0	0	RO	32 bit boundary required
1	0	RO	
2..31	MRAR_2..31	RW	Used for Bus Master (DMA) Read operations, Memory buffer -> DOM message buffer, Points to the first 32 bit memory word to be transferred to the selected (MRTC reg. bit 24…26) DOM message buffer.

#8 MRTC @ 20h **Master Read Transfer Count** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..1	MRTC_1..0	RW	Can be used, but a 4 byte boundary is preferred
2..19	MRTC_2..19	RW	Used for Bus Master (DMA) Read operations, Memory buffer -> DOM message buffer, Amount of 32 bit memory words to be transferred to the selected (bit 24…26) DOM message buffer.
20..23	0	RO	Reserved
24..26	RXBF_0..2	RW	DOM message buffer number the transfer is associated with
27..31	0	RO	Reserved

#9 MWAR @ 24h **Master Write Address Register** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0	0	RO	32 bit boundary required
1	0	RO	
2..31	MWAR_2..31	RW	Used for Bus Master (DMA) Write operations, DOM message buffer -> Memory buffer , 32 bit memory address, the data from the selected DOM (MWTC reg., bit24…26) message buffer have to be written to

#10 MWTC @ 28h **Master Write Transfer Count** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0	0	RO	32 bit boundary required
1	0	RO	
2..19	MWTC_2..19	RW	Used for Bus Master (DMA) Write operations, DOM message buffer -> Memory buffer , Amount of 32 bit memory words to be transferred from the selected (bit24…26) DOM message buffer.
20..23	0	RO	Reserved
24..26	TXBF_0..2	RW	DOM message buffer number the transfer is associated with
27..31	0	RO	Reserved

#11 UTCRD0 @ 2Ch **UTC Read #0** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..31	UTC_TIME_0..31 3)	RO	DOR Timer (later UTC time), lower 32 bits,50ns ticks. CTRL reg. bit10 (TIMER_SNAP) must be used before.

#12 UTCRD1 @ 30h **UTC Read #1** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..31	UTC_TIME_32..63 3)	RO	DOR Timer (later UTC time), upper 32 bits,50ns ticks

3) counts up immediately after FPGA load

#13 CURL @ 34h **Current Limits** power on state: **E19E:8C14**h

Bit	Signal	Attribute	Description
0..7	CUR_MIN_0..7	RW	DOM power, minimum current in mA, When DOMs are powered on and the current is smaller, an status bit get set. See DSTAT reg. and GSTAT reg.. The appropriate channel gets shut down. recommended range 20mA…25mA, default value = 20mA,
8..15	CUR_MAX_0..7	RW	DOM power, maximum current in mA, When DOMs are powered on and the current is bigger, an status bit get set. See DSTAT reg. and GSTAT reg.. The appropriate channel gets shut down. recommended range 100mA…200mA, default value = 200mA, IMPORTANT !!!! A short term (max 100sec) of up to 255mA is allowed. **But permanently using more than 200mA could damage the inductances L100, L101, L200 .. L401 !**
16..23	VOLT_MIN_0..7	RW	DOM power, minimum voltage * 2.25 in V, When DOMs are powered on and the voltage is lower, an status bit get set. See DSTAT reg. and GSTAT reg.. The appropriate channel gets shut down. recommended range 60V…70V, default value = 70V,
24..31	VOLT_MAX_0..7	RW	DOM power, maximum voltage * 2.25 in V, When DOMs are powered on and the voltage is bigger, an status bit get set. See DSTAT reg. and GSTAT reg.. The appropriate channel gets shut down. recommended range 96V…110V, default value = 100V,

#14 DCUR @ 38h **DOM Current** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..11	DCUR _VAL_0..11	RO	DOM (wire pair) current value current (mA) = DCUR_VAL_11..0 / 2 Use the bit12…13 to select the wire pair
12..13	DCUR_SEL_0..1	RW	Wire pair, the current to measure from
14	0	RO	unused
15	0	RO	unused
16..27	DVOL _VAL_0..11	RO	DOM voltage value voltage (V) = DVOL_VAL_11..0 / 36 (exactly 35.64) Use the bit28..29 to select the wire pair
28..29	DVOL_SEL_0..1	RW	Wire pair, the voltage to measure from
30..31	0	RO	unused

#15 FLASH @ 3Ch **FLASH access register** power on state: **xx00:0000**h

Bit	Signal	Attribute	Description
0	FL_ADR _-1	RW	FLASH address -1, (identical to DQ15)
1..19	FL_ADR _0..18	RW	FLASH address bus
20	FL_WE	WO	FLASH write enable, read back value is zero, if set, FL_DATA_0…7 are written to FL_ADR-1…18
21	*FL_ADR_19*	RW	DOR_rev1 - only
22	FL_RESET	RW	FLASH reset is on if FL_RESET==1
23	FL_RESET_VID	RW	Enforces +12V (VID) at the reset pin, if FL_RESET==0. This bit allows sector protect / unprotect operations, if jumper J3 is set.
24..31	FL_DATA_0..7	RW	FLASH data bus, accessing this register set the FLASH always into byte mode. A write operation takes one cycle; a read operation might need two cycles, if the address has to be written before.

#16…19 MBF0…3 @ 40..4Ch **Rx/Tx Message Buffer 0…3** power on state: **00000000**h

Bit	Signal	Attribute	Description
0..31	MBFn _0..31	R/W	Read selects the Rx FIFO of DOM_n, Write selects the Tx FIFO of DOM_n,

#20…23 MBF0…7 @ 50..5Ch **Rx/Tx Message Buffer 4…7** power on state: **00000000**h

Bit	Signal	Attribute	Description
0..31	MBFn _0..31	R/W	Reserved, not yet implemented.

#24 **DOMC** @ 60h **DOM Control** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..7	DOM_SYS_RES_0..7	RW1O	DOM Mainboard **System Reset** (Softreset). Read / Write_1_Only (cannot be cleared by writing a zero) Get blocked if the DOM is in CONFIGBOOT, see DOMS bits 24..31. The appropriate Tx-and RxFIFO of MBF_n get cleared as well.When ready the DOM replys with an IDLE command. Gets cleared by: IDLE command received, Appropriate WP_RES_x, see CTRL reg. 4..7, DOM_COM_RES_x,
8..15	DOM_COM_RES_0..7	RW1O	DOM Mainboard **Comunication Module Reset**. Read / Write_1_Only (cannot be cleared by writing a zero) The appropriate Tx-and RxFIFO of MBF_n get cleared as well.When ready the DOM replys with an IDLE command. Gets cleared by: IDLE command received, Appropriate WP_RES_x, see CTRL reg. 4..7, **Hardware timout** (after 4.1 sec.)
16..23	DOM_ID_REQ_0..7	RW1O	DOM Mainboard **ID Request** Read / Write_1_Only (cannot be cleared by writing a zero) When ready the appropriate DOMID reg. contains the 6 byte DOM mainboard ID. Gets cleared by: DOM-ID received, Appropriate WP_RES_x, see CTRL reg. 4..7, DOM_SYS_RES_x. DOM_COM_RES_x,
24..31	DOM_TCAL_REQ_0..7	RW1O	**TCAL Request** Read / Write_1_Only (cannot be cleared by writing a zero) To enforce a time calibration. When ready the appropriate TCAL buffer (TCBUF) contains the TCAL data packet. Get blocked if the DOM is in CONFIGBOOT, see DOMS bits 24..31. Gets cleared by: TCAL packet received, Appropriate WP_RES_x, see CTRL reg. 4..7, DOM_SYS_RES_x. DOM_COM_RES_x,

#25 CERR @ 64h **Communication Errors** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..15	COMM_ERR_0..15	RO	Amount of received corrupt control or data messages. Get cleared by: - CERR_29..31, DOMC_0..7, DOMC_8..15, CTRL_4..7
16..27	COMM_DEBUG_0..12	RO	dependend on the firmware revision, debugging information per DOM
28	DOMb_notDOMa	RW	DOM select, '0' = DOMa, '1'=DOMb
29..30	WIRE_SEL_0..1	RW	To select wire pairs 0..3
31	COMM_ERR_CLR	WO	Communication Error Clear for the selected DOM (bit30..28), a single write clears the selected error counter

#26 TCSIC @ 68h **TCAL Status & Interrupt Ctrl. Reg.** power on state: **0000:000F**h

Bit	Signal	Attribute	Description
0..7	TC_EF_0..7	RO	TCAL buffer empty flag of DOM_n
8..10	TC_BUF_SEL_0..2	RW	to select the TCAL data buffer, **mirrored (!!!) CTRL bits 12..14**
11	TC_BUF_SEL_ENA	RW	if set TC_BUF_SEL_0..2 are being used, else CTRL bits 12..14 are valid
12..15	0	RO	unused
16..23	TC_PRCVD_0..7	RWC	TCAL Packet Received for DOM_n
24..31	TC_ INT_EN_0..7	RW	TCAL interrupt enable for DOM_n

#27 TCBUF @ 6Ch **TCAL data Buffer** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..31	TC_DATA _0..31	RO	TCAL data, see format description, select the DOM by using CTRL reg., bits 14..12

#28 TSTRG @ 70h **Time String buffer** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..7	TSTRG_F_DATA[0..7]	RO	Time String Fifo Data
24..31	0	RO	unused

#29 DOMID @ 74h **DOM-ID Register** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..7	DOM_ID_0..7	RO	DOM-ID, byte0 / 3
8..15	DOM_ID_8..15	RO	DOM-ID, byte1 / 4
16..23	DOM_ID_16..23	RO	DOM-ID, byte2 / 5
24 25..27	DOM_ID_UPPER DOM_0..7	RW RW	Bits 0..23 represent the Upper 3 bytes, else lower… DOM_0..7, the ID is read from
28..31	0	RO	unused

#30 DCREV_read @ 78 **DOM Comm. Module Revision** power on state: **0000:xxxx**h

Bit	Signal	Attribute	Description
			if CTRL_bit21=0 (CPAR_RD_EN): DCOM firmware rev., if CTRL_bit21=1 (CPAR_RD_EN): DOR comm. parameters, see below
0..15	DC_REV_0..15	RW	16 bit number, this revision must be the same used at the DOM;
16..31	0	RO	unused

#30 DCREV_write @ 78 **DOR Comm. Parameters**

Bit	Signal	Attribute	Description
0..7	COMM_THRESH_0..7	WO	length of the lowgoing edge of the comm. Signal after 4 clocks (200ns) in multiples of 0.4 mV (cable signal) or 2mV (comm. ADC input); Power up value dependend on the firmware revision;
12..13	DAC_MAX_0..1	WO	0…3, initial comm_DAC_amplitude = 128 + dac_max[]*32 + 31 ; can get changed if CTRL_bit15 = 1; Power up value dependend on the firmware revision,;
16..23	REC_DELAY_0..7	WO	1..255, multiples of 50ns to fade out reflections from the last DOR-send, seen by the DOR receiver stage; receiver_enable_delay = shortest_cable_length (m) / 5 + 10 Power up value dependend on the firmware revision;
24..31	SEND_DELAY_0..7	WO	1..255, multiples of 50ns to fade out reflections from the last DOM-send, seen by the DOM receiver stage; Power up value dependend on the firmware revision;

#31 FREV_read @ 7c **DOR Firmware Revision** power on state: **xxxx:xxxx**h

Bit	Signal	Attribute	Description
			if CTRL_bit21=0 (CPAR_RD_EN): firmware revsion, if CTRL_bit21=1 (CPAR_RD_EN): bit9..0= CLEV_MIN, bit21..12 (!!!)=CLEV_MAX
0..7	CHAR_TAG_ID_0..7	RO	Character tag
8..15	SUB_MINOR_ID_0..7	RO	0..99
16..23	MINOR_ID_0..7	RO	0..99
24..31	PCI_CTRL_FREV_0..7	RO	0..255, these bits are located in the PCI_CTRL FPGA

#31 FREV_write @ 7c **Limits for the comm. level adaption** power on state: **xxxx:xxxx**h

Bit	Signal	Attribute	Description
0..9	CLEV_MIN_0..9	WO	Lower limit, gets compared to the comm. ADC output, typical value = 800..960; Power up value dependend on the firmware revision,
10..11		RO	unused
12	DEBUG_LEDS_0	WO	first row, green LED (wire pair #0 range)
13	DEBUG_LEDS_1	WO	first row, yellow LED
14	DEBUG_LEDS_2	WO	second row, green LED
15	DEBUG_LEDS_3	WO	second row, green LED
16..25	CLEV_MAX_0..9	WO	Upper limit, gets compared to the comm. ADC output, typical value = 810..970; Power up value dependend on the firmware revision,
26..17		RO	unused
28	DEBUG_LEDS_4	WO	third row, green LED (wire pair #0 range)
29	DEBUG_LEDS_5	WO	third row, yellow LED
30	DEBUG_LEDS_6	WO	fourth row, green LED
31	DEBUG_LEDS_7	WO	fourth row, green LED

Debug (Shadow) Register Space, for accessing Ctrl-bit22 must be set

#3 DBCS @ 0ch **Debug Control & Status** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0	CDLT_GO	RWSC	1K Byte comm. DAC look up table burst start, ready when bit returned to zero
1	CDLT_RES	RW	Resets the look up table write-address and read-address counters
2..3	0	RO	unused
4..5	CDWP_SEL_0..1	RW	Comm. DAC (wire pair = = noise channnel) select,
6..7	0	RO	unused
8	WF_RECORD	RWSC	Starts the waveform recorder. Gets cleared if the waveform read buffer is full. The waveform buffer address **gets reset automatically** before the recording starts. Don't try to read, before WF_RECORD went low !
9	0	RO	reserved
10	WF_ADR_RES	RW	Resets the waveform buffer address counter
11	WF_SEL	RW	Waveform select, 0 -> wire pair #0,1; 1-> wire pair #2,3
12	0	RO	Reserved
13..15	0	RO	Reserved
16	STM_FIFO_EF	RO	wire pair #0 states recording Fifo is empty
17	STM_FIFO_FF	RO	wire pair #0 states recording Fifo is full, 1000 words can get read
18	STM_FIFO_CLR	RW	clears wire pair #0 states recording Fifo
19	0	RO	Reserved
20	STM_ADV_ON_WP	RW	recording if wire pair control state has been changed
21	STM_ADV_ON_RX	RW	recording if rx buffer write control state has been changed
22	STM_HLT_ON_TOUT_A	RW	recording stops, if DOM_A timeout has been detected
23	STM_HLT_ON_TOUT_B	RW	recording stops, if DOM_B timeout has been detected
24..31	0	RO	Reserved

#4 CDLT @ 10h **Communication DAC Look up Table** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..7	CDLT_DATA_7..0	RW	1K Byte comm. DAC look up table
31..8	0	RO	unused

#6 WFRB @ 18h **Waveform Read Buffer** power on state: **0000:0000**h

Bit	Signal	Attribute	Description
0..9	WP0_WF_9..0	RO	Wire pair #0 or # 2, 256Kx10bit recorded comm. ADC waveform, the read address counter gets incremented per read
10..15	0	RO	unused
16..25	WP1_WF_9..0	RO	Wire pair #1 or # 3, 256Kx10bit recorded comm. ADC waveform
26..31	0	RO	unused

#11 **STMF** @ 2ch **State Machine Fifo** power on state: **0000:0000**h

Depending on bits 20..23 of DBCS the states of the wire pair #0 control state machine and / or the states of the Rx buffer- write control state machine get recorded continously. The recording stops in case of a selected halt condition (see DBCS).

Bit	Signal	Attribute	State Name	binary	decimal
2..0	WP_CTRL_SV_2..0	RO	POFFA	001	1
			A_AVAL	000	0
			CLR_CRESA	100	4
			CRES_RDYA	010	2
			CRESA	110	6
			TOUT_A	101	5
7..3	WP_CTRL_SV_7..3	RO	IDL_AB	00010	2
			CRESACKWT	00000	0
			CRESET	10000	16
			DBUFCHK	01000	8
			DBUFWT	11000	24
			DRBT_CRES	00100	4
			DRBT_IDLE_WT	10100	20
			ID_REC	01100	12
			ID_REQ	11100	28
			RACKWT	10010	18
			RDREQ	01010	10
			REC_IDLE	11010	26
			SND_IDLE	00110	6
			SRES_WT	10110	22
			SRESET	01110	14
			TC_RDAT	11110	30
			TC_WAIT1	00001	1
			TCALIB	10001	17
			TCPULSND	01001	9
			TCPULSREC	11001	25
			TCWFMCPY	00101	5
			WACKWT	10101	21
			WDAT	01101	13
8	WP_CTRL_SV_8	RO	DOMBUF_A	0	0
			NODOMBUF_A	1	1
9	WP_CTRL_SV_9	RO	DOMBUF_B	0	0
			NODOMBUF_B	1	1
12..10	WP_CTRL_SV_12..10	RO	PRIO_A	000	0
			PRIO_B	100	4
			RUN_A	010	2
			RUN_B	110	6
			SEL_A	001	1
			SEL_B	101	5
15..13	WP_CTRL_SV_15..13	RO	POFFB	001	1
			B_AVAL	000	0
			CLR_CRESB	100	4
			CRES_RDYB	010	2
			CRESB	110	6
			TOUT_B	101	5
17..16	WP_CTRL_SV_17..16	RO	WRA	10	2
			RDA	00	0
			WTA	01	1

19..18	WP_CTRL_SV_19..18	RO	WRB	10	2
			RDB	00	0
			WTB	01	1
24..20	RX_FWR_SV_4..0	RO	BYTE0	00000	0
			BYTE0_TCAL	10000	16
			BYTE1	01000	8
			BYTE2	11000	24
			BYTE3	00100	4
			CLR_BYTES	10100	20
			CPY_BYTE0	01100	12
			CTRL_EOF_WT	11100	28
			CTRL_OK	00010	2
			DAT_OK	10010	18
			DCMD_SEQ1	01010	10
			LAT_WADR	11010	26
			LEN0	00110	6
			MTYPE_LEN1	10110	22
			PIDL	01110	14
			PTYPE_SEQ0	11110	30
			RxTIME_H	00001	1
			RxTIME_L	10001	17
			RxWFM_CPY	01001	9
			RxWFM_WT	11001	25
			STF_WAIT	00101	5
			TCAL_HDR	10101	21
			TCWF_CLR,	01101	13
			TxTIME_H	11101	29
			TxTIME_L	00011	3
			WFM_RDY	10011	19
			WR_HOLD	01011	11
			WRITE	11011	27
26..25	RX_FWR _SV_6..5	RO	REC_DOM_ID	01	1
			DOM_ID0	00	0
			DOM_ID1	10	2
30..26	RX_FWR _SV_10..7	RO	REC_IDLE	1110	14
			CTR_ERR	0000	0
			CTR_MSG	1000	8
			DAT_ERR	0100	4
			DAT_MSG	1100	12
			DATA_OK	0010	2
			ID_ERR	1010	10
			ID_MSG	0110	6
			SEQ1	0001	1
			TCAL_ERR	1001	9
			TCAL_MSG	0101	5

Revision History

Revision	Changes
2.8	Reg. #0 (**CTRL**), bit 11=DCUR_CONT, removed. Current / voltage get measured ALWAYS now Reg. #1 (**GSTAT**), bit 3, new Reg. #2 (**DSTAT**), bits 8..11, 16..31 new / changed Reg. #13 (**CURL**), in use now, set max./min. current and max./min. voltage Reg. #14 (**DCUR**), bit 15=START_notRDY, removed, not needed anymore Reg. #31 (**FREV**), bits 31..24 = pci control FPGA revision
2.9	Reg. #14 (**DCUR**), document correction, it is NOT a Fifo
3.0	Reg. #0 (**CTRL**), bit 23=GLOBAL_RES, new (pci_009 and later) Reg. #5 (**INTEN**), bit 3, new (com_102j and later)
3.1	Reg. #26 (**TCSIC**), bits 8..11 new defined, TCSIC bits 8..11 = mirrored CTRL bits 12..14
3.2	Reg. #1 (**GSTAT**), bits 24,25 new defined Reg. #25 (**CERR**), bits 16..27 new defined
3.3	see figure 3, TCAL cycle symplified, extra DRREQ within the TCAL cycle removed, Reg. #0 (**CTRL**) new: bit 11 = DBG_REG_SEL Reg. #1 (**GSTAT**) new: bit 4 = HW_TIMEOUT, bit 5 = DOM_REBOOT Reg. #5 (**INTEN**) new: bit 4 = INT_ON_HW_TIMEOUT, bit 5 = INT_ON_DOM_REBOOT Reg. #6 (**DOMS**) bits 8..15 RWC now, bits 24..31 DOM_REBOOT now
3.4	Reg. #6 (**DOMS**) bits 24..31 = DOM_NOT_CFG_BOOT_0..7 now Reg. #24 (**DOMC**) check description of bits 0..7, 24..31 Reg. #1 (**GSTAT**), bit 5 = DOM_REBOOT removed Reg. #5 (**INTEN**), bit 5 = INT_ON_DOM_REBOOT removed for upgoing messages headers, the sequence number field bit12 = NOT_CONFIGBOOT now Reg. #25 (**CERR**) , see description Shadow register added (firmware rev. 104p), see Reg. #11 (**STMF**) , see Reg. #3 (**DBCS**), new bits 16..23 see Figure 5, TCAL – Rx time sample point description changed: **"…Rx_time sample point at ADC[n] > ADC[n-1] "** was " **>= "** before
3.4a	Reg. #0 (**CTRL**) new: bit 11 = ENC_8B10B_DIS; bit 21 = CPAR_RD_EN