Solar Orbiter
HET/EPT Digital FPGA

Stephan I. Böttcher

SVN Revision 8147
Date 2020-12-03 21:53:06 +0100 (Do, 03 Dez 2020)
Document code: SO-EPD-KIE-DA-0001 Issue 1 Rev 0.3

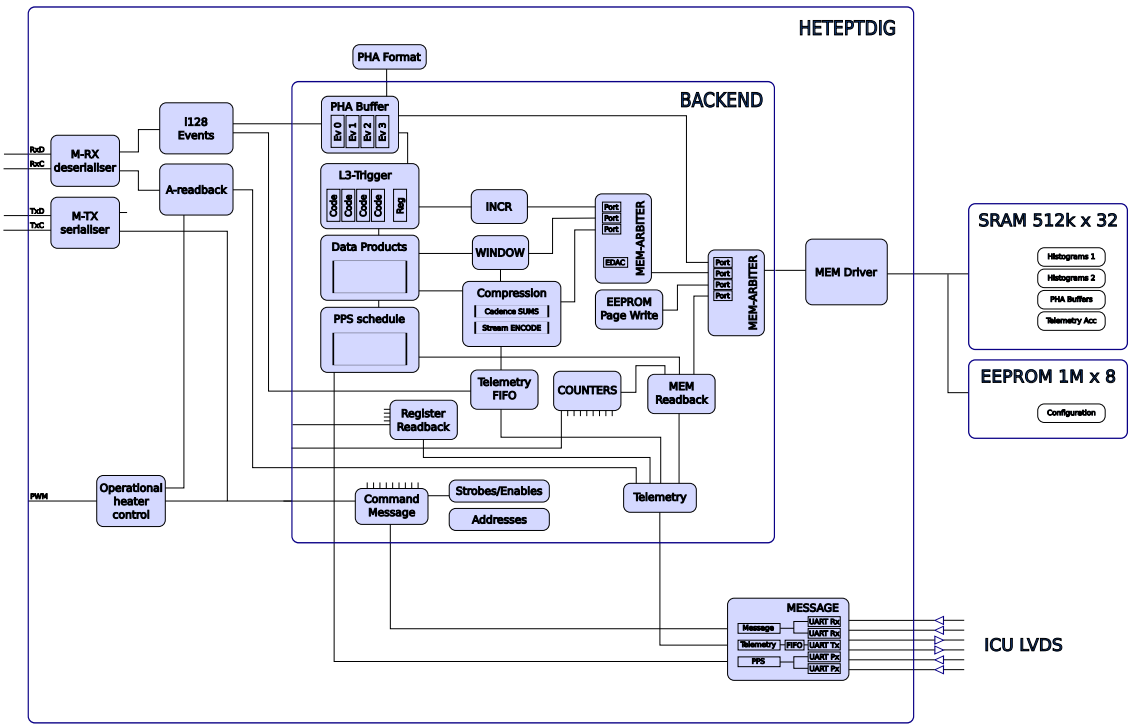
Contents

List of Tables

List of Figures

1 Introduction

The HET/EPT sensor employs two FPGAs, one each on the analog and the digital boards. The FPGA on the analog board will control the ADCs, perform digital filtering, L1 and L2 trigger processing, and housekeeping acquisition. The digital board FPGA is described in this document.



2 Mode of Operation

The operations of the FPGA are grouped in three threads of execution:

1. data acquisition,
2. data checkout, and
3. slow control.

All functions are controlled via command messages and a *Pulse Per Second* (PPS) signal. Both command messages and PPS signals can be received from the ICU and generated autonomously in the FPGA.

During nominal operation, the PPS is received once per second from the ICU and the necessary command messages are generated inside the FPGA synchronized to the PPS. No command messages from the ICU are required during nominal operation.

Command messages for configuration and slow control need to be send by the ICU via LVDS serial communication.

2.1 Data Acquisition

Data acquisition involves

1. reception of event data packets from the analog FPGA,
2. buffering (up to four packets),
3. L3 trigger processing,
4. histogram counting, and
5. PHA data storage.

Histograms and PHA data are stored in the external SRAM. The storage areas are double buffered, one buffer for acquisition and one being checked out for telemetry.

Once configured and enabled, the data acquisition operates completely autonomously.

2.2 Data Checkout

In parallel to the data acquisition, the checkout proceeds from data acquired previously. The execution of the checkout is controlled by a command table in the PPS scheduler. The scheduled tasks include

- starting and stopping data acquisition between acquisition periods,
- switch buffers between acquisitions,
- perform analog housekeeping acquisitions and telemetry,
- send counters and other status housekeeping,
- send PHA data buffers,
- kick the operational heater,
- reconfigure the frontend, to clear out SEU,
- calculate and send histogram data telemetry.

2.3 Slow Control

Once configured and operational, there shall be no or very little commanding of the sensor from the ICU. Tasks that may arise during operation include

- changing trigger parameters,

- initiate diagnostic telemetry, or
- writing new data into the EEPROM.

Writing into configuration memory to tune trigger parameters should be safe at any time as long as the changes do not change the trigger algorithm and leave the trigger operational between all writes. The analog frontend configuration is not normally changed directly, but stored in the PPS scheduler and updated once per second between acquisitions.

Diagnostic telemetry must not be initiated during telemetry checkout, else the data streams collide and garbage is emitted. The PPS schedule shall include spots that are reserved for synchronous actions, like EEPROM page write or diagnostic telemetry. The command to execute the action will be written into reserved slots in PPS schedule table. A further entry in the table will automatically clear the command message from the table after is was emitted, so it will be executed only once.

Writing EEPROM pages is safe as long as the memory port occupation does not interfere. EEPROM access is rather slow and blocks the external memory bus. The execution of the EEPROM page write shall be synchronized to the PPS schedule.

2.4 Command Messages

A command message includes three items:

- an address (14 bit),
- a size tag (2 bits), and
- a data word (64 bits).

The address identifies the recipient of the message. The size tag indicates how many bits shall be interpreted from the data word:

0: zero bits,

1: 16 bits,

2: 32 bits, or

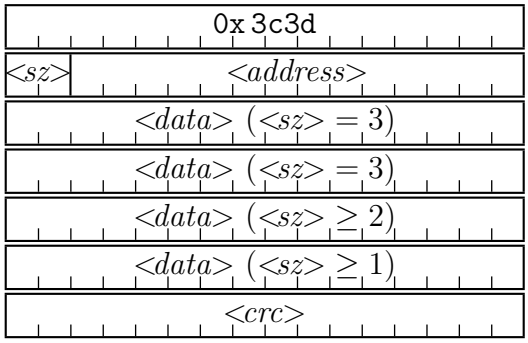
3: 64 bits.

Most recipients ignore the size tag and just accept as much of the data word as they need. Bits beyond the indicated size are expected to be zero.

2.4.1 Serial Commands via LVDS

Messages are received from the ICU via LVDS signaling through a UART operating at 115 200 baud. The serial message format is specific to the Kiel

sensor units:



The CRC is computed according to the following parameters as defined in `http://www.ross.net/crc/download/crc_v3.txt`

WIDTH	= 16
POLY	= `0x 1021`
INIT	= `0x ffff`
REFOUT	= 0
REFIN	= 0
XOROUT	= `0x 0000`

Only as many data words are transmitted as indicated by the <*sz*> tag. The received message data is padded to the left with zeros to fill 64 bits. Word and byte ordering is big endian.

2.4.2 PPS scheduled messages

The PPS scheduler emits messages that are indistinguishable from serially received messages, i.e., the recipients cannot tell where they came from. There is only one difference: the PPS scheduler can emit data words with nonzero bits beyond the indicated size. That should not matter, because those recipients that do look at the size tag ignore the data beyond the indicated size.

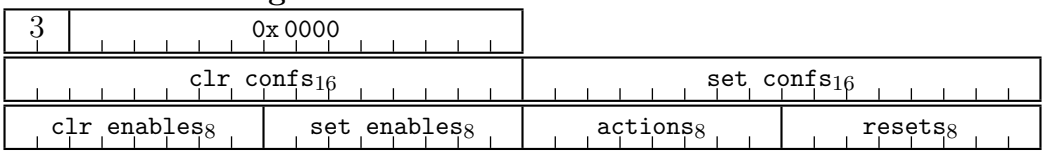
2.5 Master Control Message

A message to address `0x 0000` issues $\texttt{strobes}_{64}$. These are simple command pulses to trigger some action or effect. Most of the `strobes` set or clear $\texttt{enables}_8$ or $\texttt{confs}_{16}$ bits, so these act more like master control registers.

$\texttt{strobes}_{64}$[23:16] set the corresponding bits in $\texttt{enables}_8$. $\texttt{strobes}_{64}$[31:24] clear those bits. A bit that is both to be set and cleared will be toggled.

Similarly, $\texttt{strobes}_{64}$[47:32] set, and $\texttt{strobes}_{64}$[63:48] clear bits in $\texttt{confs}_{16}$. Most bits in $\texttt{confs}_{16}$ are unused.

Command message:

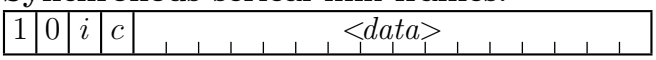


2.6 Link to the Analog Board

Communication between the digital board FPGA and the analog board FPGA goes over synchronous serial links, with a data and a clock signal for each direction, with LVDS signaling. The clock frequency is 48 MHz. The clock towards the analog board is used as the master clock of the analog FPGA.

The serial links employ 20-bit frames to transfer 16-bit words over two channels. Two bits provide a synchronization edge, one bit selects the channel, one bit can be used for line balancing, but here it is just toggled for each frame.

Synchronous serieal link frames:



Bit c is zero for control channel data and one for stream channel data. When no data is to be sent, the link transfers a control message of zero. When the i bit is set, all the remaining bits of the frame are inverted. A stream of control messages zero will include exactly one falling edge per frame, regardless of the i-bit. Link synchronization can always be immediately recovered when the link is idle.

These links provide four channels, two in each direction, which operate all independently.

Control Digital to Analog: Enables and strobes,

Stream Digital to Analog: command messages,

Control Analog to Digital: status and configuration readback, and

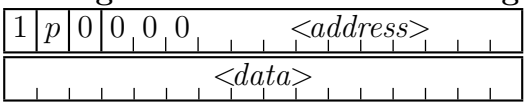
Stream Analog to Digital: data acquisition.

2.6.1 Command Messages to the Analog Board

Command messages can be sent to the analog board FPGA via the stream channel. The messages complies to the IRENA command format, with 14

address bits and none or 16 data bits.

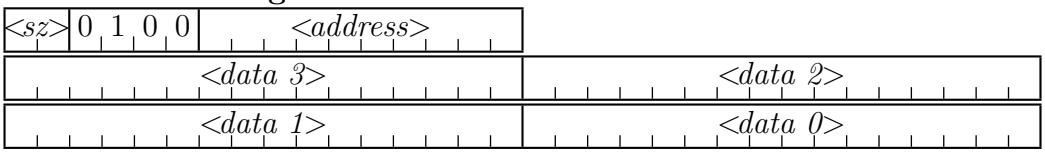
Analog Board Command Message



The <*data*> frame is only present when the p bit is set. The analog board responds to a subset of the address range with 10 valid address bits, and four most significant bits zero.

There are two ways to generate messages to the analog board, either with the command message described here, or with an analog readback command described in section 3.5.

Command message:



If the command has <*sz*>=0, a message without data is sent. Otherwise the command data is send in multiple messages to consecutive addresses on the analog board. The word ordering is little endian.

2.6.2 Analog Board Responses

When the analog board returns data in response to a command messages, the data is sent through the control channel, using two frames per 16-bit data word. Each returned word is prefixed with a fixed header. This allows to distinguish data words with content zero from the idle link.

The analog board returns data only when it receives an analog readback command described in section 3.5.

2.6.3 Analog Frontend Control

The control channel to the analog board is used to send states and strobes. The channel can support up to 15 state bits and up to 15 strobes.

When a strobe needs to be sent, a control word is sent with the most significant data bit zero, and the strobe set in the corresponding bit position.

a $\texttt{strobes}_{16}$**[1]**=$\texttt{strobes}_{64}$**[14]:** random, trigger,

a $\texttt{strobes}_{16}$**[3]**=$\texttt{strobes}_{64}$**[3]:** timestamp clock reset,

a $\texttt{strobes}_{16}$**[13]**=$\texttt{strobes}_{64}$**[4]:** readback FIFO reset,

$\texttt{a strobes}_{16}[\mathbf{14}]=\texttt{strobes}_{64}[\mathbf{5}]$**:** data FIFO reset.

Every few microseconds, the digital board will send a control message with the most significant data bit set. The remaining bits are enable bits which will be latched by the analog board to control is operations.

$\texttt{a states}_{16}[\mathbf{0}]=\texttt{enables}_{8}[\mathbf{5}]$**:** analog master enable,

$\texttt{a states}_{16}[\mathbf{1}]=\texttt{confs}_{16}[\mathbf{1}]$**:** samples mode enable,

$\texttt{a states}_{16}[\mathbf{3}]=\texttt{enables}_{8}[\mathbf{4}]$**:** timestamp clock enable,

$\texttt{a states}_{16}[\mathbf{13}]=\texttt{confs}_{16}[\mathbf{6}]$**:** readback disable,

$\texttt{a states}_{16}[\mathbf{14}]$**:** event buffer full.

2.6.4 Analog Trigger Event Data

The stream channel from the analog board is used to transfer trigger event data packets. Two types of packets are supported, pulse height event packets, and sample packets. When sample packets are enabled, the analog board will not send pulse height event packets.

2.6.5 High Speed Calibration Mode

In nonflight operation during calibration campains it is very desirable to acquire data at high rate from the analog board. For this purpose it is possible to send the data stream from the analog board directly to the redundant LVDS link.

This mode is enabled by setting bit $\texttt{confs}_{16}[0]$. Reception of commands in the redundant receiver is not impacted by activating high speed mode.

The data and clock from the analog board are time multiplexed on the single redundant line. A microframe of four bits at 96 Mbps consists of the sequence 0, 1, A, and B, with two data bits A and B. The GSE samples the stream at 384 MHz, identifies the rising edges, and extracts data and clock, which are then decoded by an instance of the same module as present in the digital board FPGA.

3 Telemetry packets

Telemetry emitted by the FPGA is tagged with an APID. Eight bits xx of the APID can be arbitrarily configured for each packet emitted. These bits shall steer the processing of the packet in the ICU. The upper eight bits distinguish the included data types:

$\texttt{0x 01}xx$**:** Register readout.

`0x 03`xx**:** Histogram data.

`0x 05`xx**:** Memory readout.

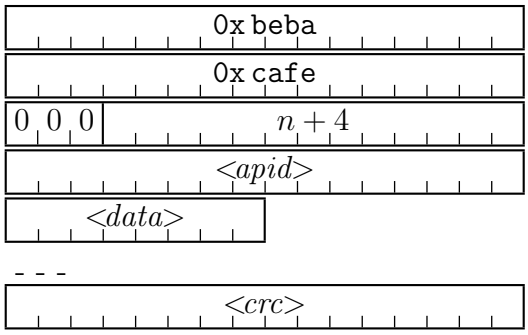
`0x 08`xx**:** Frontend readout.

`0x 57EA:` L2 streaming. (TBC)

The last packet type shall not ever be issued when the sensor is integrated. These packets are issued in L2 streaming mode, where all frontend data packets are sent directly to the serial line. This function will be used for sensor calibration. This mode is someimes referred to as *Slow Speed Streaming*, to distinguish it from the *High Speed Streaming* of the analog board bitstream via the redundant channel.

3.1 Serial Packet Format

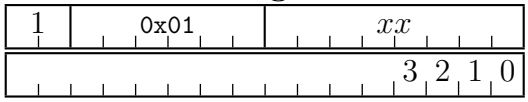
Telemetry packets are sent to the ICU in the STF packet format defined for EPD. The first two bytes of the STF packet payload is the APID.



3.2 Register Readout

A register readout is initiated with a corresponding command message. The message address is the same as the issued APID, i.e., `0x 01`xx. The message shall have 16 bits of data, which select from a set from 16 items to be sent in the resulting telemetry packet. Each item results in 64 bits of data.

Command message:



The items selected by the given bit number are

0: status and error registers,

1: address registers,

2: modulus timer registers,

3: scratch register,

4: Nominal UART and PPS status.

5: Redandant UART and PPS status.

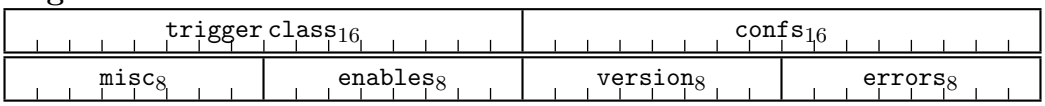
6: heater registers,

All other items are reserved, they return zeros.

3.2.1 Status and Error Registers

$\texttt{confs}_{16}$ and $\texttt{enables}_{8}$ are control bits that are set and cleared with a master control message. $\texttt{version}_{8}$ is a constant identifier of the FPGA and the version of the code. $\texttt{errors}_{8}$ are bits that are set when something happened.

Register 0 Readout:



The $\texttt{misc}_{8}$ bits are zero, except for bit $\texttt{misc}_{8}[0]$, which is the state of the EEPROM BUSY signal.

The error bits are cleared by $\texttt{strobes}_{64}[7]$.

$\texttt{errors}_{8}$**[7:6]:** 0 (unused)

$\texttt{errors}_{8}$**[5]:** lost synchronization to analog board,

$\texttt{errors}_{8}$**[4]:** ATBRK received,

$\texttt{errors}_{8}$**[3]:** single-bit error in the external SRAM,

$\texttt{errors}_{8}$**[2]:** multi-bit error in the external SRAM,

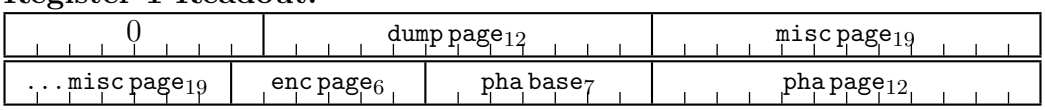
$\texttt{errors}_{8}$**[1]:** single-bit error in FPGA RAM,

$\texttt{errors}_{8}$**[0]:** multi-bit error in FPGA RAM.

3.2.2 Address Registers

These are address pointers for various memory related operations.

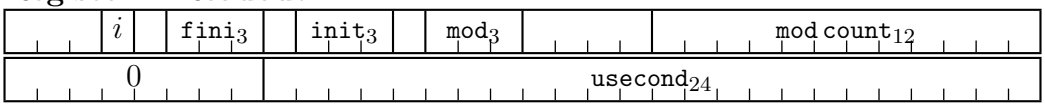
Register 1 Readout:



3.2.3 Modulus Timer Registers

All of the FPGA's notion of time.

Register 2 Readout:



The i bit set when the PPS scheduler is not active.

The modulus scheduler counts μseconds divided down from the FPGA clock. At each μsecond it issues a tick. Messages from the PPS scheduler are issued in phase with the μsecond tick, messages from the ICU are delayed to fall in the middle of the tick period, to not collide with PPS scheduler messages. The EEPROM page write delays are derived from the μsecond tick.

A PPS or $\texttt{strobes}_{64}[2]$ resets the μseconds counter. The counter is 24-bits long, so we can support PPS intervals up to 16 seconds.

The modulus scheduler counts seconds (as defined by the PPS interval) in a cycles of length 3600, i.e., one hour. The $\texttt{mod count}_{12}$ reflects the current second within that cycles. The cycle is reset by a $\texttt{strobes}_{64}[1]$. The cycle is factored into seven data acquisition cadence counters

$$3600 = 5 \times 2 \times 3 \times 2 \times 5 \times 2 \times 6$$

Three moduli in the range $0 \ldots 7$ mark significant times in the cycle. $\texttt{mod}_3$ marks the beginning of a data acquisition cadence periods. All cadence levels at or below $\texttt{mod}_3$ are beginning in the current second.

$\texttt{fini}_3$ marks the end of the cadence periods. All cadence levels at or below $\texttt{fini}_3$ end in the current second. This modulus tells how many data products are generated in the current second.

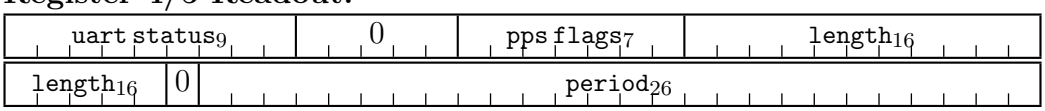
$\texttt{init}_3$ is a rather complicated modulus used to initialize encoding sequences. All encoded data product with acquisition cadence equal $\texttt{fini}_3$ and encoding cadence at or below $\texttt{init}_3$ will initialize an new compression sequence in this second.

$\texttt{fini}_3$ and $\texttt{init}_3$ are part of the science data packet headers. The parser obviously needs to know these moduli to decode the data bitstream.

3.2.4 UART Status

The status of the UART and PPS receiver.

Register 4/5 Readout:



`uart status`$_9$**[8]:** AT-break active.

`uart status`$_9$**[7]:** RX break active.

`uart status`$_9$**[6]:** RX busy.

`uart status`$_9$**[5]:** RX active.

`uart status`$_9$**[4]:** RX frame error.

`uart status`$_9$**[3]:** RX line status.

`uart status`$_9$**[2]:** PPS line status.

`uart status`$_9$**[1]:** TX empty.

`uart status`$_9$**[0]:** TX break (not used, alway zero).

`pps flags`$_7$**[6]:** PPS is good.

`pps flags`$_7$**[5]:** PPS is selected.

`pps flags`$_7$**[4]:** PPS is too seldom, i.e., two PPS received on the other line.

`pps flags`$_7$**[3]:** PPS is too long, longer than 500 μs.

`pps flags`$_7$**[2]:** PPS is too short, shorter than 1 μs.

`pps flags`$_7$**[1]:** PPS is too fast, more than 5 ms too early.

`pps flags`$_7$**[0]:** PPS is too slow, more than 5 ms too late.

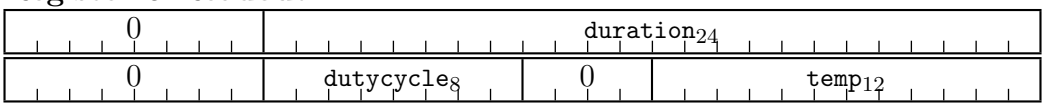
`length`$_{16}$**:** PPS length, in units of 24 MHz clocks.

`period`$_{26}$**:** PPS period error, in units of 24 MHz clocks.

3.2.5 Heater Registers

The temperature, resulting duty cycle of the operational heater, and remaining heater time in PWM cycles. See section 5.

Register 6 Readout:

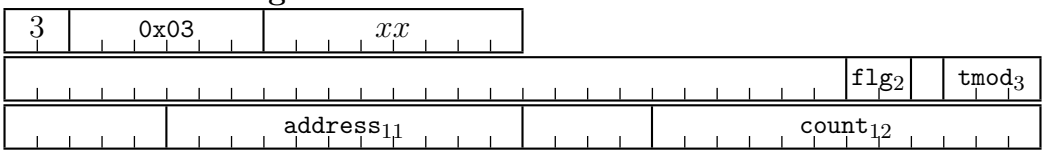


3.3 Histogram Data

A command message to the *Data Product Scheduler* (DPS) will initiate the assembly of a histogram data packet. These are the main data products of

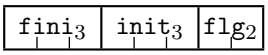
the sensor.

Command message:



This command instructs the DPS to execute $\texttt{count}_{12}$ items starting at $\texttt{address}_{11}$ from the data products table. If the current modulus is at least $\texttt{tmod}_3$, the resulting data will be sent in a telemetry packet.

The payload of the data packet is a stream of items with varying bit sizes, starting with an 8-bit header telling the cadence modulus, followed by optionally compressed histogram counter sums. The last byte is filled with unspecified bits. The format of the header is



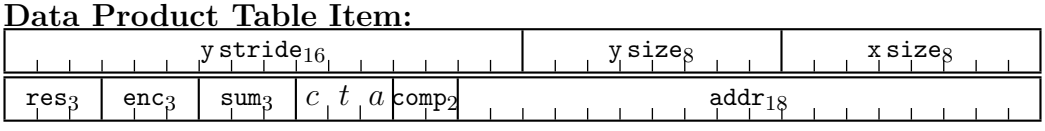
The $\texttt{flg}_2$-bits from the command message are included in the 8 bit packet header. They carry no semantics within the sensor unit and can be used to multiplex up to four data product streams into one CCSDS telemetry packet stream.

The packet is assembled in a special telemetry FIFO, to establish the packet size before transmission is started. This FIFO can assemble packets of up to 2 kBytes.

3.3.1 Data Product Scheduler

A table with 2048 entries configures individual histogram window sums. A window sum is calculated as the sum of histogram counters from a 2-dimensional region in the external memory.

Data Product Table Item:



Three bits have these meanings:

- c: Clear each histogram memory word after reading.
- t: Send the result to the telemetry.
- a: Add the result of the previous data product.

The c-bit is used to clear the histogram memory for the next acquisition. The DPS can be started with a message-id `0x 02`xx, instead of `0x 03`xx, which will prevent the assembly of a data packet, e.g., for clearing a memory region.

The a-bit can be used to build more complex data products. An item with the a-bit shall follow an item without the t-bit. The first sum is not sent for compression, but included in the following data item.

The memory window is defined by $\texttt{addr}_{18}$, $\texttt{x size}_8$, $\texttt{y size}_8$, and $\texttt{y stride}_{16}$. The sum starts at $\texttt{addr}_{18}$. The address is incremented $\texttt{x size}_8$ times by one. Then it is incremented by $\texttt{y stride}_{16}$ and the run through the x is repeated. The $\texttt{y stride}_{16}$ increment happens $\texttt{y size}_8$ times. The total number of memory locations being added is

$$(\texttt{x size}_8 + 1) \times (\texttt{y size}_8 + 1)$$

The MSB of the memory address is the complement of $\texttt{enables}_8[3]$, i.e., the `hist page` bit, selecting the part of memory not currently used for data acquisition.

3.3.2 Data Compression

A data item that has the t-bit set is sent to the compression engine. The compression is controlled by $\texttt{sum}_3$, $\texttt{enc}_3$ and $\texttt{res}_3$.

The DPS will execute every second. Most data products shall be produced at a longer cadence. The $\texttt{sum}_3$ modulus tells the compressor for how long to sum the data before transmission. The data packet will be generated every second, but include only data items that have run their summing period. The modulus values are given in Table 13 on page 43.

The $\texttt{res}_3$ modulus shall be the same as the $\texttt{sum}_3$ modulus. If $\texttt{res}_3$ has a lower value than $\texttt{sum}_3$, a sparse count rate is produced. The data is generated at the cadence specified in $\texttt{sum}_3$, but the counter is reset at the cadence given in $\texttt{res}_3$. Counts before the last reset are lost.

If $\texttt{res}_3$ has a larger value than $\texttt{sum}_3$, the data sent will accumulate over multiple data items. This does not work with compression.

The $\texttt{enc}_3$ modulus tells the compressor for which stretches of time the lossy running difference compression shall be applied to the stream of counts. If $\texttt{enc}_3$ is not larger than $\texttt{sum}_3$, the data will be send unencoded according to the format specified in $\texttt{comp}_2$. Otherwise they are subject to further compression. The compression cycle is reset at the cadence given by $\texttt{enc}_3$.

The compressor maintains a data record for each data product in main memory. Each record occupies four words. The base address is configured in $\texttt{enc page}_6$, i.e., bits [18:13] of the memory address.

If a telemetry packet includes only data products that require summing, i.e., all $\texttt{sum}_3 > 0$, the data packets between the sums would include only the 8-bit header. The DPS shall be started with $\texttt{tmod}_3$ equal to the smallest $\texttt{sum}_3$, to suppress these empty telemetry packets in between.

3.3.3 Unencoded Data

Most science data will be encoded with variable bit sizes. The format of unencoded data is defined with two bits $\texttt{comp}_2$, selecting one of four representations for the counters. The unencoded data is unsigned.

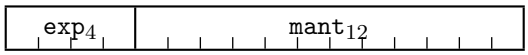
$\texttt{comp}_2$**=00:** 16 bit unsigned floating point, 4-bit exponent.

$\texttt{comp}_2$**=01:** 8 bit representing $8{\cdot}(\log_2(c) + 1)$.

$\texttt{comp}_2$**=10:** 24 bit unsigned integer.

$\texttt{comp}_2$**=11:** encoded uncompressed counter.

The format of an unsigned 16-bit float is



The represented counter value is

$$C = \texttt{mant}_{12} \qquad \text{for} \quad \texttt{exp}_4 = 0,$$
$$C = (\texttt{mant}_{12} + 2^{12}) \cdot 2^{\texttt{exp}_4 - 1} \qquad \text{for} \quad \texttt{exp}_4 > 0.$$

The floating point or $\log_2$ representations do not need to be converted to plain integers for comparisson to some threshold. The $\log_2$ representation can be used in trigger or monitoring logic for ratio cuts.

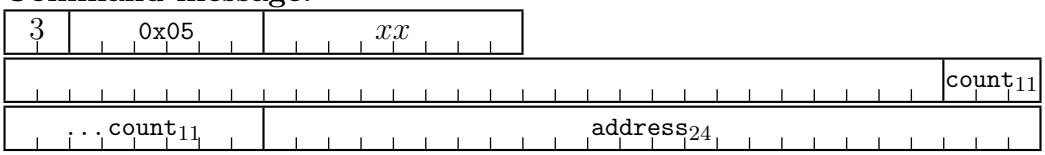
3.4 Memory Readout

The memory readout engine can read words from various sources:

- external SRAM.
- external EEPROM.
- configuration tables in the FPGA
  - PPS schedule table,
  - data products table,
  - L3 code memory,
- counters.

Readout is initiated with

Command message:

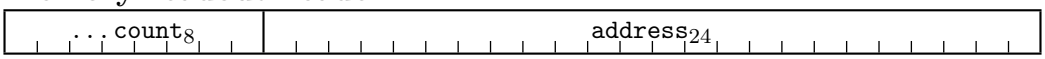


The read data will be packed into a telemetry packet. Using message address 0x 04xx will also perform the reads of the memory, but not transmit a packet. This can be used to perform EDAC scrubbing on configuration memories.

Which memory is read and how the final memory addresses are calculated is encoded in $\texttt{address}_{24}$. The number of words read is $\texttt{count}_{11}$. The word size depends on the kind of memory or is encoded in $\texttt{address}_{24}$ as well.

The telemetry packet payload is prefixed with 32 bits

Memory readout header:



Three most significant bits of $\texttt{count}_{11}$ are omitted. The $\texttt{address}_{24}$ is the calculated read address of the first data word. This may be different from the $\texttt{address}_{24}$ given in the command message.

3.4.1 External SRAM

The external SRAM is read in words of 32 bits. No EDAC decoding is performed. The most significant bits of $\texttt{address}_{24}$ encode how the address shall be calculated:

0x8*aaaaa*: direct addressing,

0x9*aaaaa*: histogram page,

0xA*aaaaa*: PHA page,

0xB*aaaaa*: misc $\texttt{page}_{18:0}$ indexed.

Direct addressing leaves the given address unchanged.

Histogram page access selects the half of the SRAM which is not currently being used for data acquisition, unless $\texttt{address}_{24}[18]$ is set, to access the active half.

For PHA page readout, three bit ranges of the memory address are replaced, based on the content of the pha $\texttt{base}_{18:12}$ and pha $\texttt{page}_{11:0}$ registers. pha $\texttt{base}_{18:12}$ points to the memory where PHA data is being stored by data acquisition. Readout address bit 18 is the complement of pha $\texttt{base}[18]$. Address bits [17:12] are replaced by the corresponding bits from pha base. The

lowest 12 bits are the sum of the address given in the command message and the `pha page` register.

The `misc page`$_{18:0}$ register can be used for dribble readout of large memory regions. The same readout command is issued repeatedly from the PPS schedule, followed by an appropriate increment of the `misc page` register. For PHA data readout the `pha page` register can be used in this fashion.

3.4.2 EEPROM Readout

Readout of the EEPROM yields bytes, i.e., 8-bit words. Two modes are supported:

`0x C`*aaaaa*: direct addressing,

`0x F`*aaaaa*: `misc page`$_{18:0}$ indexed.

3.4.3 Configuration Table Readout

The configuration tables are stored in 72-bit wide memory blocks inside the FPGA. Eight bits are used for EDAC redundancy, leaving 64-bits of information. Readout yields error corrected 64-bit words. Dribble readout is available via the `dump page`$_{11:0}$ register:

`0x 0C2`*aaa*: direct addressing,

`0x 2C2`*aaa*: `dump page`$_{11:0}$ indexed.

The address ranges are

`0x 0C2400-0x 0C25FF`: L3 code.

`0x 0C2600-0x 0C26FF`: PPS schedule table.

`0x 0C2800-0x 0C2FFF`: data products table.

The word size is encoded in `address`$_{24}$[19:18] of the command message (`0x 0C`). If a smaller word size is specified, the words will be only partially transmitted. Don't do that.

3.4.4 Counter Readout

The FPGA keeps a set of 128 counters in a memory block of width 36 bit. After EDAC correction the counters are 29 bits wide. `dump page`$_{11:0}$ indexing is available:

`0x 0800`*aa*: direct addressing,

`0x 2800`*aa*: `dump page`$_{11:0}$ indexed.

The word size is encoded in $\texttt{address}_{24}$[19:18]. For the counter readout, three word sizes are implemented:

0x 0800*aa*: 32-bit unsigned integers.

0x 0400*aa*: 16-bit unsigned floating point, 12-bit mantissa,

0x 0000*aa*: 8-bit unsigned floating point, 3-bit mantissa.

Tables 1 to 5 list the counters provided by the HET-EPT digital board. Unused counters up to index 127 are present, but nothing increments them.

Only one counter can increment per clock cycle. An event to be counted will be latched until it gets access to the counter memory. When further events for the same counter occur before it could be submitted, the additional events will be lost.

Table 1: Digital Board Counters, frontend L2 triggers

Index	Description
0	×1 triggers lost on the analog board.
1	×2 triggers lost on the analog board.
2	×4 triggers lost on the analog board.
3	×8 triggers lost on the analog board.
4	×16 triggers lost on the analog board.
5	×32 triggers lost on the analog board.
6	×64 triggers lost on the analog board.
7	×128 triggers lost on the analog board.
8	L2 #0 triggers.
9	L2 #1 triggers.
10	L2 #2 triggers.
11	L2 #3 triggers.
12	L2 #4 triggers.
13	L2 #5 triggers.
14	L2 #6 triggers.
15	L2 #7 triggers.

3.4.5 Memory Index Registers

The memory index registers are manipulated by command messages. The values are read via register readout telemetry item 1.

hist $\texttt{base}_{[18]}$=$\texttt{enables}_8$[3]: This bit is used as address bit [18] for histogram data acquisition.

pha $\texttt{base}_{18:12}$: Base address for PHA data storage.

Table 2: Digital Board Counters, SEU Errors

Index	Description
16	Counter memory uncorrectable SEU errors.
17	Telemetry FIFO uncorrectable SEU errors.
18	L3 register uncorrectable SEU errors.
19	Data product schedule uncorrectable SEU errors.
20	L3 trigger code uncorrectable SEU errors.
21	PPS schedule uncorrectable SEU errors.
22	(unused) reserved for frontend SEU errors.
23	SRAM uncorrectable SEU errors.
24	Counter memory single bit SEU errors.
25	Telemetry FIFO single bit SEU errors.
26	L3 register single bit SEU errors.
27	Data product schedule single bit SEU errors.
28	L3 trigger code single bit SEU errors.
29	PPS schedule single bit SEU errors.
30	(unused) reserved for frontend SEU errors.
31	SRAM single bit SEU errors.

Table 3: Digital Board Counters, communication

Index	Description
32	PPS accepted.
33	PPS received via message.
34	PPS received via 1 Hz clock from ICU.
35	Data product packets.
36	Memory readout packets.
37	Register readout packets.
38	Frontend readout packets.
39	Packet collisions.
40	messages received from ICU.
41	messages executed from PPS schedule.
42	EEPROM bytes written.
43	EEPROM bytes read or written.
44	ATBRK received, $\texttt{errors}_8[4]$.
45	Lost analog board sync., $\texttt{errors}_8[0]$.
46	Unused.
47	Unused.

Table 4: Digital Board, Trigger Counters

Index	Description
48	Lost events received from the frontend.
49	Data products encoded.
50	L3 trigger started.
51	PHA records stored/counted.
52	Histogram windows computed.
53	Histogram bin increments.
54	Triggers received for class 0.
55	Triggers received for class 1.
56	Triggers received for class 2.
57	Triggers received for class 3.
58	TFIFO put while pushing.
59	TFIFO init while sending.
60	Window sum valid.
61	Floating point result valid.
62	$\log_2(c)$ result valid.
63	Compression result valid.

Table 5: Digital Board, UART

Index	Description
64	Valid bytes received.
65	Invalid bytes received (STOP bit missing).
66	Messages received.
67	Message header format errors.
68	Message timeouts.
69	CRC errors.
70	FIFO uncorrectable SEU errors.
71	FIFO single bit SEU errors.
72	Streaming mode: packet lost.
73	Streaming mode: packet init.
74	Streaming mode: packet submit.
75	Streaming mode: packet put.
76	Unused.
77	Unused.
78	Unused.
79	Unused.

pha page$_{11:0}$: Index register for PHA readout.

misc page$_{18:0}$: Index register for external memory dribble readout.

dump page$_{11:0}$: Index register for internal memory dribble readout.

enc base$_{18:13}$: Base address for the data compression memory.

And there is a scratch$_{64}$ register, which has no function at all, but shall be used as a versioning tool to inject into the telemetry, to help the parsers on the ground to figure out what configuration was used. The scratch$_{64}$ register is returned as readout item 3.

To manipulate the index registers the command message addresses are

0x 0004: pha base$_{18:12}$,pha page$_{11:0}$ = data$_{18:0}$.

0x 0005: pha page$_{11:0}$ = data$_{11:0}$.

0x 0006: pha base$_{18:12}$ += data$_{18:12}$, pha page$_{11:0}$ += data$_{11:0}$.

0x 0007: enc base$_{18:13}$ = data$_{18:13}$.

0x 0008: misc page$_{18:0}$ = data$_{18:0}$.

0x 0009: misc page$_{18:0}$ increment.

0x 000A: dump page$_{11:0}$ = data$_{11:0}$.

0x 000B: dump page$_{18:0}$ increment.

0x 000F: scratch$_{64}$ = data$_{64}$.

hist base$_{[18]}$ is set or cleared with strobes from a master control message. The MSB of pha base$_{18:12}$[18] can be toggled with strobes[11].

The misc page$_{18:0}$ and dump page$_{18:0}$ increment commands allow for quite a bit of flexibility to control the memory regions covered. The message data provides an increment amount, two masks, and a constant.

	misc page$_{18:0}$	dump page$_{18:0}$
<*increment value*>	data$_{18:0}$	data$_{11:0}$
<*increment mask*>	data$_{50:32}$	data$_{27:16}$
<*keep mask*>	data$_{50:32}$	data$_{43:32}$
<*constant*>		data$_{59:48}$

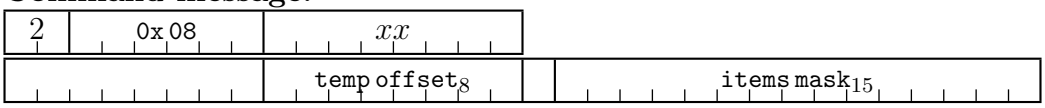
The index register will be incremented by the <*increment value*>. All bit positions not set in the <*increment mask*> will be cleared from the sum. All bit positions set in the <*keep mask*> will be keep from the old value of the index register. Finally, all bits set in <*constant*> will be set in the index register.

Let's see how much of this is useful.

3.5 Frontend Readout

A message to address `0x1ff` on the analog board will initiate the return of an analog readback packet to the digital board. This data stream will normally be ignored by the digital board. A special *analog readback* command message can be used as an alternative way to issue such a readback message to the analog board

Command message:



This command will save the telemetry tag xx and enable the transmission of the next analog readback packet wrapped into a telemetry packet with APID `0x08`xx.

The analog readback packet includes the $\texttt{items mask}_{15}$ as first data word. The digital board FPGA will use the returned mask to calculate the size of the packet. So, unfortunately, the engine that forwards the analog readback needs to have hardwired knowledge of the size of the items that may be returned.

3.5.1 Analog Readout Items

The bits in the $\texttt{items mask}_{15}$ select the items that shall be returned. The items are variable sized blocks of 16-bit words.

0: Filter configuration, 8 words.

1: Trigger configuration, 92 words.

2: Timestamp clock, 2 words.

3: HK ADC readout, packed, 12 words.

4: HK ADC readout, unpacked, 16 words.

5: Raw sample peek, 30 words.

6: Pulse height peek, 30 words.

7: Pulse phase peek, 30 words.

8: Single channel trigger counter read, 30 words.

9: Single channel trigger counter read and clear, 30 words.

10: Single channel trigger counter clear only, 0 words.

Bits [14:11] are unused/reserved. The items are send starting with the highest selected bit.

3.5.2 Analog Housekeeping

Sixteen ADC readings with 12-bit resolution are returned either packed, or unpacked one reading per words with the channel number prefixed.

Table 6: Analog Housekeeping Readings

0	400	HET preamp temperature
1	500	EPT preamp temperature
2	401	HET VREF, +2.5 V
3	501	EPT sensor temperature
4	402	HET sensor temperature 1
5	502	EPT VFET/2, +5 V
6	403	HET VFET/2, +5 V
7	503	EPT VANA/2, +6 V
8	404	HET sensor temperature 2
9	504	EPT VREF, +2.5 V
10	405	Shaper VCC/2, +5 V
11	505	Bias Current
12	406	Power board temperature
13	506	Bias Voltage
14	407	Analog board temperature
15	507	Digital board temperature

3.5.3 Temperature Capture

To control the operational heater, it may be necessary to know the temperature of the sensor unit. Among the items that can be returned from the analog board are readings of voltages and temperature sensors. When the $\texttt{temp offset}_8$ field in the command message is non-zero, it will be used as a word offset into the returned analog readback packet. Twelve least significant bits from the indicated word will be copied to the $\texttt{temp}_{12}$ register. The $\texttt{temp offset}_8$ shall point to a word from an unpacked HK ADC readout. Some packed HK ADC readout channels may also work if they are properly aligned.

To capture a temperature from an analog readout packet without issuing a telemetry packet, the command message can be issued with address $\texttt{0x 09}xx$ instead of $\texttt{0x 08}xx$.

3.5.4 Peeking

The digital filter continuously issues pulse height reconstruction parameters and raw ADC samples for each data acquisition channel. Peek readout cap-

tures a set of these values. Raw sample peek returns a set of ADC samples, to diagnose problems with the analog signal baselines. Pulse height/phase peeks may be useful to estimate noise amplitudes.

The digital filter needs to be enabled via $\texttt{enables}_8[5]$ for any peek to yield the expected data.

3.5.5 Single Channel Trigger Counters

Each signal channel has a counter that increments when its L1 trigger threshold is exceeded. The counters can optionally be cleared after reading, or even be cleared without reading. The counters values are provided in 16-bit unsigned floating point representation with 12-bit mantissa.

3.6 L2 Streaming

When the data acquisition is enabled on the analog board, a trigger event packet is sent for every valid L2 trigger. Normally, this data is sent to the L3 trigger processor and, on request from L3, saved into PHA storage.

For calibration of the sensor it will be necessary to get direct access to the event packets. The bit $\texttt{confs}_{16}[2]$ enables L2 streaming mode, all event packets received from the analog board will be wrapped into a telemetry packet. L2 streaming uses the telemetry FIFO.

When the FIFO is full, the packets trying to get in will be dropped.

3.6.1 Samples Readout

The analog board FPGA can be set to samples mode. Instead of trigger event packets, L2 triggers will issue a configurable stream of sample packets, including consecutive sets of raw ADC readings. This is like a 30 channel digital oscilloscope, and can be used to diagnose the pulse shapers.

Samples mode is enabled by setting bit $\texttt{confs}_{16}[1]$. If streaming mode is enabled as well, the packets will be sent to telemetry.

Sample packets will be forwarded to the L3 trigger, and can be saved in PHA buffers. This may lead to scenarios to use sample mode in flight for troubleshooting, with readout through PHA storage.

3.7 Monitoring Parameters

The Sensor unit will send several packets to the ICU for purposes other than being forwarded to the spacecraft and to ground, i.e,

- monitoring sensor status and health,

- data to be sent in S20 packets,
- trigger burst mode, and
- detect high particle rate conditions.

These functions need information from the sensor unit. The ICU will extract values from sensor packets and store them in its global parameters storage. Parameter extraction is governed by a table in the ICU configuration that specifies for each extractable parameter:

- a parameter-ID,
- the sensor-ID,
- the packet-APID,
- byte offset into the packet,
- length in bytes of the value to extract, big endian,
- a mask (32 bits) to clear selected bits from the value.

A parameter is identified by a 16-bit parameter-ID.

This section lists possible/proposed parameters to extract from sensor packets, and how they may become useful.

3.7.1 Register readout

Sensor status is extracted via register readout, as described in section 3.2. Table 7 assumes that the first seven registers are read, i.e., the register mask is `0x007F`. With a different mask the offsets will change. The STEP sensor may define more registers that may require monitoring, these may be added to the table in the future.

Table 7: Parameters to extract from the register readout packet. Offset is in bytes, length is in bits.

Name	offset	length	description
status	2	64	variius status and error bits
time	22	12	cadence counter, second of the hour
duty cycle	56	8	OP heater duty cycle
temperature	57	12	OP heater temperature reference

3.7.2 Histogram Data

A packet with data derived from the acquired histograms will be formatted for the ICU, i.e., without variable length encoding. Most of these will be

8 bits values representing the eight times the logarithm of base two of the counter value. This format is suitable to be directly put into S20 packets as requested by RPW.

The format is also suitable for relative trigger cuts, because differences in parameter values represent ratios of counter values.

A second useful format available is 16 bits unsigned floating point representations of counter values. These can be directly compared to thresholds, since the representation interpreted as an unsigned integer is monotonous with the represented counter value.

For absolute difference trigger cuts, the counters can be represented as plain integers in 24 bits.

The packet from an HET-EPT sensor shall include

- ten $\log_2$ values from EPT as requested by RPW, covering four energy ranges for electrons and one energy range for ions, from two directions, at a time resolution of 1 s.
- the same ten values at time resolution 10 s,
- further data from HET and EPT as required.

The data items must be sorted by time resolution. Longer cadence data needs to be appended to shorter cadence data, so that each data item is always at the same byte offset in the packet.

The ICU will reserve a set of parameter-IDs for trigger parameters that can be configured during the mission to extract the required data items for monitoring and trigger purposes. The parameters to be sent in S20 will be alloated with fixed IDs. Triggers can use any parameters available, including the S20 parameters.

3.7.3 Counter Memory Readout

The digital board counters may include items which are useful for monitoring or trigger purposes. Event or trigger counters can be used to identify high rate environments. SEU error counters may be used to monitor sensor health. Message counters can flag unusual activity, i.e., for a flag in a status word that indicates a message reception in a sensor unit.

The available counters are listed in tables 1 to 5.

3.7.4 HET-EPT Analog Frontend Readout

A packet from the frontend board will contain at least the housekeeping ADC readings in unpacket format. The single channel trigger counters may be

usefull for high rate triggers. The frontend packet with `items` $\texttt{mask}_{15}$=`0x0110` will provide the parameters at the offsets given in Table 8.

The ICU may define parameters with fixed semantics for the housekeeping ADC readings. These readings are prefixed with a 4-bit channel number which may need to be masked out when extracting the parameters. That's why the length is givel as 12 bits.

The single channel counters are 16 bits unsigned floating point. These may be extracted into general trigger parameters as required.

Table 8: Frontend parameters

Name	offset	length	description
SC 00	2	16	L1 trigger counter channel 0.
SC 01	4	16	L1 trigger counter channel 1.
...			
SC 29	60	16	L1 trigger counter channel 29.
HK 400	62	12	HET preamp temperature
HK 500	64	12	EPT preamp temperature
HK 401	66	12	HET VREF, +2.5 V
HK 501	68	12	EPT sensor temperature
HK 402	70	12	HET sensor temperature 1
HK 502	72	12	EPT VFET/2, +5 V
HK 403	74	12	HET VFET/2, +5 V
HK 503	76	12	EPT VANA/2, +6 V
HK 404	78	12	HET sensor temperature 2
HK 504	80	12	EPT VREF, +2.5 V
HK 405	82	12	Shaper VCC/2, +5 V
HK 505	84	12	Bias Current
HK 406	86	12	Power board temperature
HK 506	88	12	Bias Voltage
HK 407	90	12	Analog board temperature
HK 507	92	12	Digital board temperature

4 Data Acquisition

Various bits from the master control registers control the data acquisition process:

$\texttt{enables}_8$**[0]:** event reception enable,

$\texttt{enables}_8$**[1]:** L3 trigger enable,

$\texttt{enables}_8$**[2]:** PHA storage enable,

$\texttt{enables}_8$**[4]:** timestamp clock enable.

$\texttt{enables}_8$**[5]:** analog frontend enable,

$\texttt{confs}_{16}$**[0]:** high speed calibration mode enable,

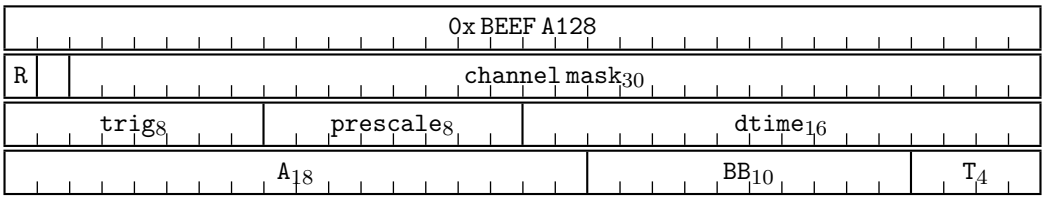
$\texttt{confs}_{16}$**[1]:** sample mode enable,

$\texttt{confs}_{16}$**[2]:** L2 streaming mode enable,

4.1 Pulse Height Data Packets

When the L2 trigger in the analog board FPGA identifies a valid particle event, a data packet is sent to the digital board. There are eight L2 triggers, and each has an associated channel mask identifying the signal channels that are important to analyze the particle hit. Data is sent for all channels that are selected by any valid L2 trigger. The data packet starts with the channel mask.

Table 9: Event Pulse Height Packet Format



- - -

The MSB of the channel mask word identifies random triggers that are activated by $\texttt{strobes}_{64}[14]$. Those event packets will contain data for all 30 channels.

The header further contains three fields

$\texttt{trig}_8$**:** bitmask of L2 triggers.

$\texttt{prescale}_8$**:** number of event packets that where dropped because the FIFOs were full, since the last transmitted packet.

$\texttt{dtime}_{16}$**:** time since the last trigger in μseconds.

For each selected channel a record of 32 bits will be sent, containing three data fields

$\texttt{A}_{18}$**:** pulse height, i.e., filter channel A.

$\texttt{BB}_{10}$: pulse phase, i.e., filter channel B, ten bits selected by the leading bit in $\texttt{A}_{18}$.

$\texttt{T}_4$: pulse age.

4.1.1 Event Packet Reception

The digital board monitors the stream channel from the analog board for the sync header of event packets `0x BEEF A128` or sample packets `0x 5A61`, and initiates the packet parser when it is idle, or when there where unrecognized data in the stream before the currently parsed packet. Event packet reception cannot be disabled at this stage.

The parser will capture the $\texttt{channel mask}_{30}$, and expect as many channel records as requested. The data will be send to three destinations

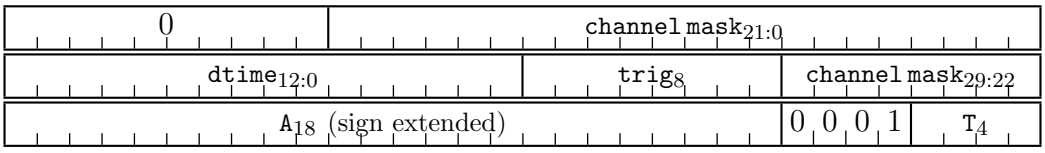
- the L3 data set,
- the PHA data set, and
- in streaming mode to the telemetry FIFO.

The telemetry FIFO receives the packet unchanged, if streaming mode is enabled.

The L3 and PHA data sets are transferred simultaneously to the event buffer. Each data set contains 32 words with 32 bits each. The L3 data set will later be loaded into the L3 register file, the PHA data set will be saved in a PHA buffer when so instructed by the L3 processor.

The PHA data set receives the words from the event packet unchanged. The data for each signal channel will we written to a fixed position in the data set, i.e., unselected channels will be substituted with zeros in the data sets. PHA data is saved without EDAC protection into the external SRAM.

Table 10: L3 Data Set Format



- - -

The word size of the L3 trigger processor is only 29 bits because of EDAC overhead. Some reformatting is necessary to make sure all relevant data fits into the registers.

The L3 trigger will not receive the $\texttt{BB}_{10}$ fields of the channel records, nor the $\texttt{prescale}_8$ field of the headers. The $\texttt{dtime}_{16}$ field is truncated to 13 bits with overflow check.

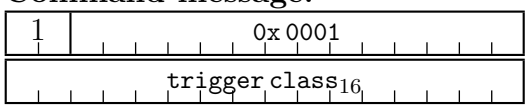
L3 data set word 0 receives the part of the channel mask corresponding to HET channels. Word 1 receives the channel mask bits for EPT channels, and further header data. The remaining data set words are each associated with a fixed signal channel. For unselected channels all zeros are written. Available channel data are formatted such that the whole register value can be used as the pulse height, the data in the LSB contributing insignificant noise.

4.1.2 L3 Event Class

The L3 trigger processor has four entry points, i.e., there can be four different programs called for different event classes. The event class is established by the packet reception unit from the L2 $\texttt{trig}_8$ bits.

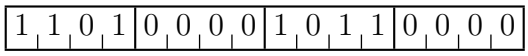
The mapping from $\texttt{trig}_8$ to event class is provided by the $\texttt{trigger class}_{16}$ register, which can be set with

Command message:



The event class is a 2-bit number. The MSB of the event class is set, when any of the L2 $\texttt{trig}_8$ bits is active that is also set in the upper half of $\texttt{trigger class}_{16}[15{:}8]$. The lower bit of the event class is set when any of the L2 triggers selected by the lower half $\texttt{trigger class}_{16}[7{:}0]]$ is set.

The proposed value for the $\texttt{trigger class}_{16}$ register is



with a proposed configuration of L2 triggers

$\texttt{trig}_8$**[0]:** EPT ion forward,

$\texttt{trig}_8$**[1]:** EPT electrons backward,

$\texttt{trig}_8$**[2]:** EPT ion backward,

$\texttt{trig}_8$**[3]:** EPT electrons forward,

$\texttt{trig}_8$**[4]:** HET forward stopping,

$\texttt{trig}_8$**[5]:** HET forward penetrating,

$\texttt{trig}_8$**[6]:** HET backward stopping,

$\texttt{trig}_8$**[7]:** HET backward penetrating,

With this L2 trigger menu, the L3 trigger classes are:

0: EPT trigger, no HET triggers present.

1: HET forward stopping, no other HET triggers.

2: HET backward stopping, no other HET triggers.

3: HET penetrating.

EPT triggers are presumably more frequent than HET triggers. The class 0 program does not need to bother with HET. The HET programs need to check for possible EPT triggers as well. HET stopping trigger classes do not need to look for signals in the downstream detectors. HET penetrating needs to do the most work. The downstream hits may be crosstalk or it may be a real penetrating particle.

4.1.3 Trigger and Prescale Counters

The data reception unit counts the occurrences of set bits in $\texttt{prescale}_8$ and $\texttt{trig}_8$ for all received event packets. The counters are located at address 0 to 7 ($\texttt{prescale}_8$) and 8 to 15 $\texttt{trig}_8$ of the counter memory.

The counts of the $\texttt{prescale}_8$ bits can be weighed by bit significance and added together give the total number of events lost in the analog board.

4.2 Event Buffer

L3 and PHA data sets are written into the event buffer memory, which is 128 word of 64 bits, excluding EDAC overhead. The words contain the L3 and PHA parts, 32 words for one data set. Four event data sets fit into the buffer memory.

Incoming event data is always written into the next available data set space. When the set is complete,

- if $\texttt{enables}_8[0]$ is not set, the data set will be discarded and ignored.
- Else, when the event buffer is full, the data set will be discarded, and counted as lost.
- Else, the data set is queued for processing by the L3 processor, i.e., the data set write pointer advances to the next slot.

4.2.1 Feeding the L3 trigger processor

The L3 trigger processor operates on a register file with 256 registers. The L3 event data sets are loaded into the last 32 registers before the trigger is

executed.

When events are queued in the buffer, and the L3 trigger processor is idle, and the PHA storage unit is idle, the L3 processor is loaded and started. The data set memory space is then released from L3 but still locked for PHA storage. But all previous data sets are released from PHA at this point and become available for new incoming data.

The event buffer keeps track of the event class that was assigned to each data set. The L3 trigger is started at the instruction address with the event class as the two MSB and zeros in the LSB, i.e., at addresses `0x 000`, `0x 100`, `0x 200`, or `0x 300`.

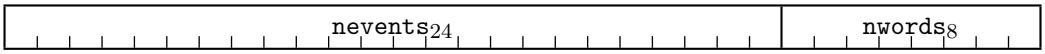
4.2.2 PHA storage

The L3 trigger processor may decide to save the PHA data sets into a PHA buffer in external memory. 16 buffers with 256 memory words of space are available, 32 bits each, 1kByte per buffer 16 kBytes total. L3 decides into which buffer the data shall go.

The idea is that rare events are saved in separate buffers, so they are not lost. The frequent events will fill their buffers fast and all further events will be dropped.

The PHA storage is not protected with EDAC redundancy. The data is not considered essential for science analysis. EDAC protected memory words leave only 26 bits, which is not a good match for the storage of PHA data. The location of the PHA buffers in the external memory is configured via $\texttt{pha base}_{18:12}$.

The PHA storage unit keeps two counters for each PHA buffer. $\texttt{nevents}_{24}$ is the number of events that where supposed to be stored in the buffer, and $\texttt{nwords}_{8}$ is the pointer to the last word in the buffer that was written. These two counters are written into the first words of the PHA buffer for each event, i.e., $\texttt{nevents}_{24}$ continues to count when the buffer is full.



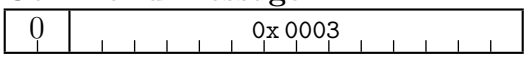
The PHA data set is appended to the buffer data, until it is full. The end of a full buffer will likely contain an incomplete event data set. Only the event header and valid channel data are saved. The headers allow to reconstruct the storage size of each event.

The MSB of $\texttt{pha base}_{18:12}$ is toggled with $\texttt{strobes}_{64}[11]$. That effectively saves the set of filled PHA buffers for telemetry readout, and make a new set available for data acquisition. The new set needs to be cleared with $\texttt{strobes}_{64}[8]$, which resets all internal copies of $\texttt{nevents}_{24}$ and $\texttt{nwords}_{8}$ to zero, and writes zeros into the first word of each PHA buffer.

4.2.3 Test Data Injection

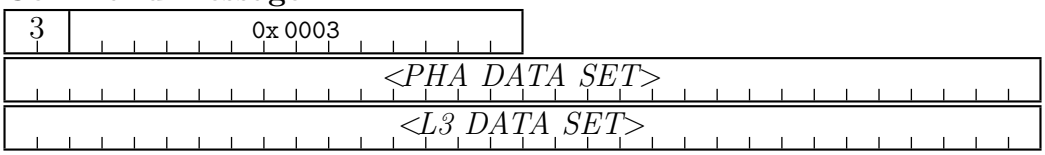
Test data can be injected into the PHA buffer via command messages to address 0x0003. A message with size tag zero resets the input register pointer, i.e., the next data will go into slot 0 of the event data set.

Command message:



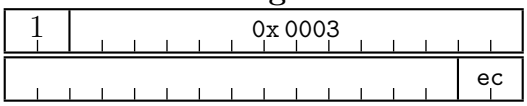
Messages with size 64-bits write into the next word of the event buffer, with the PHA data in the upper half of the data word and the L3 data in the lower half.

Command message:



The data set is submitted by a message with data size 16-bits. The two LSB encode the event class.

Command message:



The analog frontend should be disabled when the injection is in use, else the data may collide with real triggers.

4.2.4 Event Data Formats

. . . TODO

4.3 L3 trigger

The L3 trigger is a programmable processor with separate program and register memory.

Both memories are EDAC protected. EDAC errors in the program are corrected in the memory when encountered. EDAC errors in the register memory are corrected for read but not in memory.

The program memory stores 1024 instructions. An instruction word is 32 bits. Two instructions are stored in one configuration memory word. The

registers are 29 bits, stored in 36 bit wide memory blocks including 7 bits of EDAC redundancy.

The result of the computation of each instruction is stored in the register file at the address that is the same as the eight LSB of the instruction address. The program leaves a trail of values behind as it executes. The processor maintains two condition bits. Execution of any instruction is conditional depending on the value of those bits. An instruction where the condition is false will save the result of the previous instruction.

The opcodes are designed specifically for the work of a L3 trigger. There are instructions for calibration and corrections of pulse height values to energy units. It can compute $\log_2$ of pulse energies, for histogram binning and ratio cuts. The experience gained from the MSL RAD trigger went into the design of this processor.

Table 11: L3 instruction opcodes

STOP		`cc00 0000 0--- ---- ---- ---- ---- ----`
NOP		`cc00 0000 10-- ---- ---- ---- ---- ----`
GOTO	$\mathtt{u}_{10}$	`cc00 0000 11-- --uu uuuu uuuu ---- ----`
LOG	R_x	`cc00 0001 ---- ---- ---- ---- xxxx xxxx`
POKE	$\mathrm{R}_d = \mathrm{R}_x$	`cc00 0010 ---- ---- dddd dddd xxxx xxxx`
BITC	$\mathrm{R}_x\{\mathtt{u}_5\}$	`cc00 0011 n0SC ---- ---u uuuu xxxx xxxx`
BITS	$\mathrm{R}_x\{\mathtt{u}_5\}$	`cc00 0011 n1SC ---- ---u uuuu xxxx xxxx`
BRNG	$\mathrm{R}_x\{\mathtt{v}_5 : \mathtt{u}_5\}$	`cc00 0100 ---- --vv vvvu uuuu xxxx xxxx`
TRIM	$\mathrm{R}_x, \mathtt{u}_8, \mathtt{v}_8$	`cc00 0101 vvvv vvvv uuuu uuuu xxxx xxxx`
MULI	$\mathrm{R}_x * \mathtt{m}_{12} \gg \mathtt{e}_4$	`cc00 0110 eeee mmmm mmmm mmmm xxxx xxxx`
PHA	$\mathrm{R}_x + \mathtt{u}_{16}$	`cc00 0111 uuuu uuuu uuuu uuuu xxxx xxxx`
ADD	$\mathrm{R}_x \gg \mathtt{i}_4 + \mathrm{R}_y \gg \mathtt{j}_4$	`cc00 1000 yyyy yyyy jjjj iiii xxxx xxxx`
SUB	$\mathrm{R}_x \gg \mathtt{i}_4 - \mathrm{R}_y \gg \mathtt{j}_4$	`cc00 1001 yyyy yyyy jjjj iiii xxxx xxxx`
HIST	$\mathrm{R}_x \gg \mathtt{i}_4 + \mathrm{R}_y \gg \mathtt{j}_4$	`cc00 1100 yyyy yyyy jjjj iiii xxxx xxxx`
HIST	$\mathrm{R}_x \gg \mathtt{i}_4 - \mathrm{R}_y \gg \mathtt{j}_4$	`cc00 1101 yyyy yyyy jjjj iiii xxxx xxxx`
CMP	$\mathrm{R}_x + \mathtt{u}_8 <=> \mathrm{R}_y$	`cc01 0ooo yyyy yyyy uuuu uuuu xxxx xxxx`
CMP	$\mathrm{R}_x <=> \mathrm{R}_y + \mathtt{u}_8$	`cc01 1ooo yyyy yyyy uuuu uuuu xxxx xxxx`
ADDI	$\mathrm{R}_x + \mathtt{i}_{21}$	`cc1i iiii iiii iiii iiii iiii xxxx xxxx`

Three opcodes produce output.

- STOP: Stop the processor, to accept another event.
- HIST: Increment a histogram counter.
- PHA: Submit the PHA data set to a PHA buffer.

Please refer to the *Solar Orbiter Level 3 Trigger Instruction Set Manual* for details. An assembler, disassembler and simulator is available.

4.3.1 HIST Instruction

The HIST opcode computes the same value as an ADD or SUB instruction. The result is taken as the memory address in external SRAM. The memory location is read, incremented by one and the result written. The memory access is EDAC protected, the histogram counters are 26 bits. The increment is checked for overflow, the counters do not wrap.

Bits [17:0] of the memory address are the result of the HIST computation, the MSB is $\texttt{hist base}_{[18]}$=$\texttt{enables}_8[3]$. $\texttt{hist base}_{[18]}$ shall be toggled every second. The PPS scheduler shall issue a command to the data product scheduler that clears the memory areas used for histogramming after the data products were extracted.

The HIST opcode executes an ADD because it may be common to add two $\log_2$ energy values as an index into 2-dimensional histograms, or add a histogram index to a base address.

4.3.2 PHA Instruction

The PHA opcode computes an ADDI. Four LSB bits of the result are transmitted to the PHA storage unit to select the PHA buffer where the data shall be stored, if space permits.

4.4 Streaming mode

Do not enable streaming mode, unless you are prepared to handle a lot of packets on the serial line. Disable all other telemetry sources when you do.

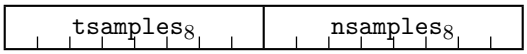
4.5 Sample Mode

When sample mode is enabled, the analog board sends sample packets instead of event pulse height packets. Three conditions must be true to activate sample mode,

- the bit $\texttt{confs}_{16}[1]$ must be set,
- the register $\texttt{nsamples}_8$ must be nonzero, and
- the register $\texttt{tsamples}_8$ must be nonzero.

$\texttt{nsamples}_8$ is the number of sample packets that shall be sent for each enabled L2 trigger, and $\texttt{tsamples}_8$ is the mask of L2 triggers enabled for sample mode.

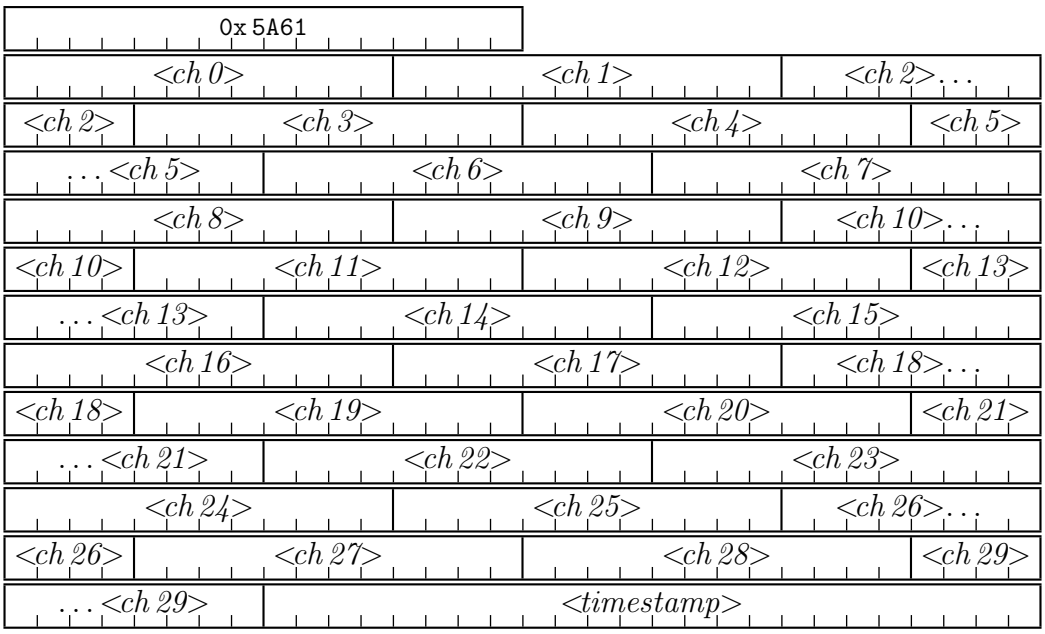
Both registers are part of the filter configuration, at address `0x0107` in the analog board command address space



A sample packet includes a timestamp with μseconds resolution, and 30 ADC readings sampled simultaneously from the channel ADCs. Recording about 16 consecutive sample packets captures the pulse shapes that were analyzed for the peak heights that lead to the trigger.

The L3 trigger probably cannot do much with the sample packets, except to store them in a PHA buffer.

Table 12: Sample Packet Format



4.6 Timestamp Clock

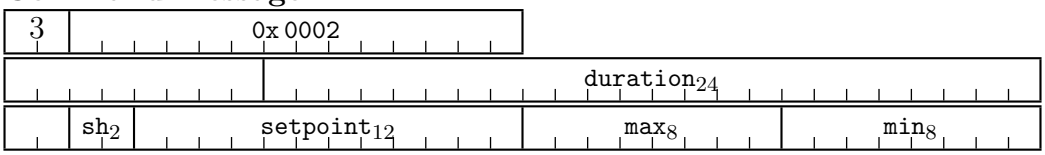
The analog board FPGA has a 32-bit clock incrementing at 1 MHz when enabled. The timestamp is included in sample packets. The clock can be reset with $\texttt{strobes}_{64}[3]$.

5 Operational Heater

To drive an operational heater, the digital board FPGA provides a PWM output. At a frequency of 125 kHz the output can drive a square wave with duty cycle adjusted in 192 steps from 0 % to 100 %. The output shall be filtered to provide a DC voltage to the gate of a heater n-MOSFET.

The PWM output cannot be turned on permanently. A command message needs to be send periodically to keep it operating

Command message:



The heater will be turned on for $\texttt{duration}_{24}$ cycles. The duty cycle will be calculated from the $\texttt{temp}_{12}$ register and the parameters of the message.

The $\texttt{temp}_{12}$ is assumed to hold a temperature reading spied from a recent analog housekeeping reading. The lower the value of $\texttt{temp}_{12}$ the higher the temperature. The duty cycle is calculated by the formula

$$\texttt{dutycycle} = (\texttt{temp} - \texttt{setpoint})/2^{\texttt{sh}}$$

The result is clipped to the range $\texttt{min}_8$ to $\texttt{max}_8$. If $\texttt{min}_8$ and $\texttt{max}_8$ have the same value, the duty cycle is set to that fixed value. Values from 192 up result in 100 % duty cycle.

The duty cycle is computed once when the message is received and will not update when $\texttt{temp}_{12}$ changes.

5.1 Heater Modes of Operation

The heater can be autonomously operated by the sensor unit. The PPS schedule may include commands for the temperature readout and to kick the heater periodically. In case of a temperature sensor failure the heater will misbehave, but the probability for such a failure is low, and the consequences not likely dramatic.

Alternatively, the ICU shall analyze the sensor housekeeping readings periodically, figure out if the sensors deliver plausible results and kick the heater via command messages. The ICU can consider sensor readings from multiple temperature sensor.

A combined approach requires the ICU to monitor the housekeeping readings and heater behavior and reconfigure the sensor in case of anomalies, e.g.,

select a different heater. Reconfiguration involves manipulation of the PPS schedule, which is non-trivial for the ICU to do autonomously. There may be a few canned sequences for switching between two temperature sensors or to turn the heater off.

6 Count Rate Series Compression

The main data products of the sensor are time series of count rates. The L3 trigger classifies energetic particle triggers and counts the occurrences of triggers of a large number of classes for consecutive periods of one second. The checkout units needs to reduce the information in these counters to telemetry products of a few hundred bits per second. This reduction is achieved in three steps:

1. Binning of the trigger classes into fewer, physically relevant particle classes.
2. Rebinning of the resulting time series to longer cadence.
3. Representation of the counter values in a compact format, which involves removing statistically insignificant information.

The reduction process is controlled by the data products scheduler, and executed by the histogram window unit and the compression unit.

6.1 Histogram Windows

The counters accumulated during data acquisition are mostly one or two dimensional histograms. The events are classified by one or two energy parameters, binned equidistantly in units $\log_2(E)$. The rebinning will provide projections of the data on physically relevant dimensions, or specific selections of particle types.

Finer binned products will be produced with longer cadence, some smaller sets will be provided with higher time resolution.

6.2 Counter Cadence

Governing the cadence periods of the data products is the modulus unit. This is a set of seven small counters that divide time into eight cadence units, each being a multiple of the previous. At the root is the period of the PPS signal from the ICU. The $\texttt{modulus}_3$ value is the level of the longest cadence interval that ended with the currently processed acquisition. All shorter cadences end

at the same time. The modulus unit also provides the information to the compression unit which cadence interval begin with the current acquisition.

The PPS scheduler uses the modulus unit to qualify tables entries for execution.

Table 13: Cadence Modulus Table.

`mod`	div	cadence
0		1 second, no summing
1	5	5 seconds
2	2	10 seconds
3	3	30 seconds
4	2	1 minute
5	5	5 minutes
6	2	10 minutes
7	6	1 hour

6.3 Compression Unit

The heart of the compression unit is a machine that can execute seven basic operations. The input is a histogram window sum acquired during the last second. The unit further computes and uses three numbers stored in external memory separately for each data product, to save state from one acquisition to the next.

The encoding scheduler controls which of the basic operations need to be executed for the given data product at the current modulus.

6.3.1 Encoder

The very central part of the compression unit is the encoder, which computes a variable length encoding of an approximation of a signed integer value.

The encoder operates with one parameter $\texttt{drop}_2$, governing how many LSB of the input value will be omitted from the encoded result. When this parameter is zero, the result will include representation for all bits that are significant when the Poisson distribution applies. The parameter can call for up to three more bits to be dropped.

An input value of zero will be encoded in a single bit. With $\texttt{drop}_2$ larger than zero, a range of inputs around zero will be represented as zero. Larger inputs will be represented by increasingly larger representations, about as large as the number of significant bits of the original value.

The encoded representation is sent to the telemetry FIFO.

A second output of the encoder unit returns the exact value represented by the encoded bits, i.e., the number that the decoder finds in the received telemetry.

6.3.2 Basic Encoder Operations

Three numbers are stored in external memory for every data product to compute sums and compressed time series data

- A the accumulator to build the sum of consecutive acquisitions,
- $-L$ minus the number of counts that the receiver on the ground decodes from the telemetry for the previous acquisition cadence,
- R the residue, i.e., the number of counts that should have been included in the number last sent, but were lost due to the encoding.

The encoder machine is started by issuing a nonzero $\texttt{op}_3$ instruction, and with input value D. The operations are

1: Encode the residue at the end of a compression cadence

$$E = \text{encode}_0(R)$$

2: Encode an intermediate value in full time resolution

$$\begin{aligned} Q &= D + R - L \\ E &= \text{encode}_3(Q) \\ R &= Q - \text{decode}(E) \\ -L &= -L - \text{decode}(E) \end{aligned}$$

3: Encode the first value of a compression cadence in full time resolution

$$\begin{aligned} Q &= D \\ E &= \text{encode}_0(Q) \\ R &= Q - \text{decode}(E) \\ -L &= -\text{decode}(E) \end{aligned}$$

4: Add to the accumulator

$$A = A + D$$

5: Initialize the accumulator for a new summing cadence period

$$A = D$$

6: Encode an intermediate sum

$$\begin{aligned} Q &= A + D + R - L \\ E &= \text{encode}_3(Q) \\ R &= Q - \text{decode}(E) \\ -L &= -L - \text{decode}(E) \end{aligned}$$

7: Encode the first sum of a compression cadence

$$\begin{aligned} Q &= A + D \\ E &= \text{encode}_0(Q) \\ R &= Q - \text{decode}(E) \\ -L &= -\text{decode}(E) \end{aligned}$$

All E or some A may be sent to the telemetry FIFO, at appropriate times. A, $-L$, and R are read from, and/or stored in external SRAM to be available for the next acquisition period, as indicated.

There is a special case: if $-L \geq -8$, then $-L$ will be set to zero. The next value will be transmited as absolute value, not a running difference. This is to avoid oscillations in the data stream with low count rates.

(Early versions of the module were treating small L differently: if $-L$ was zero, or not read at all, and Q is so small that it encodes to the smallest representation, but not zero, and $\texttt{drop}_3$ is nonzero, then the new $-L = 0$.)

6.3.3 Cadence Sequence without Compression

For a data product without compression, i.e., $\texttt{sum}_3 \geq \texttt{enc}_3$, only $\texttt{op}_3$=4 or 5 will be called. In the first or only second of a summing period $\texttt{op}_3$=5, all further seconds $\texttt{op}_3$=4. In the last second, the resulting A will be converted to one of four possible representations, and put into the telemetry FIFO.

This is useful for very low cadence data products or for data products that shall be analyzed on board the S/C, e.g., for burst mode trigger evaluation.

The available representations were described in section 3.3.3 on page 19.

6.3.4 Compression Sequence

For a data product with full time resolution a value will be sent every second, with compression. The data are counters which are expected to deliver almost the same number of hits every second distributed according to the Poisson distribution. This gives two handles for compression:

- The lower half of the bits representing the counter are noise and can be dropped.
- By computing running differences, the numbers become smaller.

The stream of counters is divided into encoding periods by the $\texttt{enc}_3$ parameter of the data product. The encoding period is started by sending a compressed representation of the first value without dropping excess bits using $\texttt{op}_3$=3

$$\begin{aligned} Q &= D \\ E &= \text{encode}_0(Q) \\ R &= Q - \text{decode}(E) \\ -L &= -\text{decode}(E) \end{aligned}$$

Two number are kept: R is the difference between the encoded value D and the number represented by the encoding. This number will be added to the next value, so the counts will not be forgotten. L is the encoded number, i.e., the value that is received for this counter on the ground after decoding the telemetry.

For the following counter values the difference between the new value and the last will be encoded, plus the residue that was dropped by the encoder. The last value that is subtracted must be the value known on the ground L, not what we liked to have sent, so that it can be added when decoding.

$$\begin{aligned} Q &= D + R - L \\ E &= \text{encode}_3(Q) \\ R &= Q - \text{decode}(E) \\ -L &= -L - \text{decode}(E) \end{aligned}$$

Q is the new value that we want to send. It will be encoded with three extra bits dropped.

When the last encoded value was send, one further number will be encoded and sent to the telemetry packet, so there is one more number in the data stream than there were counter values

$$E = \text{encode}_0(R)$$

The last residue R is send without dropping bits. For very low count rate products, the stream will likely contain only zeros. One or two hits during the encoding periods will be dropped. At the end of the period those hits have accumulated in the residue R. This last encoding makes sure they are not lost. The time resolution degrades to the encoding period, but that cannot be a problem for count rates that low.

6.3.5 Compression Sequence with Summing

The same compression sequence will be performed for data with reduced time resolution, except that each value that is submitted to the procedure is the sum of a series of inputs.

Each summing period starts with $\texttt{op}_3$=5 to initialize the accumulator with the first D, followed by the required number of $\texttt{op}_3$=4 to compute the sum. The last input value is fed into an encoding operation. The first sum is encoded without loss of bits using $\texttt{op}_3$=7

$$\begin{aligned} Q &= A + D \\ E &= \text{encode}_0(Q) \\ R &= Q - \text{decode}(E) \\ -L &= -\text{decode}(E) \end{aligned}$$

The remaining sums use $\texttt{op}_3$=6, which does it all

$$\begin{aligned} Q &= A + D + R - L \\ E &= \text{encode}_3(Q) \\ R &= Q - \text{decode}(E) \\ -L &= -L - \text{decode}(E) \end{aligned}$$

When the last sum of a compression period was encoded, the residue R will be immediately encoded and sent as well, as described above.

6.3.6 Encoding scheduler

The compression unit is driven by the encoding scheduler, which is kicked into action when the histogram window unit delivers a window sum. Depending on the $\texttt{sum}_3$ and $\texttt{enc}_3$ parameters of the data product and the status of the modulus unit, the scheduler decides which operation(s) the compression unit needs to perform. Secondly, it decides if and how the results shall be transfered to the telemetry FIFO.

Only when the histogram window unit, the compression unit and the encoding scheduler finished their action will the data product scheduler initiate processing of the next data product, or signal to the telemetry FIFO to close the packet and send it out.

6.4 Encoding Format

The format of the encoder output may be best visualized with a table, Tables 14 and 15.

This first column is a range of input numbers, the third shows the resulting bit pattern. The second column shows the encoding error. The last two columns compare the length of an unsigned number of that magnitude with the length of the encoding.

Table 14: Encoding Format without Drop

input	error	encoding	input/output length
0	±0	`0`	0/1
1-15	±0	`1s0xxxx`	4/7
16-31	±1	`1s10xxx-`	5/7
32-63	±2	`1s1100xxx--`	6/9
64-127	±4	`1s1101xxx---`	7/9
128-255	±4	`1s11100xxxx---`	8/11
256-511	±8	`1s11101xxxx----`	9/11
512-1023	±8	`1s111100xxxxx----`	10/13
1024-2047	±16	`1s111101xxxxx-----`	11/13
2048-4095	±16	`1s1111100xxxxxx-----`	12/15
4096-8191	±32	`1s1111101xxxxxx------`	13/15
8192-16383	±32	`1s11111100xxxxxxx------`	14/17
16384-32767	±64	`1s11111101xxxxxxx-------`	15/17
32768-65535	±64	`1s111111100xxxxxxxx-------`	16/19
65536-131071	±128	`1s111111101xxxxxxxx--------`	17/19
...			

6.4.1 Encoding in Full Poisson Resolution

The number zero is represented by a single bit `0`.

All other pattern must start with a `1`, followed by the sign of the input value `s`. The rest of the pattern represents the absolute value of the input.

The length of the encoding depends on the number of bits needed to represent the input. It is encoded in a stream of `1`-s followed by a `0`, followed by another bit that tells if the length of the input value is even or odd.

After this length encoding follow a number of bits from the binary representation of the absolute value of the input, starting with the bit after the leading `1`, obviously. These are marked `x` in the table. The dashes `-` represent the bits from the input number that are not included in the encoding.

The first three lines represent special cases. Numbers in the range $-15 \ldots 15$ are represented exactly, encoded in seven bits. Numbers up to 31 are also encoded in seven bits but loose the LSB.

This encoding is denoted $\text{encode}_0()$ in the previous section.

Table 15: Encoding Format with Drop=3

input	error	encoding	input/output length
0-3	±3	`0`	2/1
4-15	±4	`1s0x---`	4/4
16-31	±8	`1s10----`	5/4
32-63	±16	`1s1100-----`	6/6
64-127	±32	`1s1101------`	7/6
128-255	±32	`1s11100x------`	8/8
256-511	±64	`1s11101x-------`	9/8
512-1023	±64	`1s111100xx-------`	10/10
1024-2047	±128	`1s111101xx--------`	11/10
2048-4095	±128	`1s1111100xxx--------`	12/12
4096-8191	±256	`1s1111101xxx---------`	13/12
8192-16383	±256	`1s11111100xxxx---------`	14/14
16384-32767	±512	`1s11111101xxxx----------`	15/14
32768-65535	±512	`1s111111100xxxxx----------`	16/16
65536-131071	±1024	`1s111111101xxxxx-----------`	17/16
...			

6.4.2 Encoding with Drop=3

Table 15 shows the ecoding with $\texttt{drop}_2$=3, i.e., each line has three fewer `x` and three more `-`.

This encoding is denoted $\text{encode}_3()$ in the previous section. The compression scheme in this application does not use $\texttt{drop}_2$=1 or $\texttt{drop}_2$=2.

6.4.3 Decoding

The decoder assumes that the first dash in a pattern is a zero, and all futher dashes represent ones, i.e., it rounds pessimistically. The exception is the non-normalized case of the second line, `1s00---`. With $\texttt{drop}_2$=3 the three dashes represent `101`, i.e, the value 5.

6.4.4 NaN

In the case $\texttt{drop}_2$=0, two bit pattern represent redundant zeros: `1s00000`. The encoder does not generate these patterns.

The largest pattern emitted are 27 bits long

```
1s1111111111101xxxxxxxxxxxx------------
```

for inputs in the range 33554432...67108863.

7 EEPROM

Each sensor unit is equipped with an EEPROM of size 128 kBytes, for persistent storage of the sensor configuration. Storage space in the ICU is not appropriate for two reasons: there is not enough space available, and there are two ICUs where it is difficult to keep the configuration stores consistent.

The EEPROM content is not used directly by any logic in the FPGA. The ICU is assumed to read the content at boot, check if it is valid, and use the information to configure the sensor.

The EEPROM is connected to the data and address lines that also drive the external SRAM, so that any access needs to go through the same memory port arbiter. Most memory access ports to not provide access to the EEPROM address space, the two exceptions are

- the memory readout unit described above, and
- the EEPROM page write unit.

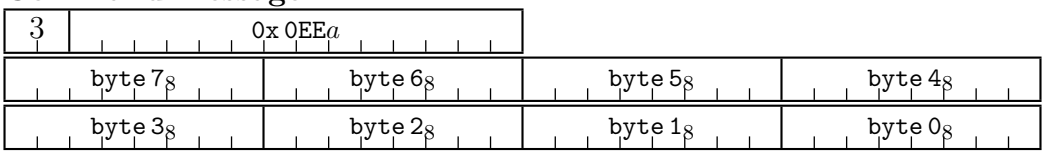
7.1 EEPROM Page Write

The EEPROM chip is organized in pages of 128 Bytes. The page write unit can write to a range of Bytes in a single page. It automatically performs the magic incantations to turn the EEPROM into write mode, writes the desired data, and waits for the prescribed period of time for the EEPROM to burn the data into the storage cells.

7.1.1 Page Upload

A range of 16 command message addresses is used to upload data into the 128 Byte page buffer in FPGA internal memory. This memory is not EDAC protected. The messages must be tagged for 64 bits data size, as each uploads eight bytes. The bytes are ordered little endian.

Command message:

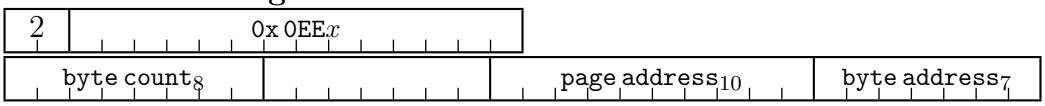


a represent four MSB [7:3] of the address in the page. The whole page can be uploaded with 16 messages.

7.1.2 Page Write

A message to any address in the same range, but with size tagged for 32 bits data submits the page to the EEPROM

Command message:



This command will be ignored if it is received less than 16 milliseconds since the previous page write. The datasheet claims that the EEPROM needs at most 15 milliseconds to complete a write. The flight EEPROM chip provides a separate pin to indicate if it is busy. This pin is not used.

The page write is performed in four phases:

- A series of magic writes turns the EEPROM into write mode.
- Writing bytes to a range of addresses, all in the same page, i.e., only the 7 LSB may change.
- About 100 μs of inactivity convince the EEPROM that the page is complete, so it starts the burning process.
- The actual transfer to the persistent storage takes up to 15 ms.

Obviously, the FPGA must perform the first two actions and then wait.

Writing data bytes starts at the page address$_{10}$,byte address$_7$ given. and proceeds for byte count$_8$ bytes, incrementing only the byte address$_7$, which may wrap around. At least one byte will be written, even if byte count$_8$ is zero. It is not an error to wrap around and to write previously written bytes again, but pointless.

The bytes will be read from the page buffer at the byte address$_7$, so if a partial page shall be written, the data needs to be uploaded to the corresponding location in the buffer. With byte count$_8$>= 128, the initial byte address$_7$ does not matter much.

7.1.3 Access Time

The memory driver implements access cycles that allow to drive an EEPROM chip with up to 250 ns access time.

8 Inputs and Outputs

The I/O ports are

- clock input,
- ICU asynchronous serial communication,
- synchronous serial ports to the analog board,
- external memory bus, and
- PWM output for the operational heater.

Table 16: Input and Output Ports

Name	Dir	I/O Standard	EM pin	Description
xclk	in	CMOS	101, CLKFP	48 MHz clock
ARxD	in	LV-PECL	217/216, HCLK	deserializer data
ARxC	in	LV-PECL	229/228, HCLK	deserializer clock
ATxD	out	LV-PECL	210/211	serializer data
ATxC	out	LV-PECL	208/209	serializer clock
Cx[1:2]	in	CMOS	147,119	PPS inputs
Rx[1:2]	in	CMOS	148,122	UART inputs
Tx[1:2]	out	CMOS	125,117	UART outputs
TxE[1:2]	out	CMOS	123,118	LVDS driver enables
RAMD[31:0]	inout	CMOS	...	SRAM/EEPROM address
RAMA[18:0]	out	CMOS	...	SRAM/EEPROM data
RAMnCE[0:3]	out	CMOS	242,240,47,50	SRAM chip selects
RAMnWE[0:3]	out	CMOS	51,234,112,113	SRAM write enables
RAMnOE	out	CMOS	241	SRAM output enable
EEPROMnCE	out	CMOS	38	EEPROM chip select
EEPROMnWE	out	CMOS	25	EEPROM write enable
EEPROMnOE	out	CMOS	34	EEPROM output enable
EEPROMnRES	out	CMOS	24	EEPROM reset output
EEPROMBUSY	in	CMOS	n/a	EEPROM busy
OHPWM[1:2]	out	CMOS	184,185	heater PWM output

8.1 Power

The RTAX FPGA core is powered with 1.5 V, all I/O banks are powered with 3.3 V.

8.2 The clocks

An external crystal oscillator is driving a CLKP input with LV-CMOS signaling. The frequency is 48 MHz. This clock is driving a global HCLK via an HCLKINT buffer. This clock is called `xclk`.

The input clock is devided by two, the 24 MHz clock is driven to another global HCLK called `mclk`.

The `xclk` is used to drive the synchronous serializer towards the analog board and so provide a clock to the analog board FPGA.

The HCLK inputs are used for the synchronous deserializer inputs from the analog board. The deserializer uses the clock received there.

The `mclk` drives all other logic.

8.2.1 TODO

The EM digital board uses the CLKFP input for the master clock. Future revisions may move it to an HCLK input. The `ARxD` input need not use an HCLK.

This move has not been implemented for the EM2 model RTAX proto board, and will most probably not make it into QM and flight models.

8.3 ICU Asynchronous Serial Link

Communication with the ICU requires four LVDS inputs and two LVDS outputs. The RTAX2000 chip supports the LVDS I/O standard, but we will not use that for two reasons.

- LVDS requires an I/O supply voltage of 2.5 V.
- There is a perceived risk to connect an external I/O line directly to the FPGA.

External LVDS drivers/receivers convert CMOS levels to/from LVDS.

The ports are

- the PPS receivers `Cx[1:2]`,
- the UART receivers `Rx[1:2]`, and
- the UART transmitters `Tx[1:2]`.

8.3.1 ICU Cold Redundancy

Two sets of LVDS lines connect to the cold redundant ICUs. The drivers for each set share an LVDS driver/receiver chip. Each chip includes two drivers and two receivers, one driver is unused.

8.3.2 Driver Enable

The LVDS driver can be disabled by the FPGA. These enables are connected to the inverted value of $\texttt{confs}_{16}[5{:}4]$. When all bits in $\texttt{confs}_{16}$ are reset at boot, the drivers are enabled. The running ICU may set the respective bit to disable the driver for the other ICU, to save power. This is not implemented on the PQM.

8.3.3 LVDS Termination

The standard LVDS specification calls for loop current $I_\mathrm{L} = 3.5\,\mathrm{mA}$ terminated at the receiver with $R_\mathrm{T} = 100\Omega$. The flight LVDS drivers will be implemented with two chips UT54LVDM055LV. These chips drive a loop current of $I_\mathrm{L} = 10\,\mathrm{mA}$, and are supposed to be terminated with $R_\mathrm{T} = 35\Omega$.

For interoperability with GSE and to minimize EMI emissions, the termination circuit shall be implemented as schown in Fig. 1, with en extra shunt at the driving end, and a center tap on the receiver termination.

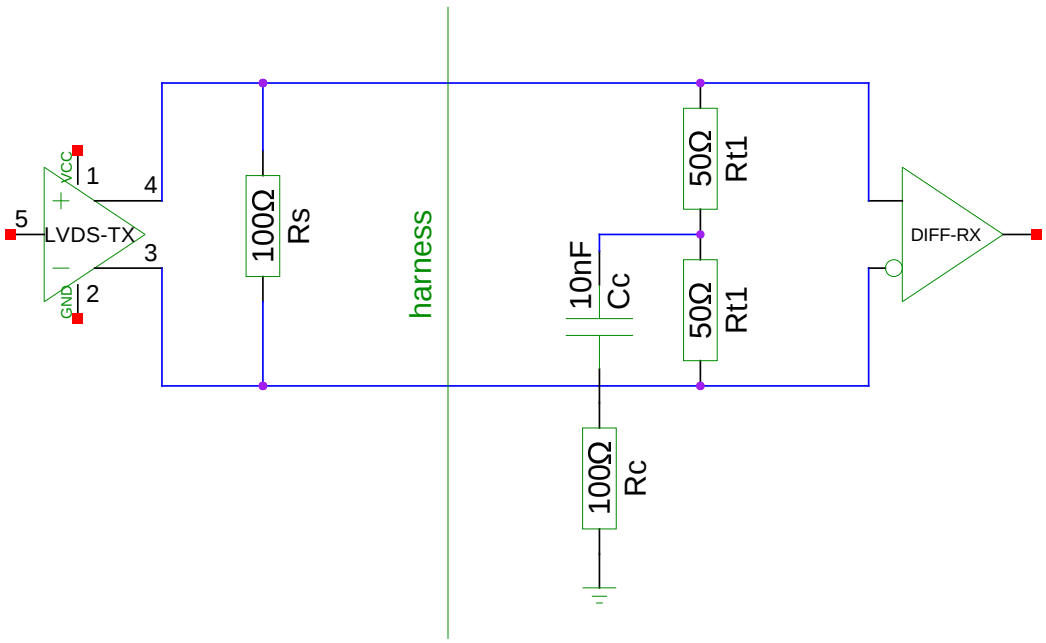


Figure 1: Proposed LVDS Termination Circuit.

The shunt limits the loop current to $I_\mathrm{L} = 5\,\mathrm{mA}$ and the voltage swing to 500 mV, closer to the standard. It also provides a backtermination to swallow any backward traveling signal, from EMI or reflections.

The center tap on the split termination resistor terminates limits the

common mode EMI at high frequency, to avoid stress on the reciever and EMI susceptibility.

8.4 Synchronous Serial Ports

The communication with the analog board is accomplisched with synchronous serial ports, with a data and a clock signal in each direction. LVDS signaling shall be used to minimize EMI with the analog circuits, but proper LVDS is not available. The RTAX FPGAs provide an LV-PECL I/O standard to be used with 3.3 V on the power lines.

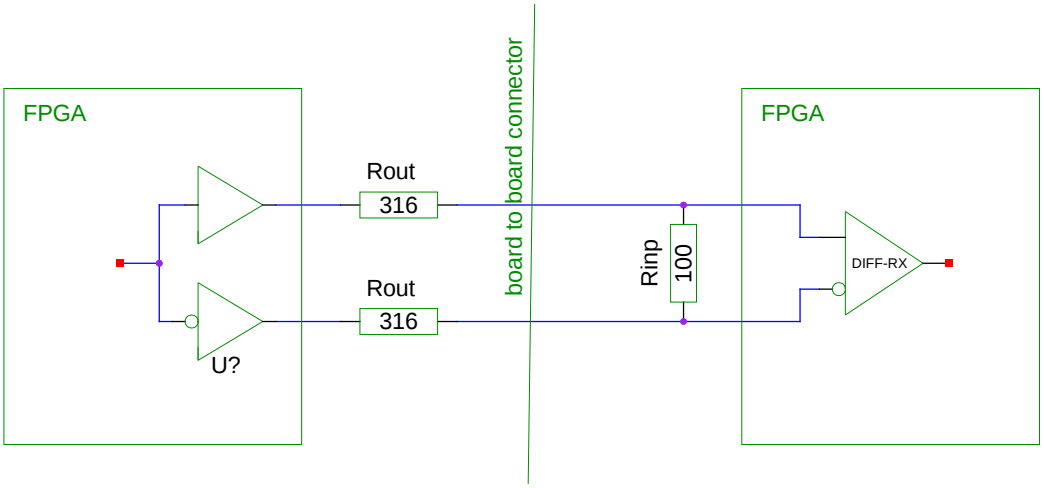


Figure 2: LV-PECL Termination Schematic

8.4.1 Termination

The LV-PECL I/O standard will be used with slightly nonstandard voltage swing and termination, to reduce power consumption and EMI. The outputs have series termination resistors with $R_{\text{out}} = 316\,\Omega$, the input is terminated with $R_{\text{inp}} = 100\,\Omega$. The current in the loop is

$$I = \frac{V_{\text{ccIO}}}{2R_{\text{out}} + R_{\text{inp}}} = \frac{3.3\,\text{V}}{732\,\Omega} = 4.5\,\text{mA}$$

The voltage swing is

$$\Delta V = \pm I R_{\text{inp}} = \pm 450\,\text{mV}$$

which is well beyond the requirement mentioned in the RTAX datasheet.

8.5 Memory Bus

The external memory bus includes

- address outputs RAMA[18:0],
- bidirectional data port RAMD[31:0],
- low active chip enables separate for SRAM and EEPROM, RAMnCE[0:3] and EEPROMnCE,
- low active write enables, RAMnWE[0:3] and EEPROMnWE,
- low active output enables, RAMnOE and EEPROMnOE,
- a reset output for the EEPROM, EEPROMnRES.

The flight EEPROM chip provides a busy indicator, EEPROMBUSY, and a reset pin EEPROMnRES. Engineering EEPROM do not provide EEPROMnRES nor EEPROMBUSY. The memory driver does not make use of them.

The EEPROMnRES is controlled by $\texttt{confs}_{16}[7]$, i.e., the EEPROM will be enabled when $\texttt{confs}_{16}$ is cleared at boot. The state of EEPROMBUSY is available in the register 0 readout.

The EEPROM in the PQM is not functional, because the write cycle time of the driver is too short. It cannot be written.

The flight SRAM can be addressed byte-wise. The FPGA does not use that capability. The four chip and write enables emit the same internal signals. All access is full 32 bits wide.

8.5.1 Engineering Model Memory

Prototype and engineering models with RTAX FPGA use the board layout for flight, but with the memory chips replaced by non-flight parts on a daughter board. The daughter board is soldered on the footprint of the flight SRAM and holds both a 16-bit SRAM and an EEPROM chip.

The EEPROM chip is functionally equivalent. The SRAM chips requires a different memory driver that performs two physical 16-bit accesses for one logical 32-bit access.

The additional address bit and the control signals are mapped to the control pins of the fight SRAM footprint. The non-flight chips share the write and output enables. This is supported by the mode in which the memory drivers work.

8.5.2 Engineering Model 2 Memory

The second EM model uses the same non-flight memory chips, but two of them, for real 32-bit operation. This setup operates with the flight mem-

Table 17: Engineering to Flight Mapping of Memory Ports

EM	FM	FPGA	EM	FM	FPGA
A[0]	A[0]	57	D[0]	D[5]	83
A[1]	A[1]	58	D[1]	D[6]	110
A[2]	A[2]	58	D[2]	D[7]	111
A[3]	A[3]	60	D[3]	D[4]	82
A[4]	A[4]	52	D[4]	D[3]	81
A[5]	A[5]	46	D[5]	D[2]	68
A[6]	A[6]	45	D[6]	D[1]	53
A[7]	A[7]	44	D[7]	D[0]	56
A[8]	A[8]	27	D[8]	D[8]	15
A[9]	A[9]	30	D[9]	D[9]	14
A[10]	A[10]	35	D[10]	D[10]	13
A[11]	A[11]	31	D[11]	D[11]	12
A[12]	A[12]	41	D[12]	D[12]	9
A[13]	A[13]	26	D[13]	D[13]	8
A[14]	A[14]	40	D[14]	D[14]	7
A[15]	A[15]	21	D[15]	D[15]	6
A[16]	A[16]	39	nOE	RAMnOE	241
A[17]	A[17]	235	nWE	RAMnWE[0]	51
A[18]	A[18]	116	RAMnCE	RAMnCE[2]	47
A[19]	RAMnWE[3]	113	EEPROMnCE	RAMnCE[0]	242

ory driver. This provides the necessary testing before the design shall be commited to flight chips.

8.5.3 Engineering Model 1 Memory

The first EM model has later been fitted with the same SRAM setup as EM 2, and with a PQM EEPROM. The model now uses the flight memory driver and confirms the functioning of the flight EEPROM.

8.6 PWM output

The PWM output for the operational heater is avialable in both polarities. The inverted pin is not connected on the digital board.

The idea was to use AC-coupling, rectification and filtering to drive the gate of an n-MOSFET. The heater would turn off if the output is stuck either high or low. The EM board layout supports such a circuit. But the gate voltage does not reach the required values.

The PWM output shall be filtered such that the mean output voltage drives the gate of the heater transistor.

8.7 Device Pinouts

...

9 Implementation Details

This section is decidedly incomplete. It may get expanded ...

9.1 External Memory Access

The digital board FPGA has access to external memory chips through a common bus. Several logic units need access to the memory for read and/or write. Memory arbitration units provide prioritized access to multiple memory ports.

During development of the logic, and for prototype models, several different memory architectures were employed. The arbitration units drive an abstracted memory interface to a memory driver. The driver can be transparently substituted for any physical memory architecture in use.

Some memory access need EDAC protection, some do not. Direct access is granted by the primary arbitration unit, which interfaces to the driver of the physical memory. EDAC protected access is granted by the secondary

arbitration unit, which interfaces to a virtual memory driver. The physical port of the virtual memory driver connects to a memory port of the primary arbitration unit.

The primary memory is accessed in units of 32 bit words. EDAC protected access yield words with 26 bits, six bits are used for EDAC redundancy.

9.1.1 Memory Port to Arbitration Unit Interface

The arbitration units are configured for the required number of ports. For each port a `req` input is provided, a high level indicates the desire to perform an access. Access is granted to each port via the `ack` output. When `ack` is high, the port must put the $\texttt{addr}_{19}$, `write`, and $\texttt{data}_{32}$ on the global bus.

For read access, the arbitration unit provides a `val` output to each port, to tell when the returned data is valid for that port. There is a global data output port $\texttt{q}_{32}$, and the data is valid for only one `mclk` cycle.

To avoid deadlocks, each port is supported with a `pending` output that indicates that a read is pending for that port. The idea is that `val` is used to trigger the use of the returned data, but use `pending` to perform state transitions in the scheduler. `pending` is implemented with a guarantee that is will deassert eventually. Waiting for a missed `val` may take forever.

9.1.2 Memory Driver Interface

The memory driver is signaled with a pulse on `go` from the arbitration unit to execute an access cycle. The driver responds by asserting `busy` until is is ready to accept the next `go`. `busy` will be deasserted for the cycle before the next `go` may be issued. If the cycle time is only two clock cycles the busy will not be asserted.

For read access, the driver issues a pulse on `valid` one cycle before the data on $\texttt{q}_{32}$ is valid. `pend` is asserted between `busy` and `valid` if necessary.

9.1.3 Memory drivers

Several memory drivers have been written, some of them used on prototype units.

`memasync32`: Asynchronous SRAM with 32 bit data bus. Two `mclk` cycles per access. Can drive the flight SRAM.

`memasync32ee`: Asynchronous SRAM with 32 bit data bus and EEPROM connected to the MSB of the data bus. This is the flight memory driver for the digital board. The driver needs two `mclk` cycles for SRAM access.

EEPROM access can be synthesized for 24 MHz or 48 MHz `mclk` to use 7 or 15 `mclk` cycles.

This driver is used on the second engineering module to drive a pair of 16-bit SRAM.

`memasync16ee`: Asynchronous SRAM with 16 bit data bus and EEPROM connected to the LSB of the data bus. This driver needs two physical SRAM accesses for one logical cycle. It is used on the engineering models where the SRAM is substituted by a 16-bit version.

The 24 MHz variant issues a physical access cycle in every `mclk`, it uses the `xclk` at 48 MHz to do the timing of the control signals. The driver is as fast as the flight driver in this case.

The 48 MHz variant needs two `mclk` cycles for each physical access, four cycles for a logical cycle.

EEPROM cycles are the same as for `memasync32ee`.

`memasync8`: This driver is used on SIRENA boards with a rather slow asynchronous SRAM with 8-bit data bus. A physical access cycle requires four `mclk` cycles, or one more to switch data bus direction. A logical cycle requires four physical cycles, obviously.

`memasync8m`: This driver is used on the FLYRENA prototype digital board. The SRAM chip is the same as on the SIRENA. 16 bits of the address bus are multiplexed on the data bus into external latches. A physical access cycle requires at least four `mclk` cycles, more if the address latches need update, or to switch data bus direction. A logical cycle requires four physical cycles.

`mem26edac`: Virtual driver, that connects the secondary arbiter for EDAC access to the primary arbiter. This driver also takes care of the hamming coding and error correction, but is does not automatically rewrite corrected errors into the SRAM.

9.1.4 Memory Port

Memory port modules interface between the clients and the arbitration unit. A client issues an access request with a pulse on `aen` and/or `den`. The memory $\texttt{addr}_{19}$ must be provided together with `aen`. `den` initiates a write cycle, the $\texttt{data}_{32}$ must be provided as the same time. `den` without `aen` reuses the last address. This is convenient for read–modify–write cycles.

The port module asserts `busy` until the request was accepted by the arbitration unit. For read cycles a `ppend` signal is asserted as well until `busy` is deasserted. `ppend` must be combined with the arbiters `pend` to wait for

outstanding reads to finish.

The memory port units take care to deliver $\texttt{addr}_{19}$, `write`, and $\texttt{data}_{32}$ to the arbiter and memory driver at the correct time.

9.1.5 Memory Port Priorities

The primary arbiter supports four memory ports, the secondary arbiter supports three EDAC protected ports, from highest to lowest priority

1. write only port for PHA data storage,
2. write only port for EEPROM page writes,
3. secondary arbiter for EDAC access,
   (a) histogram counter increment port,
   (b) data compression state memory port,
   (c) histogram window sum port,
4. read only memory telemetry readout port.

Data acquisition ports are higher priority than telemetry readout ports. The gaps between processed triggers should not interfere with the timely execution of checkout activity.

EEPROM write have a rather high priority, to make sure there are no gaps between write that make the EEPROM think the page is complete and can be burned. A histogram window sum could easily block access for long enough if it were higher priority than the EEPROM write.

9.1.6 Memory EDAC

Six Hamming code bits can protect at most 26 data bits, and can be stored together in a 32 bit memory word. The data bits are located in the physical memory word in their natural order, i.e., the binary digits of the bit number indicate the parity bits that the bit position contribute to. Bits 1, 2, 4, 8, and 16 contribute to only one parity, so these bits are five of the six hamming codes. Bit number 0 is the sixth parity bit, to which all bits contribute that have an even number of 1s in their bit number.

All physical bits contribute to an odd number of parity bits, the stored parity is even, i.e., when the parities are calculated for reads, they must come out zero. When a single bit is flipped, an odd number of parities are wrong. The five upper parities encode the flipped bit's number. When two bits are flipped, an even number of parities are wrong, indicating an uncorrectable error.

The data bits are at the bit positions with two or more 1s in their bit number, 3, 5–7, 6–15, and 17–31. All this is important to know if an EDAC protected memory is read for diagnostic telemetry.

9.2 UART

The *Universal Asynchronous Receiver and Transmitter* operate at the bit rate of 115 200 baud. Reprogrammable prototypes may be configured to operate at 1.5 Mbaud, or even 3 Mbaud with 48 MHz `mclk`, for calibration runs with high data rates.

The asynchronous serial receiver is designed to be very picky. The bit levels must be stable for half a bit period. This is to avoid accepting noise in the inactive line as input. The downside is that the bit rate must match the nominale rate of 115 200 baud within 1.5 %.

9.2.1 Baud Rate Generator

The bit clock is provided by the fractional baud rate generator. For the receiver, a clock with 16 times the bit rate is provided. The tranmistter is running from a clock with the nominal bit rate.

The generator has three parameters, `DIV`, `FRAC`, and `BAUDMODULO`. The input clk is `MCLK` = 24 000 000. The receiver bit clock is

$$16 \times \texttt{BAUD} = \frac{\texttt{MCLK}}{\texttt{DIV} + \dfrac{\texttt{FRAC}}{\texttt{BAUDMODULO}}}.$$

For the desired bit rate `BAUD` = 115 200, the parameters can be derived as

$$\begin{aligned} \texttt{DIV} &= \frac{\texttt{MCLK}}{16 \times \texttt{BAUD}} \\ \texttt{FRAC} &= \frac{\texttt{MCLK} \times \texttt{BAUDMODULO}}{16 \times \texttt{BAUD}} - \texttt{DIV} \times \texttt{BAUDMODULO} \end{aligned}$$

with `DIV` rounded towards zero. `BAUDMODULO` shall be chosen that the division in the formula for `FRAC` yields an integer number. With `BAUDMODULO` = 48 we get

$$\begin{aligned} \texttt{DIV} &= \frac{24\,000\,000}{16 \times 115\,200} = 13 \\ \texttt{FRAC} &= \frac{24\,000\,000 \times 48}{16 \times 115\,200} - 13 \times 48 = 1 \end{aligned}$$

The resulting baud rate is exactly 115 200 baud, as precise as the crystal oscillator. This must be hardwired into the FPGA, obviously.

The ICU cannot provide exactly 115 200 baud. Their algorithm is

$$\texttt{BAUD}(x) = \frac{1}{(x+1) \times 300\,\text{ns}},$$

and with $x = 28$, we get $\texttt{BAUD}(28) = 114\,942.53$ baud. The baud rate generator in the FPGA has been instantiated with the following parameters:

$$\begin{aligned} \texttt{BAUDMODULO} &= 240 \\ \texttt{DIV} &= \frac{24\,000\,000}{16 \times 114\,942} = 13 \\ \texttt{FRAC} &= \frac{24\,000\,000 \times 240}{16 \times 114\,942} - 13 \times 240 = 12 \end{aligned}$$

resulting in a bitrate of

$$\texttt{BAUD} = \frac{1}{16}\,\frac{24\,000\,000}{13 + \frac{12}{240}} = 114\,942.53.$$

This is close enough to work with GSE that provide exactly 115 200 baud, and matches the ICU bitrate exactly.

9.2.2 Abort Sequence

The logic in the FPGA has been carefully designed from the start to not ever lock up in an unresponsive state. Especially the UART receiver must have this property, since there is no RESET facility, except for a power cycle that can force the system out of such a state. When the UART and the message receiver respond, it is possible to send a master control message to reset any misbehaving unit, but that should never be necessary.

Well, you never know. The UART monitors the received input for a special sequence that will issue an `attn` signal to the core logic. The sequence consists of three character <*LF*>, 'A', 'T', followed by a <*break*>, i.e., a zero with a frame error. A <*break*> is not part of any nominal message, so the sequence cannot match by accident.

The `attn` signal has almost the same effect as a $\texttt{strobes}_{64}[0]$.

9.2.3 PPS Receiver

The PPS receivers performs stringent checks before the pulse is accepted and forwarded to the core logic.

- The length of the pulses must be between 1 μs and 500 μs.
- The time between pulses must be a multiple of a second within 5.5 ms.

To be precise, a rising edge on a `Cx` line is considered a valid PPS, if

- the time between the previous rising edge and the previous falling edge is at least 1 μs and at most 500 μs, and
- the time between the previous rising edge and this rising edge is a multiple of a second, with a total error of not more than 5.5 ms.

The delivery of the PPS to the core logic is delayed by about ten `mclk` cycles, about half a μs.

There are two PPS receivers that are accepting pulses. When both receivers see good pulses, the selection will not switch between them, but continue to use the one that came online first. The period between PPS may be any multiple of a second. If the active source becomes quiet, the channel is still marked good, because no bad pulse was received. This prevents the switch to the receiver that is still sending.

This misfeature was corrected in version v04. A channel is considered inactive after two good pulses were received on the other channel since the last pulse was received.

9.3 Serializer/Deserializer

...

10 Changelog

Current SVN Revision 8147 .

SVN Revision 4005

Section 2.6.3
Update $\texttt{strobes}_{64}$ and $\texttt{enables}_{8}$ allocations.

Section 3.2
Update register readout allocations, scratch register, two UART status registers, move heater register.

Section 3.2.1
Add $\texttt{misc}_{8}$ register for chip periphery status, i.e., EEPROM BUSY.

Section 3.2.4
Move and update UART status register description.

Table 4
Add counter 60: Streaming mode: packet lost.

Section 3.4.5
Add $\texttt{scratch}_{64}$ register description.

Section 3.7.1
Update for register reallocation.

Section 4
Update for $\texttt{strobes}_{64}$ and $\texttt{enables}_{8}$ reallocations.

Section 6.3.3
Mention that there are four available representations for count rate numbers.

Section 8.3.2
Describe the LVDS driver enable bits.

Section 8.5
Describe the EEPROM nRES bit.

Section 8.5.3
Describe rework of EM 1 with PQM EEPROM.

Section 9.2.1
Update the baud rate generator parameters for the real ICU-supported values.

Since SVN Revision 2415

Section 9.2.3
Add note about changed PPS source selection.

Section 8.5.2
EM2 memory driver.

Section 3.2.4
Register 4 definition.

Section 2.6.5
Explain high speed calibration mode implementation. Removed the old section that outlined a different implementation.

Section 3.3.1
Sparce count rates via $\texttt{res}_{3}$.

Since SVN Revision 2268

Section 3.2.1
Use $\texttt{states}_{8}$ as FPGA and version identification.

Section 3.3
Science data header format.

Section 3.2.3
Explain time.

Section 3.3
Remove B bit, add $\texttt{flg}_2$.
FPGA design change: Data products header is always 8 bit, with init and fini modulus, and two informational flag bits.

Section 5
Fix: $\texttt{max}_8$ – $\texttt{min}_8$ were swapped.

Since SVN Revision 2249

Titlepage
Change Document code: SO-EPD-KIE-DA-0001 Issue 1 Rev 0.

Document
Minor spelling fixes and clarifications.

Section 3.2.1
$\texttt{errors}_8[5{:}4]$ documented.

Section 3.3.3
Add byte aligned, uncompressed histogram data.
Floating point format, 16-bit unsigned.

Section 4.3
Seven EDAC bits in L3 register words. Mention the instruction set manual.

Section 7.1.2
EEPROM page write waits 16 milliseconds.

Table 17
EM memory pins table fixed.

Section
Calibration Mode Analog Data Streaming added.
This section was later removed.

Section 9.1.3
Memory driver `memasync16ee` drives 16 data pins.
Memory driver `memasync8` is operational in SIRENA.

Since SVN Revision 2230

Section 3

Memory readout command is `0x 05`xx.

Since SVN Revision 2118

Titlepage

Add Document code: SO-EPD-KIE-DA-0001 Issue 0.

Since SVN Revision 2110

Section 8.3.3

LVDS Termination, added, with Fig. 1.