FPGA Design for the Solor Orbiter EPD Sensor Units HET/EPT and STEIN

Stephan I. Böttcher

Revision: 846
This document will grow.

The FPGAs for HET/EPT and STEIN shall implement all digital functions required in these sensor units.

1 Development process

The functions of the FPGAs will be expressed in the hardware description language Verilog. These functions will be simulated, both individual higher-level modules, as well as the whole module, with models for the connecting circuits.

Most of the functions will be synthesized, loaded and tested in sensor demonstrator units based on Altera Cyclon 3 FPGAs.

The development will follow a bottum-up fow, where indiviual subsystem modules will be developed, with major consideration of FPGA resource use at each level.

In the end we will write a user's manual, or datasheet describing all functions and configuration registers.

1.1 Simulation

We will use Iracus Verilog to simulate the designs during development.

1.2 Revision control

The Verilog HDL code and all supporting configuration files and scripts will be maintained in a subversion repository.

1.3 Demonstrator units

Altera Cyclon 3 based demonstartor units have been built for the EPT and HET sensors, with readout through an IRENA. Further demonstrtators are being developed for the three ASICs: VIRENA, RAL-SIXS, and IDEF-X. All these units may host test of modules for the flight FPGA.

2 Functional blocks

The three major functions are:

- Interfaces to the frontend.
- Data processing.
- Downstream interface and communication with the ICU.

In case we need to adopt a solution without ASICs for HET/EPT, there will be DIRENA-type filter and trigger modules to process the sensor signals digitally.

2.1 Frontend

The frontends include three different ASICs,

- VIRENA, for HET,
- RAL-SIXS, two instances for EPT, and
- IDEF-X, for STEIN.

Each sensor unit uses at least one ADC to capture analog signals from the frontend ASICs, and an ADC for housekeeping.

(The readout of the VIRENA and the IDEF-X may be implemented differentially with a pair of the ADCs foreseen for housekeeping. The housekeeping will use seven channels each, while channel 0 is used for science data.)

Each FPGA interfaces to an SRAM, 2 MBytes×8, to store the science data.

TBC: Operational heater control may be required.

2.2 Data processing

Three kind of processes are required,

- particle event processing,
- datacompression, packaging, and transmission, and
- slow control and configuration.

There will not be a CPU core.

2.2.1 Event processing

The analog frontend will trigger on particle hits in the ASIC. The FPGA needs to process each trigger, with the following steps:

- Read the hit register, i.e., which detector channels caused the trigger.
- L2 trigger, decide which detector channels shall be digitized, based on the hit register. This includes (anti-)coincidence logic for an early event classification.
- Digitization. Obtain digitized values for the energy deposit in the selected channels.
- Calibration of the peak heights, computation of logarithms.
- L3 trigger, final event classification based on the digitized peak heights.
- Histogramming, increment a histogram counter in the SRAM.
- PHA storage, save the event data in one of several buffers, depending on event class priority.

2.2.2 Histograms

The accumulated histograms are two-dimensional arrays of counters in the SRAM. The indices into the histograms are typically logarithms of energy values.

2.3 Compression and packaging

At the end of each acquisiton period of nominally 1 s, the SRAM is switched to the other of two banks for event processing. During that next acquision into one bank, the completed acquisition in the other bank is packaged for telemetry.

2.3.1 Histogram compression

The telemetry data product is a rather small list of counters. Each counter is computed as the sum of a rectangular region of a histogram. The idea is to upload a list of data products during unit configuration, in the form of indices, ranges and strides to compute these sums. (Address, number of x-bins, number of y-bins, y-stride.) The FPGA executes that list during checkout.

2.3.2 PHA data

The PHA store can be drained to fill excess telemetry allocation.

2.3.3 Lossy timeseries compression

Instead of fooling around with burst mode (or in addition to, but then it's the ICU's job) we can compress the data volume with a lossy timeseries compression in the FPGA. The idea is to only send the difference to the previous counter value for each data product. And to drop from that difference the non-significant bits in the poisson noise. The difference will be computed versus the sent value, not the original value, so that in the end no counts get lost. In regular intervals (minutes) the stream is closed by sending the residue, and reset to a starting value of zero, so that the next counter is sent full.

The counter values are encoded with a stream that is roughly as long as the number of bits to represent the number, except that half of those bits are header, and the least significant bits of the number are dropped.

In the agressive mode, all numbers between -3 and 3 are sent a a single bit 0. Nobody needs one second resolution when there are only a few hits per minute. The agressive mode relies on the fact that the differences themselves are almost completely noise most of the time, because they represent the poisson noise of the original counters. That means that a few more bits can be dropped.

The starting value and the residue must be sent in non-agressive compression mode to the real poisson limit. That way, all the single hits during the minute that were dropped, are accounted for.

For this compression we need to store the previously send counter value, and the residue that was not represented in that value for each data channel inside the SRAM.

This method has been implemented in Python. An HDL implementation is pending.

2.4 Downstream interface

The interface to the ICU are asynchronous serial lines, two redundant receivers and transmitters, plus redundant 1 Hz clock signals.

Communication happens via binary data packets, with sync headers and CRC checksums.

2.4.1 Protocols

Command packets are sent by the ICU to the sensor unit. The payload is small, up to ten bytes, because the FPGA tranforms the packets into parallel commands. Command packets may be sent to the sensor at any time and are executed immediately.

Commands that are not well formed or unknown are silently ignored. The sensor accepts command packets on both receiver lines at all times, but not simultaneously.

A valid, well formed command may or may not cause a reponse in the form of a sensor telemetry packet.

Sensor telemetry packets are sent from the sensor unit to the ICU. The payload may be larger, within limits of the capability of the ICU to process them. The payload size may depend on command or configuration parameters, there may be cases where it is the ICU's responsibility to make sure that the packet size stays within limits.

Generally, sensor telemetry packets are sent as direct response to a message from the ICU, either via a command packet or by the 1 Hz clock signal. Different modules in the FPGA may generate telemetry packages. It is the responsibilty of the ICU to ensure that no collisions happen. The required timing parameters shall be documented in a sensor users handbook. In most cases, command responses are generated faster than it takes to receive a new command, and buffered in a FIFO, so that collisions are not to be expected in normal cases.

2.4.2 Packet formats

The ICU proposed a packet format they called ITF for all packets between ICU and sensor units and between sensor units.

These FPGAs will *not* support receiving ITF packets, we implemented a similar format for command packets:

- 16 bit SYNC: 0x3c3d,
- 2 bit SIZE: 0, 2, 4, or 8 data bytes,

- 14 bit ADDR: command code,
- 0..64 bit DATA,
- 16 bit CRC.

There will be no need to send packets directly between sensor units. The sensor telemetry packet format is ITF as proposed by the ICU:

- 32 bit SYNC: 0xbebacafe,
- 3 bit FLAGS: unused,
- 13 bit SIZE: number of data bytes,
- 16 bit ADDR: response code,
- var DATA: data bytes,
- 16 bit CRC.

2.5 Design hardening

The design must operate reliably in hostile ratiation fields, without means to reset via external wires, apart from power cycle of the sensor unit.

2.6 Memories

We will evaluate the need to implement memory from SRAM blocks in a triple redundant fashion, with Hamming codes for Error Detection And Correction, or with parity, based on the radiation fields expected, and the lifetime of the stored data. Configuration data will need to be protected well, or rewritten from safer storage regularly. Transient data does not require protection against SEU. The processing units shall be designed to accept any data without corrupting the data flow.

We may ask the ICU to refresh all configuration of the unit regularly, a few registers each seconds. A register write with the same value shall not disrupt the operation of the unit.

2.7 Reset input

There will be none.

The design simulation will start with all registers undefined and is expected to start up to a know state in a finite number of clock cycles.

This does not work in case of some loops, which will be analysed carefully and initilized to some random value for simulation. E.g., a counter that is supposed to count to zero and then stop will not work in simulation when its value is undefined, but will work in hardware.

The serial receiver UART in particular does not need any such initialization. From completely undefined state it comes up into a state to properly receive serial bytes.

2.8 Reset sequence

Well, there will be sort of a reset pin after all. When a serial line receives a sequence of three bytes followed by a break, an attention signal is generated that will be used to reset the FPGA into a clean responsive state, whatever it might be currently doing. The idea is though, that this shall not be necessary, ever.

The sequence is: <LF> A T <BREAK>.

This feature needs to be negotiated with the ICU developers.

2.9 Redundant ICU interface

The ICU will only power one set of serial lines at a time. That means, the lines that are not powered may present random noise to the serial receivers. Both the serial UART and the 1 Hz clock receivers will be very selective before accepting valid data. This may limit the baud range at which the receiver can properly receive bytes frames. But since the magnetometer requires very tightly controlled crystal frequencies anyhow, this should not impose serious limitations.

The sensor units will send all telemetry packets on both redundant transmitters in parallel.

2.9.1 Fractional baud rate generator

The UARTs are driven with a fractional baud rate generator from the main FPGA clock. There is a module available to synchronise the baud rate to the 1 Hz clock when it is properly received, and reset it to nominal what no 1 Hz clock is available. Right now I consider the use of such a module too dangerous.

We may require the ICU to provide a means to change the bit-rate in small increments (0.5 %) to recover from clock stability failures in a sensor, in case such failure modes are non-negligible.

3 STEIN

3.1 IDEF-X

The IDEF-X controller supports four functions:

- Reset. The reset sequence according to the datasheet reset all configuration registers to their default values.
- Slow-control. Reading and writing of the configuration registers in the ASIC.
- Temperature readout. The builtin temperature sensor can be read with a special type of frame.
- Trigger processing. When enable, this function responds to triggers in the ASIC with a readout frame.

3.1.1 Event readout

The IDEF-X supports three modes of readout. Only two modes are supported by this module. Each mode starts with the serial readout of the 32 hit register bits, which tell which channels were hit. Subsequently, a certain set of channels may be read out via the analog multiplexer and ADC.

1. In the nominal mode, the ASIC automatically presents only the triggered channels on the multiplexer. This is the only usefull mode for physics data taking, because channels that did not trigger may present old charge, that is not related in time to the particle that hit the trigger.
2. All channels mode. This mode presents all 32 channels on the multiplexer. The mode works very similar to the nominal mode frame, so that the implementation of this mode is cheep. The readout mode is selected before the readout of the hit registers is complete, so that the L2 trigger cannot influence the mode selection. That makes this mode useless for physics data, but it may serve for tests.
3. Selected channels mode. This mode requires 32 extra serial bits to be sent to the ASIC to select which channels shall be read. This mode is

not implemenmted by this module, because it does not serve any useful purpose and presents extra cost in design complexity.

3.2 Timing

The timing parameters for communication with the IDEF-X are fixed with compile-time parameters, except for the multiplexer settle time. The fixed time include:

- Reset cycle timings.
- Bit clock frequency and pulse width.
- Multiplexer advance pulse width.
- Miscellaneous setup and hold times.

The multiplexer settle time is configurable at runtime up to 256 clock cycles. This is the time to wait at each analog multiplexer setting before that analog output is sampled, and the multiplxer is allowed to advance. This affects the temperature readings as well.

4 L2 trigger

The level 2 trigger decides which channels need to be digitized, if any. The input is a set of bits telling which trigger descriminators in the ASIC fired.

4.1 EPT and STEIN

EPT and STEIN require triggers of single detectors in anticoincidence to every other channel in the particular telescope. This calls for a rather simple L2 trigger, with most complexity in modes for testing and calibtration.

4.2 HET

HET will need a level 2 trigger similar to the one in MSL-RAD, to decide which channels need to be digitized for the event class.

5 L3 trigger

The level 3 trigger classifies the event for inclusion into a histogram, and provides the histogram indices.

5.1 EPT and STEIN

Since these telescopes are supposed to register single detector hits, the L3 trigger only needs to figure out which channel was hit the most, and if the other hits are compatible with crosstalk. I.e, if more than one channels in a telescope were hit, is the signal in one of them significantly larger than in the others.

For EPT we could accumulate a 2D histogram of the energy in the opposite central segments. The data products will then be drawn from the counters on the axis, and a few large debugging bins in the coincidence area.

5.2 HET

The HET will trigger several event classes, depending on the path, direction, and penetration depth of the particle. This will require some math, which will be mostly done in the $\log_2(E)$ space, with 8-bit representation of the energy in units $8\log_2(E)$.

One idea is to design a trigger processor with a very limited intruction required for L3 processing. The current list, very preliminary:

- ADDI $R_x + I$, add a signed constant
- MULI $R_x \times I$, multiply a constant
- LOG2 $(R_x << i_1) << i_2$, compute logarithm
- ADD $(R_x << i_1) + R_y << i2$
- SUB $(R_x << i_1) - R_y << i2$
- CMP $R_x + I <=> R_y$
- CMP $R_x <=> R_y + I$
- TRIM $\max(\min(R_x, I_1), I_2)$
- PHA $R_x + I$
- HIST $R_x + R_y << 8$
- STOP

All instructions are conditional on the result of the last CMP. We'll probably need a jump instruction. Each instruction leaves it's result in a register. There are 256 instruction slots, each with a register in core SRAM. Such a

machine will need four blocks of memory, two for the instructions, two for the registers. The digitization results will be preloaded into the registers at startup. Instructions and registers are 32-bit wide.

The multiplication will take a few clock cycles to complete. We may be able to pipline it, blocking only on another MULI instruction or on a target register access. MULI and ADDI are required for calibration.

PHA instruction instructs some external data buffer to push the event into a PHA buffer at the given address.

The HIST instruction provides the address of a histogram counter to be incremented. The TRIM instruction can limit the histogram boundaries.

The CMP instructions are optimized for $\log_2(E)$ arguments, with unsigned numbers. That's why there are two variants, to prevent negative numbers anywhere. CMP instruction and instructions with false condition copy the previous result.

Will 256 instructions be enough? How many such machines will we need? One per HET direction?

We will need a fifo of triggered events to derandomise the datastream.

6 DIRENA128

A DIRENA-type readout for flight, based on the ADC128S102 from National needs to be implemented with very limitted FPGA resources, compared to the Altera-base implementation. There will be no multiplier units, the RTAX2000 provides fewer SRAM blocks. There is a lot more to be done in the FPGA. The sample clock frequency is limited to 1 MHz. The HET/EPT units needs 30 channels.

A filter module has been developed for up to 36 channles, with a pipeline of sixteen samples, and 3-bit coefficients.

The sample data requires four RTAX2000 SRAM blocks, in a 64 words by 144 bits configuration.

6.1 Algorithm

At the sample frequency of 1 MHz two linear combinations over fifteen samples are computed for each channel. The coefficients are limited to the integer range -3 to 3. The computation performs at 48 MHz clock in serial adders.

Each channel keeps a record of the maximum peak height within a configurabe coincidence time window. When a recorded peek ages out of the coincidence window, the last reconstructed peak height value is recorded instead, until another local maximum is observed.

Each channel provides three numbers each microsecond, for consideration by the trigger logic.

- A (18 bits): linear combination A, peak height.
- B (18 bits): linear combination B, phase parameter.
- T (4 bits): peak age.

6.1.1 Digital filter

The linear combinations are computed as

$$A = \sum_{i=1}^{15} a_i S_i \quad (1)$$

$$B = \sum_{i=1}^{15} b_i S_i \quad (2)$$

With the 3-bit coefficients a_i and b_i, and the 12-bit ADC samples S_i. Each channel is using the same coefficients.

6.1.2 Digital peak detector

The age T tells when the recorded maximun was observed:

- $T = 0$: Falling edge. No maximum in A was observed in the window, the last reconstructed A,B are recorded, the last A is smaller than the previous one.
- $T = 1$: Rising edge. No maximum in A was observed in the window, the last reconstructed A,B are recorded, the last A is larger than the previous one.
- $T > 1$: The recorded A/B were observed T samples ago, there was no larger unmasked A observed in the window.

Caveat: A peak will be masked by a larger preceeding peak inside a coincidence window. The large first peak ages out, and the second one is forgotten.

6.2 Trigger

A trigger module has been implemented that provides two trigger levels L1 and L2. This trigger module is deadtime free. It can serve multiple sensors independently.

6.2.1 L1

A memory block stores L1-trigger parameters, 28-bit for each channel: an 18-bit threshold and ten L1-trigger flags. A channel registers a hit if the A-peak height exceeds the threshold and the age is greater than one of two minimum ages $T \geq T_1$ or $T \geq T_2$, where $T_1 \leq T_2$. A $T \geq T_2$ condition ensures that the event is sufficiently old that slow detectors had a chance to show up.

The L1-trigger flags define to which L1-trigger the channel contributes. In other words, they define ten sets of channels. For each set, the trigger provides three L1-trigger conditions:

1. COINC, all channels in the set hit with $T \geq T_1$, at least one channel hit with $T \geq T_2$.

2. ANTI, none of the channels in the set hit with $T \geq T_1$.

3. ANY, at least one channel in the set hit with $T \geq T_2$.

6.2.2 L2

The same memory block provides storage for eight L2-triggers. Each trigger is defined by 30 bits, corresponding to the thirty L1-trigger conditions. Each L2-trigger requires a set of the L1-trigger conditions to be met.

Eight triggers are sufficient to provide one for each of the six telescope apertures of EPT/HET.

6.2.3 Derandomizing Buffer

Triggered events will transfered to a buffer, where they wait for the L3-trigger processor to become available. This buffer shall be eight events deep, to cover fluctuations of the instantanious trigger rate.

The output of the filter is always written into the next available buffer slot, unless it is full. When a trigger is accepted, the write pointer advances to the next slot.

The data in the buffer includes: a timestamp (32-bit, 1 μs resolution), a delta-time (16-bit, time since last trigger), the L2-trigger (8-bits), and the number of triggered events while the fifo was full, since the last event (8-bit). And for each channel, the A-peak height, the peak age T, and ten bits from the peak phase B, selected according to the leading bit of A. That gives a total of 64 bits of header information and 32 bits per channel.

The current implementation drains the buffer into an XRENA readout fifo.

7 Hardware Models

To implement and test the FPGA functionality on hardware, we have a series of options with rising cycle time and cost.

1. Initial tests on xRENA. The sensor head demonstrators are based on IRENA or ARENA. For tests of the downstream interface we developed the SIRENA. A downstream interface to the ICU can also be added to the ARENA.

   The xRENAs all have an Altera Cyclon 3C25 FPGA, an Arm 7 microcontroller LPC2148, with a USB interface to a computer.

2. Initial engineering digital boards will carry a flash-programmable emulator for the RTAX2000. The bitstream for this FPGA will be converted from the RTAX bitstreams.

3. Later engineering boards will employ RTAX engineering samples. These are anti-fuse programmable. They may consume less power than flight parts, because they lack SEU mitigation in the sequencial cells. These FPGAs should be on sockets initially.

4. Flight boards will employ flight RTAX parts. These may need to go an sockets as well, initially, depending on the state of the design at the time the boards are made. I do not know how easy that is.

7.1 SIRENA

The SIRENA is an adaption of the xRENA design as a simulator of the EPD sensor units towards the downstream interface, the ICU. This simulator shall allow to test FPGA design modules for this interface inside the sensor units, and the communication to the ICU.

7.1.1 Components

The SIRENA mainboard includes the following hardware components:

- FPGA: Altera Cyclon 3, EP3C25-144. The simulator functions shall be implemented on this chip. A Verilog module library is available that facilitates the communication with the microcontroller.

- LVDS: the interface to the ICU is provided in form of four LVDS receivers and two LVDS transmitters, connected to the FPGA. The downstream connector is a SUB-D 25 with a pinout corresponding to the EPD-ICD.

- RAM: 2 MByte BS62LV1600E. This memory is conected to the FPGA to be used to simulate sensor functions.
- ARM7: NPX LPC2148. This microcontroller interfaces the simulator FPGA to a controlling computer.
- USB: The Arm 7 connects to a computer via a full-speed USB interface with a custom protocol.
- FLASH: 1 MByte AT45DB081D. A small read-only filesystem stores the FPGA bitstream and startup scripts. It is connected to the SPI interface of the Arm 7. The filesystem is prepared on a computer and can be replace in whole via the USB interface.
- RS232: A serial interface to the computer allows reprogramming of the ARM 7. This interface can also be used to control the simulator.
- MISC: Some spare FPGA pins are provided to a frontend connector, in case someone wants to test some extra hardware with the simulator. The housing has a spare slot for a board of size 106×70 mm.
- SD-CARD: A micro-SD card slot is available for non-volatile storage, connected to the Arm 7 SPI interface.
- SPI: The Arm 7 microcontroller employs a serial port interface that connects to the FLASH, SD-CARD, and to the power supply board.

The SIRENA power supply board provides the following services:

- The 28 V power input, either from the ICU interface connector or supplied externally, is converted to power the simulator.
- A power load simulator can adjust the power consumption under control of the Arm 7 microcontroller. The controller can read input volatge and current, and control the extra load. A cron script that executes once per second adjusts the load towards a target power.
- A two-line LCD display can be addressed by the Arm 7 via the SPI interface. The display is driven by an ATMEGA microcontroller.

8 Changelog

- 2011-12-22: Add description of DIRENA 128 trigger.
- 2011-12-22: Add *Hardware Models* chapter 7, witha descriptions of the SIRENA.