CMX Firmware Specifications

Wojtek Fedorko, Pawel Plucinski, Yuri Ermoline

Introduction

There are three FPGA circuits in the CMX design:

- Base function FPGA
- Topo function FPGA
- Support FPGA

This document provides description of functionality and interfaces of the firmware components of the base FPGA firmware as well as the support FPGA firmware. Topo FPGA firmware has not yet been addressed however we expect to be able to re-use modules handling communication over GTX transceivers, readout, CTP communications, VME configuration and control and TTC/BCID from the base-FPGA.
Overall design and state of development of individual modules is described. Some unresolved questions are mentioned in the text.

Overall layout of the base FPGA firmware

A rough diagram of the functional blocks is shown in the diagram below. It is expected that functionality of backplane data capture and synchronization, data transmission over MGTs to L1Topo as well as readout, and to the CTP and crate/system CMX over LVDS links as well as VME communication will be common to all types of the CMX. Type-specific firmware modules will conform to the common interfaces with the common modules. Presently work is proceeding on the jet-CMX. Firmware for board testing and other types will follow.

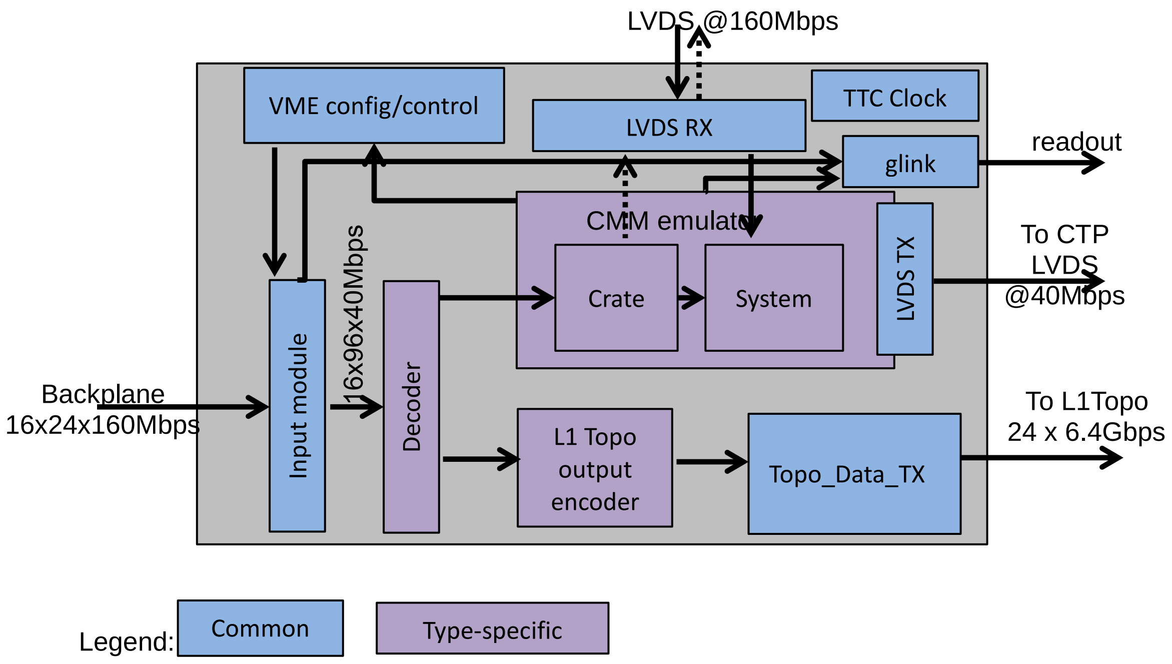

Figure 1. A rough diagram of the functional blocks of the base function FPGA firmware.

Clock Domains

In the CMX design there are be 18 clock domains:

- One 'processor input' clock domain exists for each of the 16 processor inputs forwarding 80 MHz clocks.
- Two 320 MHz clock 'gtx' domains exist within the gtx_TX module – each domain is associated to a group of 12 gtx transceivers within neighbouring 3 'quads'. Clock sharing is not possible among more than 3 quads
- 'System domain' encompasses clocks generated from the TTC clock with a well-defined frequency and phase relation to the TTC clock and one another.

Input module

The function of the input module is to capture the backplane data, time-demultiplex it and bring it to the system time domain as well as detect parity errors. The inputs of the module are the FPGA IOBs connected to the backplane transmission lines. Each processor input provides 24 data bits at 160 Mbps and one clock line at 80 MHz with edges centred in the data windows. Each of the data and clock inputs are piped through an IODELAY module which provides a capability to delay the signals by up to 2.4 ns in up to 31 'taps' of 78 ps. Data is captured and time de-multiplexed to 80 Mbps using the IDDR circuits built into each IOB.

Two schemes for de-multiplexing to 40 Mbps and synchronization are considered:

De-multiplex first, then synchronise:

Data is de-multiplexed to 96 bits x 40 Mbps using the forwarded clock. It is then captured into a system clock domain register. Data becomes available for further processing 30.4 ns after the arrival of the first word on the most delayed (‘slowest’) processor input (furthest away in the crate). This latency is the primary disadvantage of this scheme. Another disadvantage is the necessity of a data framing pattern to be initially sent so that first two and last two words in the event can be identified. The advantage of this scheme is relative robustness of the clock domain crossing. The system clock has a wide margin (~25 ns) to latch the data from the input processor clock domains. The latency quoted above includes the phase delay between the clock of the slowest processor input and the system clock. Post Place-and-Route timing analysis indicates that this delay can be as low as 2 ns. Timing analysis also indicates that data capture will be robust with data valid window as narrow as 50% and forwarded clock jitter of up to 1 ns, however under these conditions the forwarded clock will have to be advanced with respect to the center of the data window by a small amount (0.5 – 1 ns). Fig. 2 shows a time diagram of the backplane data arrival, time-demultiplexing and synchronization to the system domain.

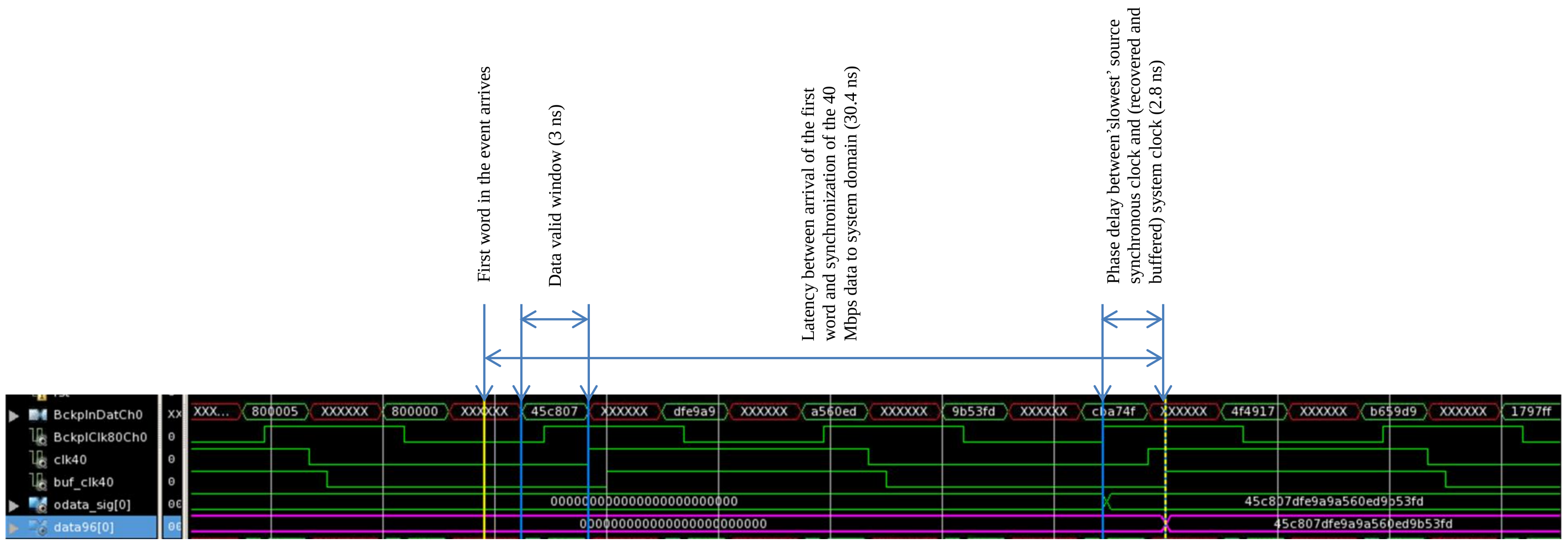

Fig 2 Timing diagram of the backplane data capture, time de-multiplexing and synchronization to the system domain in behavioral simulation. BckplnDatCh0 is the data (24 bits at 160 Mbps) (‘X’ indicates data is not stable) from the processor 0, BckplClk80Ch0 is the 80 MHz clock forwarded with the data, clk40 is the 40 MHz clock from the TTC, buf_clk40 is the same clock regenerated by the MMCM and globally buffered (used as the system clock), odata_sig[0] is the time de-multiplexed data in the clock domain of the forwarded clock and data96[0] is the data synchronized to the system clock domain. White vertical lines indicate 5 ns intervals, yellow lines indicate latency from arrival of the first word to synchronization to system domain, blue lines indicate various events in the sequence as described above the diagram.

Synchronize first, then de-multiplex:

In this scheme the data from IDDR is captured in the input clock domain register and then in a system clock domain register (at 80 MHz). Further de-multiplexing to 40 Mbps is possible in the system clock domain. Compared to the 'de-multiplex first' scheme the first two words of the event are made available half bunch crossing earlier for further processing while the last two words have the same latency. The disadvantage of this scheme is the difficulty of synchronizing the clock domains within a quite narrow window of 6.25 ns. Given that skew as large as ~2 ns is possible in clock networks extending through large areas of the FPGA and unknown jitter of clocks forwarded from the processor modules this may be difficult to achieve. Note that in this scheme no data framing pattern is necessary since the edges of 80 MHz clock can be identified (as edge 0 and 1) in the system domain using the 40 MHz clock which has a known phase relation to the TTC clock.

Scheme choice and tests:

At present the demultiplex-first scheme has been chosen due to significant simplification of the further design components (particularly the decoder block) and a modest penalty on latency.

This scheme has been tested in a scaled-down environment using a ML605 Virtex 6 evaluation kit coupled to a XM105 board. In the test three simulated processor inputs each carrying four bits have been output on the FPGA's GPIO's and looped back via the FMC interface and XM105 card. The three 'channels' are captured in three FPGA clocking regions mimicking the arrangements of inputs in the final system. Parity check has been performed in overnight tests and no error found.

Ports:

- `buf_clk40` needed in both schemes (globally buffered)
- `buf_clk200` needed for IODELAY circuits calibration (globally buffered)
- `P              : in mat_var (numactchan-1 downto 0);` Backplane input to FPGA
- `ODATA          : out arr_4Xword (numactchan-1 downto 0);` Output data – time de-multiplexed and synchronised to the system 40 MHz clock domain.
- `PAR_ERROR      : out std_logic_vector(numactchan-1 downto 0);` Parity error detection for each processor input
- `rst_rx: in std_logic;` Needed if 'de-multiplex first' scheme is used – when asserted puts internal FSM in waiting for a startup pattern
- `counter_enable_out: out std_logic_vector(numactchan-1 downto 0);` Needed if 'de-multiplex first' scheme is used. If '1' indicates that FSM (for a given processor input) has detected a pattern.
- `del_register: in del_register_type;` Holds 5-bit delay values for all FPGA backplane inputs
- `upload_delays: in std_logic;` signal synchroneous to the clk40 clock – tells the iodelays to use the delays stored in del_register

Topo_Data_TX module

This module implements 24 GTX transmitters operating at 6.4Gbps line rate each (5.12 Gbps data rate). In each bunch crossing 3072 bits are transmitted (128 per gtx TX). Two internal clock domains are necessary, each operating at 320 MHz. The reference clocks are shared among two groups of three transciever 'quads'. GTX transmitters implement 8b/10b encoding with 20 bit internal data width (16 bit user data width). Two MMCMs are required internally in the module to provide user interface clocking signal.

Four first-word-fall-through FIFO's are implemented internally interfacing the system clock domain (40 MHz) and the two GTX clock domains (320 MHz). The FIFOs have 8:1 write to read width ratio. Note: this design will be changed as the FIFO displays unacceptable latency (41 ns).

The GTX transmitters are set up to bypass the TX buffer minimizing latency with an added benefit of phase synchronization of the outputs. Depending on parametrized switch in the VHDL code the receiver portion of GTX transceivers is powered and support circuitry instantiated enabling data readout to the top module. Such setup will enable internal PMA loopback tests of the megabit interfaces even though base FPGA will not be instrumented with optical gigabit receivers.

Ports:

In the target system num_GTX_groups=2 and num_GTX_per_group=12

- `MGTREFCLK_PAD_N_IN : in std_logic_vector(num_GTX_groups-1 downto 0);`
- `MGTREFCLK_PAD_P_IN: in   std_logic_vector(num_GTX_groups-1 downto 0);` -reference clock inputs
- `GTXTXRESET_IN: in   std_logic;`
- `GTXRXRESET_IN: in   std_logic;` -transmitter and receiver reset signals
- `GTX_TX_READY_OUT: out  std_logic;`
- `GTX_RX_READY_OUT: out  std_logic;` - signals specifying that all TX and RX have completed synchronization and are transmitting
- `RXN_IN: in std_logic_vector((num_GTX_per_group*num_GTX_groups)-1 downto 0);`
- `RXP_IN: in std_logic_vector((num_GTX_per_group*num_GTX_groups)-1 downto 0);`
- `TXN_OUT: out std_logic_vector((num_GTX_per_group*num_GTX_groups)-1 downto 0);`
- `TXP_OUT: out std_logic_vector((num_GTX_per_group*num_GTX_groups)-1 downto 0);` -RX and TX pad input and outputs
- `clk40: in std_logic;` -globally buffered 40 MHz clock

- `indata : in  std_logic_vector(TX_indata_length-1 downto 0)` –input data vector TX_indata_length is 3456 in the target system. Data to be sent is to be arranged in groups of 18 bits where the lower bits are the data to be sent and the two upper bits designate if the data to be sent is a K character.

Status and Tests:

The module is implemented and satisfies timing constraints. In order to satisfy the timing some care had to be taken in manual placement of the fifo components and Map and PAR effort level had to be switched to maximum level. Simulation on behavioral level shows correctness of the design however indicates an unacceptable latency of the FIFO component. This portion of the design will be modified to reduce latency. A two-clock FIFO will be implemented where both clocks are running at 320 MHz.

A rudimentary test of the component excluding the FIFO synchronization was performed in ML605 at lower line rate allowed by -1 speed grade FPGA on the test board. A single GTX (num_GTX_groups=1 and num_GTX_per_group=1) was instantiated and a random data pattern was sent and received in PMA loopback mode. Chipscope was used to confirm the reception of the data pattern however no design was implemented to test for bit error rate.

Encoder (jet type)

This module is not yet implemented. The role of this module is to transform array of TOBs provided by the decoder as well as the BCID information provided by the TTC module and encode this information in the vector provided to the transmitter module. Very little logic is expected in this module – mostly signal renaming and synchronization of TTC signal into the same register as the TOB data, however data format and protocol for CMX-L1Topo data transmission needs to be specified. Development is proceeding on the jet type

Decoder (jet type)

The main function of the CMX decoder (see Figure 1) is to fetch the data from the input module, to process it and provide two data streams for the 'CMM emulator' and L1Topo Encoder. The output data consists of the trigger objects (TOBs) multiplicities and parity bits for the 'CMM emulator' and an array of the trigger objects for the L1Topo output block. Based on the data analysis, the CMX decoder has to send up to 24 trigger objects to the L1Topo. In addition, in order to reduce the data volume and decrease the time needed to sort the data only non-empty trigger objects are provided. The input data for the decoder consists of 16 channels x 96 bits at 40MHz. Two 'de-multiplexing to 40Mbps and synchronization' schemes are possible to implement and in this version, 'de-multiplex first, then synchronise' method is being used. The time needed to provide the trigger output is estimated to be only one 40MHz clock cycle.

Ports:

- `CLK40MHz       : in` needed to process the 'CMM emulator' output data.
- `CLK160MHz      : in` needed to process the L1Topo output data.
- `RESET          : in std_logic;` reset signal.
- `CTRL_FLG       : in std_logic;` control flag, not used at the moment
- `THRESHOLD      : in THRESHOLD_TYPE(numactchan-1 downto 0);` Threshold sets
- `IDATA          : in arr_4Xword (numactchan-1 downto 0);` Input data
- `OVERFLOW       : out std_logic;` overflow bit
- `TOB_MULT_OUT   : out TOB_MULT_TYPE(numactchan-1 downto 0);` CMM emulator output
- `TOB_ARRAY_OUT: out TOB_ARRAY_TYPE(max_tobs-1 downto 0);` L1Topo output

CMM emulator (jet type)

The real-time output of the CMX decoder is sent to the "CMM emulator" which consists of two part that perform crate and system level merging, respectively, at 40 MHz. The function of the "CMM emulator" is to receive the trigger objects multiplicities, process it and transmit it to the Central Trigger Processor (CTP). Readout to the DAQ and RoI RODs is carried out by a pair of emulated G-link protocol (Figure 3) in Virtex 6, using GTX transmitters clocked at 960 Mbits/s. The G-link protocol was successfully implemented and tested in Virtex 6. The scope tests of the optical output executed with ML605 board, so called "an eye diagram" (Figure 4) proved that there is no problem to emulate the G-link protocol in Virtex 6. The rise and fall time was measured below 240 ps which is enough to fulfill the G-link protocol requirements. In order to adopt the "CMM emulator" to the Virtex 6, the following parts were implemented: the G-link protocol and GTX serializer. And, two parts updated: the clock manager (MMCM is currently being used) and a new block RAM in the readout fifo/memory. The design was fully simulated with ISIM (Figure 5).

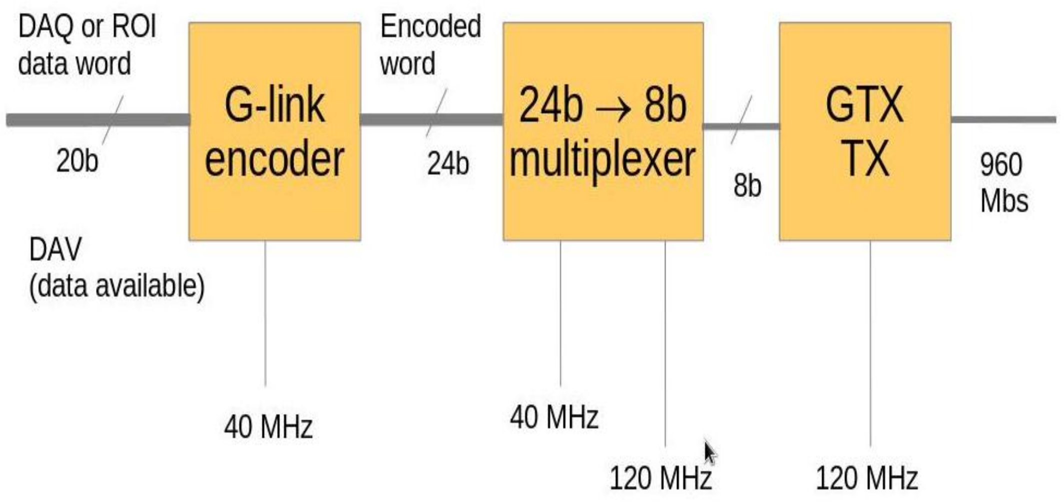

Figure 3. The general idea of emulated G-link protocol in Virtex 6.

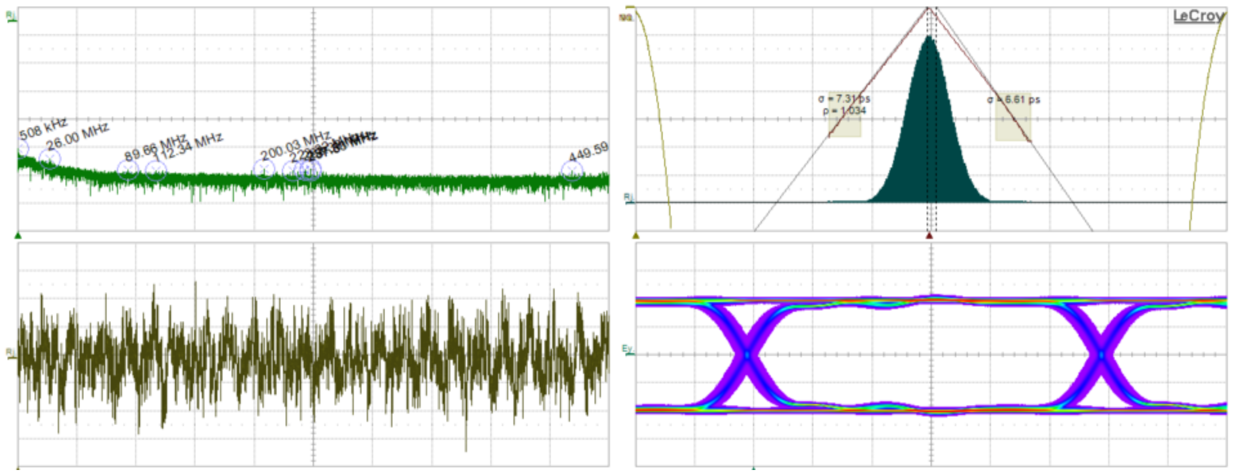

Figure 4. The scope tests of the optical output (an eye diagram) is presented. The rise and fall time is below 240ps.

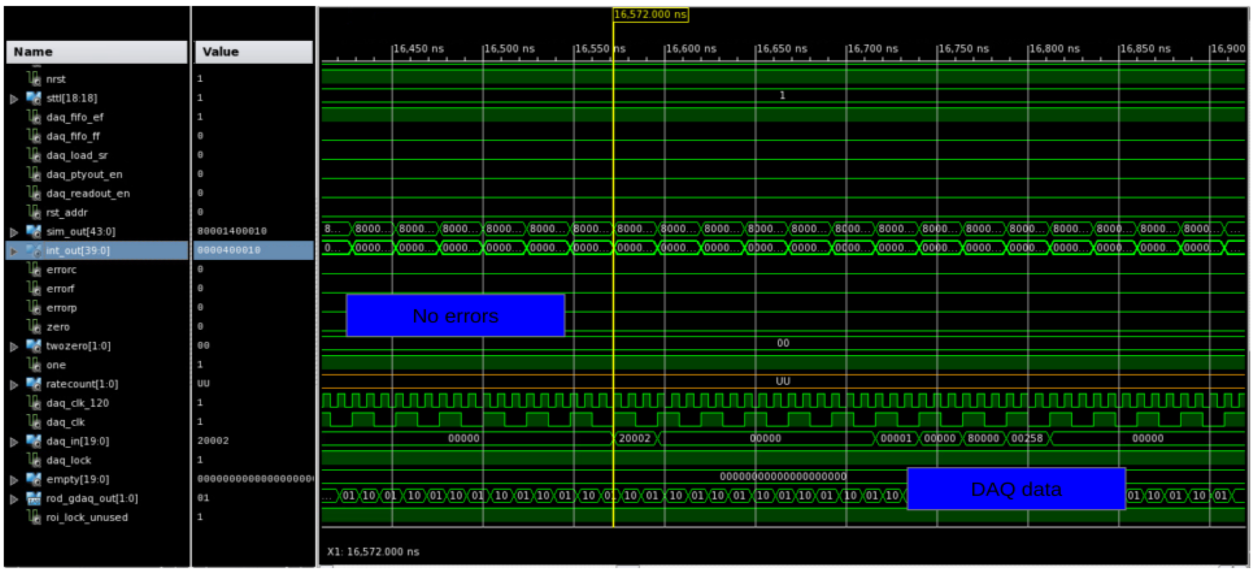

Figure 5. The ‘CMM emulator’ (crate level) simulation results. The random input data were used to feed the algorithm and the output data was carefully analyzed, no errors found.

Support FPGA firmware

The implementation of the Support FPGA firmware is based on the firmware for the current CMM design, which is composed of the following hardware parts:

- non-volatile VME CPLD (XCR3384XL-10FT256C) which contains some basic registers in a case of a malfunction in the FPGA configuration process,
- non-volatile ACE CPLD (same type as VME) which provides access to the CMM XILINX System ACE controller,
- TTC FPGA (XCV100E-6FG256C), which provides an access to the CMM TTC daughter card.

For the CMX, these 2 CPLDs and the TTC FPGA are merged in a single FPGA Spartan-3AN (XC3S400AN-FG400). This FPGA has an internal flash memory for the initial configuration after power-on. The internal flash memory can be re-loaded via the JTAG interface.

The Support FPGA firmware at the time being is a compilation of 3 VHDL firmware codes from original CMM design.

Test

The VHDL code for 2 CPLDs and the TTC FPGA of CMM are merged in a single Support FPGA design and implemented in smaller Spartan-3AN FPGA - XC3S200AN-FTG256, the result is: slices used 286/1792(15%) and pins used 165/195(84%).

VME address map

The CMM VME-- map is described in the CMM specification, and in the vme_cmm.vhd package. The current VME-- address ranges allocate 0x80000 bytes for each CMM (512k), of which less then half is currently used:

• CMM0 (slot 3):	0x700000-0x77FFFE
• CMM1 (slot 20):	0x780000-0x7FFFFE

The top of used VME address space in the CMM is 0x7171FE, therefore addresses starting from 0x717200 can be used in CMX.

The CMX VME memory map will try to keep as much as possible the location of all CMM registers and bits inside the registers. In a case, where new hardware design will need new registers, LUTs, FIFOs or RAMs, they will be implemented in a spare VME address space.

For large memories, instead of direct mapping of the CMX to the VME-- memory address space, an indirect addressing might be used - a moveable window, where a register on the CMX defines the base address of this window. This would allow the many megabytes of CMX to be accessed through a smaller VME-- address space. Provided the window is big enough to encompass any of the single blocks of RAM this solution would not be expected to slow access significantly.

VME interface

The following 2-byte registers are implemented in the Support FPGA:

00000	RO	ModuleIdA	Module ID Register A
00002	RO	ModuleIdB	Module ID Register B
00004	RW	ControlModeReg	Control Mode Register
00006	RW	ControlPulseReg	Control Pulse Register
00008	RO	StatusReg	Status Register
0000A	RO	FifoStatusReg	FIFO Status Register
00054	RO	I2cid	I2C FPGA firmware version
00056	RO	VmeId	VME CPLD firmware version
00058	RO	SystemAceVMEIf	System Ace VME Interface CPLD firmware version
0005C	RW	CanAccessA	CAN Access Register A
0005E	RW	CanAccessB	CAN Access Register B
001FA	RW	TtcI2Cid	TtcI2cId Register
00200	RAM	ia_ttc_dqram	I2C FPGA RAM for testing the TTC
00300	RW	ia_ace_ctrl	
00302	RW	ia_ace_d_msb	
00304	RW	ia_ace_rst	
00306	RO	ia_ace_out	
00308	RO	ia_ace_stat	

APPENDIX: Data types in base FPGA firmware

```
SUBTYPE T_SLV5 is std_logic_vector(4 downto 0);
SUBTYPE T_SLV9 is std_logic_vector(8 downto 0);
SUBTYPE T_SLV10 is std_logic_vector(9 downto 0);
SUBTYPE T_SLV25 is std_logic_vector(24 downto 0);
SUBTYPE T_SLV640 is std_logic_vector(639 downto 0);
TYPE JeTOB is record
     Et1: T_SLV9;
     Et2: T_SLV10;
     eta: T_SLV5;
     phi: T_SLV5;
end record;
type TOB_ARRAY_TYPE is array (integer range <>) of JetTOB;
type THRESHOLD_TYPE  is array(integer range <>) of T_SLV640;
type TOB_MULT_type is array(integer range <>) of T_SLV25;
TYPE mat_var is array (integer range <>) of
std_logic_vector(numbitsinchan downto 0);
TYPE arr_word is ARRAY (integer range <>) of STD_LOGIC_VECTOR
(numbitsinchan-1 downto 0);
```

```
TYPE arr_wordData is ARRAY (integer range <>) of
STD_LOGIC_VECTOR (numbitsinchan-2 downto 0);
TYPE arr_4Xword is ARRAY (integer range <>) of
STD_LOGIC_VECTOR ((numbitsinchan*4)-1 downto 0);
TYPE arr_2Xword is ARRAY (integer range <>) of
STD_LOGIC_VECTOR ((numbitsinchan*2)-1 downto 0);
TYPE del_register_type is ARRAY (numactchan - 1 downto
0,numbitsinchan downto 0) of STD_LOGIC_VECTOR (4 downto 0);
```