GCU Firmware Trigger and Timing

M. Bellato, A. Bergnoli, R. Brugnera, D. Corti, A. Garfagnini, J. Hu,
R. Isocrate, I. Lippi, D. Pedretti

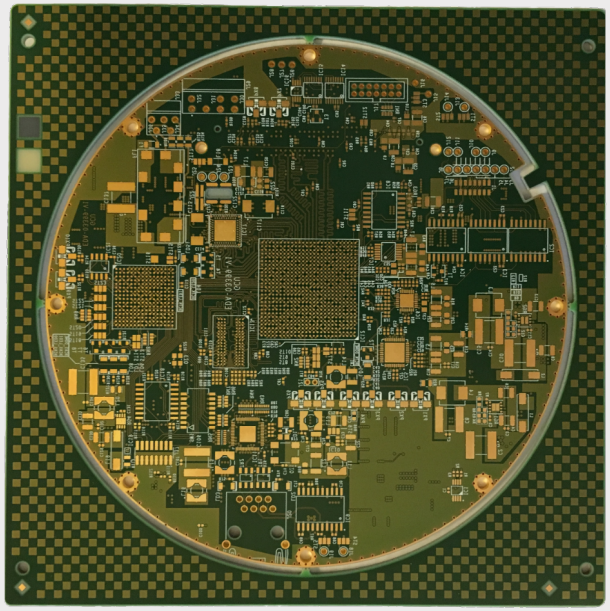
INFN and University of Padova

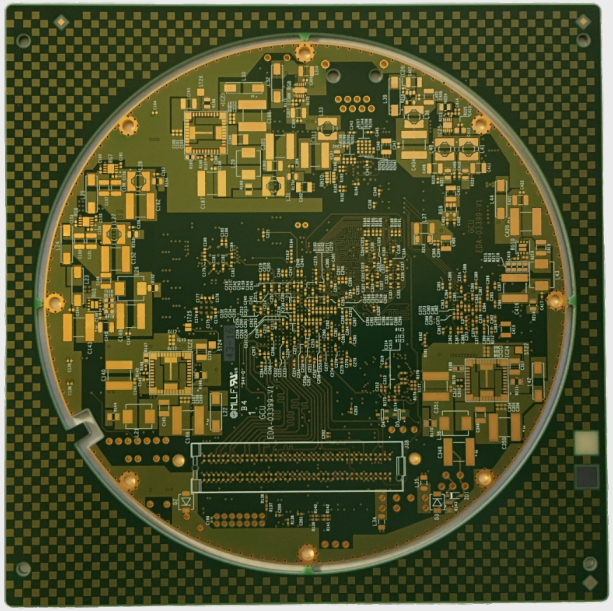
UNIVERSITÀ DEGLI STUDI DI PADOVA

Outline

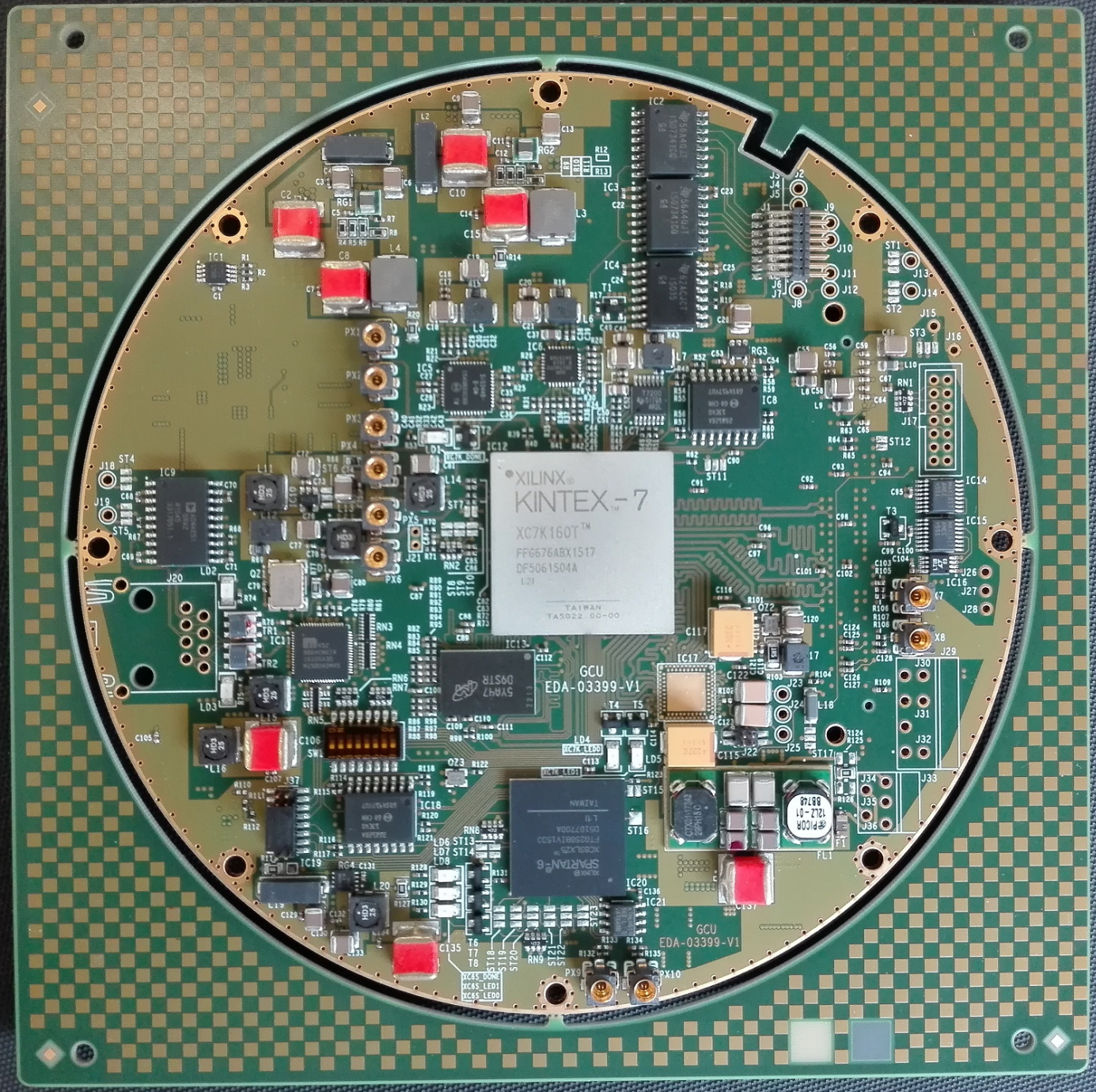
- GCU hardware status and time schedule
- Firmware Overview
- Management Software Overview
- IPBUS and Remote Debugging - Xilinx Virtual Cable
- Trigger Overview
- Timing and Clock Synchronization

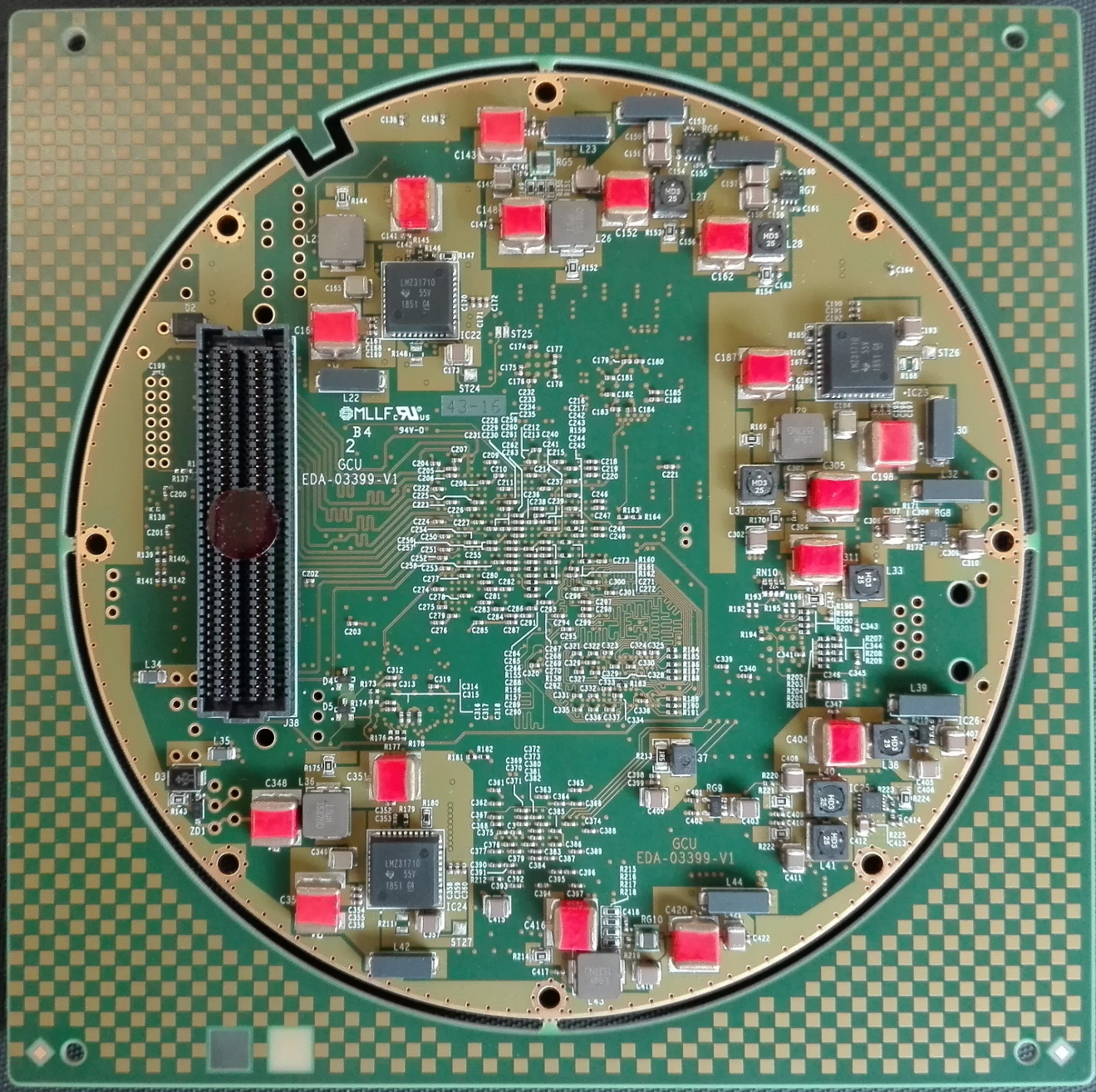
Hardware Status



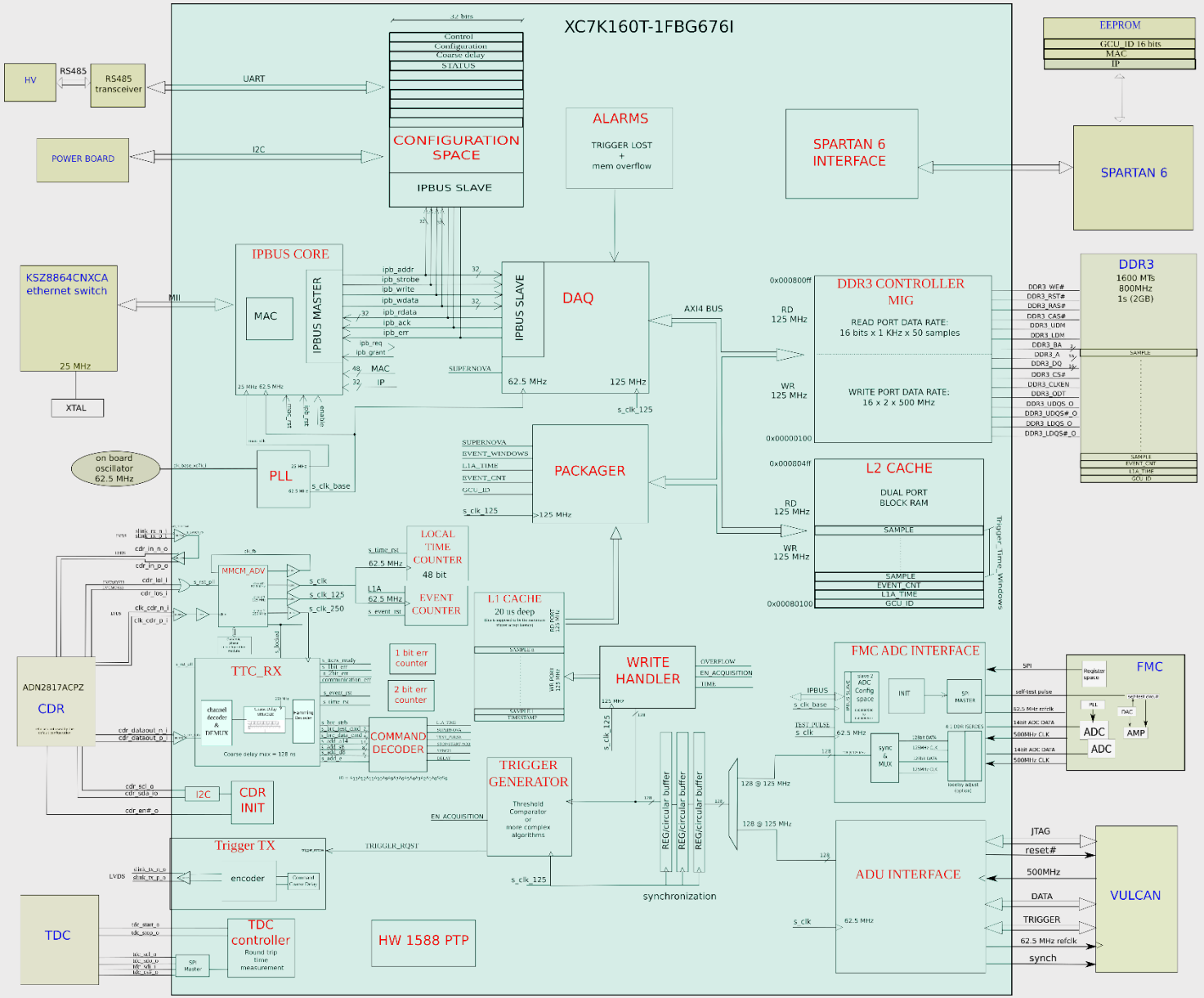


- PCBs arrived on 2nd of November
- Now being assembled at an external company
  - Delivery foreseen for this week
  - Waiting for a picture …
- Power on tests will start next week
- Complete test with custom firmware immediately after

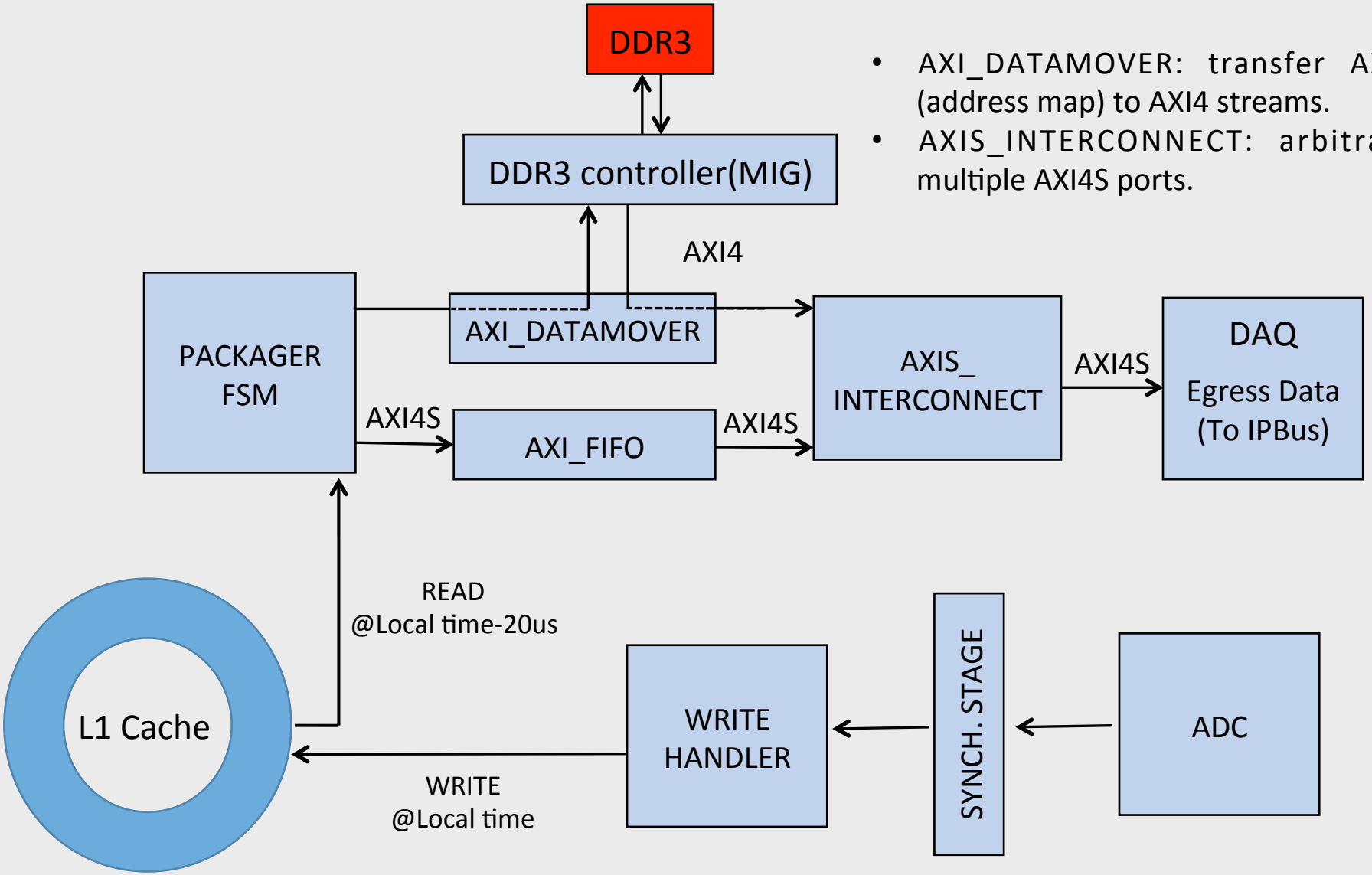

XILINX
KINTEX-7
XC7K160T
FF676ABX1517
GCU
EDA-03399-V1
GCU
EDA-03399-V1


MLLF
B4
2
GCU
EDA-03399-V1
43-16
94V-0
GCU
EDA-03399-V1

Firmware Block Diagram



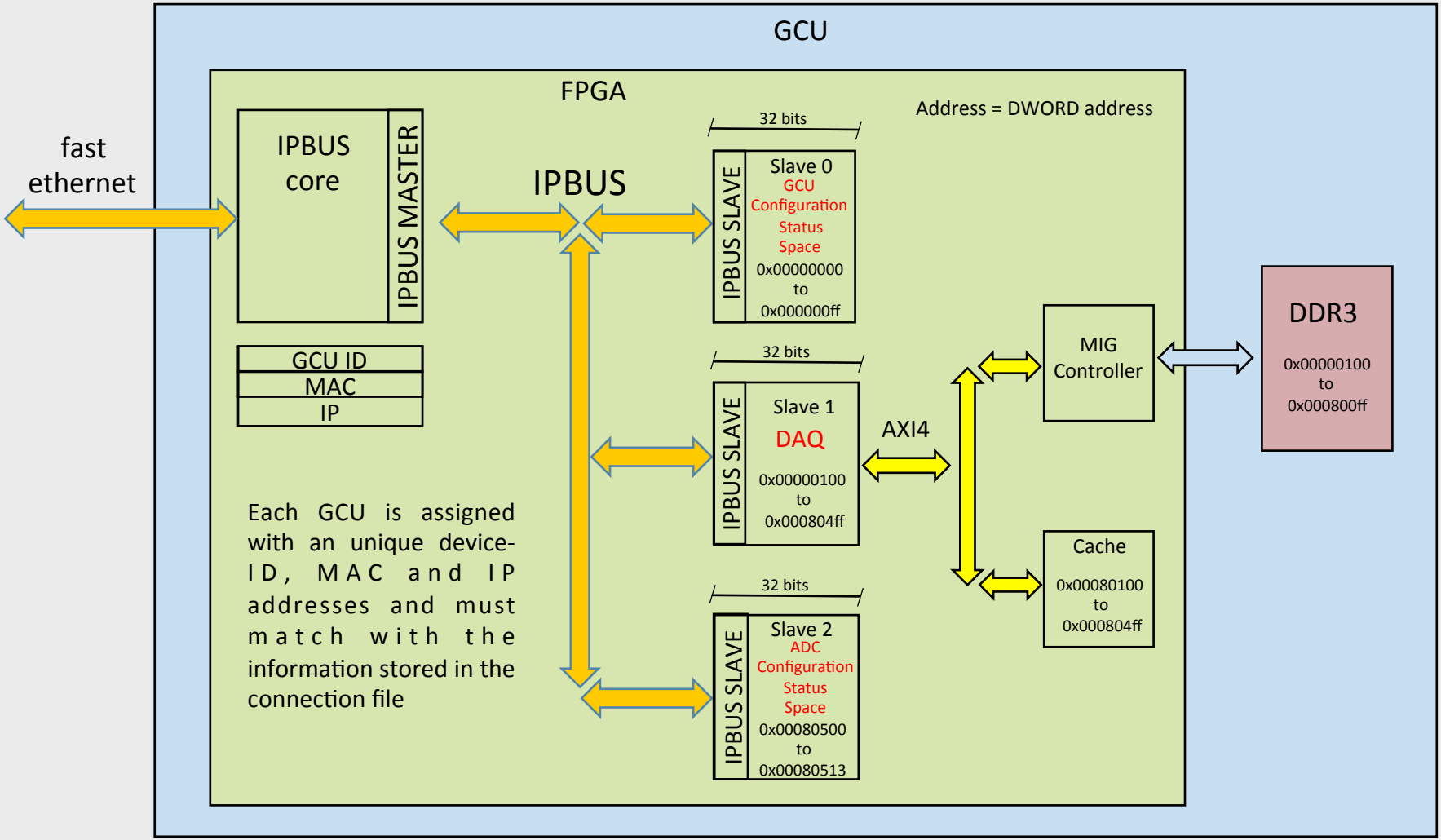
ADCs I/F and Data Readout



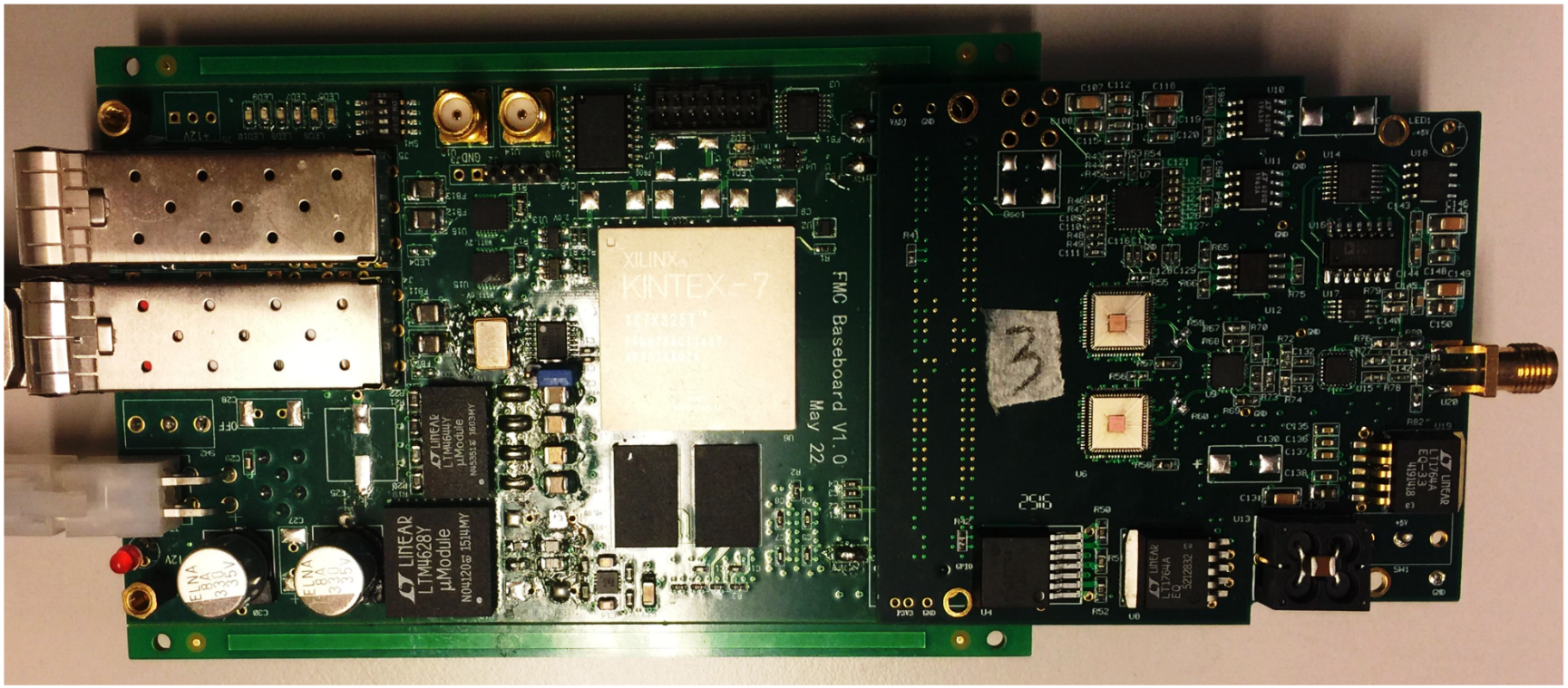
- AXI_DATAMOVER: transfer AXI4 (address map) to AXI4 streams.
- AXIS_INTERCONNECT: arbitrate multiple AXI4S ports.

Data Readout

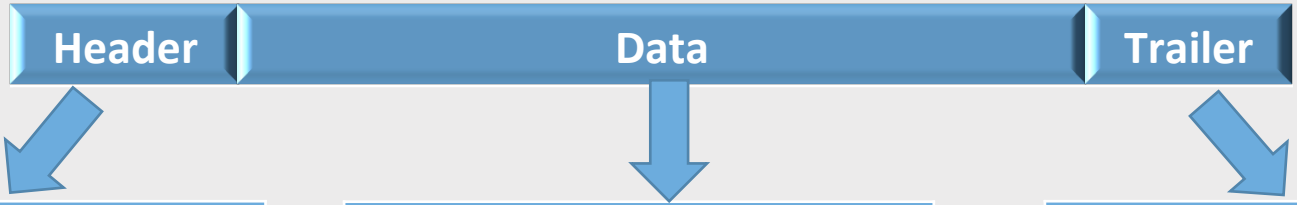
Data Readout is performed via IPBUS. The data rate achieved is ~90 Mbit/sec with block data transfers (the IPBUS efficiency increases with the payload). Accepted events are stored in the cache waiting to be pulled out by software in bunch of 5 in 5.



TsingHua ADC Test SetUp



Data Package Structure



15 0
x805A(maker)
GCU ID
Trigger No.
Packet size 8bit
Time stamp 64bit

15 14	13 0
reserved	14bit ADC Data . . . . (64 samples for normal 1G samples for SN)

(optional)
15 0
0x00
0x12
0x34
0x56
0x78
Trigger No.
GCU ID
x8069

Data Buffer Capability

- L1 Cache => Ring buffer
  - Maximum trigger latency: 20us
  - Cache1 size = 16Gbps x 20us = 320kbits
- L2 Cache => internal ram & DDR3

  For internal ram (normal event)
  - Readout window width: 90 ns
  - Event size : 128 bit header + 16bit x 90 samples ~ 1600 bits
  - Event cache capability 20 events = 32Kbit

  For DDR3 (Supernova)
  - Readout window width: 1s
  - Event size : 128 header + 16 x 1G = 2GB
  - Event cache capability 1events = 2GB

All the calculation is without data compression.

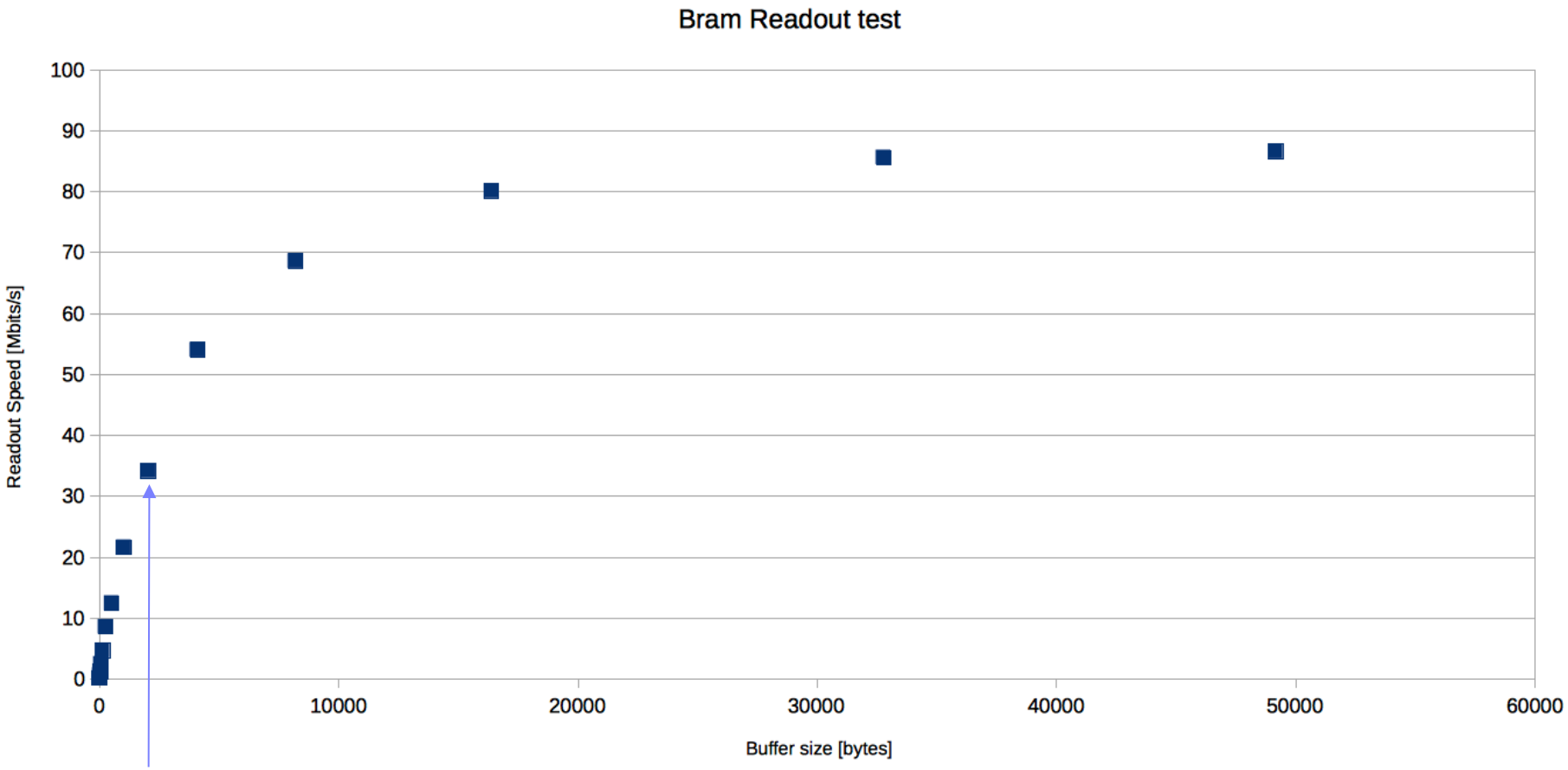
Data rate

- The raw data rate generated by ADC is 16 Gbps/1ch
  - FMC to FPGA bandwidth is 32 Gbps(2 TsingHua ADC).
  - After selecting one out of two chips: 16 Gbps
- DDR3 write bandwidth is 800 MHz x 16bits x 2 x 85%(efficiency) = 21.76 Gbps
- Max Ethernet's bandwidth (IPBus) is ~ 90 Mbps
  - Normal mode (event validation), 1600b x 1KHz = 1.6 Mb/s
  - In Autotrigger mode : 40 samples x 16 bit x 50 KHz = 32Mb/s

DAQ Readout tests(1)

- *IPBUS Readout works in pull mode, e.g. Server retrieves data from GCUs memory*
- *Test code reads many times from blockram istantiated in Kintex7 as IPBUS slave*
- A C++ routine reads back 10000 times a variable size buffer and measure elapsed time
- Link speed forced to 100BaseTx
- Topology: 1 ipbus client (PC) → 1 target (KC705) (no control hub)

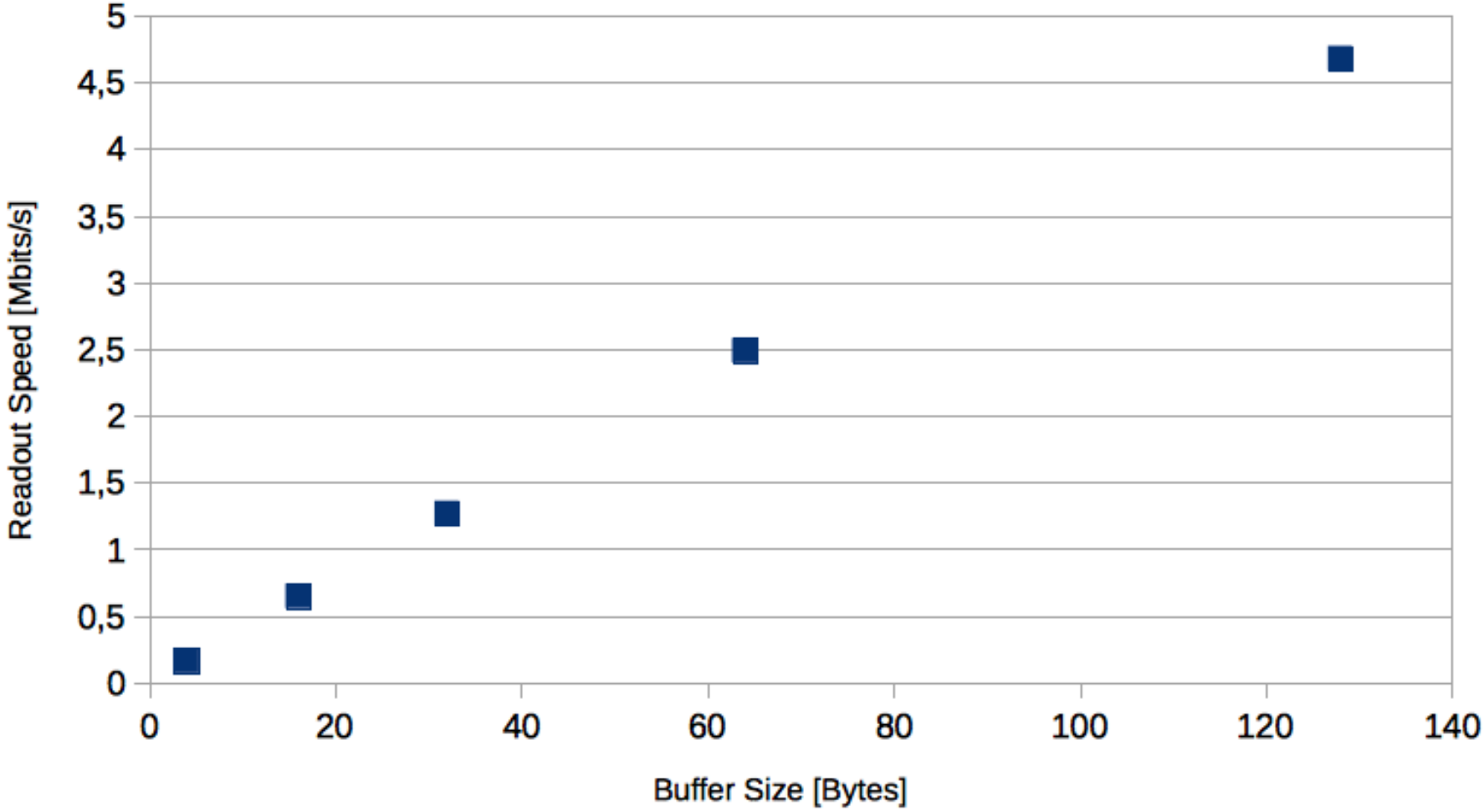
DAQ Readout tests(2)



Buffer Size = 16Kbit

DAQ Readout tests(3)

Details for smallests buffer sizes



Remote debugging – Virtual JTAG over IPBUS(1)

Virtual JTAG components

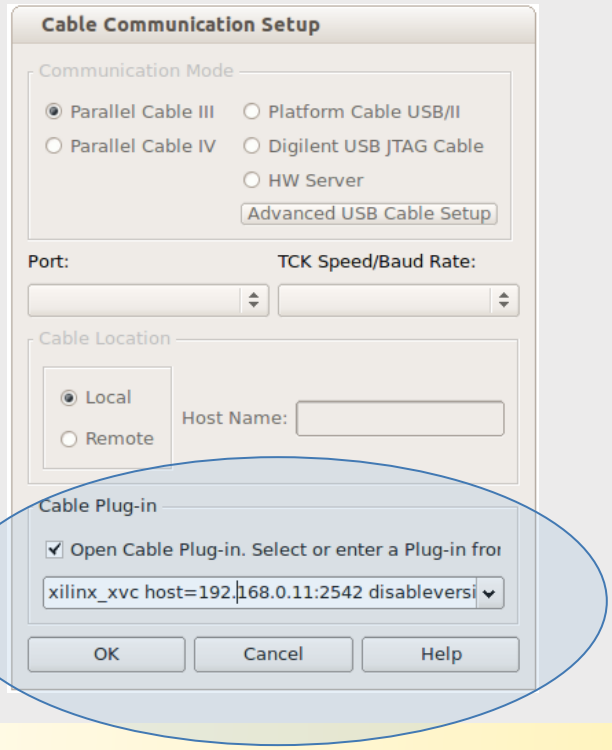
- Xilinx Debug & Programming tools (***Impact***™ and ***Chipscope***™, also ***Vivado***™) are able to connect to a TCP port service named xilinx_xvc

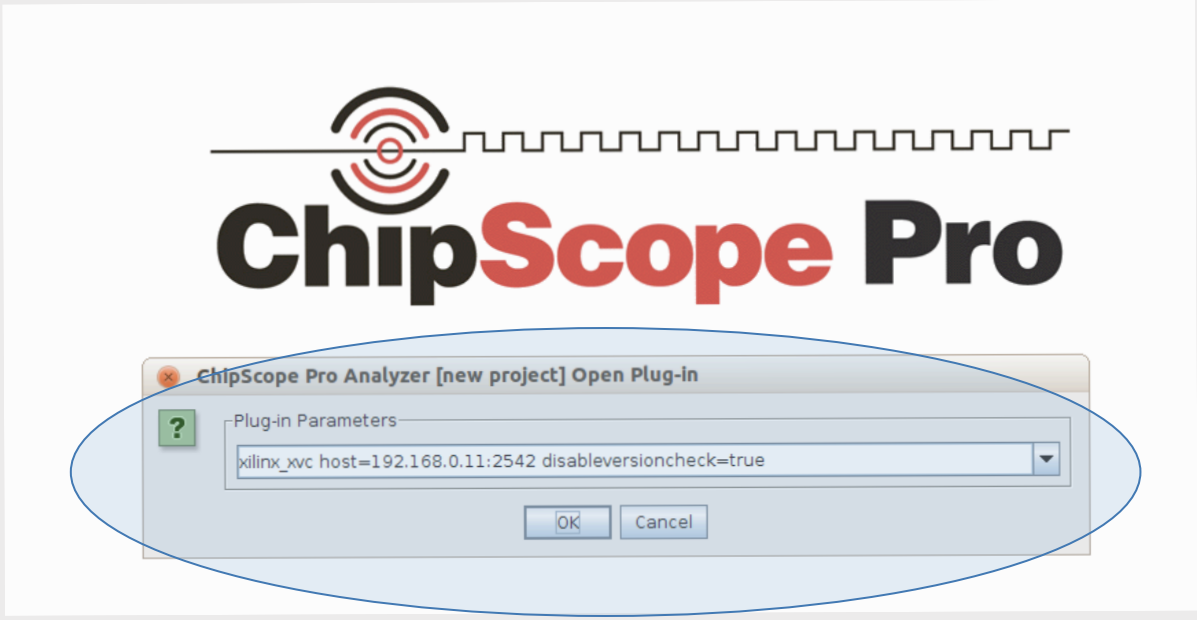
- A Tcp server implementing this *xvc protocol* and able to perform ipbus operations (uhal library).

- IPBUS slave on Spartan6 (named JTAG TAP) drives TDI-TMS-TCK-[TDO] of the kintex7)

Remote debugging – Virtual JTAG over IPBUS(2)

- Trivial configuration from the tools side: just specify the protocol and the address:

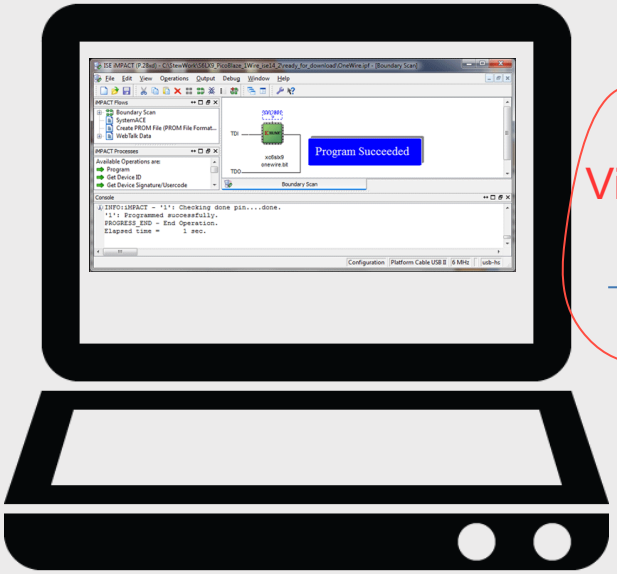




Remote debugging – Virtual JTAG over IPBUS(3)

Component on PC:

- Xilinx tools
- xvc server
- uhal library → ipbus



Virtual JTAG cable

Cat6

GCU

MII

Kintex7

JTAG wires
TDI
TDO
TMS
TCK

ETH switch

MII

Spartan6

Small Spartan6 FPGA acts as *Xilinx remote cable,* IPBus slave (JTAG TAP) lives here

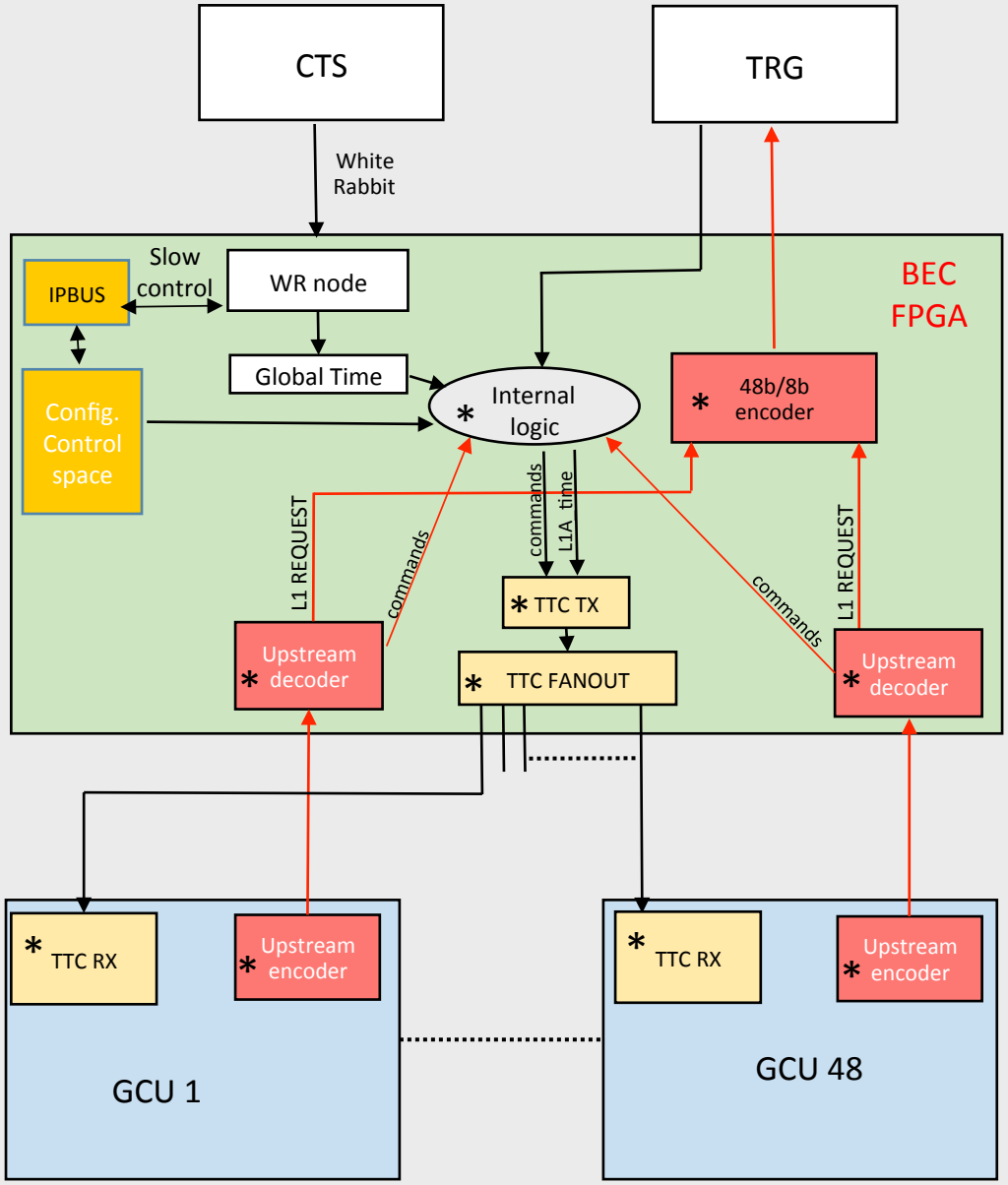
Remote debugging – Virtual JTAG over IPBUS(4)

Current status:

- Kc705 board successfully programmed with Impact (running on PC) + Xvc server (running in a small sigle board PC) able to mimic JTAG with onchip GPIO [*takes several minutes*];

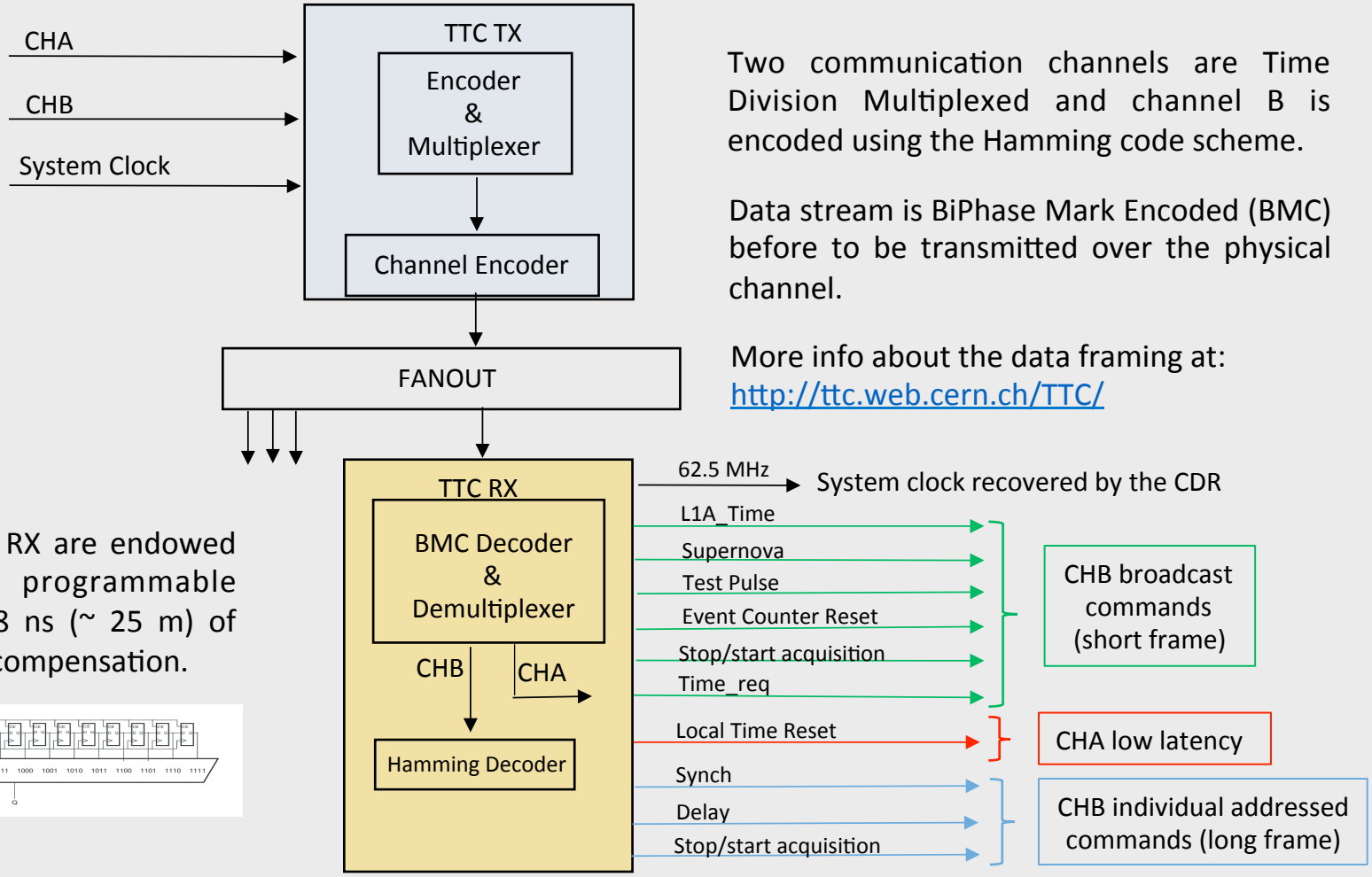
- Ongoing developement on IPBUS JTAG slave; simple bitbanged GPIO is very slow → The buffering defined in the XVC protocol should be exploited.

GCU Trigger Interface Overview



- Synchronous upstream trigger request channel.
- Downstream trigger validation command: asynchronous (max latency); the trigger validation command is tagged with the time windows.
- Synch links bandwidth requirement: 250 Mbps.
- The sync links downstream and upstream should not carry information only on trigger but we need to encode asynchronous and synchronous commands.
- To open the possibility to send to all the GCU synchronous command all the blocks marked with * must have a fixed latency.
- The Downstream protocol is based on CERN TTCrx/tx.
- Upstream we have to accommodate 48 decoders in the BEC FPGA; simple protocol. BEC FPGA resources issue.
- Level trigger. Trigger request information is updated every 16 ns.
- Both TTC RX as well as the upstream encoder incorporate programmable coarse delay to compensate for the electronics and cable propagation delays.

TTCrx/tx Overview

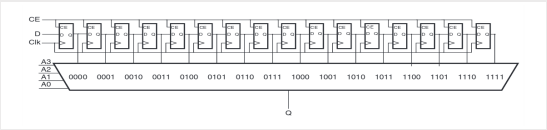


Two communication channels are Time Division Multiplexed and channel B is encoded using the Hamming code scheme.

Data stream is BiPhase Mark Encoded (BMC) before to be transmitted over the physical channel.

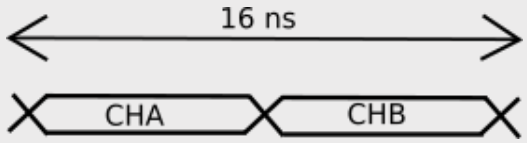
More info about the data framing at: http://ttc.web.cern.ch/TTC/

TTC TX and TTC RX are endowed with a coarse programmable delay; up to 128 ns (~ 25 m) of cable mismatch compensation.

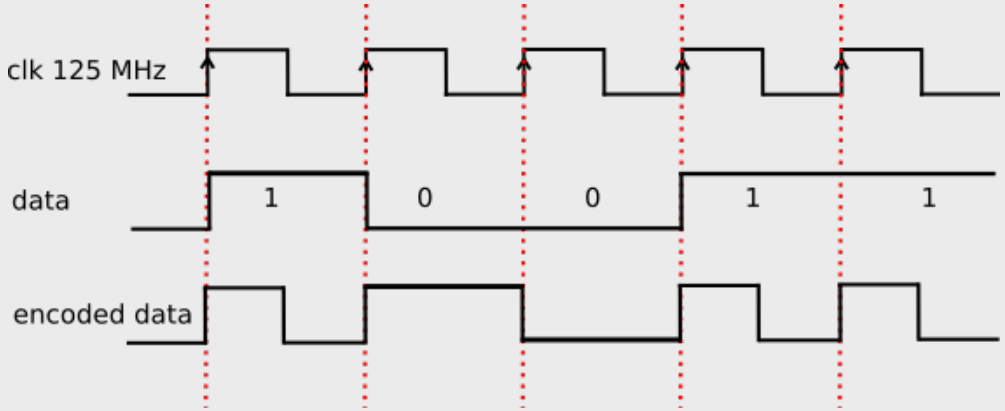


TTC – TDM and BMC

Time Division Multiplexed:



Bi-phase Mark Code:



- Transition on every rising clock edge,
- Data = 1 → bi-phase,
- Data = 0 → constant level during Tck

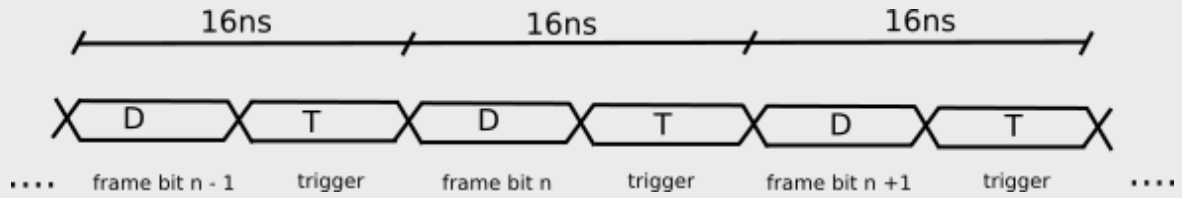
Pro and cons:
+ DC balanced
+ self-clocking
- Half channel bandwidth available

All this does not come for free! We cannot fit many TTC RX in a single FPGA therefore for the upstream channel we need a simpler protocol.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	162	202800	0%
Number of Slice LUTs	258	101400	0%
Number of fully used LUT-FF pairs	116	304	38%
Number of bonded IOBs	53	400	13%
Number of BUFG/BUFGCTRLs	2	32	6%

Upstream Trigger Protocol

- The upstream trigger request link, from the GCU to the TRG, must be synchronous (fixed latency).
- Trigger on the level.
- Trigger request is updated every 16 ns.
- Not only trigger; we need to send to the BEC synchronous commands. Likewise TTC, the trigger request (T) is Time Division Multiplexed with the data/command frame bits (D):



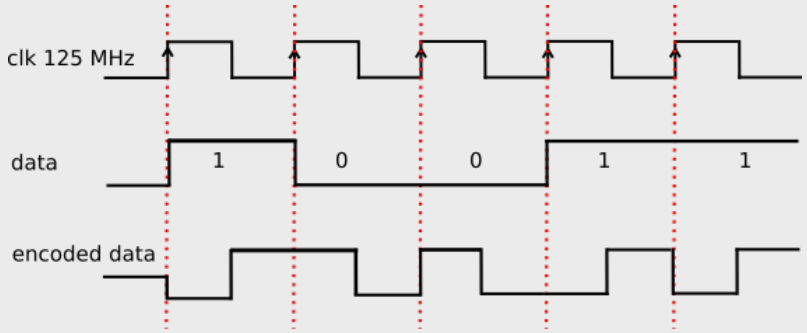
- Data framing: 1 start bit + 8 bit of data + 1 parity bit (optional) + 1 stop bit.

bit number	1	2	3	4	5	6	7	8	9	10	11
	Start	H	H	H	H	D	D	D	D	P	Stop

- Command table:

HHHH	DDDD	command
0000	0000	COMMA
0001	xxxx	BACK_PRS_ON
0010	xxxx	BACK_PRS_OFF
0011	xxxx	DELAY REQ
0100	DDDD	TIME
0101	0101	IDLE
0110	xxxx	TDB
....	....	TDB

- Channel encoding: Manchester (PE); it guarantees no DC bias and is self-clocking. The channel can be AC coupled.

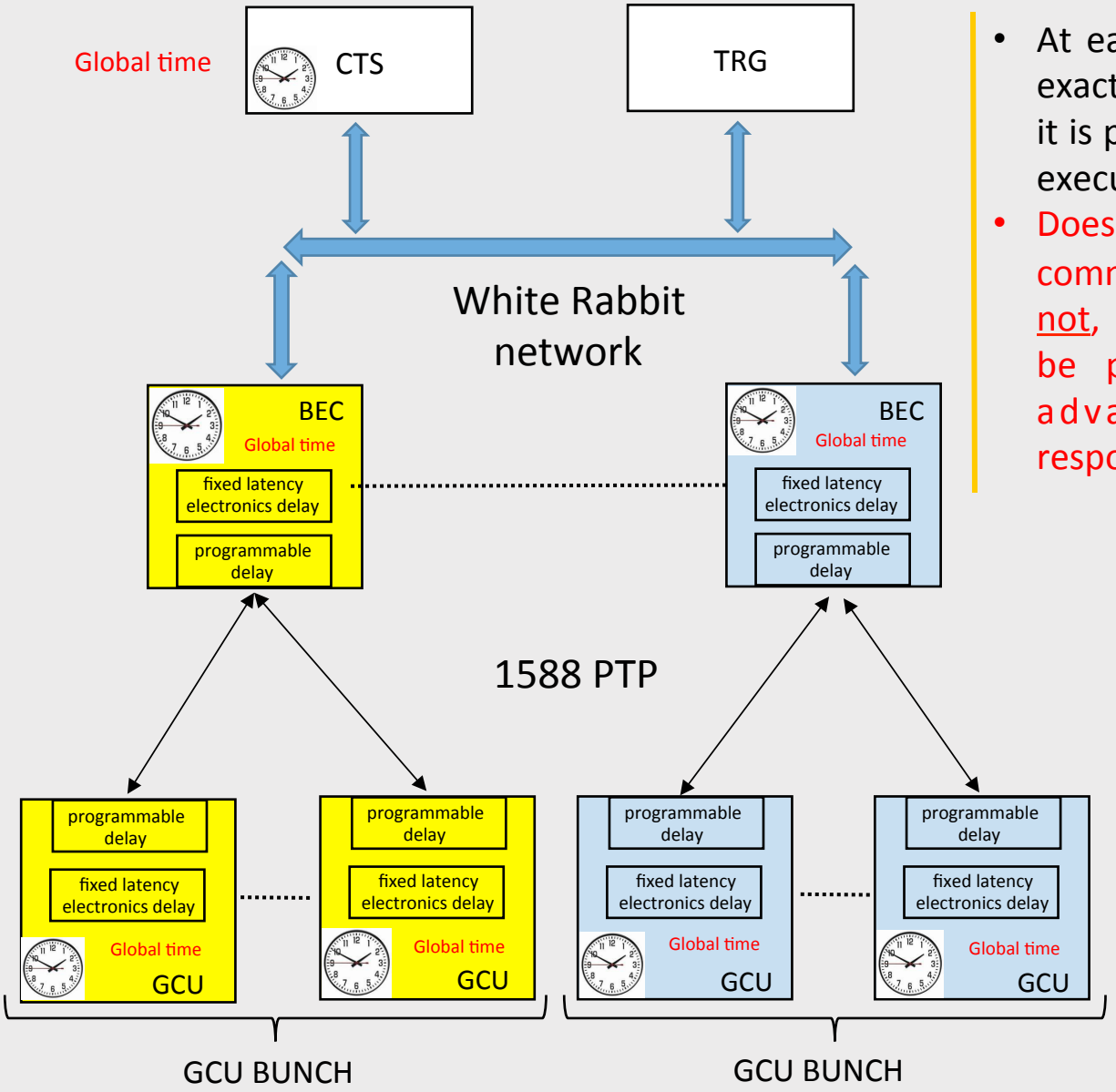


'1' logic = 0 → 1 transition

'0' logic = 1 → 0 transition

- A doubled channel bandwidth is required.

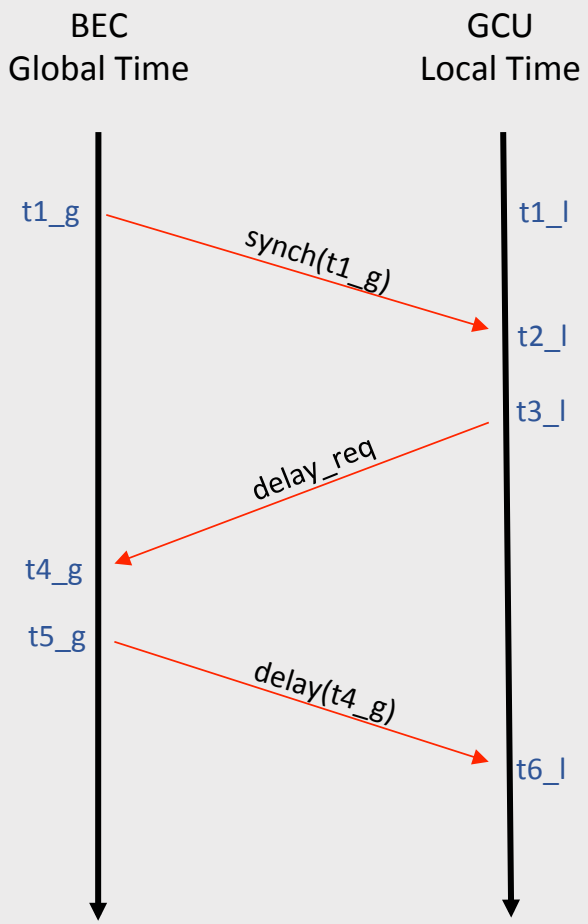
Synchronous operations



- At each level of the system there is an exact copy of the global time therefore it is possible to program all the nodes to execute synchronous operations.
- Does WR allow sending synchronous commands from TRG to all the BECs ? IF not, the synchronous operations must be predictable and programmed in advance; the system won't be responsive to non predictable events.

The programmable delays open the possibility for sending synchronous commands between BECs and GCUs. The cable and electronics delay mismatch compensation is mandatory; all the downstream and upstream delays have to be aligned toward the maximum latency in the system.

Clock alignment 1588 PTP over synch links



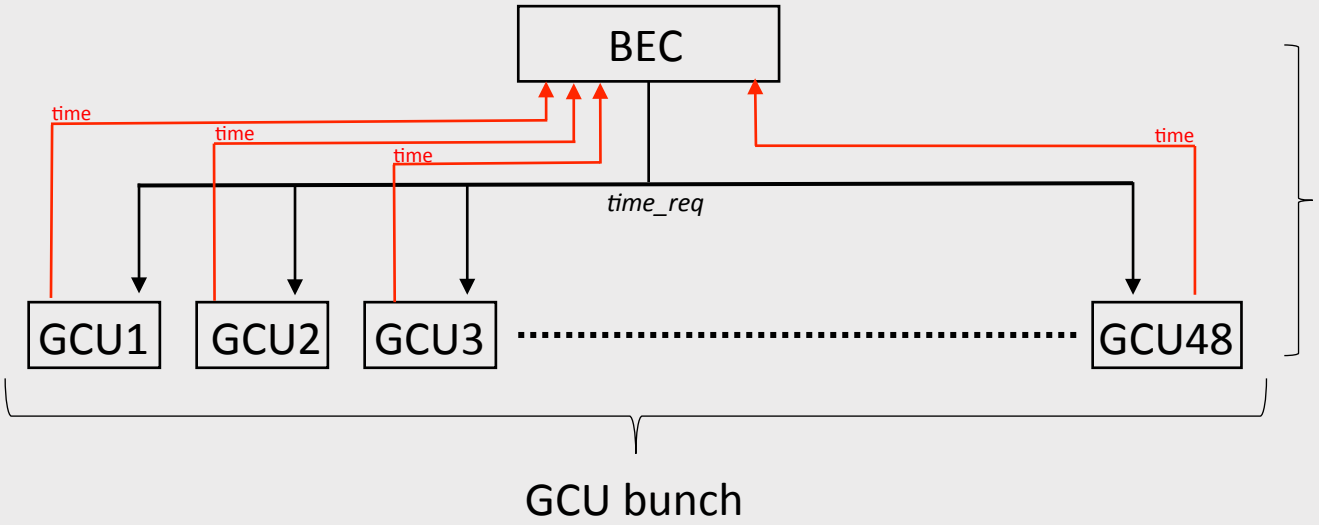
- *t1_g – t1_l* = clock offset to be compensated
- At *t1_g* BEC sends a synch message to GCU tagged with the BEC global time.
- GCU records the reception time *t2_l*.
- GCU computes *t1_g – t2_l* = *clock offset + downstream delay.*
- GCU sends a delay request message to BEC and records the transmission time *t3_l.*
- BEC records the reception time *t4_g* and sends back a delay message tagged with *t4_g*.
- GCU computes *t4_g – t3_l* = *upstream delay*.
- Let's assume *t_diff = downstream delay – upstream delay.*
- *t_diff* is mainly due to the electronics delay mismatch between the downstream and upstream channels; **known** parameter.
- GCU can finally compute the offset:

$$clock\ offset = t1\_g - t2\_l - (t4\_g - t3\_l) - t\_diff$$

This procedure, arbitrarily started by BEC must be repeated periodically and does not require the stop of data acquisition and generation of trigger requests. Indeed these synchronization messages can be sent among trigger requests and validations; t3_l – t2_l and t5_g – t4_g latencies are allowed to be non-deterministic since the important timestamp information is attached with the messages.

Clock Alignment Control

BEC card must periodically monitor the clock alignment status of the whole GCU bunch and eventually individually repeat the synchronization procedure and/or stop the data acquisition of a GCU resulting non-synchronized.



All the downstream channels have the same latency therefore all the GCUs receive *time_req* message at the same instant.

- BEC broadcasts the *time_req* command to all the GCU belonging to the bunch.
- Each GCU records the reception time and sends upstream the timestamp:

start	0	1	0	0	D	D	D	D	P	stop

- The time information is encoded in several frames (4 bits each frame) depending on the number of bit of the local time counter (open point).
- BEC checks if the timestamps are aligned within 16 ns. If a GCU loses the synchronization it must be asked to stop the data acquisition and trigger request generation until a new alignment procedure has been successfully performed.
- This alignment check can be done runtime without stopping the data acquisition.
- BEC has the possibility to start and stop the data acquisition and subsequent trigger request generation via synchronous links using both broadcast commands or individual addressed commands.

Example : Test Pulse

- The test pulse is an example of predictable synchronous operation.
- Each GCU should start the test pulse generation at a well defined time.
- It is an important test for calibration, electronics monitor and TRG training.
- There will be a register reserved for the test pulse in the GCU configuration space. Via slow control sw can write this register with the time value at which the operation have to be executed.
- All GCU may be programmed to generate the test pulse at the same time.
- Or SW can program the BEC to broadcast the test pulse command to all the GCU belonging to that bunch at a preset time.

Local Clock Reset and Event Counter Reset

- The local time counter reset operation is synchronous; even for this operation is not possible to send a synchronous reset message from the CTS to all the GCUs but instead CTS, BECs and GCUs must all be programmed to auto-generate a synchronous internal reset at a pre-programmed time t_reset.

- The local clock reset can be used to prevent the time counter overflow.

- The event counter reset operation is another example of predictable synchronous operation. The BEC card can broadcast the event counter rest command to all the GCU belonging to that bunch via TTC system.

GCU Firmware Schedule

- End 2016 - Q1/2017: prototype firmware for hardware test
  - FMC, ADC I/F
  - DDR3
  - Cable drivers/receivers + power, HV I/F
  - Ethernet
  - CPLD
  - CDR
- Q2 2017 : deployment firmware V. 0.0
  - **Needs ASAP**
    - ADCs (either Vulcan and Tsinghua)
    - BEC
    - Trigger “processor”
    - “Final” cable
    - Power Board
    - Prototype DAQ