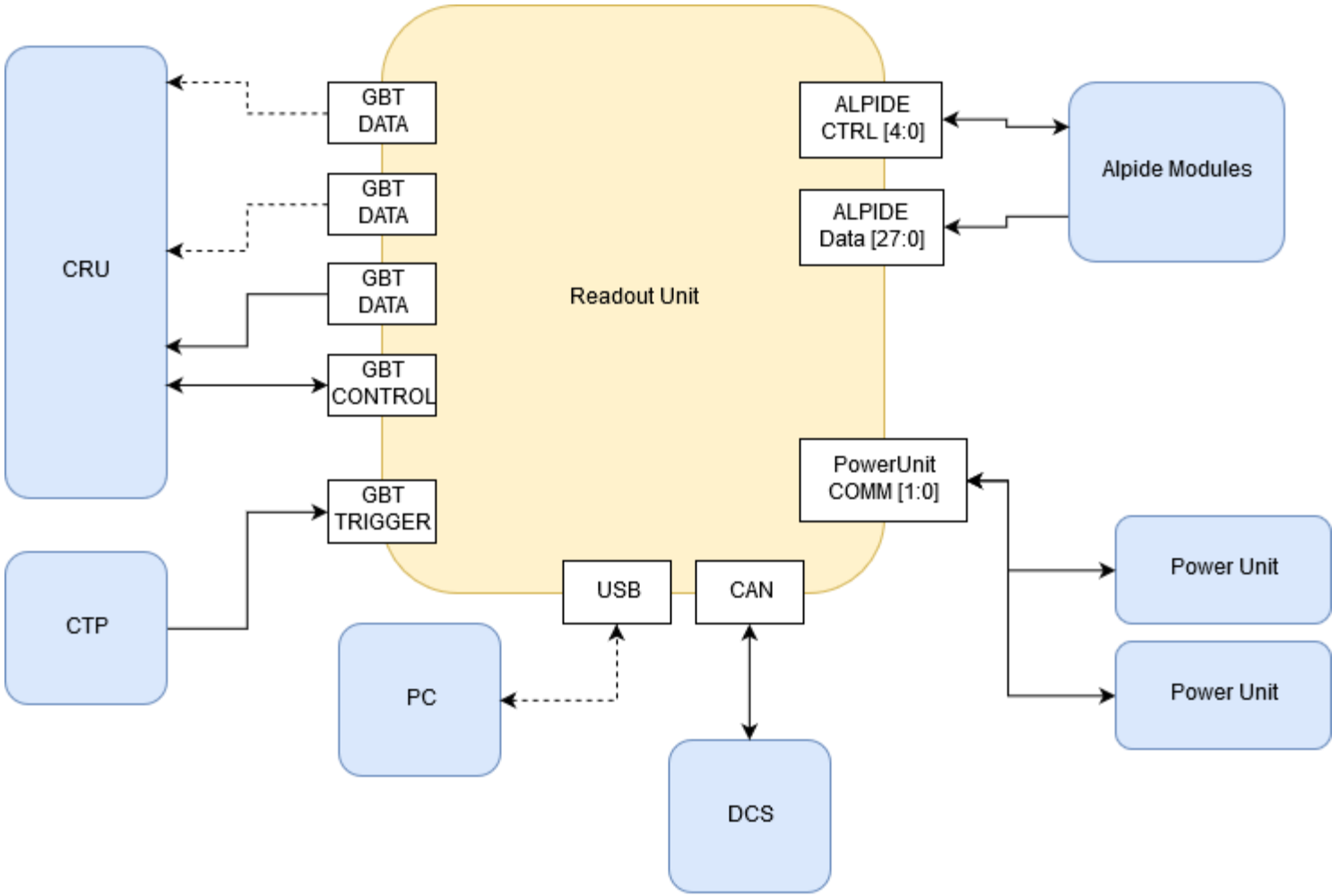
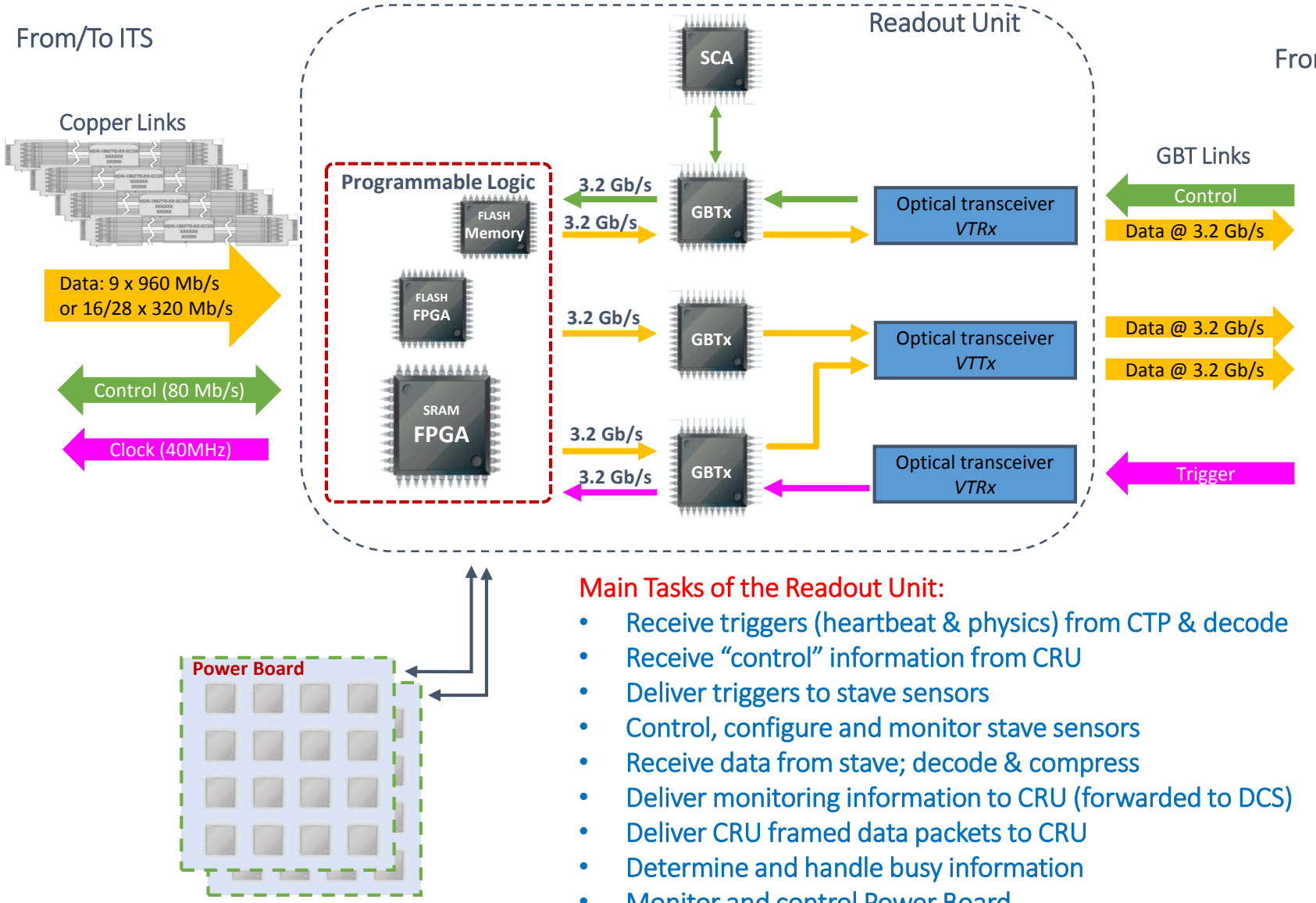
Firmware Overview

J. Schambach

Readout Unit Connectivity



Readout Unit Firmware Purpose

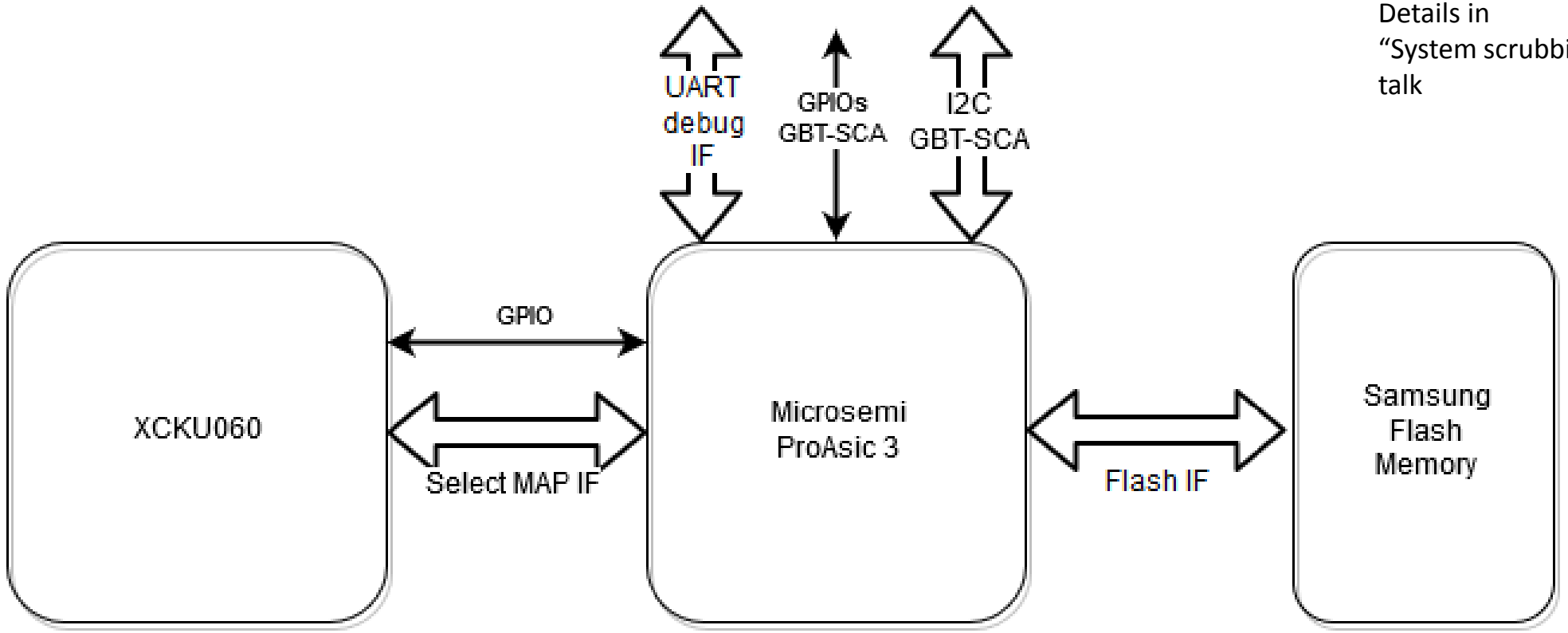


Main Tasks of the Readout Unit:

- Receive triggers (heartbeat & physics) from CTP & decode
- Receive "control" information from CRU
- Deliver triggers to stave sensors
- Control, configure and monitor stave sensors
- Receive data from stave; decode & compress
- Deliver monitoring information to CRU (forwarded to DCS)
- Deliver CRU framed data packets to CRU
- Determine and handle busy information
- Monitor and control Power Board
- Handle radiation upsets in programmable logic (& sensors)

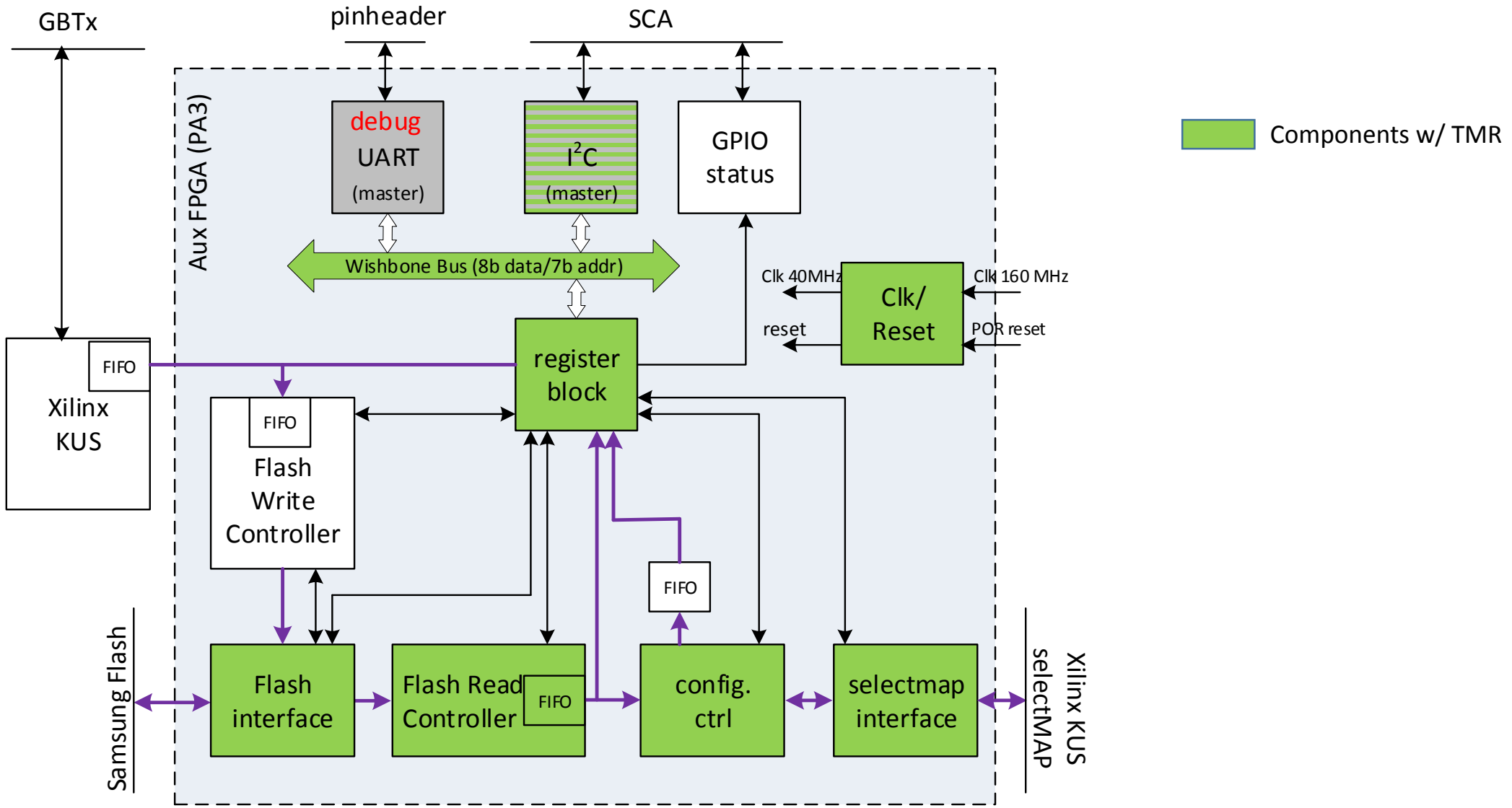
Flash FPGA

Flash FPGA: Configuration and Scrubbing



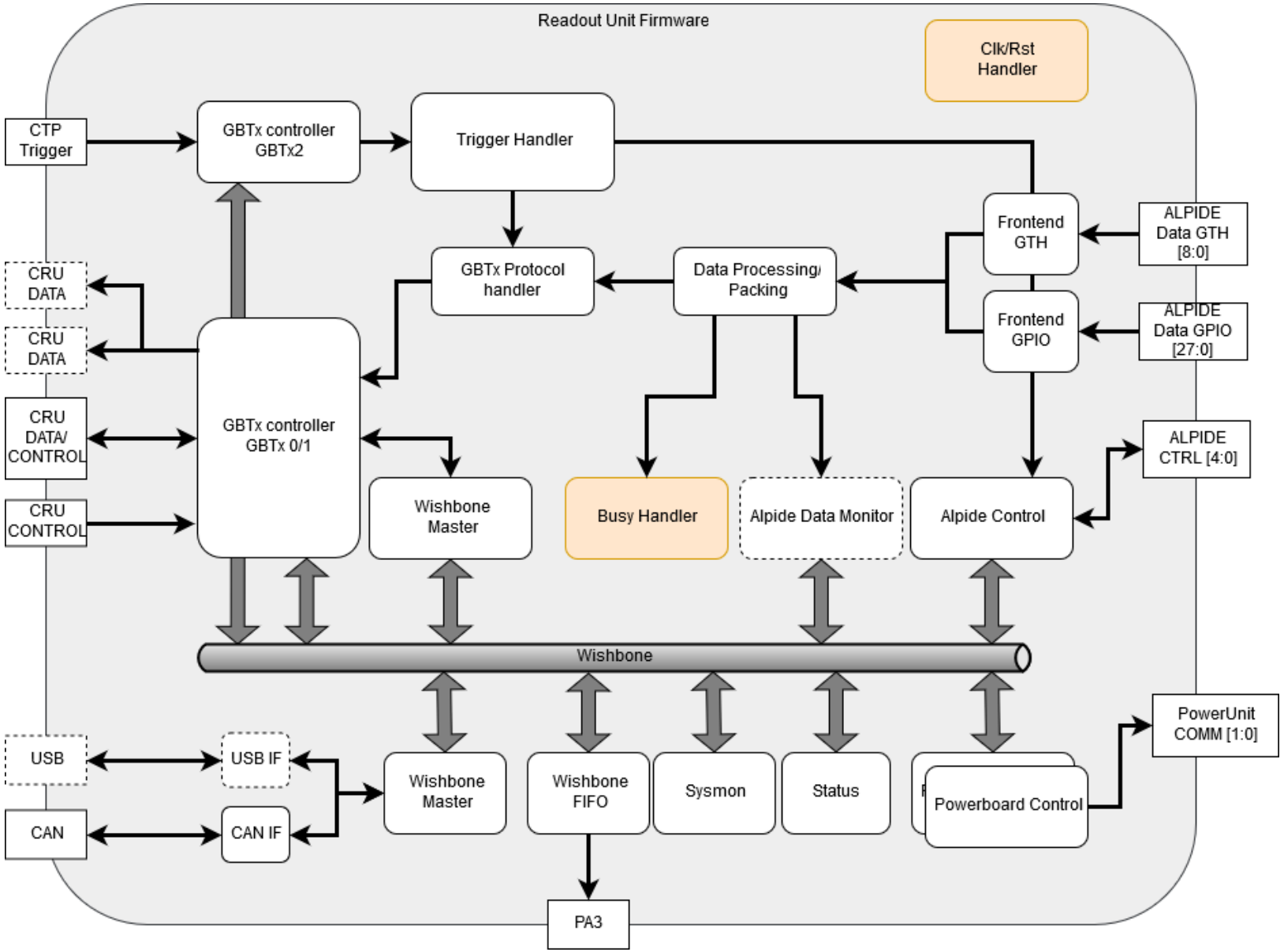
Details in
"System scrubbing & re-programming" talk

ProAsic3 Firmware



SRAM FPGA

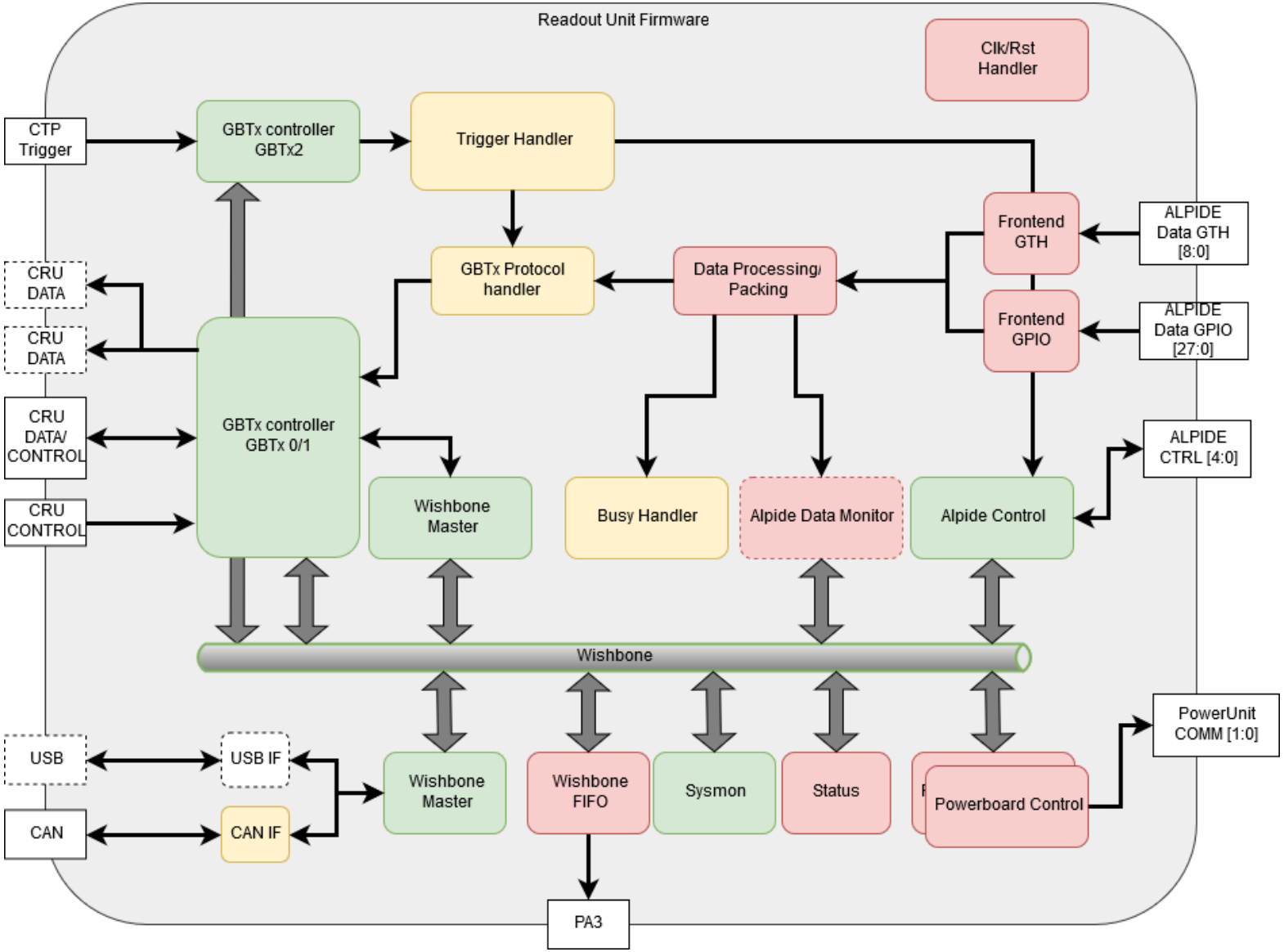
Readout Firmware Overview



Firmware modules - Status

- Control Interface (Wishbone bus)
  - Implemented and tested (lab, test beams)
  - Sending and receiving control words over USB and GBT
- Power Board Interface
  - Implemented and tested for all 4 I2C busses
  - Controls sensor power, read status, currents, and voltages
- Alpide Control
  - Implemented and tested for all 5 FireFly connectors
  - Handles configuration and triggers to sensors
- Data Readout and Packaging
  - 2 Flavors: Inner Barrel (MGT transceivers), Outer Barrel (GPIO fabric logic)
  - Implemented and tested for MGT (lab, test beams)
  - Outer Barrel path under development (only receiving logic different)
  - Receives data from sensors, deserialization, idle removal
  - Add CRU protocol and Trigger info
  - Send data packets over GBT
  - Some features still missing (OB sensor data packaging, sparsification)
- GBTx Communication
  - Implemented and tested (lab, test beams)
  - Use only single GBT link for now for trigger, control, and data
  - Full access to Wishbone via CRU protocol
- Trigger Handler
  - Proof of principle implemented and tested
  - Needs additional work (pipelining, additional triggers, …)
- Radiation Monitor
  - Implemented and tested
  - SEU observation
- System Monitor
  - Implemented and tested
  - Read voltages, temperatures
- USB interface to wishbone
  - Implemented and tested
  - Used for tabletop tests, lab setups
- FIFO interface to PA3
  - Implemented and tested
  - Used for fast firmware update of flash
  - Can be used from GBT (via CRU control protocol)
- Datapath Monitor
  - Implemented to monitor protocol, gbtx
- Error/Busy Handling
  - Still missing

Radiation Protection: Triple Modular Redundancy

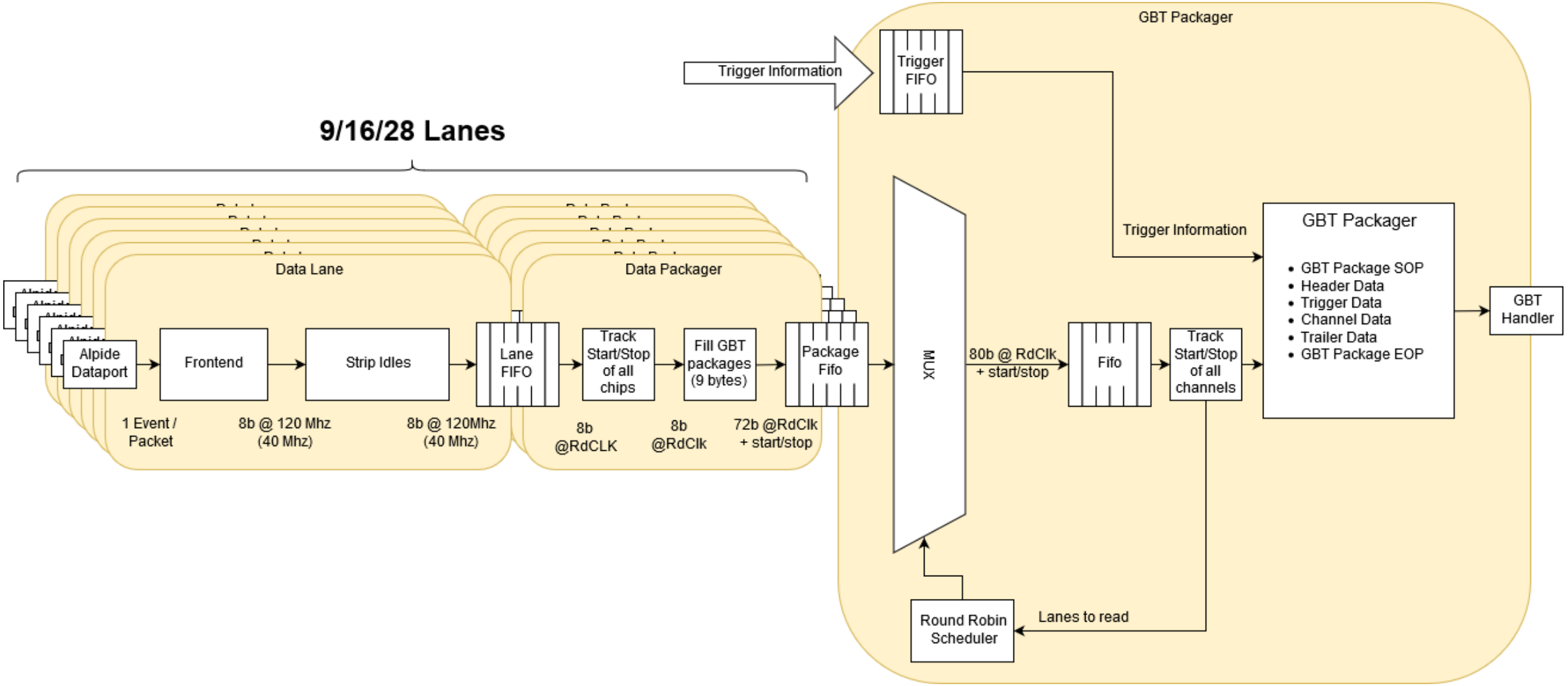


TMR Status

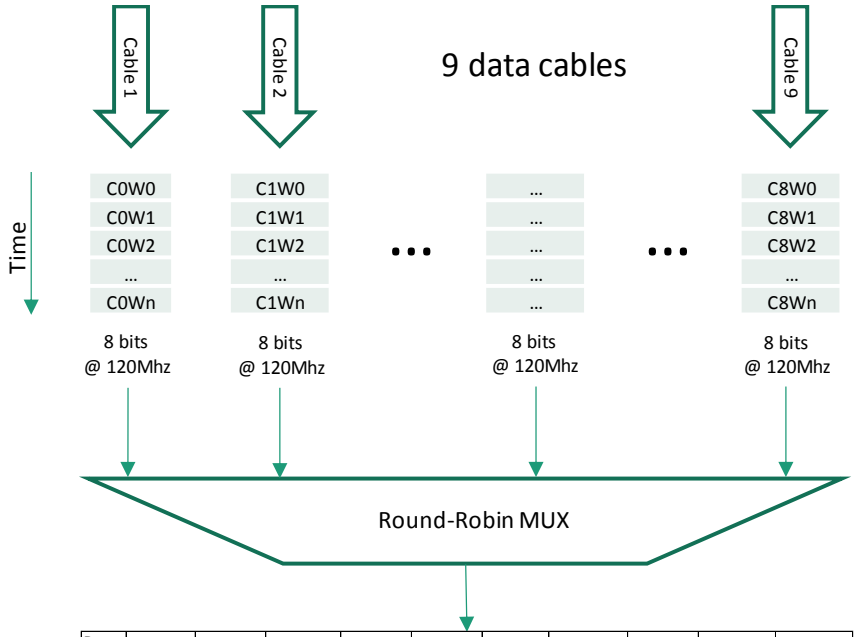
- TMR completed
- To be TMR'ed
- No TMR planned

Alpide Data Path Details

Data Flow


GBT Packager
Trigger Information
Trigger FIFO
9/16/28 Lanes
Data Lane
Data Packager
Alpide Dataport
Frontend
Strip Idles
Lane FIFO
Track Start/Stop of all chips
Fill GBT packages (9 bytes)
Package Fifo
MUX
80b @ RdClk + start/stop
Fifo
Track Start/Stop of all channels
Trigger Information
GBT Packager
GBT Package SOP
Header Data
Trigger Data
Channel Data
Trailer Data
GBT Package EOP
GBT Handler
1 Event / Packet
8b @ 120 Mhz (40 Mhz)
8b @ 120Mhz (40 Mhz)
8b @RdCLK
8b @RdClk
72b @RdClk + start/stop
Round Robin Scheduler
Lanes to read

Inner Barrel Data Format



Legend:

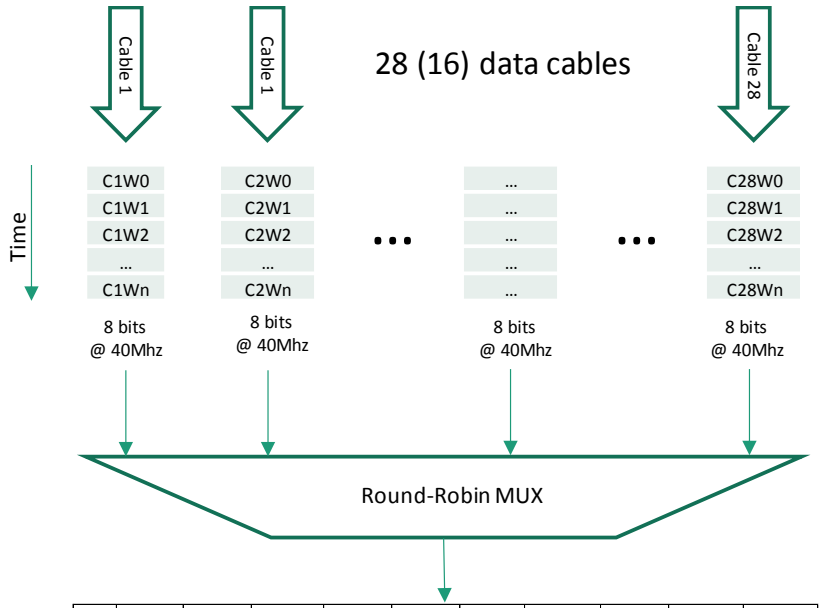
C<n>W<m>:	Cable <n>, Word <m>
Cx_Id:	FIFO identifier (1-9, 5bit)
SOP:	CRU Start-of-Packet
EOP:	CRU End-of-Packet
BC:	Bunch Crossing ID (12bit)

Header

Data Valid	72	64	56	48	40	32	24	16	8	0
0	Reserved						TTS Busy		Length	SOP
1	0x0		0x0	Priority Bit	FEE ID		Block Length		Header Size	Header Version
1	0x0		HB Orbit				TRG Orbit			
1	0x0		TRG TYPE				0	HB BC	0	TRG BC
1	0x0		0x0	Pages Counter		STOP BIT	PAR		Detector Field	
1	Status (e.g. busy)									
1	C1_Id	C1W8							C1W1	C1W0
1	C2_Id	C2W8							C2W1	C2W0
1	...	...							...	...
1	C9_Id	C9W8							C9W1	C9W0
1	C0_Id	C0W17							C0W10	C0W9
1	C1_Id	C1W17							C1W10	C1W9
1	C2_Id	C2W17							C2W10	C2W9
1	...	...								...
1	C9_Id	C9W17							C9W10	C9W9
1	...	...							...	...
1	...	...							...	...
1	C9_Id	C9W(n)							C9W(n-7)	C9W(n-8)
1	Status (e.g. missing data)									
0	0x0						Checksum		Length	EOP
0	0x0									IDLE

Data out of the Alpide Readout firmware module in the inner barrel mode arrives with a rate of 8 bits every 120MHz clock period. Nine 8-bit data words from one cable are combined with an ID for the FIFO they originate from into an 80 bit GBT word as shown here. A round-robin MUX collects data from each of the nine cables. Finally, the data is framed with the appropriate CRU protocol words, and trigger and status information is added to complete a CRU data packet.

Outer Barrel Data Format



Header

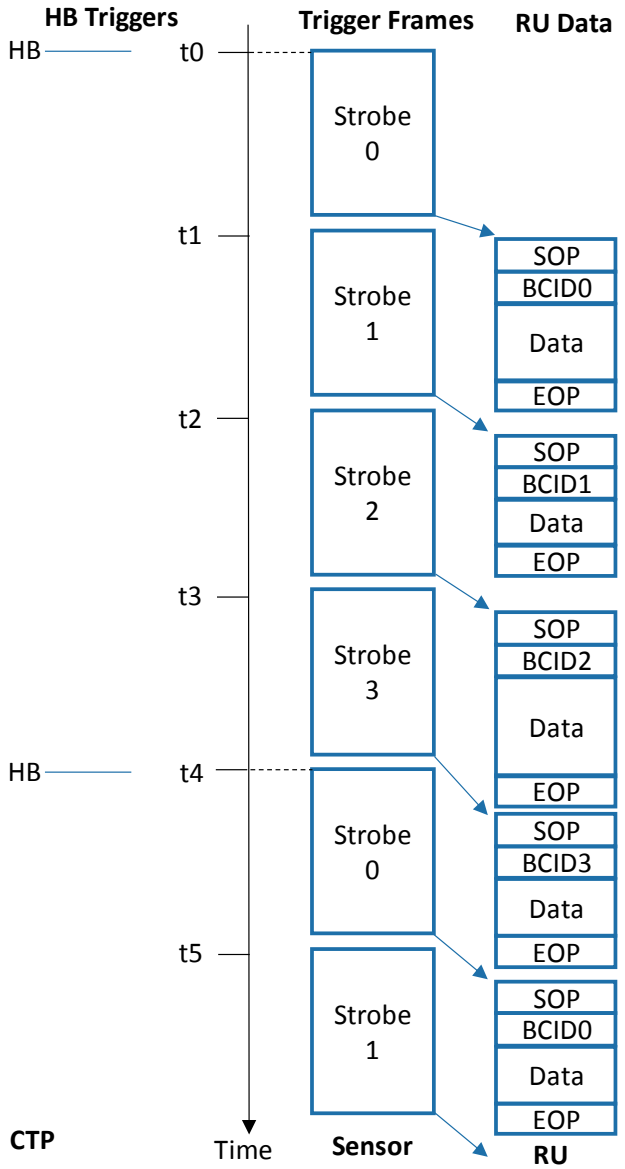
Data Valid	72	64	56	48	40	32	24	16	8	0
0	Reserved						TTS Busy		Length	SOP
1	0x0		0x0	Priority Bit	FEE ID		Block Length		Header Size	Header Version
1	0x0		HB Orbit				TRG Orbit			
1	0x0		TRG TYPE				0	HB BC	0	TRG BC
1	0x0		0x0	Pages Counter		STOP BIT	PAR		Detector Field	
1	Status (e.g. busy)									
1	C1_Id	C1W8							C1W1	C1W0
1	C2_Id	C2W8							C2W1	C2W0
1	...	...							...	...
1	C28_Id	C28W8							C28W1	C28W0
1	C0_Id	C0W17							C0W10	C0W9
1	C1_Id	C1W17							C1W10	C1W9
1	C2_Id	C2W17							C2W10	C2W9
1	...	...								...
1	C28_Id	C28W17							C28W10	C28W9
1	...	...							...	...
1	...	...							...	...
1	C28_Id	C28W(n)							C28W(n-7)	C28W(n-8)
1	Status (e.g. missing data)									
0	0x0						Checksum		Length	EOP
0	0x0									IDLE

Legend:

C<n>W<m>:	Cable <n>, Word <m>
Cx_Id:	FIFO identifier (1-28, 5bit)
SOP:	CRU Start-of-Packet
EOP:	CRU End-of-Packet
BC:	Bunch Crossing ID (12bit)

For the outer barrel mode, the same principle on the previous slide applies, but with modified rates and number of fragments combined into a GBT word.

Continuous Readout



- **HB** = Heart Beat Trigger
- **HBF** = Heart Beat Frame
- **Strobe<x>** = Trigger strobe to sensor with trigger time = BC+Orbit at time<x>
- **BCID<x>** = BC ID (part of SDH) (with BC+Orbit) at time<x> after HB
- **Data** = Sensor Pixel addresses
- **SOP** = "Start Of Packet" (CRU protocol)
- **EOP** = "End Of Packet" (CRU protocol)

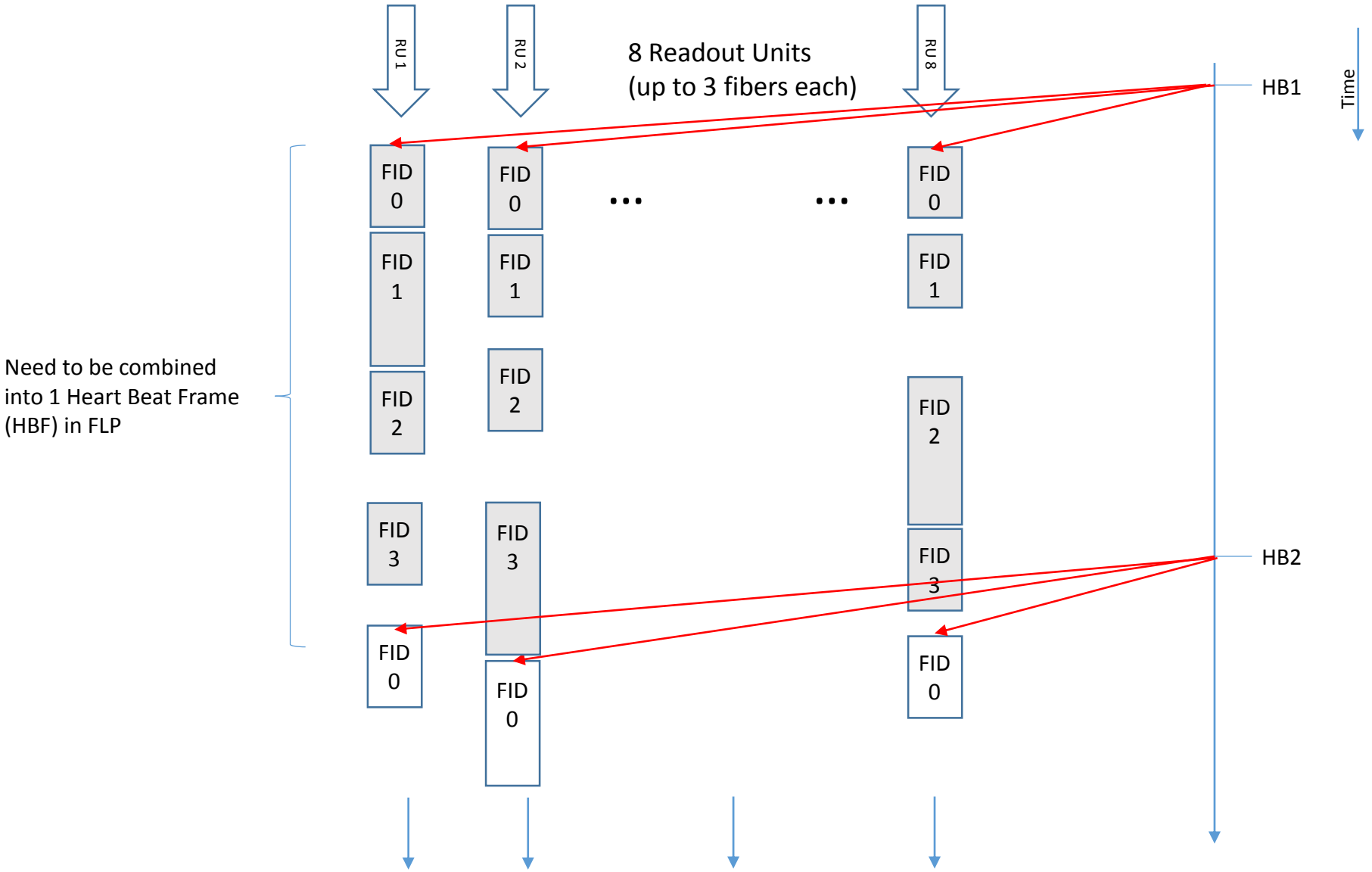
"Continuous" readout in ITS is accomplished by using "long" (10's of µ-seconds) strobe lengths (firmware triggered) with short (10's of nano-seconds) inter-strobe periods to initiate readout. ITS RU firmware therefore divides a "Heartbeat Frame" into several "Strobe" frames with fixed strobe lengths, synchronized to the received heartbeat triggers.

Heart Beat Triggers come every 89.4 µs, so trigger strobes ("frames") in this example would be approximately 89.4 / 4 = 22.3 µs long and each Heart Beat Frame (HBF) would thus contain 4 packets from the RU. This parameter is programmable and could be adjusted, e.g. to 89.4/8 to have 8 "sub-frames" in each HBF

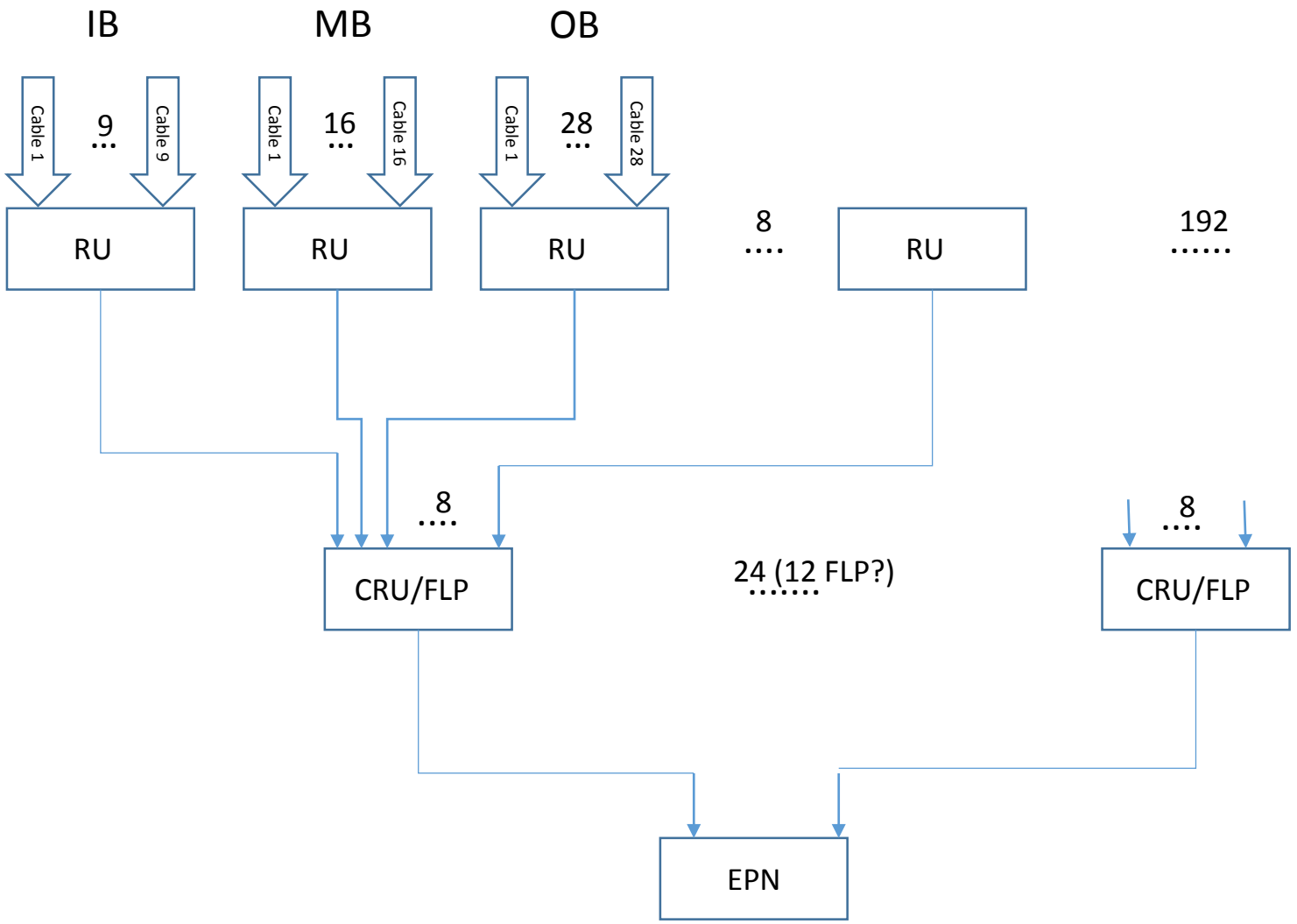
Data will appear from the sensor some tens of ns after the end of the strobe signal, and some 40MHz clock cycles later on the GBT links. Data contained in each packet corresponds to all physical signals ("events") between times tx and tx+1. The packets might also contain the busy status for individual sensors of an RU, which could then be used to determine the need for throttling at the CTP.

Although data packets are shown here contiguous, it is possible that "DATA" GBT words are interspersed with "IDLE" or "SWT" (for DCS data) during data transmission to the CRU. Since there is the possibility of SEUs in the RUs' FPGAs, there will be a "timeout provision" in the CRU protocol to determine if a packet is corrupted/incomplete/missing.

CRU Data


RU 1
RU 2
RU 8
8 Readout Units
(up to 3 fibers each)
HB1
Time
FID 0
FID 1
FID 2
FID 3
FID 0
FID 1
FID 2
FID 3
FID 0
FID 0
FID 1
FID 2
FID 3
FID 0
HB2
Need to be combined into 1 Heart Beat Frame (HBF) in FLP
DMA to FLP memory

Data Hierarchy

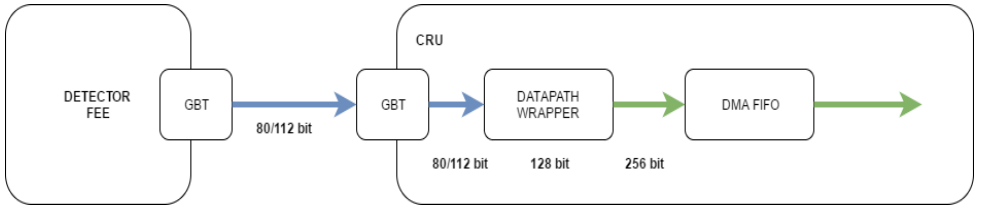


Data Format: GBT to CRU to FLP

GBT RDH

79.....0

16 bit	RDH WORD0 (64 BIT)
16 bit	RDH WORD1 (64 BIT)
16 bit	RDH WORD2 (64 BIT)
16 bit	RDH WORD3 (64 BIT)
DATA WORD 0	
DATA WORD 1	
DATA WORD 2	
DATA WORD 3	



CRU memory

127.....63.....0

16 bit	0x0	0x0	RDH WORD0 64 bit
16 bit	0x0	0x0	RDH WORD1 64 bit
16 bit	0x0	0x0	RDH WORD2 64 bit
16 bit	0x0	0X0	RDH WORD3 64 bit
16 bit	WIDE BUS		DATA WORD 0
16 bit	WIDE BUS		DATA WORD 1
16 bit	WIDE BUS		DATA WORD 2
16 bit	WIDE BUS		DATA WORD 3
16 bit	WIDE BUS		. . . . .

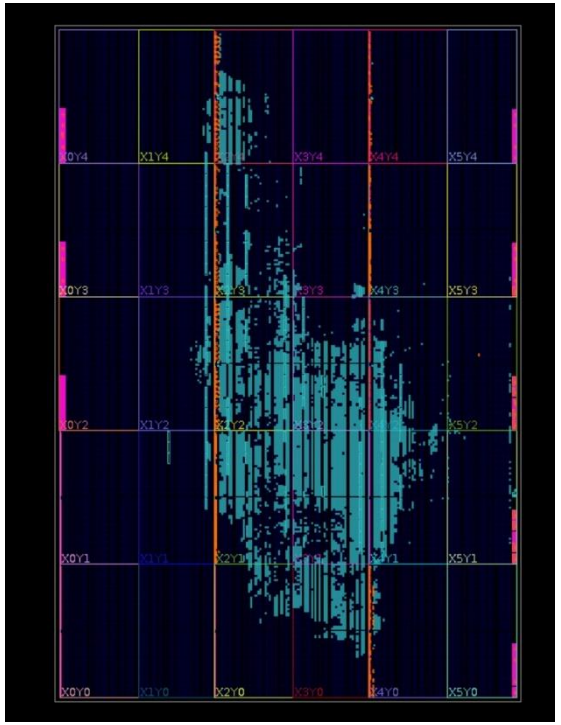
FLP memory

32 bit		0x00
32 bit		0x04
RDH WORD0 [63:32]		0x08
RDH WORD0 [31:0]		0x0C
32 bit		0x10
32 bit		
RDH WORD1 [63:32]		
RDH WORD1 [31:0]		
32 bit		
32 bit		
RDH WORD2 [63:32]		
RDH WORD2 [31:0]		
32 bit		
32 bit		
RDH WORD3 [63:32]		
RDH WORD3 [31:0]		
16 bit	WIDE 0x0	
WIDE 0x0	DATA WORD 0	
DATA WORD 0		
DATA WORD 0		
16 bit	WIDE 0x0	
WIDE 0x0	DATA WORD 1	
DATA WORD 1		
DATA WORD 1		

Utilization Report

CLB Logic

Site Type	Used	Available	Util %
CLB	6038	41460	14.56
CLBL	3062		
CLBM	2976		
LUT as Logic	23100	331680	6.96
using O5 output only	466		
using O6 output only	19961		
using O5 and O6	2673		
LUT as Memory	0	146880	0.00
LUT Flip Flop Pairs	6970	331680	2.10
fully used LUT-FF pairs	855		
LUT-FF pairs with one unused LUT output	5736		
LUT-FF pairs with one unissued Flip Flop	5160		
Unique Control Sets	931		
CLB Registers	26374	663360	3.98
CARRY8	757	41460	1.83
F7 Muxes	702	165840	0.42
F8 Muxes	287	82920	0.35
F9 Muxes	0	41460	0.00

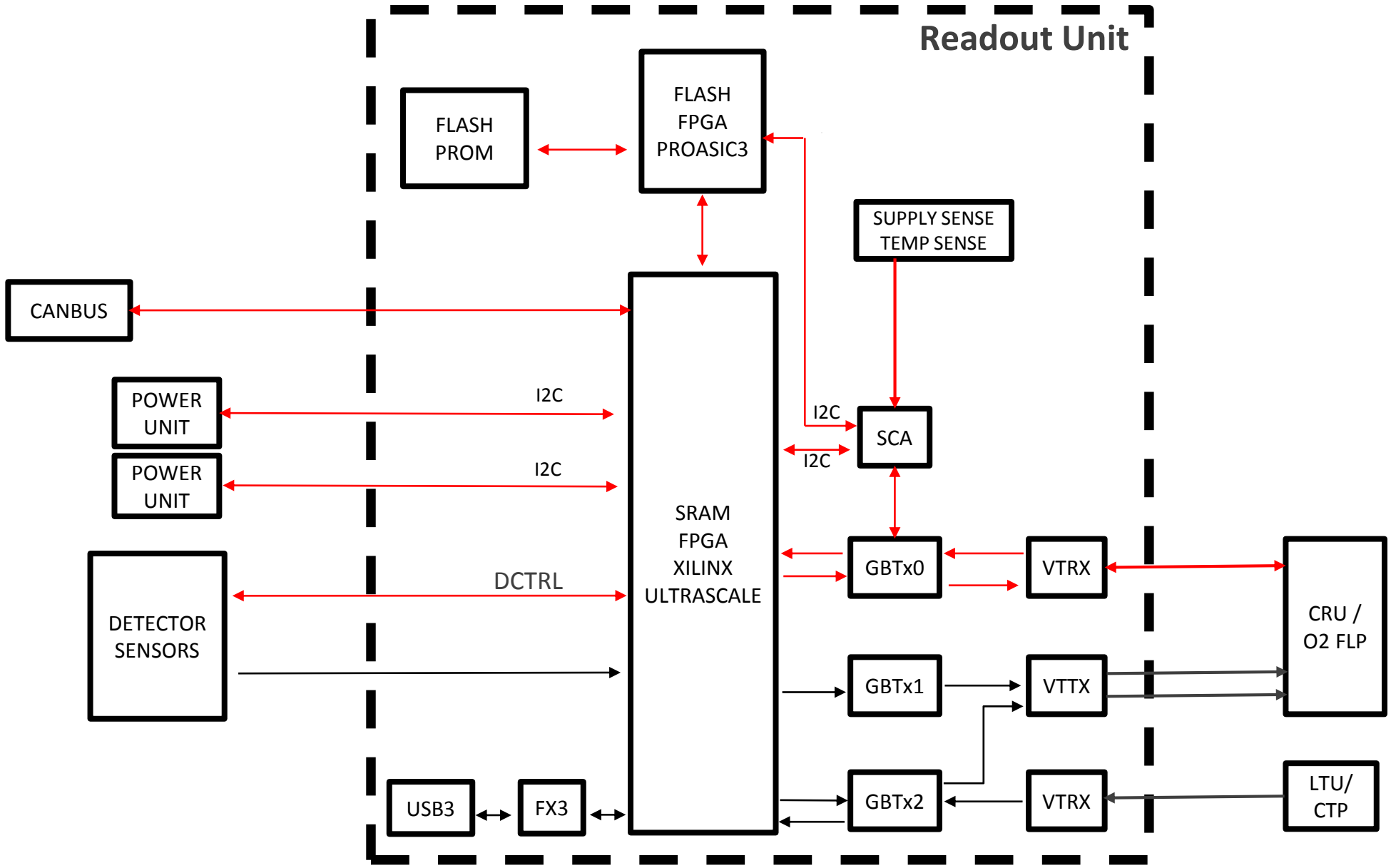


BLOCKRAM

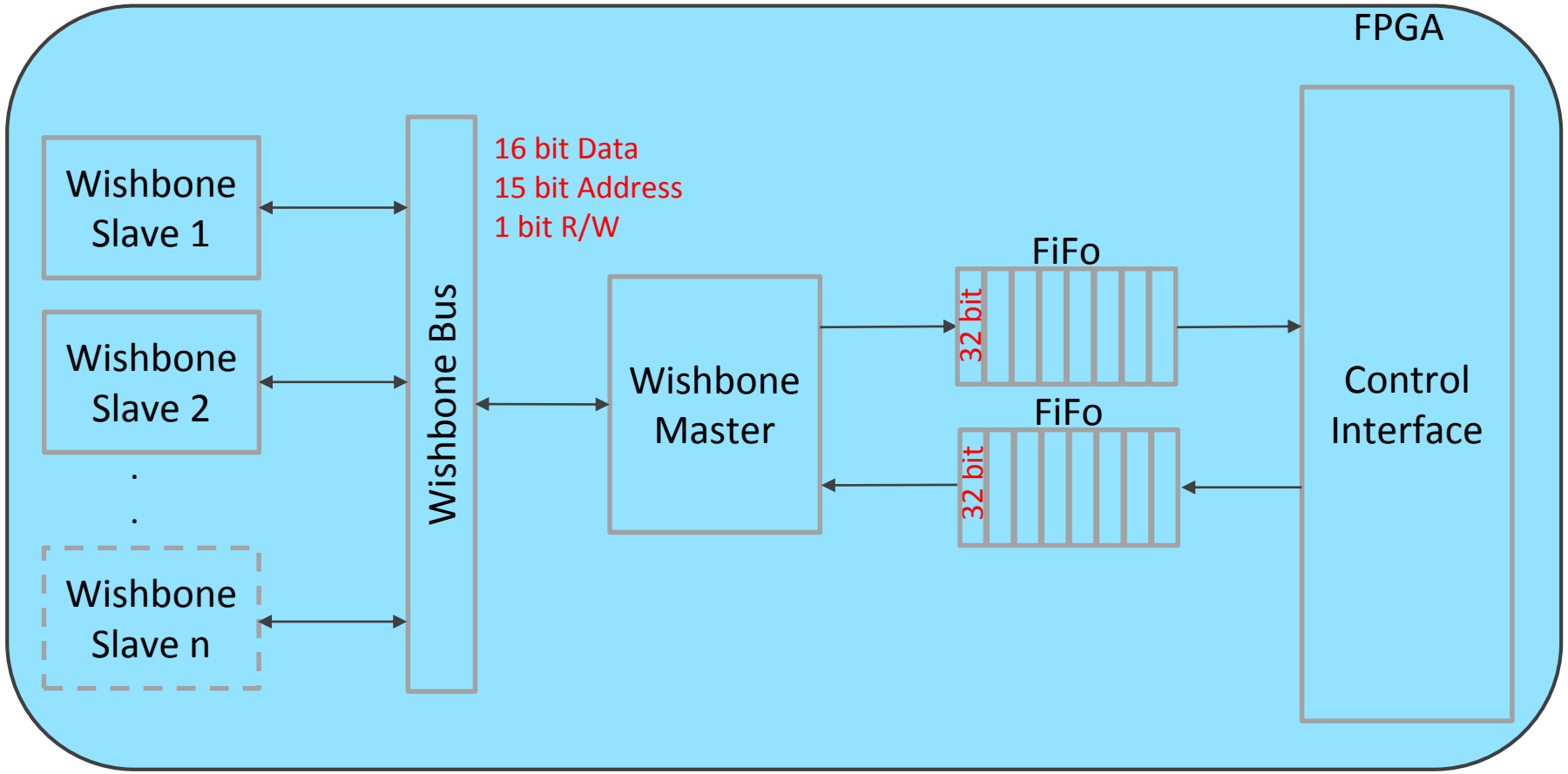
Site Type	Used	Available	Util %
Block RAM Tile	146	1080	13.52
RAM36/FIFO	134	1080	12.41
RAMB36E2 only	134		
RAM18	24	2160	1.11
RAMB18E2 only	24		

Control Interfaces

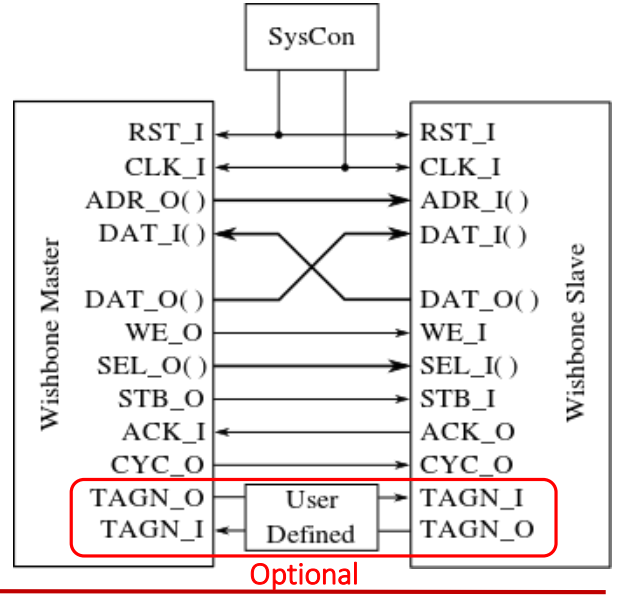
Detector Control System (DCS) Paths



Control Interface

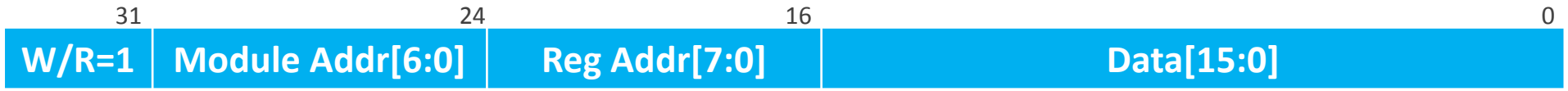


Wishbone Bus Structure

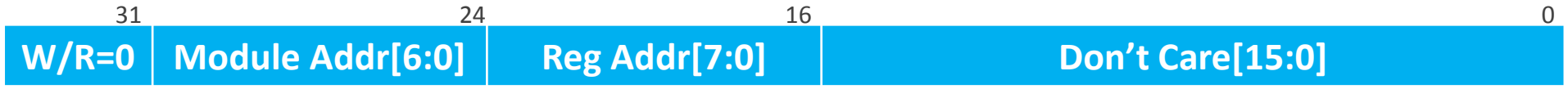


Controls Protocol

- Based originally on 32bit USB words sent over FX3
- **DCS -> FPGA**
  - *Write Request:*



  - *Read Request:*

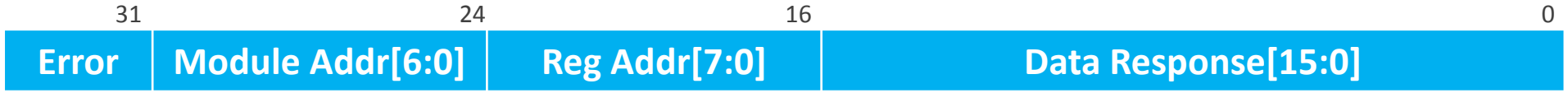


- **FPGA -> DCS**
  - *Write Response:*

    None (no acknowledgement, but errors are kept in a wishbone slave register – associated with the wishbone master - for possible later interrogation)

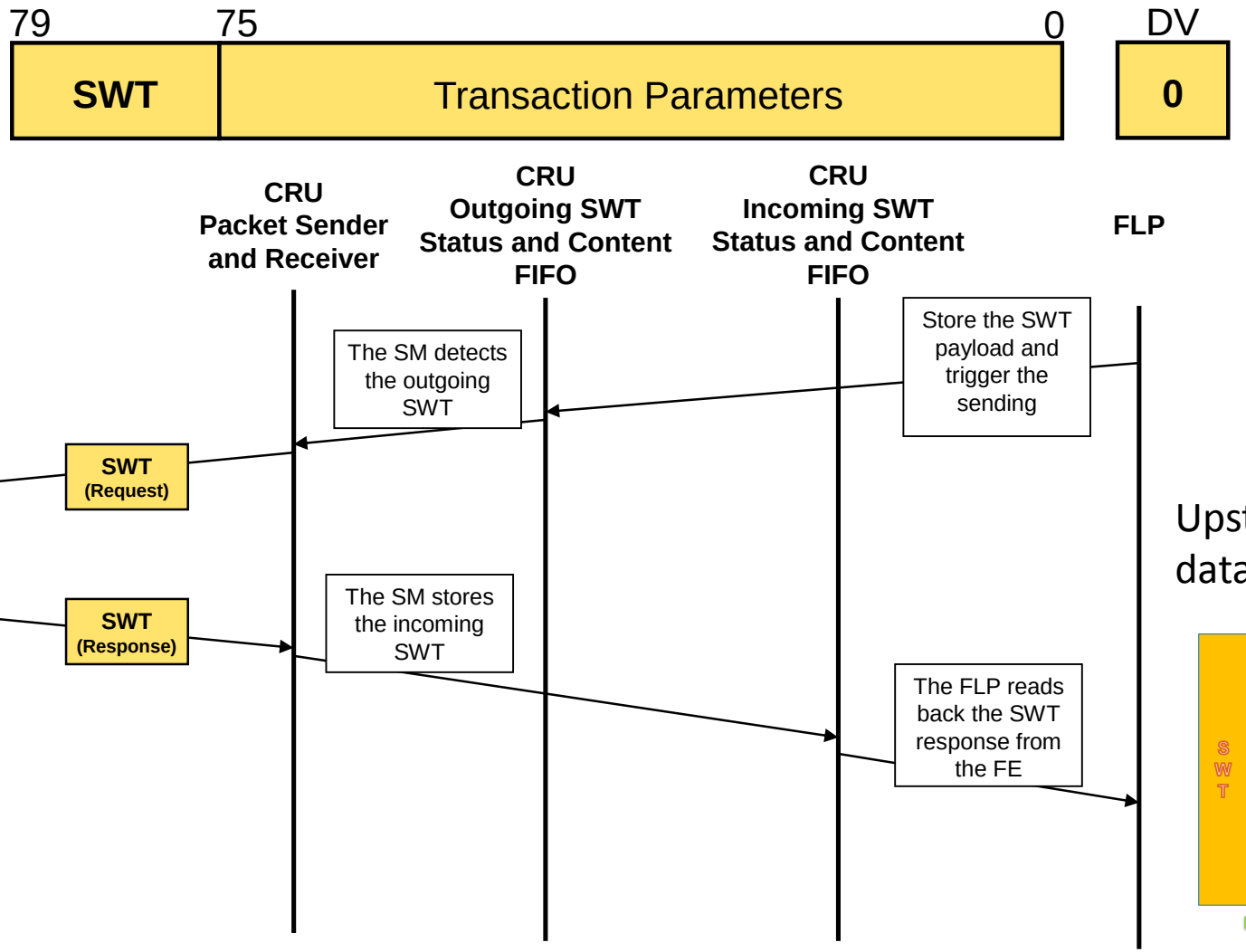
  - *Read Response:*

    Errors are kept in a wishbone slave register – associated with the wishbone master - for possible later interrogation



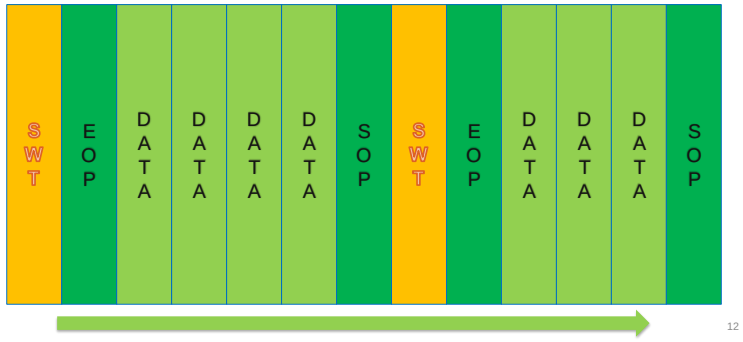
GBT Controls Transactions

GBT – SWT: Single Word Transaction



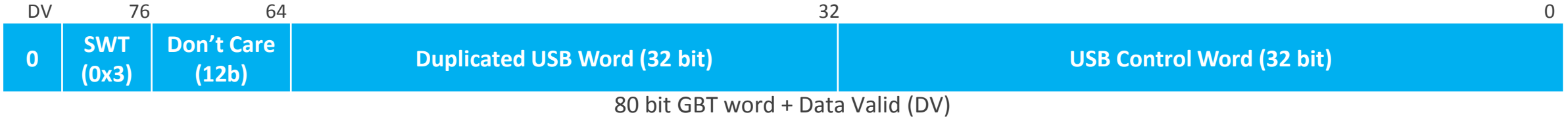
No problem on downstream path:
Only kind of communication

Upstream path: SWT interspersed between data packets



GBT Control Protocol

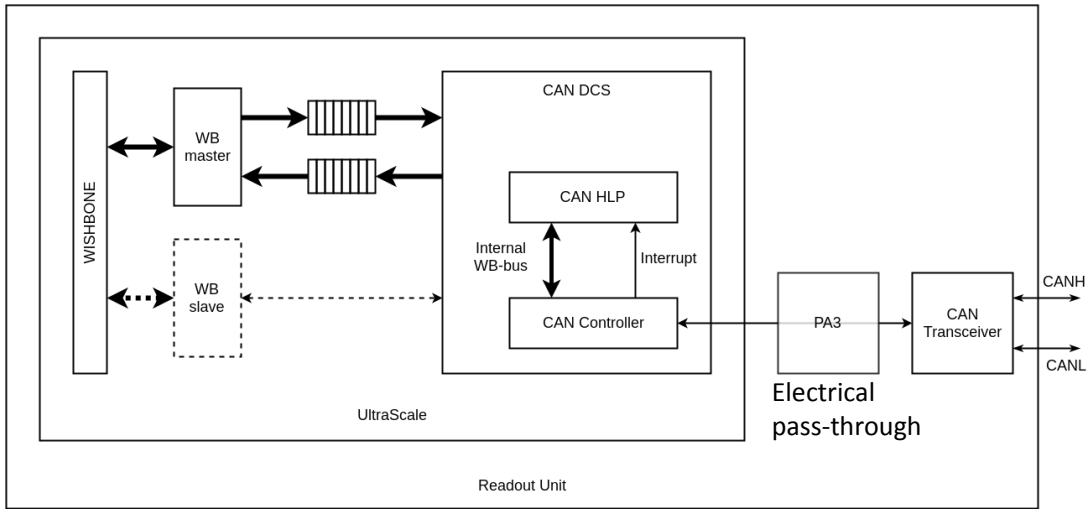
- Use CRU protocol “Single Word Transactions” (SWT) as a means for both downlink and uplink of control information
- Every USB transaction described in previous slides maps exactly to one SWT GBT word
- For radiation testing purposes, we implemented SWT words containing duplicated USB packets
- Downstream and Upstream GBT words have the same format:
  - Downstream GBT words contain request packets to the wishbone bus
  - Upstream GBT words contain response packets from the wishbone bus



80 bit GBT word + Data Valid (DV)

- A Python module for interacting with the CRU “SWT handler” using the above protocol was developed based on the O^2 Python module “libReadoutCard”
- The User Interface of this “SWT” module are the following functions:
  - clrFifos(): Clear the SWT FIFOs in the firmware
  - write(mod, addr, data): Write “data” to module “mod” at register address “addr”
  - read(mod, addr): Read register “addr” in module “mod”

CANbus Controls Transactions



- CANbus Opencores controller for raw packets
- **H**igh-**L**evel-**P**rotocol on top of controller to define a node based communication interface and to define simple READ and WRITE transactions & responses with appropriate payload
- Interface to wishbone bus same as GBT:
  - Input & Output FIFOs
  - Wishbone Master for transactions on bus
- WB slave interface in controller to configure protocol parameters like baud-rate and filters

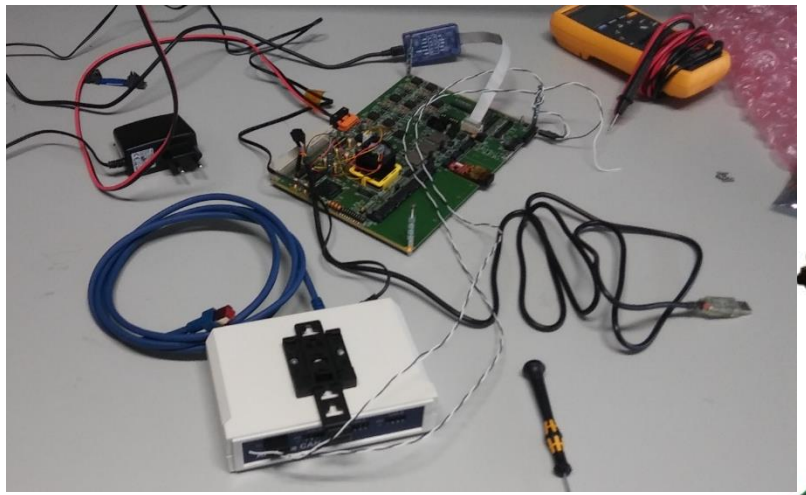
Emulate a “network topology”: Divide 11-bit CAN ID field into a “Node ID” and a “Command” field:

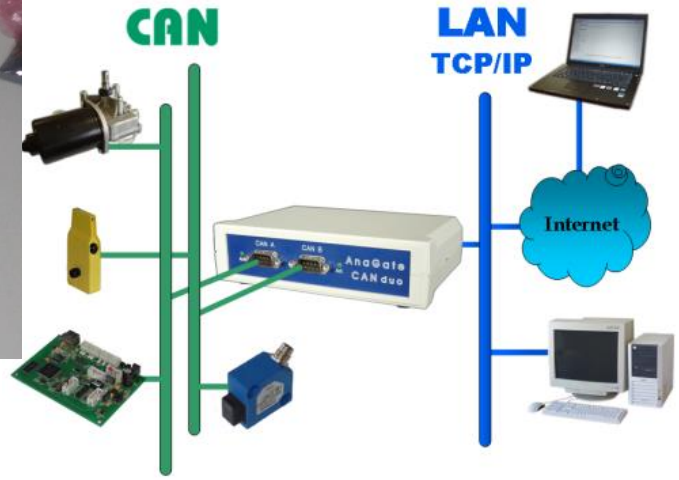
MsgID[10:4] = N[6:0] = NodeID
MsgID[3:0] = C[3:0] = Command

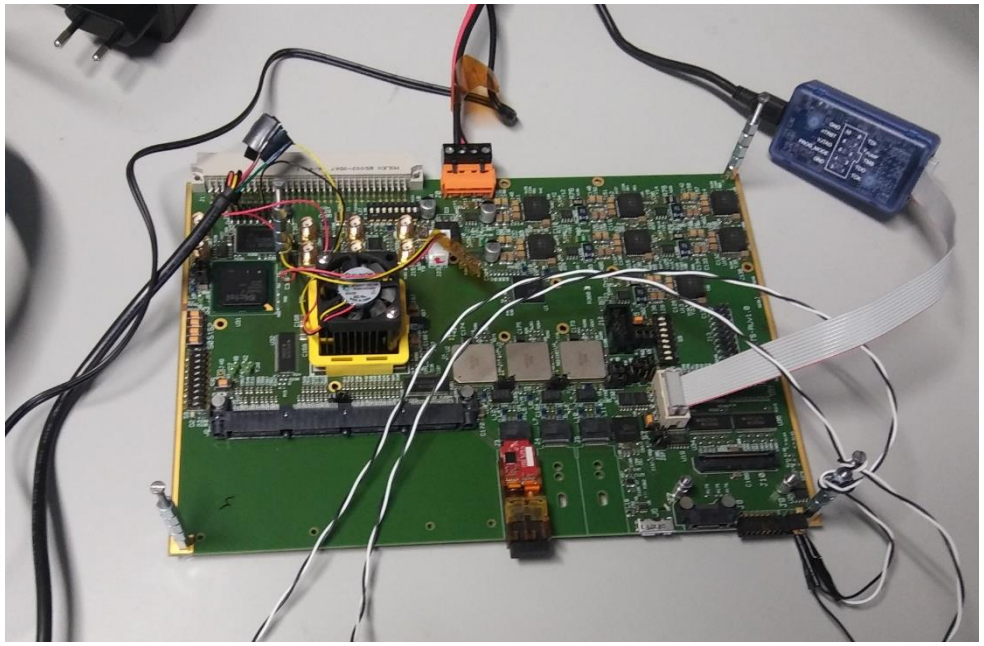
ID[10]	ID[9]	ID[8]	ID[7]	ID[6]	ID[5]	ID[4]	ID[3]	ID[2]	ID[1]	ID[0]
N[6]	N[5]	N[4]	N[3]	N[2]	N[1]	N[0]	C[3]	C[2]	C[1]	C[0]

Allows for up to 128 nodes per CAN network

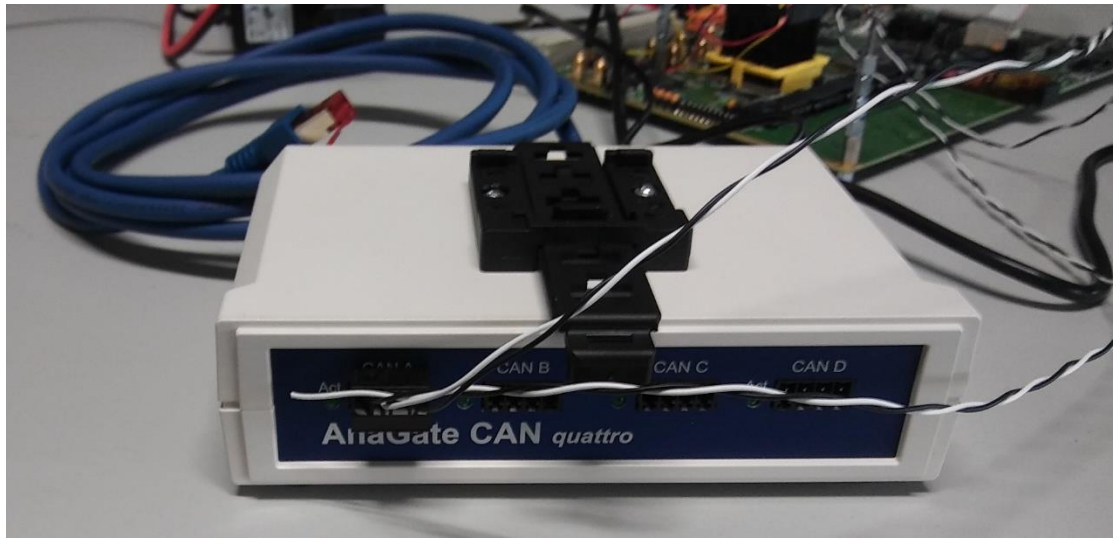
CANbus Test Setup



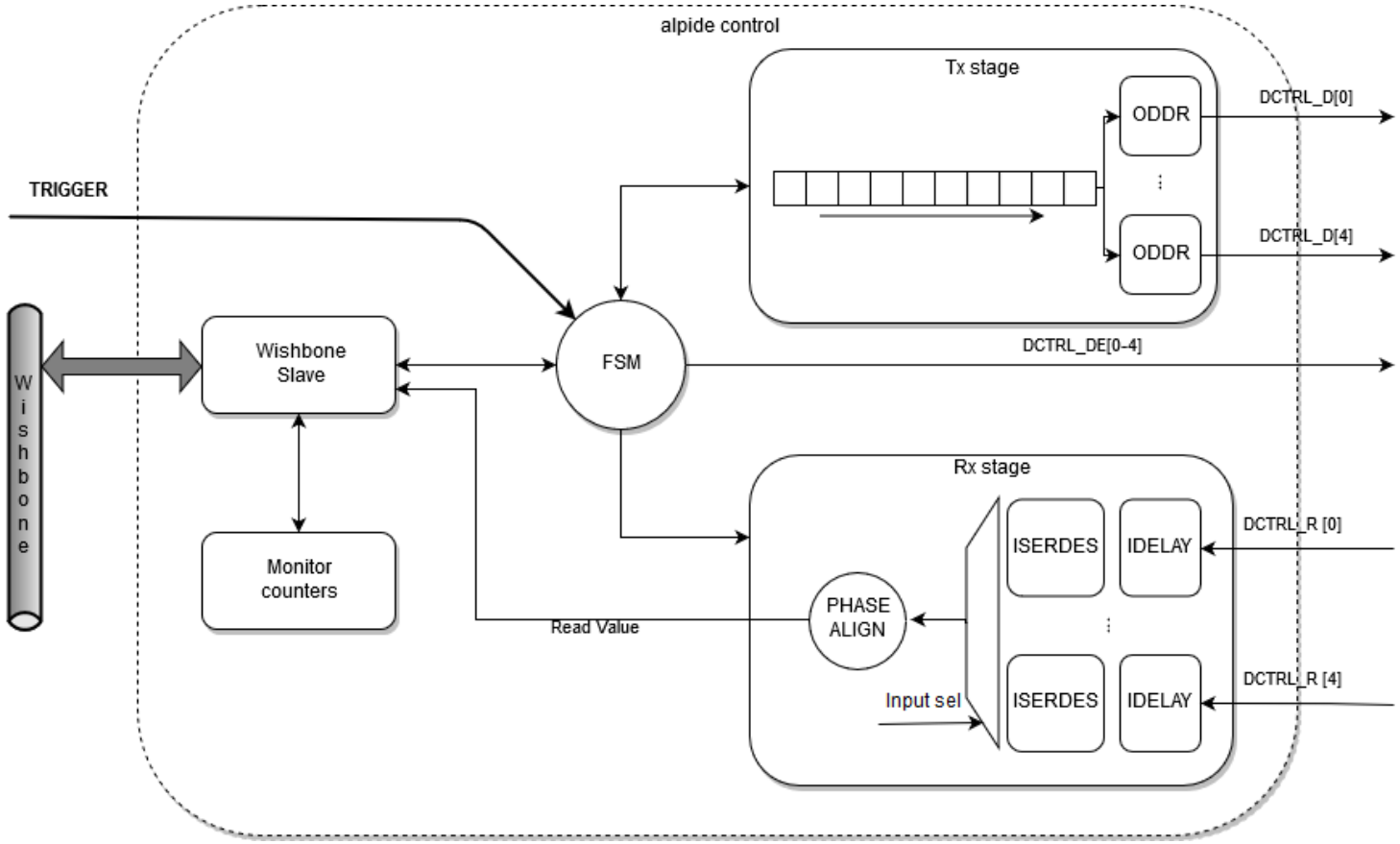




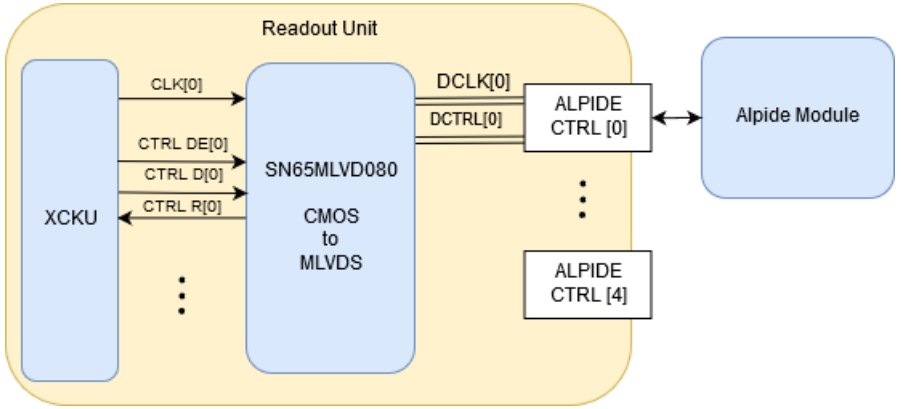
- Anagate “CAN Quattro” CAN controller
- Opencores “Project CAN” protocol controller
- Wishbone interface to PA3 firmware for configuration
- Configured for 1Mbps baud rate
- “CAN Monitor” from Anagate to generate and receive standard and extended ID CAN packets with up to 8 bytes payload



Alpide Control Implementation



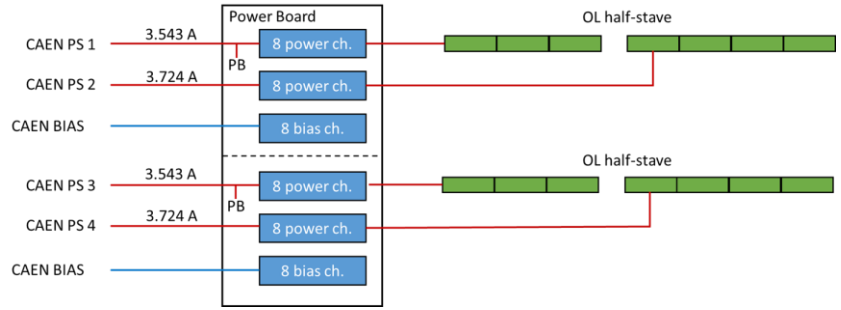
Firmware Logic



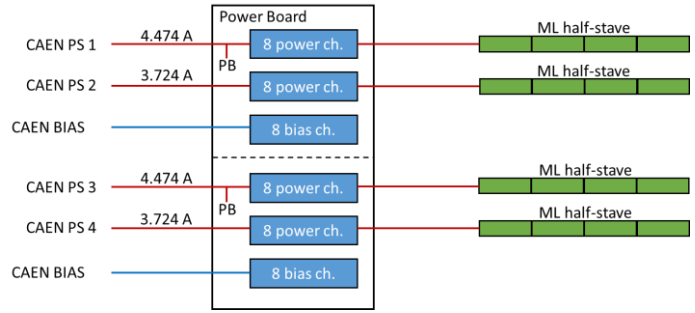
Physical Implementation

Power Board – Interface to Readout Unit

- Two 8-channel **Power Units** are combined into one board stack called a "**Power Board**".
- Each **Power Unit** provides power for up to **8** modules (16 per Power Board)
- In **layers 5 & 6** each **Power Board** provides power for **one full stave** (one power unit per half-stave)
- In all other layers, each **Power Unit** provides power for **one stave**, so there are **2** staves per **Power Board**
- Each **Power Unit** is interfaced to a Readout Unit via **2** (differential) **I2C buses**, with multiple devices per bus
- Each Power Board consisting of 2 Power Units therefore has a total of four I2C buses
- In **layers 5 & 6**, each **Power Board** interfaces to one **Readout Unit** (a total of four I2C buses per Readout Unit)
- In all other layers, each **Power Unit** interfaces to one **Readout Unit** (a total of two I2C buses per Readout Unit)
- Control via two I2C wishbone slaves (one per Power Unit) allows setting and monitoring voltages, currents, temperature thresholds and power status



Outer Layer Configuration



Middle Layer Configuration

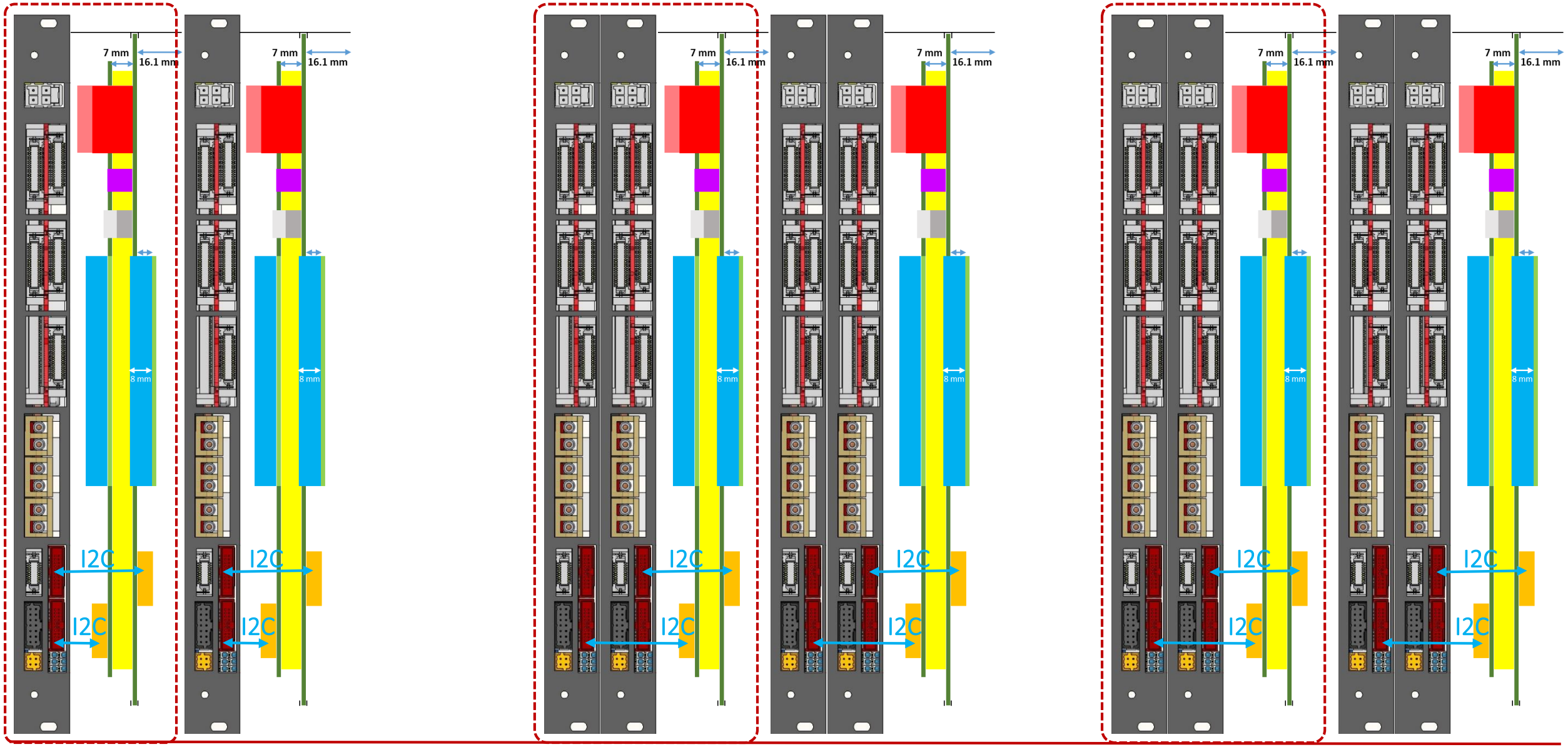
Power Board – Readout Unit Pairing

ALICE ITS UPGRADE

Outer Layers

Middle Layers

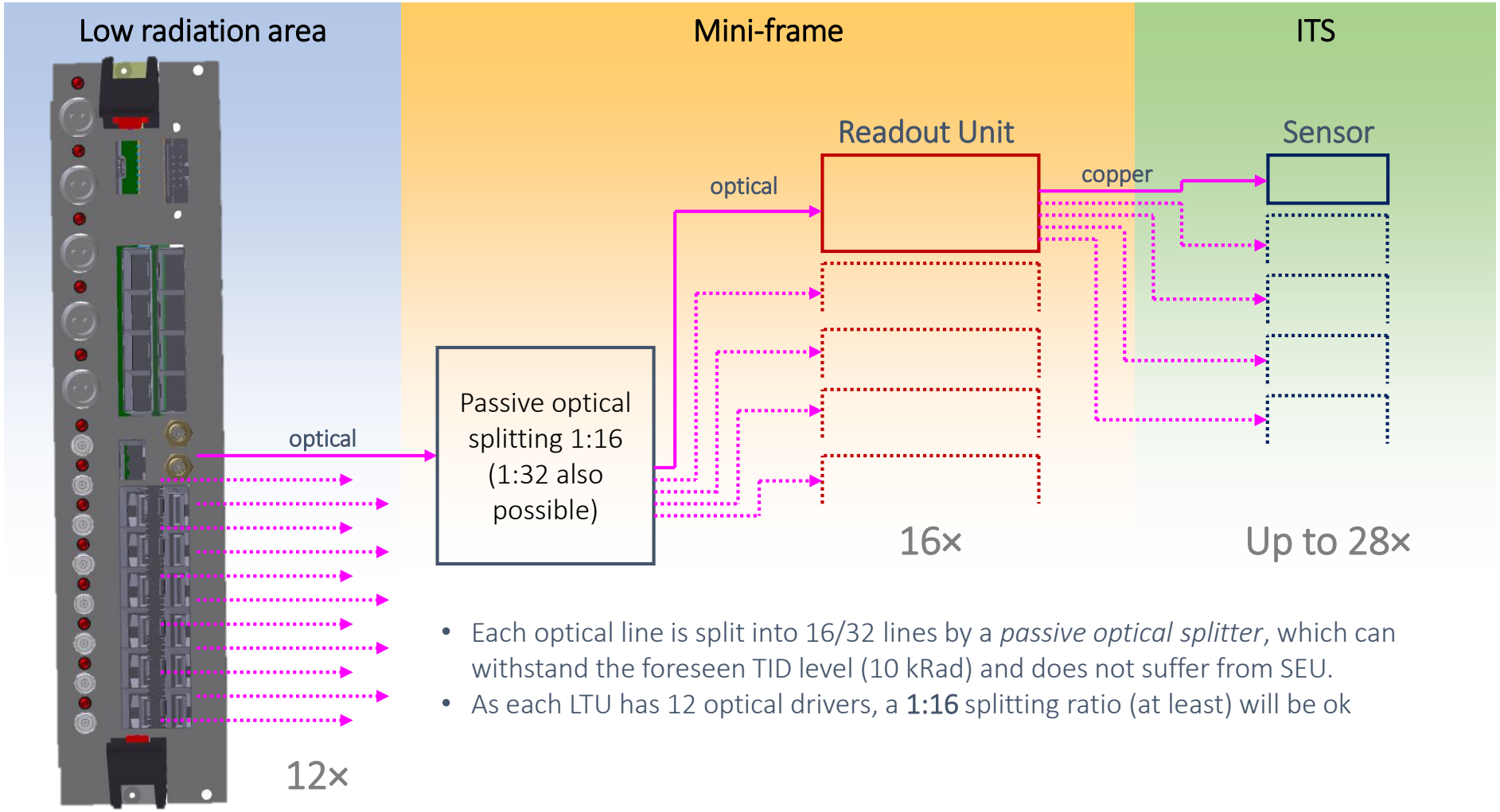
Inner Layers



Trigger Interface – Optical Splitting

The CTP directly drives a set of **L**ocal **T**rigger **U**nits (LTUs), each responsible for delivering the trigger to a specific detector.

- ITS will have its own LTU
- ITS LTU will drive 12 (or 6) Single Mode, 1030nm SFP+ Modules.



- Each optical line is split into 16/32 lines by a *passive optical splitter*, which can withstand the foreseen TID level (10 kRad) and does not suffer from SEU.
- As each LTU has 12 optical drivers, a **1:16** splitting ratio (at least) will be ok

Optical Fiber Power Loss Tests

Splitter	Power loss Ideal splitter [dB][1]	Power loss Measured [dB]	Margin [dB][2]
1:2 splitter	3	3.686	16.689
1:4 splitter	6	6.838	13.539
1:8 splitter	9	10.535	9.841
1:16 splitter	12	14.333	6.044
1:32 splitter	15	16.359	4.983

[1] The numbers for ideal power loss is taken from http://www.thefoa.org/tech/ref/testing/test/couplers.html

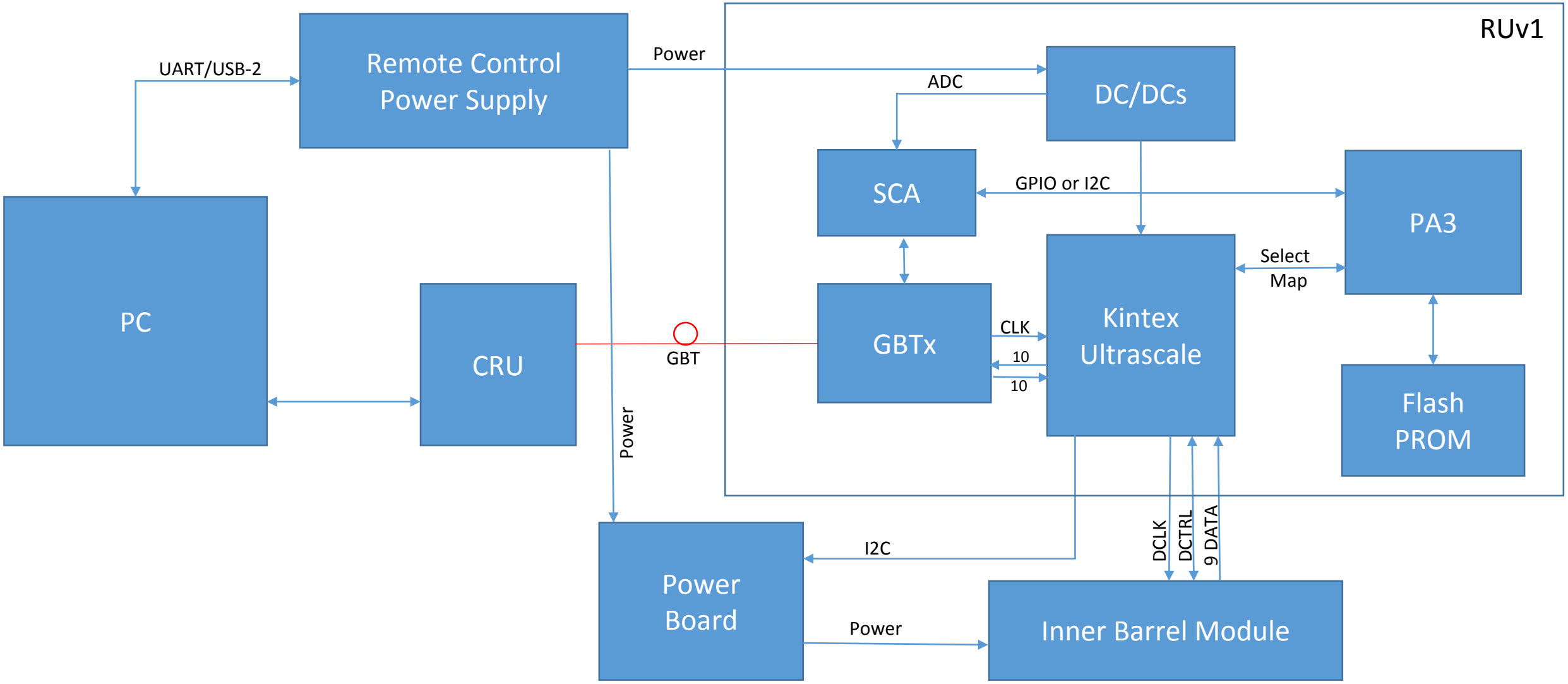
[2] Margin is given based on:
*- the **Avago AFCT-57D3ANMZ** transmitter (optical power output **+5 dBm**)*
*- the **VTRx** sensitivity (**-15.376 dBm**)*

[3] Fiber optics association: http://www.thefoa.org/tech/lossbudg.htm

- As a general rule, the link loss margin should be greater than approximately 3 dB to allow for link degradation over time[3].
- The dynamic range for the chosen Transmitter & Receiver is about 20 dB
  - The LTU has been verified to support the powerful Avago AFCT-57D3ANMZ transmitter.
- The power tests shows that all topologies have a margin > 3 dB
  - This is in line with the specifications given by the manufacturer of the splitter

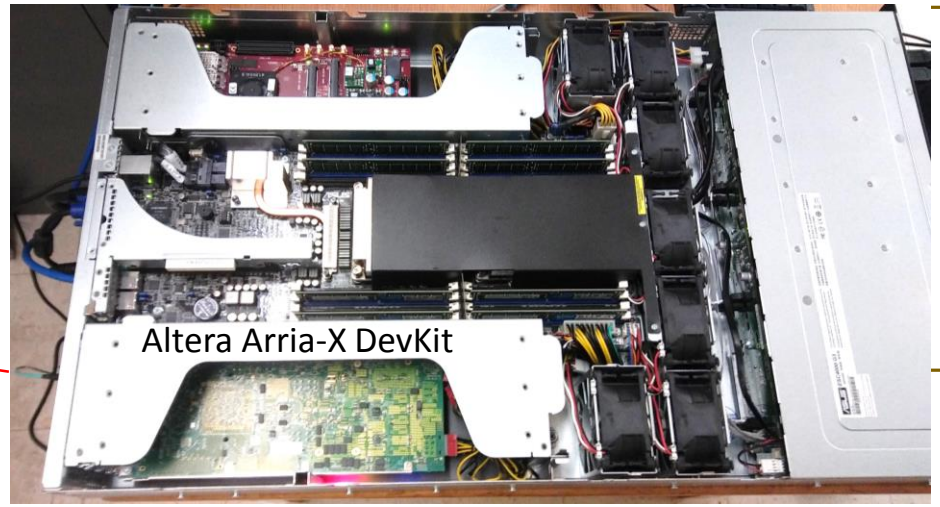
Data Streaming

Schematic Setup for Data Streaming Tests



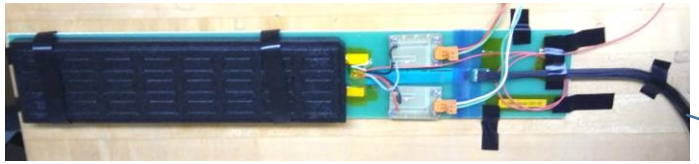
Data Streaming Test Setup

CRU

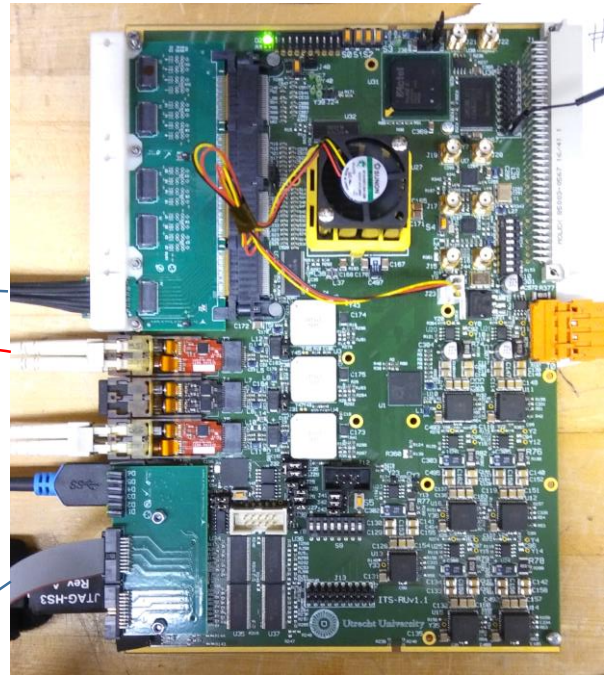


FLP

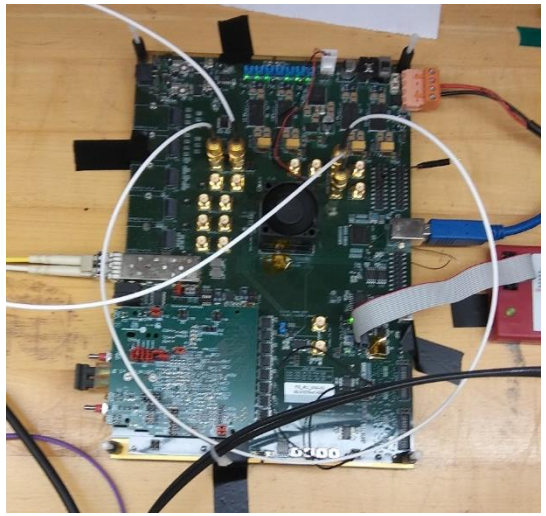
Inner Barrel Module



Power Board Prototype



RUv1

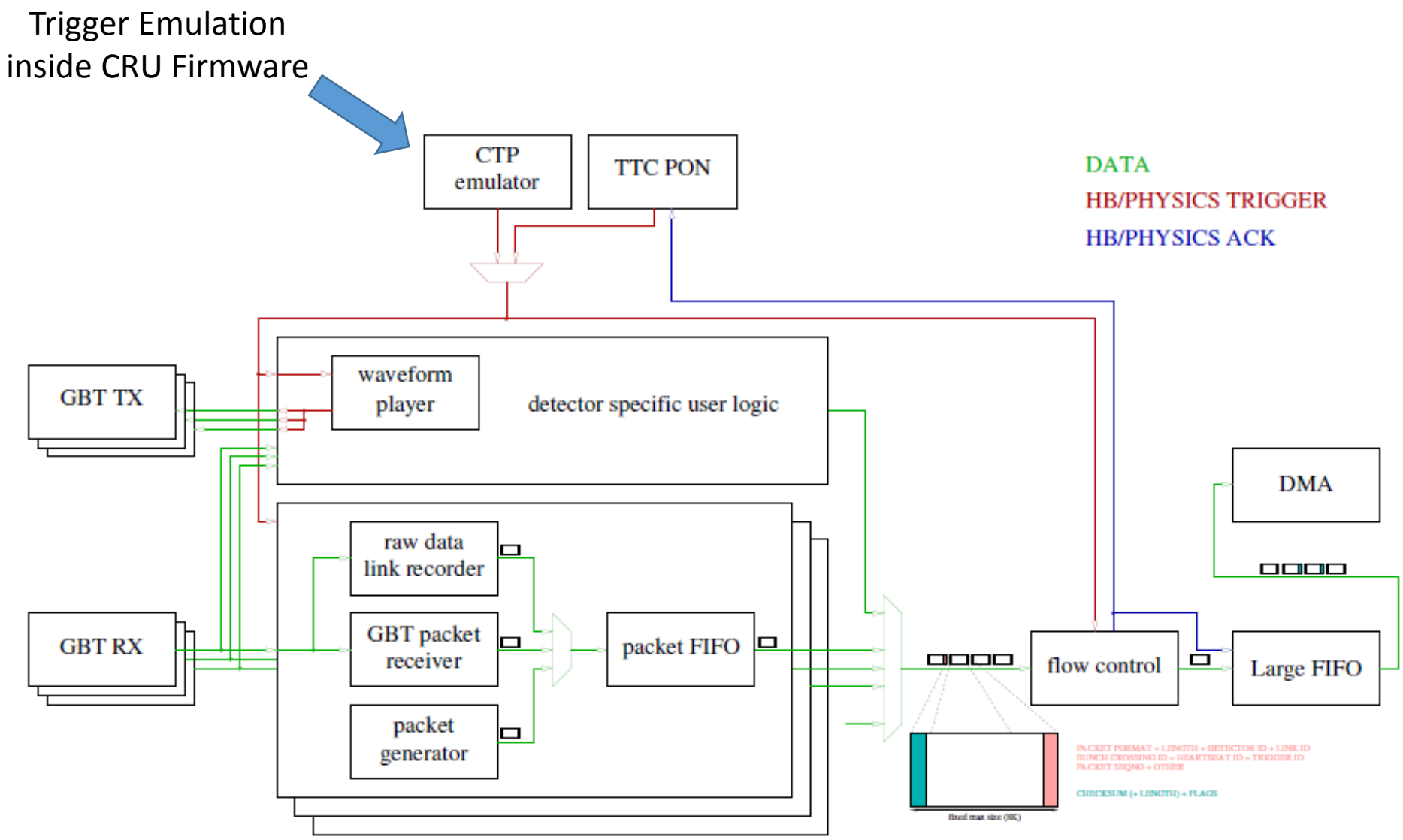


“CRU” Emulator

CRU Interface Tests

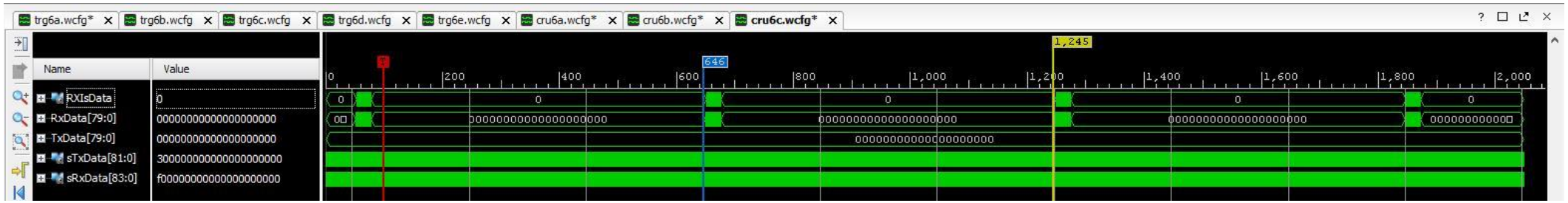
- Three different Interfaces from CRU to Readout Unit tested: **SCA, SWT, Data**
- SCA interface provided by both C++ and Python library via register write/read.
- Tested with Python interface library to read voltages, currents and temperatures from RU SCA
- SWT interface provided by both C++ and Python library via registers that access SWT FIFOs in CRU firmware
- Tested with Python interface library to read and write registers from various wishbone slaves
- Data Path Configuration interface via C++ library to interface with configuration registers
- Setup Data Streaming into one of 3 modes:
  - Packet Mode
  - Raw Link streaming (continuous mode)
  - Packet Emulator (for testing)
- Trigger Interface Path with configurable source:
  - CTP Emulator (firmware from CTP group)
  - TTC PON (from LTU)
- Data output via DMA over PCIe to memory of FLP in 8kByte packets
- Option to record memory pages to a disk file for later analysis

CRU simplified logical data flow



Data Arrival in CRU - Continuous and Triggered

Continuous Readout

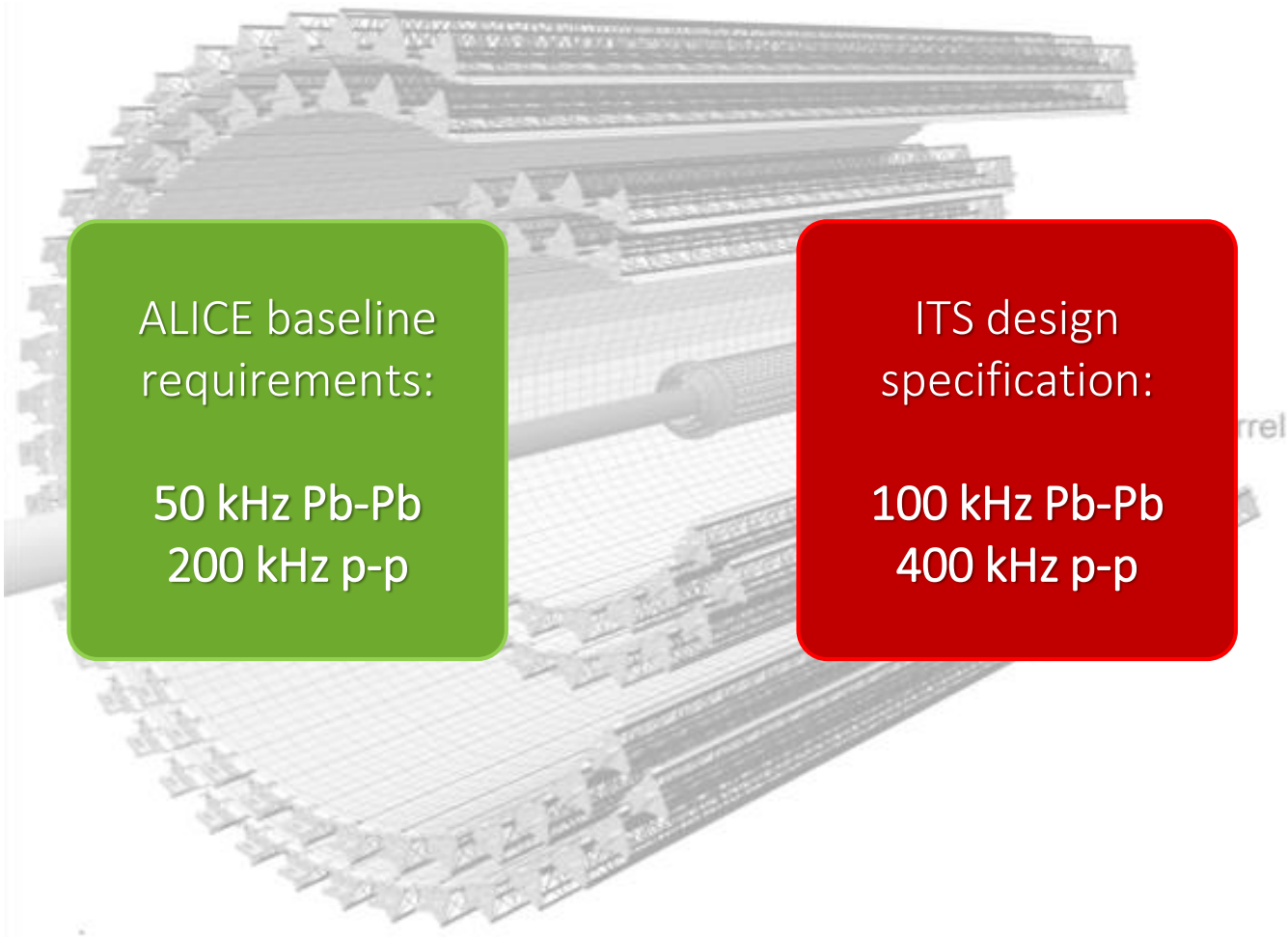


Triggered Readout

CRU: “Readout Card” (ROC) Interface library

roc-list-cards	Lists the readout cards present on the system, along with their type, PCI address, vendor ID, device ID, serial number, and firmware version.
roc-reg-read **roc-reg-write**	Writes/reads registers to/from a card's BAR. By convention, registers are 32-bit unsigned integers.
roc-reset	Resets a channel of the card.
roc-bench-dma	DMA throughput and stress-testing benchmarks.
roc-flash	Write a file to card’s flash (only supports C-RORC)

ITS Readout – Data Rate Specifications



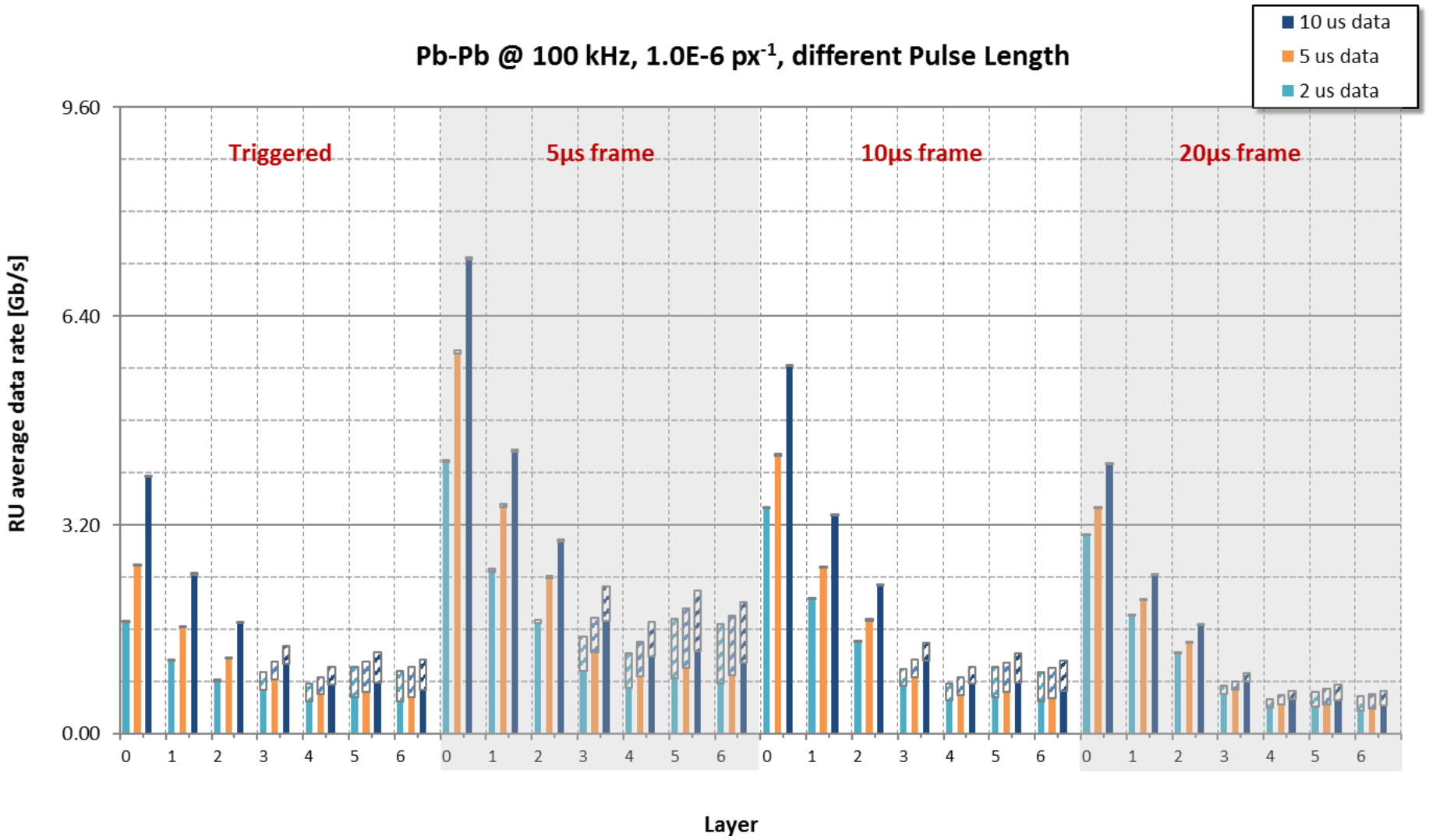
ALICE baseline requirements:

50 kHz Pb-Pb
200 kHz p-p

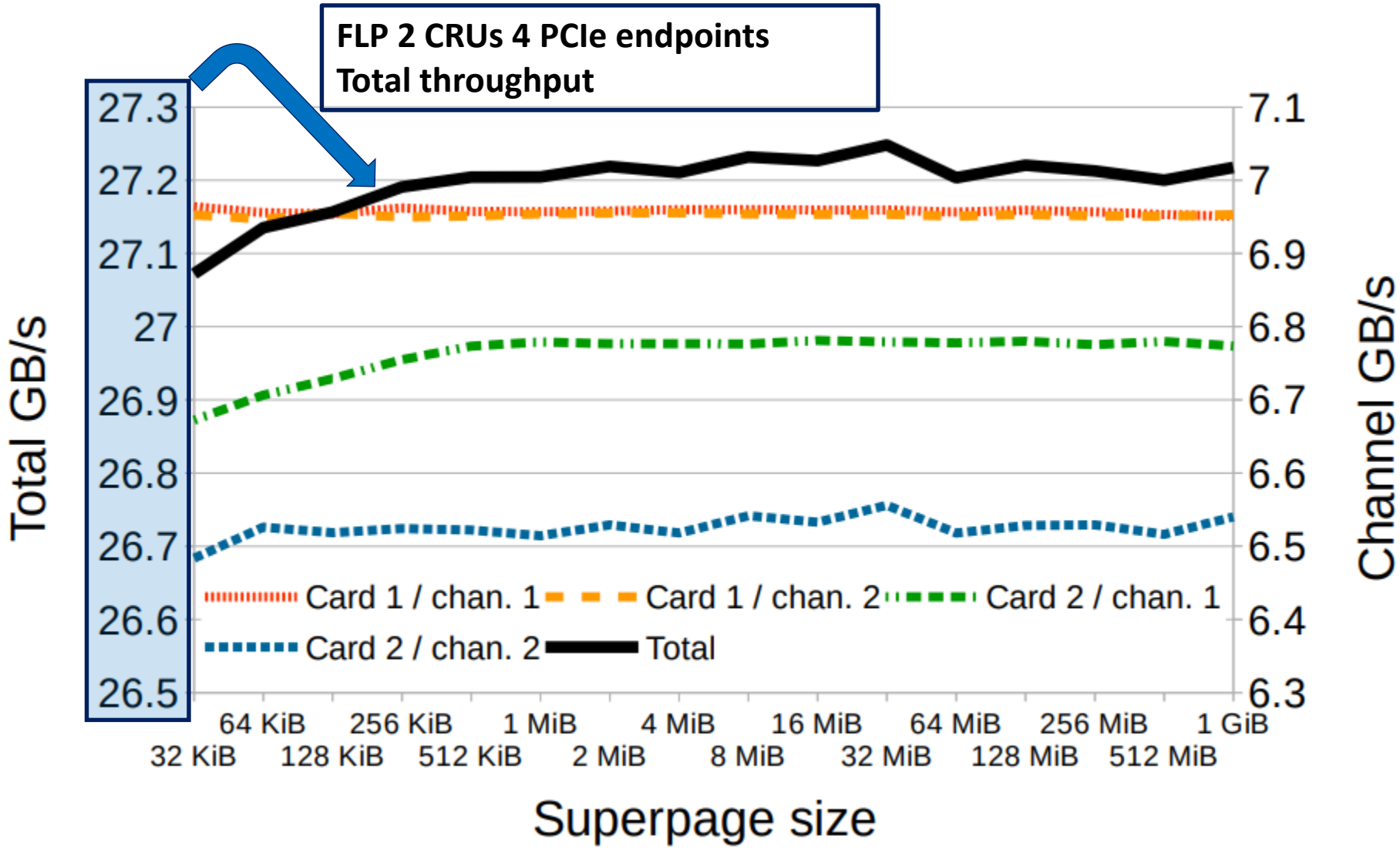
ITS design specification:

100 kHz Pb-Pb
400 kHz p-p

Data rate – PbPb @ 100kHz per Readout Unit



ROC bench tool performance



Backup Slides - Firmware

Firmware Development Environment

- **E-group**: All communication between firmware developers is organized through membership in a CERN e-group (alice-its-wp10-firmware@cern.ch). The e-group is also used for access control to the version control system (gitlab) and the issue tracking system (JIRA)
- **Online Documentation**: Documentation of hardware, Firmware, and software is posted at the WP10 Twiki page in the "ALICE Web" Twiki: https://twiki.cern.ch/twiki/bin/view/ALICE/ITS-WP10
- **Hardware Programming languages**: We decided to allow 3 different firmware languages: **VHDL**, **Verilog** and **System Verilog**. Currently, the prototype development is on the Xilinx 7-series, and the Vivado toolset supports all 3 languages.
- **Tool Sets**: The current design of the Readout Unit (RU) foresees the main programmable logic to be a Xilinx SRAM based Kintex UltraScale(+). Prototyping was started with a Xilinx Kintex-7. These devices are supported by the **Xilinx Vivado** Toolset. We are currently using Vivado version **2017.4**. The RU design also includes a flash-based device to support scrubbing. This device is foreseen to be the ProAsic-3 from Microsemi which is supported by the **Microsemi Libero SoC Design Suite**. To develop detector specific logic on the CRU requires the use of the **Altera Quartus** toolset. We are considering evaluating this toolset in the future. For FX3 (USB) firmware design, we are using the **Cypress EZ-USB Suite** & **GPIF-II Designer**.
- **Simulations**: Currently two toolsets are used for firmware test benches: **Mentor Modelsim** and **Cadence Incisive (NCSim)**. The Cadence tools is only used by CERN collaborators at the moment.
- **Coding Style**: For firmware coding in VHDL, a coding style document was developed: https://twiki.cern.ch/twiki/bin/view/ALICE/FwDevFlowMethods
- **Scripting**: Most firmware toolsets support scripting in **Tcl**. Tcl is the scripting languate integrated in the Vivado tool environment. The firmware projects developed in WP10 are required to include a Tcl script to create the project and compile it to ensure common project configuration amongst developers. An optional "Makefile" can be used to call targeted Tcl scripts for various tasks (project creation, compilation, simulation, etc…)

Software

- **Hardware Platform**: The main supported platform is currently a PC running the **CERN CentOS 7** (CC7) operating system with Vivado 2016.4 and Modelsim SE 10.4a installed. Similar features can be achieved by using the Windows mingw platform with the Msys shell, but full compatibility with the Linux development is not guarantied by WP10
- **Software Environment**: To ensure compatibility for software co-development it is recommended to install the **Linux Developer Toolset** in CC7 (https://linux.web.cern.ch/linux/devtoolset/), which provides more up-to-date compilers and libraries. Python is used for scripting to interact with the WP10 prototype hardware (currently via USB). We use the Continuum Analytics **Anaconda** platform (Python 3.6 version) to provide the python tools (https://www.continuum.io/anaconda-overview). This platform is also available for Windows.
- **Version Control**: CERN gitlab is used for version control of the firmware and software development within WP10. The gitlab group for WP10 is located at: https://gitlab.cern.ch/alice-its-wp10-firmware/. Membership in this gitlab group is via the e-group. Various projects are hosted in this gitlab group, the main prototype development is currently in the project “itswp10”.
- **Issue Tracking**: CERN JIRA is used for issue tracking. The WP10 issues are tracked underneath the ALICE project JIRA at: https://alice.its.cern.ch/jira/projects/IT.
- **File exchange**: Files that are too big or not version controlled are exchanged via a shared **CERNbox**. Currently, this is by invitation only, since the CERNbox account used is a private account.

Backup Slides – Control Interfaces

Opencores CAN Protocol Controller

- https://opencores.org/project,can
- Written in Verilog
- Supports basic CAN and extended CAN
  - 11-bit ID and 29-bit ID
  - ID mask and filtering
- Up to 1Mbps operation
- Transmit/receive/error interrupts
- Wishbone interface (8-bit address and data)
- Register map compatible with Philips SJA1000 CAN Controller IC
  - https://www.nxp.com/docs/en/data-sheet/SJA1000.pdf
- Size: 12k gates (930 flip-flops)

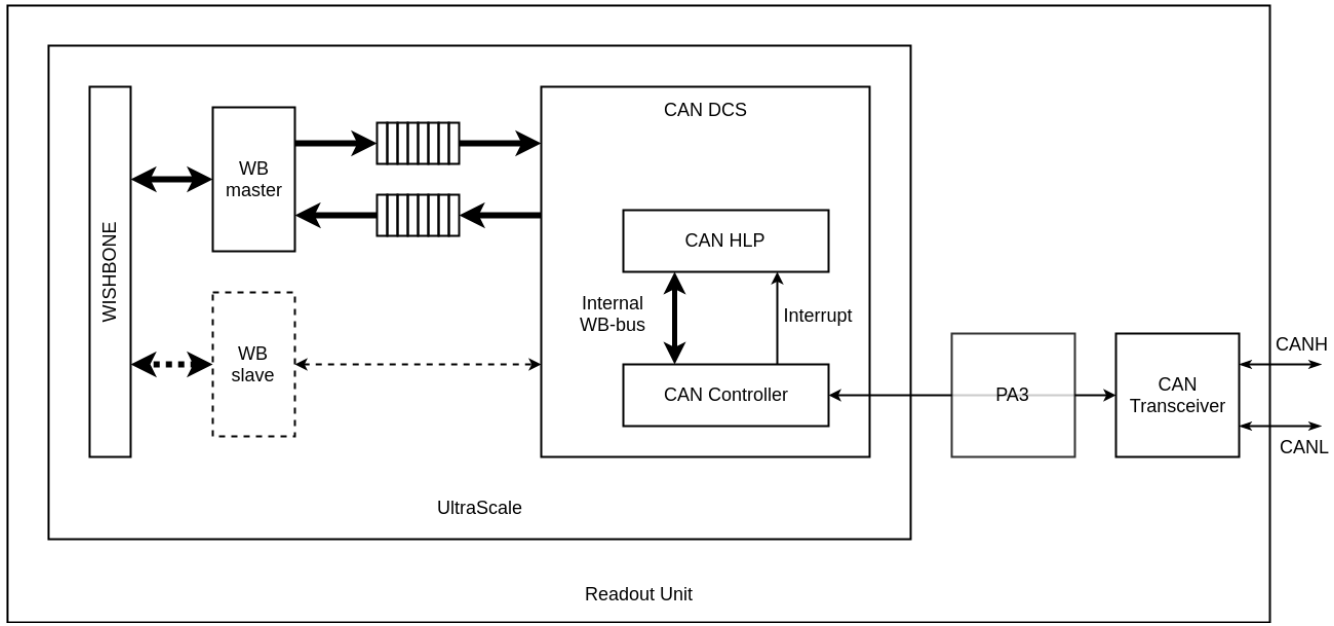
CANbus protocol proposal

Emulate a “network topology”: Divide 11 bit CAN ID field into a “Node ID” and a “Command” field:

MsgID[10:4] = N[6:0] = NodeID
MsgID[3:0] = C[3:0] = Command

ID[10]	ID[9]	ID[8]	ID[7]	ID[6]	ID[5]	ID[4]	ID[3]	ID[2]	ID[1]	ID[0]
N[6]	N[5]	N[4]	N[3]	N[2]	N[1]	N[0]	C[3]	C[2]	C[1]	C[0]

Allows for up to 128 nodes per CAN network



Command Codes

Command Code	Meaning
1	DATA
2	“WRITE” command
3	Write Response (optional)
4	“READ” command
5	Read Response
7	”STATUS” or “ALERT”

Write Request:

Write (2)	Write Command
Bytes 0 - 1:	“Address” of this “WRITE”.
Bytes 2 – 3:	“Data”

Read Request:

Read (4)	Length= 2
Bytes 0-1:	“Address” of this “READ”.

Write Response:

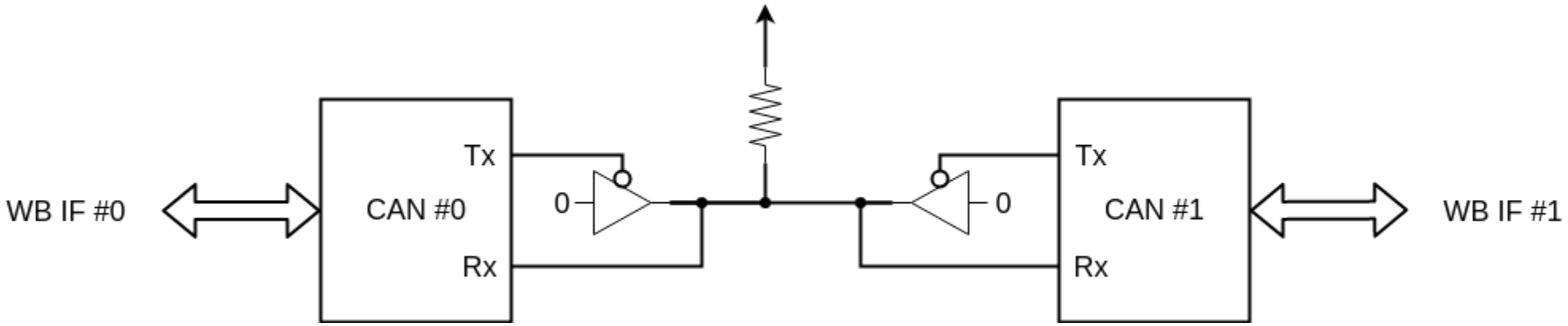
Write Response (3)	Length = 3
Bytes 0-1:	“Address” (copied from the received message).
Byte 2:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined)

Read Response:

Read Response (5)	Length 2 to 4 bytes
Byte 0-1:	“Address” of this “READ” (copied from the received message).
Bytes 2-3:	2 additional bytes representing data value resulting from “reading” the requested address. Lack of these bytes (i.e. a short message) indicates an invalid read address/request.

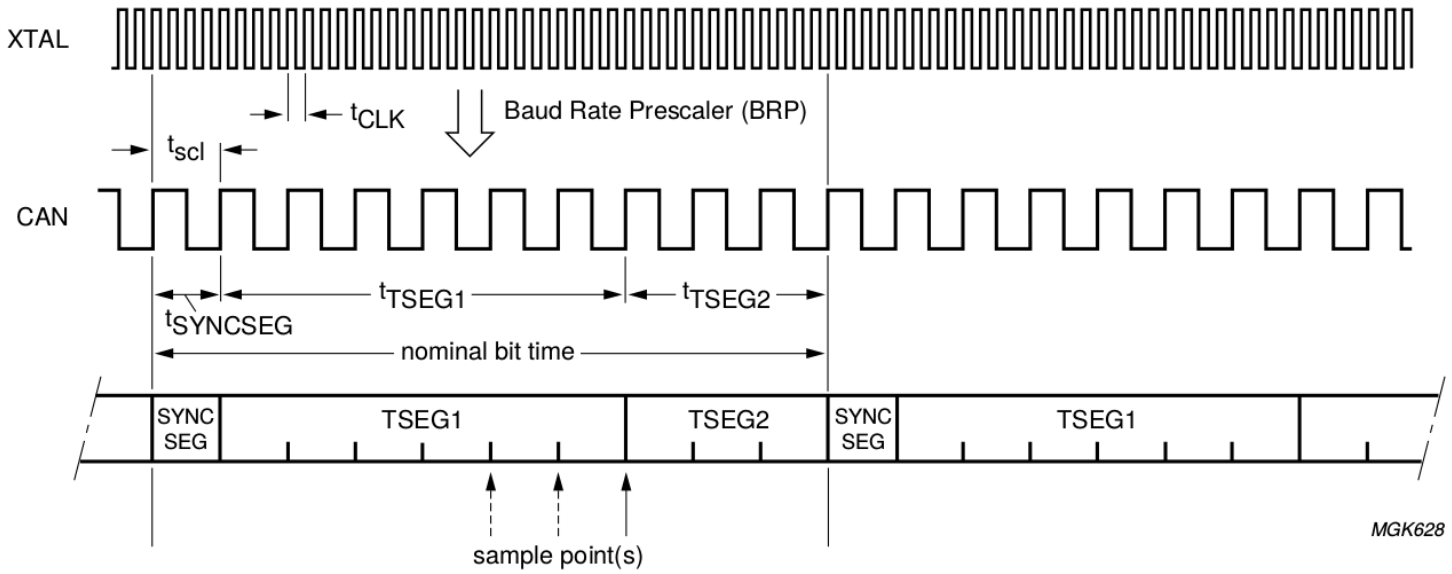
OpenCores CAN Protocol Controller Simulations

- Tested using a simple Bitvis UVVM style testbench
- Two instances of the CAN controller connected
- Two separate WB interfaces to write to each controller
- 40 MHz clock



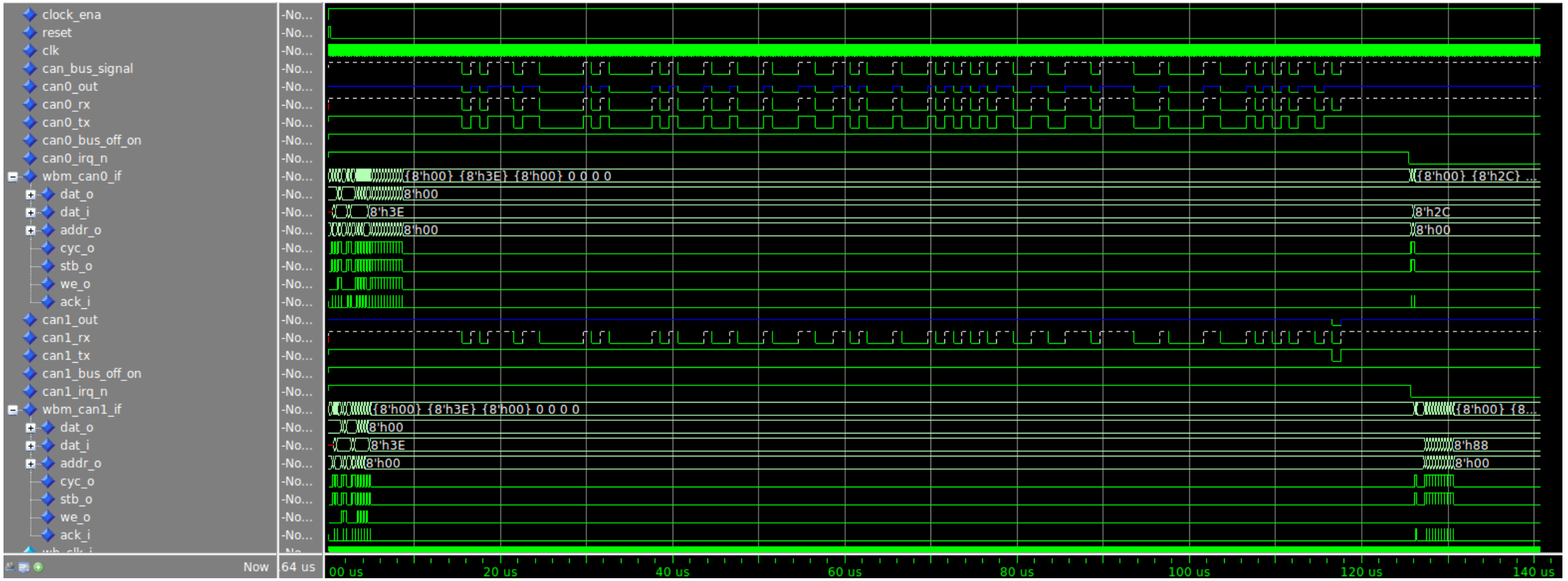
OpenCores CAN Protocol Controller Simulations

- Bit Timing Register (BTR0) configured for 4x baud clock prescale
- t_{SEG1} set to 7 baud clocks, t_{SEG2} set to 3 baud clocks, in BTR1
- $t_{SYNCSEG}$ is always 1 baud clock
- Gives us a bit rate of 40 MHz / (4 * (1+7+3)) = 1Mbps



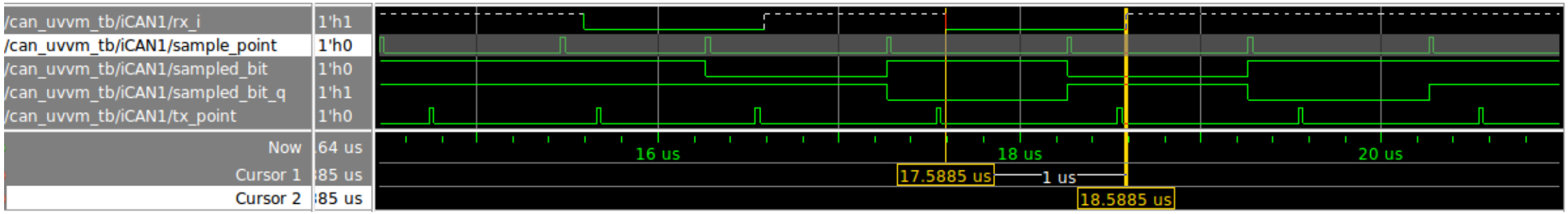
OpenCores CAN Protocol Controller Simulations

- Simulation waveforms, showing initial WB transactions on both controllers, CAN transmission, and CAN IRQ lines going low after message



OpenCores CAN Protocol Controller Simulations

- sample_point marks the point where bits on rx_i are being sampled
- The simulated bit period is 1 us, as it was configured for
- The sampling point is located at 7/10ths of a microsecond into each bit, corresponding to what t_{SEG1} and t_{SEG2} were configured for



OpenCores CAN Protocol Controller Simulations

Testbench log

Set up TX buffer on CAN0 with message for CAN1 (ID 0xBB)

Wait for CAN1 to receive message

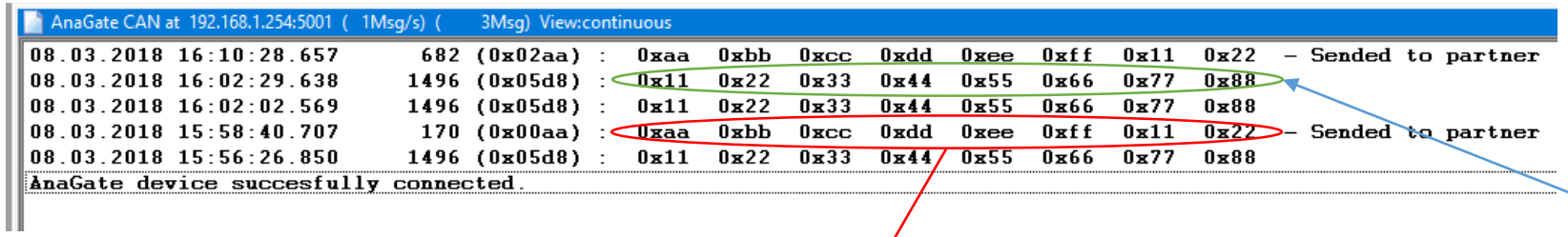
Verify message contents

```
# UVVM: ID_LOG_HDR                    4940.0 ns  TB seq.          Start a transaction from CAN0 to CAN1
# UVVM: ------------------------------------------------------------------------------------------------------------------------------------------------------
# UVVM: ID_BFM                        5115.0 ns  TB seq.          wb_write(A:x"0A", x"BB") completed. Set TXID1 to xBB
# UVVM: ID_BFM                        5465.0 ns  TB seq.          wb_write(A:x"0B", x"08") completed. Set TXID2 to x08, 8 bytes data
# UVVM: ID_BFM                        5815.0 ns  TB seq.          wb_write(A:x"0C", x"11") completed. Set data1 to x11
# UVVM: ID_BFM                        6165.0 ns  TB seq.          wb_write(A:x"0D", x"22") completed. Set data2 to x22
# UVVM: ID_BFM                        6515.0 ns  TB seq.          wb_write(A:x"0E", x"33") completed. Set data3 to x33
# UVVM: ID_BFM                        6865.0 ns  TB seq.          wb_write(A:x"0F", x"44") completed. Set data4 to x44
# UVVM: ID_BFM                        7215.0 ns  TB seq.          wb_write(A:x"10", x"55") completed. Set data5 to x55
# UVVM: ID_BFM                        7565.0 ns  TB seq.          wb_write(A:x"11", x"66") completed. Set data6 to x66
# UVVM: ID_BFM                        7915.0 ns  TB seq.          wb_write(A:x"12", x"77") completed. Set data7 to x77
# UVVM: ID_BFM                        8265.0 ns  TB seq.          wb_write(A:x"13", x"88") completed. Set data8 to x88
# UVVM: ID_BFM                        8615.0 ns  TB seq.          wb_write(A:x"01", x"01") completed. Request transmission on CAN0
# UVVM:
# UVVM:
# UVVM: ID_LOG_HDR                    8615.0 ns  TB seq.          Wait for CAN1 to receive message
# UVVM: ------------------------------------------------------------------------------------------------------------------------------------------------------
# UVVM:
# UVVM:
# UVVM: ID_LOG_HDR                  125613.5 ns  TB seq.          Got interrupt from CAN1.
# UVVM: ------------------------------------------------------------------------------------------------------------------------------------------------------
# UVVM: ID_BFM                      125790.0 ns  TB seq.          wb_check(A:x"00", x"XX")=> OK, received data = x"3E". Check that CAN0 transmit interrupt was set
# UVVM: ID_BFM                      126140.0 ns  TB seq.          wb_check(A:x"02", x"XX")=> OK, received data = x"2C". Check that CAN0 transmit complete status bit is set
# UVVM: ID_BFM                      126315.0 ns  TB seq.          wb_check(A:x"00", x"XX")=> OK, received data = x"3E". Check that CAN1 receive interrupt was set
# UVVM:
# UVVM:
# UVVM: ID_LOG_HDR                  127315.0 ns  TB seq.          Verify message received by CAN1
# UVVM: ------------------------------------------------------------------------------------------------------------------------------------------------------
# UVVM: ID_BFM                      127490.0 ns  TB seq.          wb_check(A:x"14", x"BB")=> OK, received data = x"BB". Verify received RXID1
# UVVM: ID_BFM                      127840.0 ns  TB seq.          wb_check(A:x"15", x"08")=> OK, received data = x"8". Verify received RXID2, 8 bytes data
# UVVM: ID_BFM                      128190.0 ns  TB seq.          wb_check(A:x"16", x"11")=> OK, received data = x"11". Verify received data byte 1
# UVVM: ID_BFM                      128540.0 ns  TB seq.          wb_check(A:x"17", x"22")=> OK, received data = x"22". Verify received data byte 2
# UVVM: ID_BFM                      128890.0 ns  TB seq.          wb_check(A:x"18", x"33")=> OK, received data = x"33". Verify received data byte 3
# UVVM: ID_BFM                      129240.0 ns  TB seq.          wb_check(A:x"19", x"44")=> OK, received data = x"44". Verify received data byte 4
# UVVM: ID_BFM                      129590.0 ns  TB seq.          wb_check(A:x"1A", x"55")=> OK, received data = x"55". Verify received data byte 5
# UVVM: ID_BFM                      129940.0 ns  TB seq.          wb_check(A:x"1B", x"66")=> OK, received data = x"66". Verify received data byte 6
# UVVM: ID_BFM                      130290.0 ns  TB seq.          wb_check(A:x"1C", x"77")=> OK, received data = x"77". Verify received data byte 7
# UVVM: ID_BFM                      130640.0 ns  TB seq.          wb_check(A:x"1D", x"88")=> OK, received data = x"88". Verify received data byte 8
```

CAN bus testing on RUv1

- AnaGate CAN adapter used for testing (same as DCS group is using).
- Successfully sent/received CAN messages to/from CAN controller in PA3

AnaGate CAN Monitor program



54	CAN_RXB_ID1	8212	8	FF	✓
55	CAN_RXB_ID2	8213	8	E8	✓
56	CAN_RXB_DATA1	8214	8	AA	✓
57	CAN_RXB_DATA2	8215	8	BB	✓
58	CAN_RXB_DATA3	8216	8	CC	✓
59	CAN_RXB_DATA4	8217	8	DD	✓
60	CAN_RXB_DATA5	8218	8	EE	✓
61	CAN_RXB_DATA6	8219	8	FF	✓
62	CAN_RXB_DATA7	8220	8	11	✓
63	CAN_RXB_DATA8	8221	8	22	✓

Readout Unit PA3 GUI software (RX buffer registers)

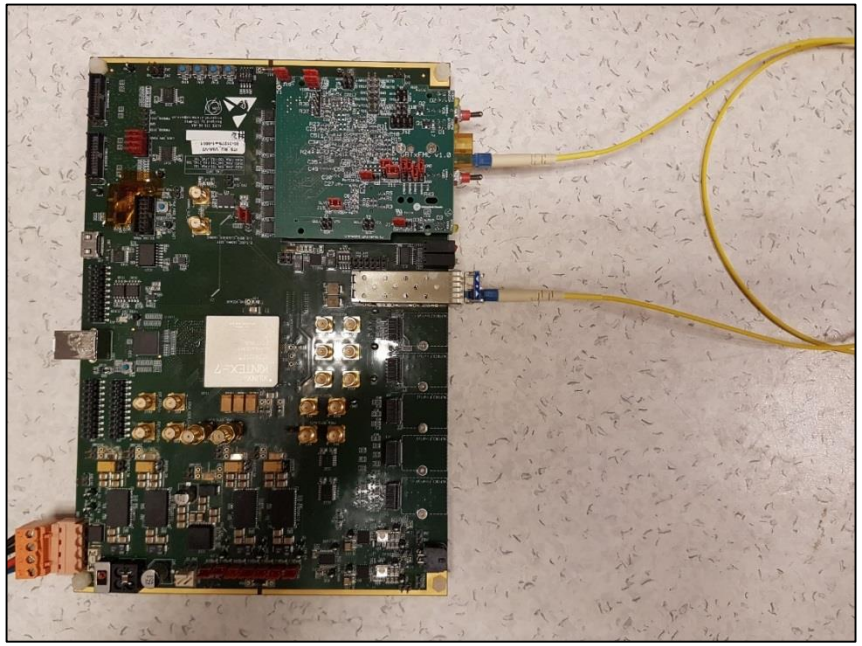
44	CAN_TXB_ID1	8202	8	BB	✓
45	CAN_TXB_ID2	8203	8	8	✓
46	CAN_TXB_DATA1	8204	8	11	✓
47	CAN_TXB_DATA2	8205	8	22	✓
48	CAN_TXB_DATA3	8206	8	33	✓
49	CAN_TXB_DATA4	8207	8	44	✓
50	CAN_TXB_DATA5	8208	8	55	✓
51	CAN_TXB_DATA6	8209	8	66	✓
52	CAN_TXB_DATA7	8210	8	77	✓
53	CAN_TXB_DATA8	8211	8	88	✓

Readout Unit PA3 GUI software (TX buffer registers)

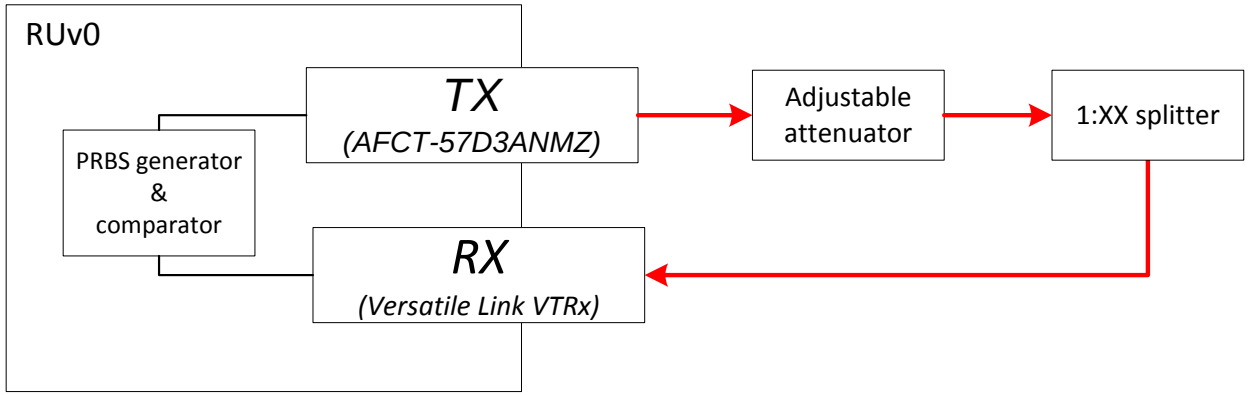
Trigger Interface – Splitter Loopback Test

- **Loopback test including:**
  - Actual SFP transmitter & receiver
  - 1:32 splitter

  Result: **24 hrs w/o errors**[1]

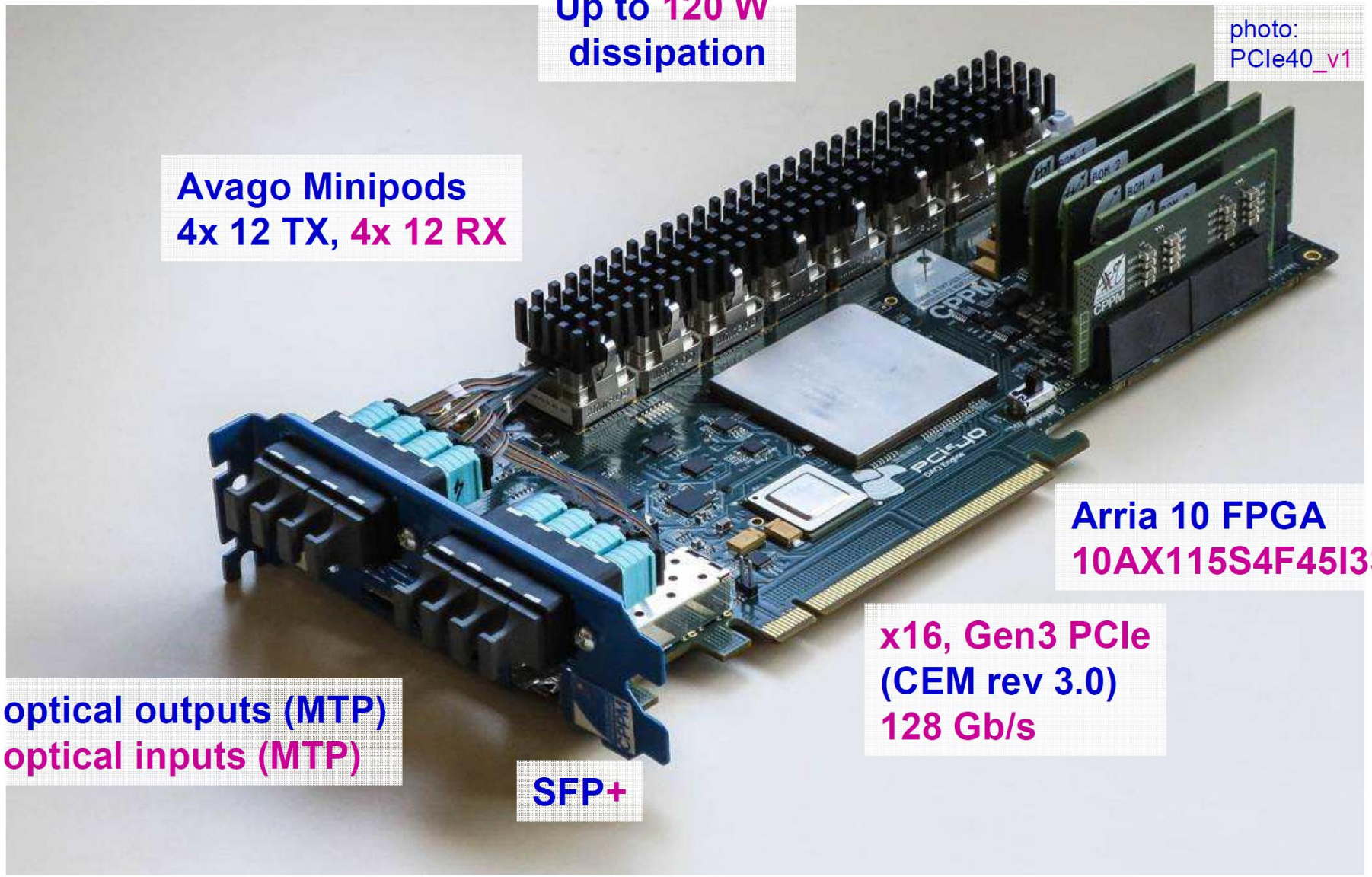


[1] Test conducted by Jo Schambach



Backup Slides – Data Streaming

PCIe40_v1


Up to 120 W
dissipation
photo:
PCIe40_v1
Avago Minipods
4x 12 TX, 4x 12 RX
Arria 10 FPGA
10AX115S4F45I3SG
x16, Gen3 PCIe
(CEM rev 3.0)
128 Gb/s
48 optical outputs (MTP)
48 optical inputs (MTP)
SFP+

CRU v2: PCIe40_v2 from LHCb

Data rate – PbPb @ 50kHz per Readout Unit