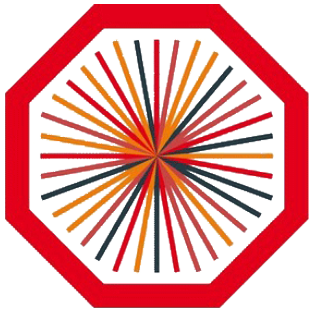
ALICE



Readout Review

System overview

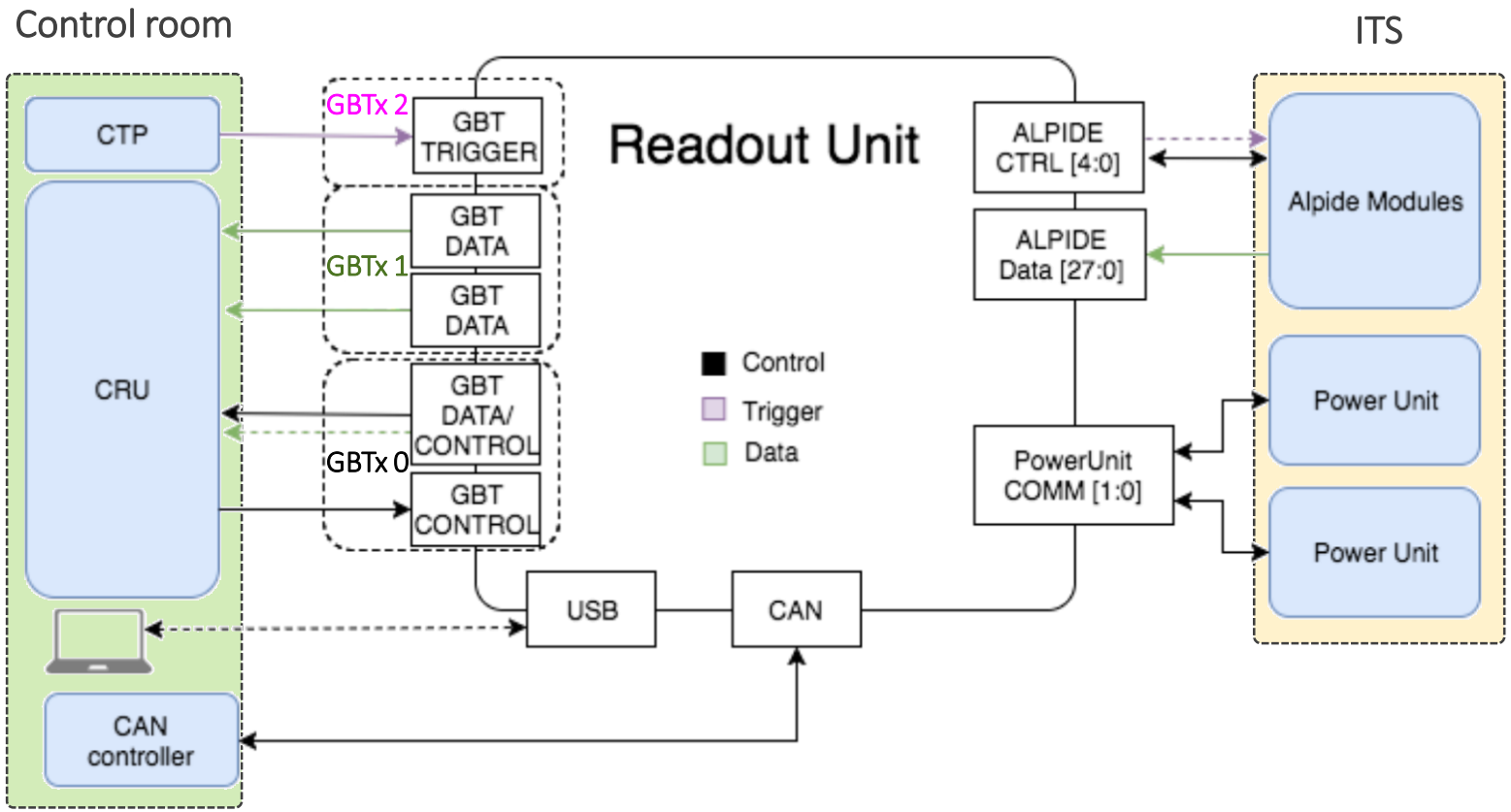
Overview – RU main interfaces

Downlink:

- GBTx0 is used for slow-control from the CRU.
- GBTx1 has NO downlink connection.
- GBTx2 is used for receiving triggers from the CTP.

Uplink:

- GBTx0 is shared between data and slow-control;
- GBTx1 and GBTx2 are reserved for data only.
- GBTx0 also communicate with the GBT-SCA (Slow-
- Control Adapter) installed on the RU.



Toward system control and DAQ

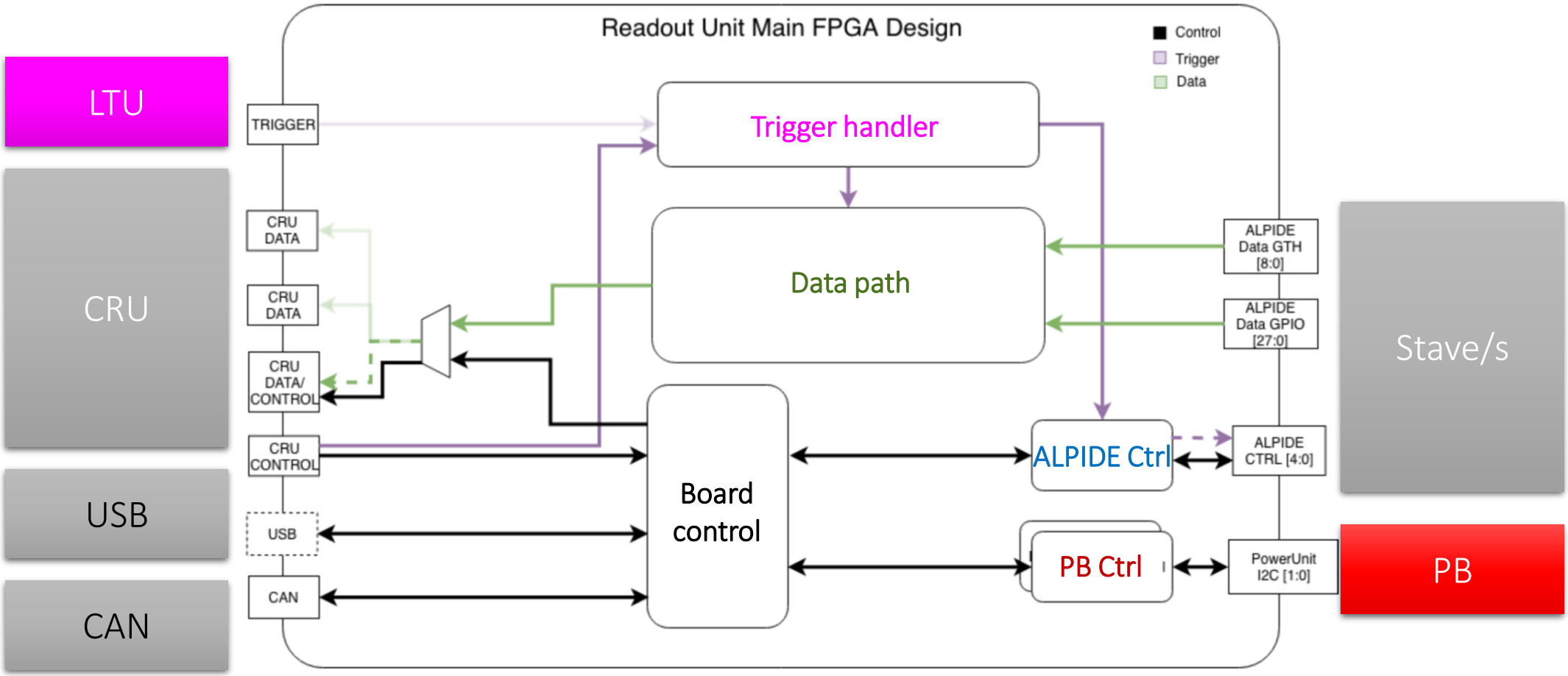
- Three GBTx chips connecting the main FPGA to the Common Readout Unit (CRU) and the Central Trigger Processor (CTP).
- One Universal Serial Bus (USB) 3.0 used for debugging.
- One CAN bus interface for DCS backup.

Toward the detector

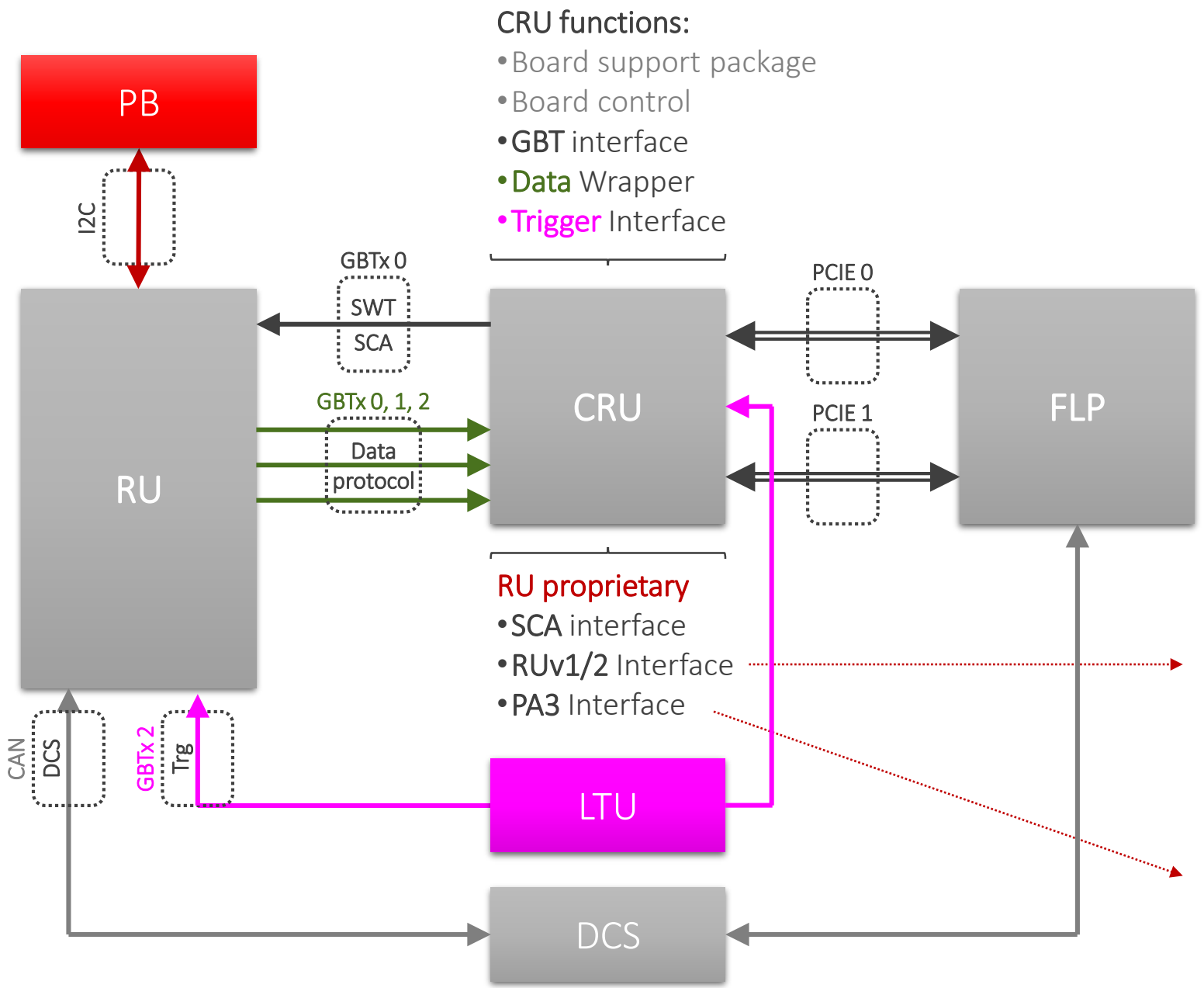
- Five clock and 5 control and trigger link toward the ALPIDE.
- 9 or 28 differential pairs for data readout
- Two Power Unit (PU) I2C interfaces controlling a Power Unit each.

Overview – RU main firmware reference structure and nomenclature

Here is represented the firmware of the main XCKU060 FPGA, which manages all the RU functions. For the firmware on the PA3 radiation support FPGA, see the radiation section.



Overview – RU control through the CRU/FLP infrastructure (and other interfaces)



CRU functions:
- Board support package
- Board control
- GBT interface
- Data Wrapper
- Trigger Interface

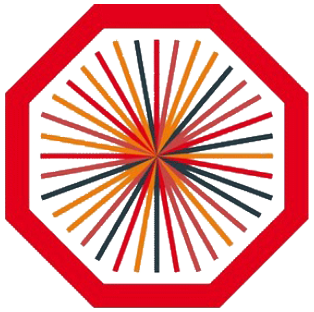
RU proprietary
- SCA interface
- RUv1/2 Interface
- PA3 Interface

- Board control/monitor
- GBT configuration
- I2C (GBT & PU ctrl)
- Alpide Control
- Data Path Configuration
- Trigger Handler Control

- RU programming
- RU scrubbing

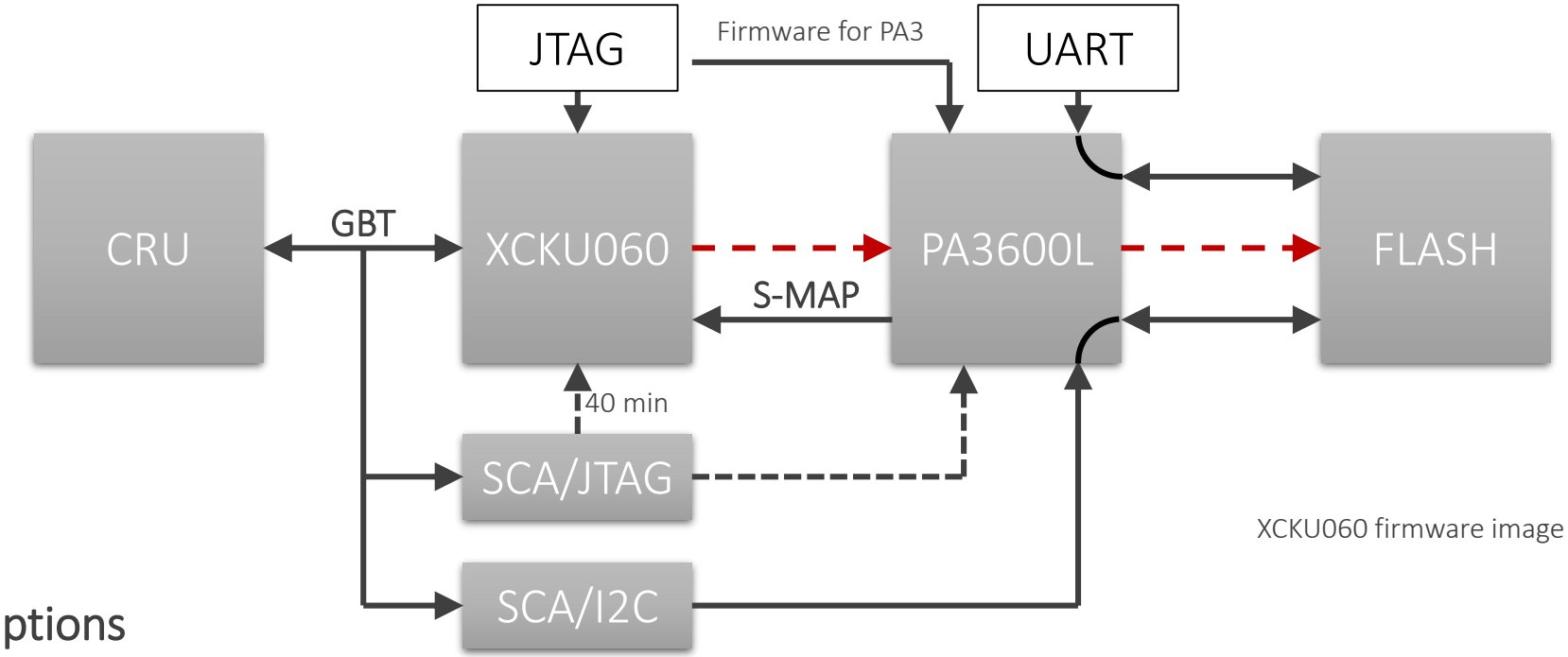
GBT/SCA

Depending on the communication class used, the RU classes can be accessed from **USB**, over the *GBT* from **RUv0** CRU, or over the GBT from the **O2 CRU**.
The access from the **CAN** bus interface to this class is under development.



System functional status

Status – RU programming



Present options

We program the PA3 FPGA and the FLASH, the PA3 then takes care of programming the XCKU060.

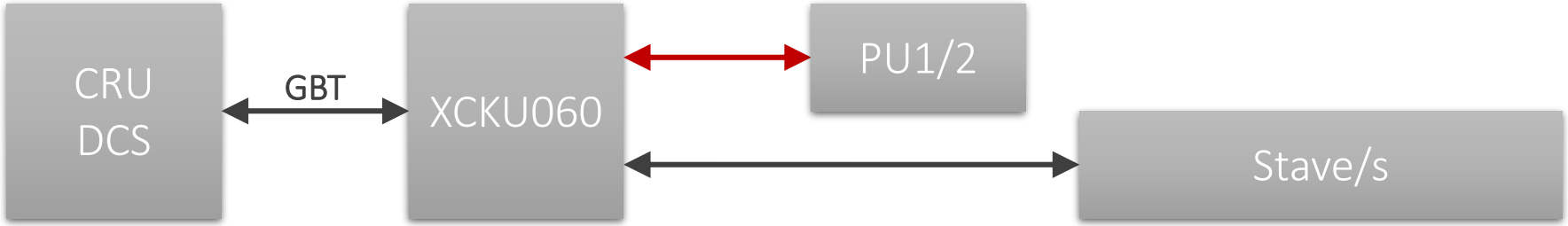
- Programming the XCKU060 firmware through JTAG dongle.
- Programming the PA3 firmware through JTAG dongle.
- Programming the FLASH through CRU → GBT/SCA/I2C → PA3 → FLASH (25 min).

To Do (proof of concept done by Gitle)

Programming the RU and PA3 firmware image directly from the CRU

- CRU → GBT/JTAG → PA3.
- CRU → GBTx → XCKU060 → PA3 → FLASH (NON rad-hard, <= 3 min).

Status – RU control and monitoring (normal operation)



Remark

The available features are controllable through the GBT/USB using the Python script classes.

Main basic functions

- Automatic interface (GBT/USB/CAN)
- GBT/USB/CAN wishbone master monitor
- Firmware and chip IDs, dates monitor
- Radiation error counters monitor
- FPGA status control (Vs, Is, Ts)
- Board status control (Vs, Is, Ts)

Power unit control

- PU #1 full control (main and aux I2C bus)
- PU #2 full control (main and aux I2C bus)

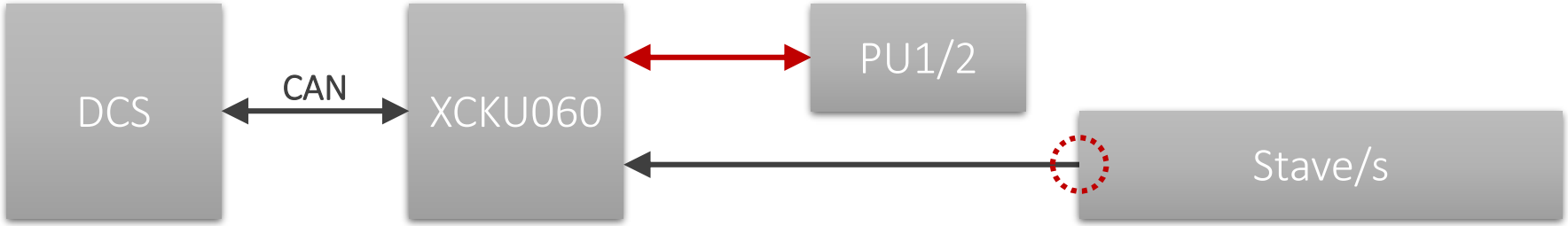
Misc

- I2C interface to the 3 GBTx chips for GBT config.
- Monitoring of GBT controllers (eLink to 80 bit word)
- GBTx flow monitor (SWT/trigger transactions)

Ongoing

- Adding further debug/flow control of various firmware IPs and conditions.

Status – RU control and monitoring (backup operation)



Remark

All the available features present in the firmware are fully controllable through the CAN interface (it is a wishbone master), only it is SLOW!

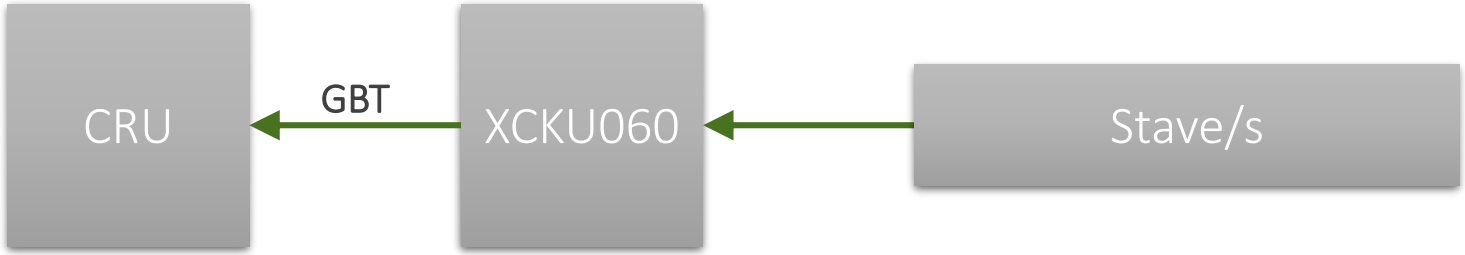
Tested

- Integrated into the main RU firmware, tested with > 118m cable (experiment length).
- Configurable bitrate, python software available.
- Tested with up to 2 RUs on the same bus.

Ongoing / To Do

- Add TMR (quite complicate, studying how to best do it).
- Improve implementation with present RUv1/2 regression system.

Status – RU data readout



Remarks

For details about the actual data protocol, see details in the Sensor/Stave data readout section

Staves/sensor readout data path control and configuration

- Data path frontend configuration for the IB (GTH transceivers)
- Data path frontend configuration for the OB (LVDS GPIO)
- Status and protocol tracker for both IB and OB

GBT data packager configuration (IB/OB)

- Configuration of GBT packer for the IB and OB (GTH / LVDS GPIO)
- Status of GBT packer for the IB and OB

Ongoing / To Do

- Management of out-of-frame data from the chip (untagged BID).
- Test the link using 3 GBT channel in parallel

Status – RU power board control

Remark

The only communication channel for the Power board is through the Readout Unit, either via CRU/GBT (default) or DCS CAN bus (backup).

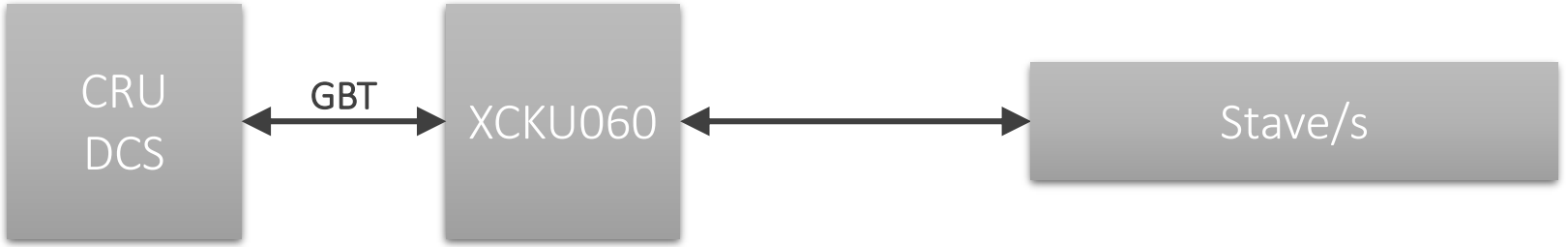
Done

- Complete Power Board control / monitoring through I2C.

Ongoing

- Standard minor improvements (mostly software)

Status – RU sensor/stave control (Alpide Ctrl and sequencer/sniffer)



Done

- Alpide control done and ready (currently in use, reads/write registers, trigger, clock management).
- Agreed on general architecture to allow flexible use of the controller during data taking.
- To allow seamless parameter monitoring during data taking, agree to exploit the abort gap (about 3 µs at 22.5 kHz repetition rate, two each LHC orbit).
- Verified the abort gap equivalent bandwidth on the control line is sufficient for what we need.

Ongoing/ To Do

- System based on programmable *sequencer* and *sniffer* (will be useful for other purposes).
- Defining memory interface for storing result within present RU firmware.
- Developing the actual IP blocks.
- Studying the interaction with the ALF/FRED system.

Status – RU trigger management

Remarks

The RU can receive the trigger either from the CRU (debug/commissioning) and the LTU (final system).

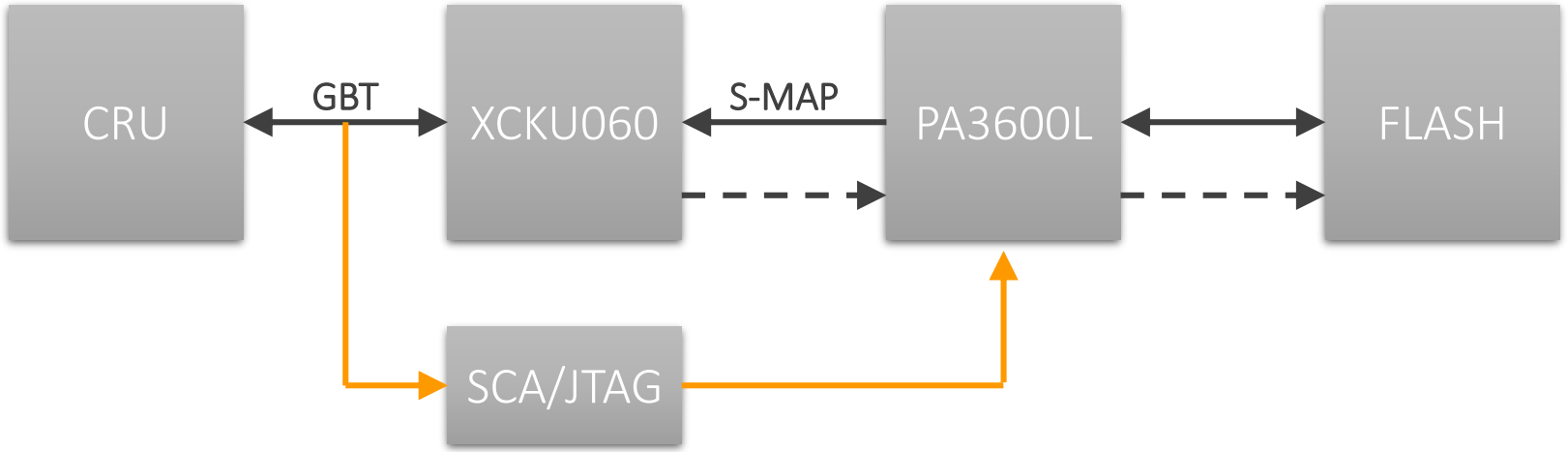
Trigger handler

- Management of both Continuous and Triggered mode
- Selectable frame period in continuous mode (min 2.4 kHz, max 4 MHz).
  - The strobe duration is programmed through ALPIDE ctrl (from 2 µs to 80 µs)
- Synchronization/labeling with data flow from the sensor
- Trigger info are recorded into the data stream for reconstruction (see protocol details)
- In case of triggered mode, halts triggers too close in time (but keep track of them, configurable).
- Provide stand-alone trigger generation (configurable pulse or trigger).

Ongoing

- Finalizing the connection with the actual LTU (still some data format problem, investigating with the CTP team)
- We have further request from commissioning

Status – RU radiation management – PA3 system



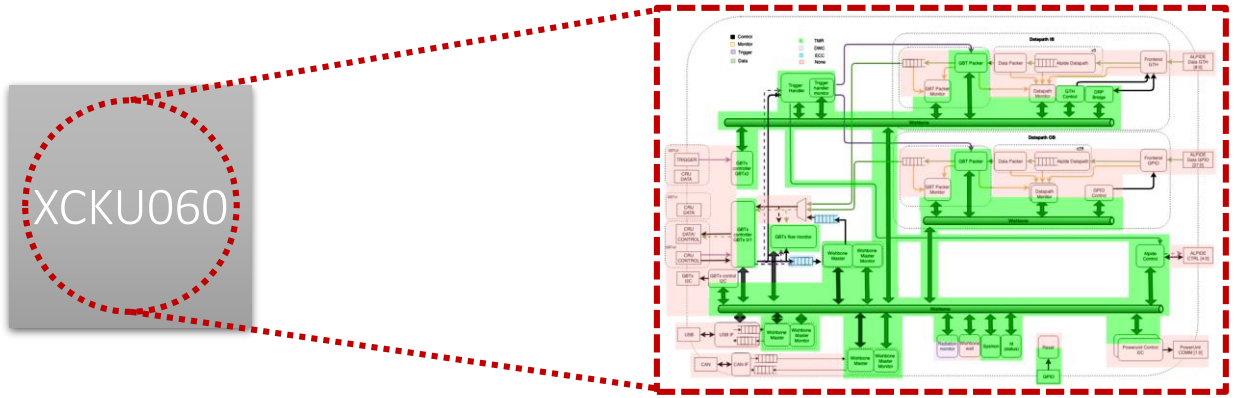
Scrubbing features

- Stable firmware release (v0206)
- Continuous scrubbing of the main XCKU060 FPGA (1.7s cycle).
- Completely controllable through the GBT/SCA channel (completely rad-hard path).
- Redundant firmware golden copy in FLASH, plus ECC per-page correction.
- TMR blocks in PA3 Firmware protects the firmware from transients.

Ongoing

- Improve the programming flexibility (see programming slide).
- Integrate the control/monitor into the main software suite (so far usable only through the GBT/SCA).
- Enable error injection while in commissioning mode to test the whole system.

Status – RU radiation management – main firmware



Remarks

To render the scrubbing effective, the main firmware MUST be hardened as much as possible.

Status

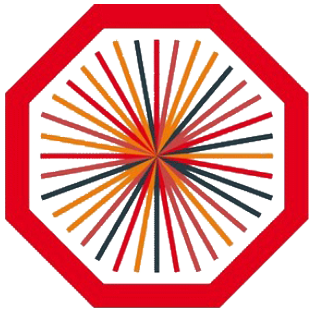
- All critical IPs have been TMR, including the entire wishbone bus.
- Critical memories and FIFOs have ECC enabled (tested to be extremely robust).
- Extensive radiation monitoring facilities in the firmware.
- Data path have NOT been triplicated where not deemed critical to save resources.
- All functional cross-section and functional interrupts measurements completed and on spec.
- Expected downtime well within specs. Errors recovery time measured.

Ongoing

- Enabling fault injection during commissioning for long-term, high statistics testing (see previous slide)

To do

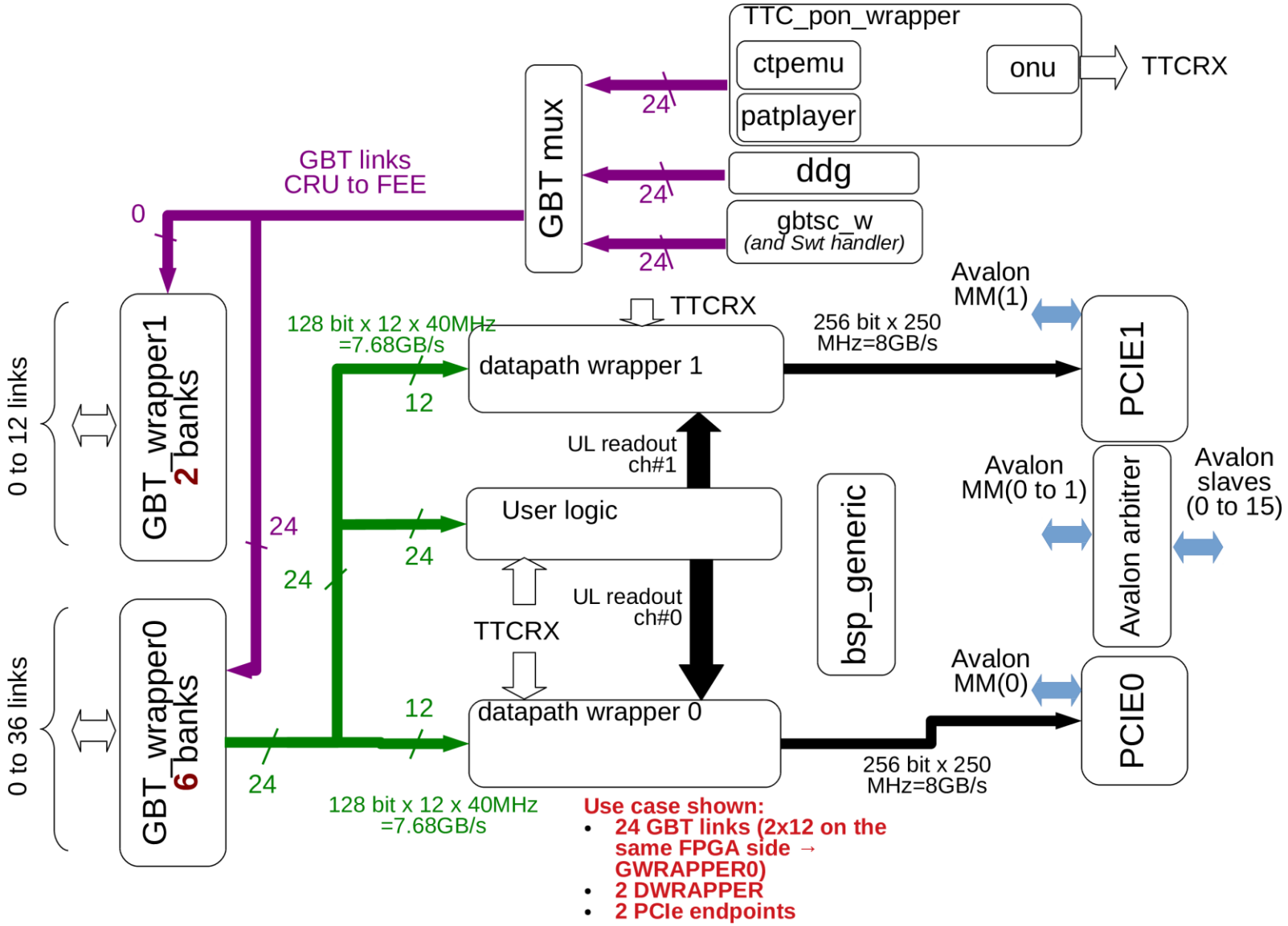
- Entire system test in radiation field for further verification, possibly this year.



More details: CRU operations

Overview – CRU main firmware reference structure and nomenclature

Here is represented the firmware of the CRU



Overview – CRU control classes available through Python scripts

- libReadoutCard ported to Python 3.6
- All CRU and WP10 python classes ported: WP10 gitlab repository – "CRU_ITS"

Classes available from **FLP** software (CRU control):

- CRU: top class containing other classes, also instantiates ROC interface
- BSP: PLL, clocks, house-keeping
- **GBT**: configuration of the GBT links
- **Data Wrapper**: data packing
- **TTC**: CTP interface, CTP emulator, pattern player

Set and control **CRU-RU communication** and trigger delivery through control GBT (temporary solution)

All **RUv1/2** classes available as well through the **CRU SWT** and **SCA** interface

- Readout Unit control and monitoring (SWT protocol)
- I2C (GBT & **Power Unit**)
- **GBT** configuration
- **Alpide** Control
- **Data path** configuration
- **Trigger Handler** Configuration

RUv1/2 specific operations (once comm. established!)

CRU operation functions – summary (relevant to ITS)

CRU itself

- Current CRU firmware version bitfile (version **2.6**, 2.7 to be merged): cru-v2-v2_7_0_24_links.sof
- Firmware control interface through 2 Avalon Masters from the 2 PCIe endpoints

GBT/SCA interface (used for RU control and monitoring)

- SCA communication available (1 RU per time).
- SWT communication is to one GBT channel only, configurable via a register (1 RU per time).
- SWT, DDG or TTC are selected as the GBT TX source by a configurable MUX (under improvement).

Data flow

- 2 Datapath Wrappers for up to 24 GBT links plus 1 User Logic input, provides data to one PCIe endpoint
- User Logic takes GBT inputs, processes them and provides data to the Datapath Wrappers

Trigger

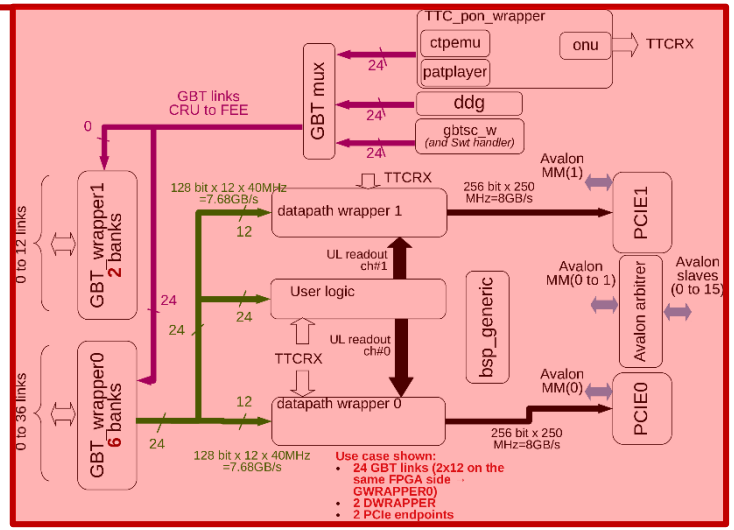
- TTC interface receives and distributes CTP messages in broadcast mode (all configured GBT links)
- TTC interface also includes a (configurable) CTP emulator and pattern player
- CTP emulator provides 4 modes:
  - Continuous
  - Triggered with periodic triggers
  - Triggered with manual triggers
  - Idle
- All CTP emulator modes always send (configurable) HB, Timeframe delimiter, Orbit triggers

CRU functions – CRU management

getBuildDate() - get firmware build date
getBuildTime() - get firmware build time
getShortHash() - get firmware short (32bit) git hash
getChipID() - get Altera chip ID
getDirtyStatus() - get git dirty status at build time
getFwInfo() - get all of the above info

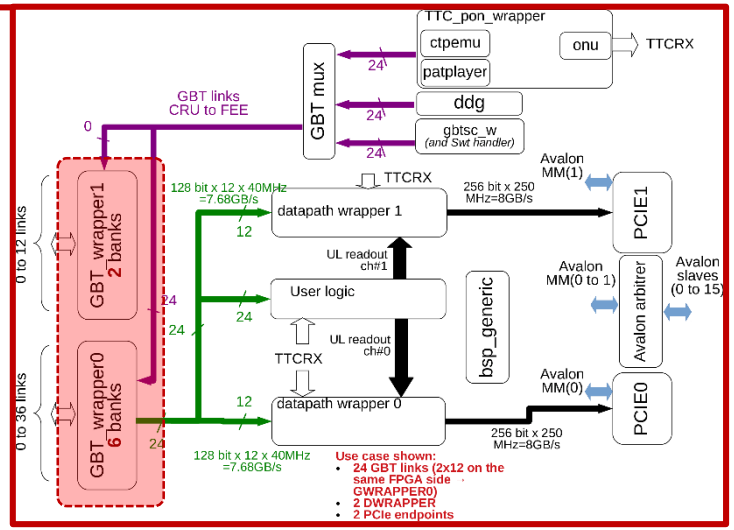
disableRun() - disable data taking
enableRun() - enable data taking
forceStartOfRun() - force start of data taking without waiting for LTU SOx
forceStopOfRun() - force stop of data taking without waiting for LTU EOx

isClk240alive() - check if clk240 is alive or stopped
ledBlinkLoop(mask) - blink front panel LED according to 4-bit mask
resetClockTree() - activate reset for TTC pon, GBT and dwrapper
setCRUid(id) - set (12bit) CRU ID to be added in the RDH



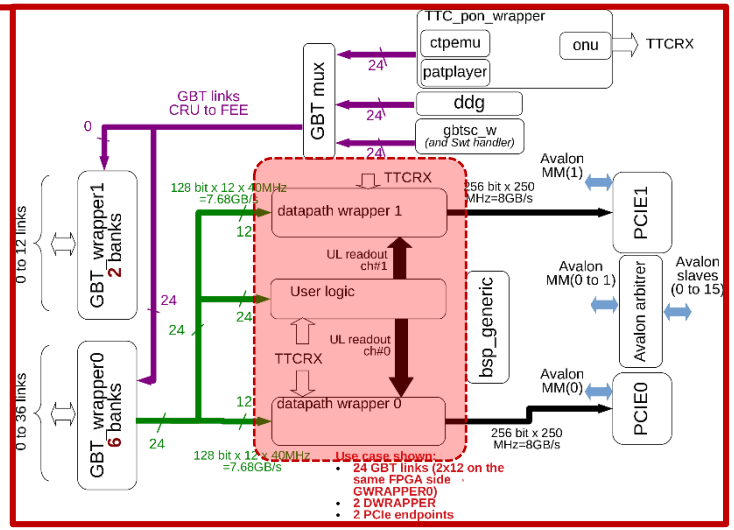
CRU functions – GBT management

resetAllLinks()	- reset all GBT links
getTotalNumOfLinks()	- return total number of links configured
getRefFrequencies()	- get GBT wrappers reference clock frequencies
getRxClkCnt(wrap, bank, link)	- get GBT RX clock frequency
getTxClkCnt(wrap, bank, link)	- get GBT TX clock frequency
checkLinkLockStatus()	- check GBT link lock status
rxmode(mode)	- “gbt” or “wb” (widebus)
txmode(mode)	- “gbt” or “wb” (widebus)
getGbtMode()	- get configured mode for each link
loopback(value)	- sets the same internal loopback mode for all links
getLoopback()	- check internal loopback mode
internalDataGenerator(val)	- select GBT TX source: upstream(0) or internal pattern generator(1)
patternmode(mode)	- “static” or “counter”
txcountertype(mode)	- select test pattern counter type “30bit” or “8bit”
rxpatternmask(hi, mid, lo)	- check mask on RX side when in test pattern mode with 8bit counter
getDataErrorCnt(wrap, bank, link)	- get Data-Not-Locked error counter
getHeaderErrorCnt(wrap, bank, link)	- get Packet-Header-Not-Locked error counter
getFecCnt(wrap, bank, link)	- get number of corrected error words (16bit only)
getRxErrorCnt()	- return RX error counter register
cntrst()	- reset error counter in specified links
cntstat(infiniteloop)	- print test pattern errors for each link
fecstat(infiniteloop)	- print FEC corrected words for each link
stat(infiniteloop, stat)	- print number of test pattern and FEC errors for each link



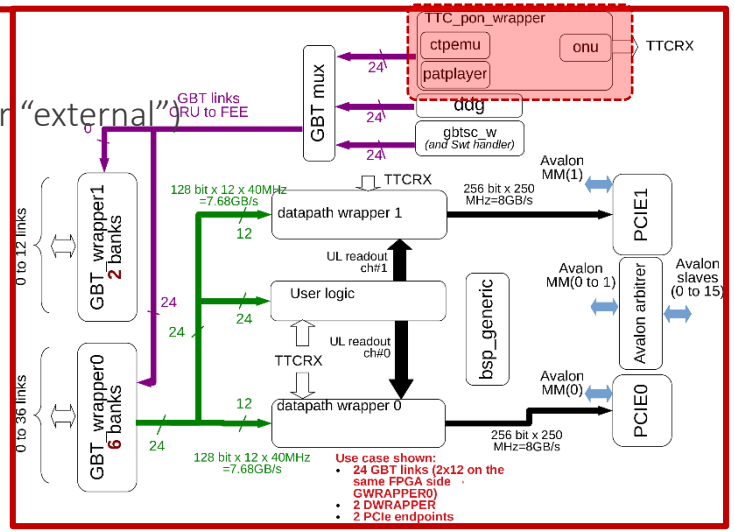
CRU functions – Data path

linkEnableMask(wrapper, mask)	- enable input link (0-23), user logic (24), statistics (31)
linkEnableBit(wrapper, link)	- enable input link (0-23), user logic (24), statistics(31)
getEnabledLinks(numAllLinks)	- return which data links are enabled
packetmuxPadLen(arbmode)	- Packet arbitration (=0, only implemented mode, "incremental")
setFlowControl(wrap, allowReject, forceReject)	- configure flow control
setGbtDatapathLink(wrap, link, isGBTpkt, RAWBYID, RAWMAXLEN)	- configures GBT datapath links
getGbtDatapathLink(wrap, link)	- get GBT datapath link configuration
setMinglerLinkID(wrap, setID, Idval, Idmask)	- configure mingler for ID mods
getBigFifoLvl()	- read big FIFO level
getLastHB()	- read last HB received
getStatistics()	- read statistics counters



CRU functions – Trigger – 1

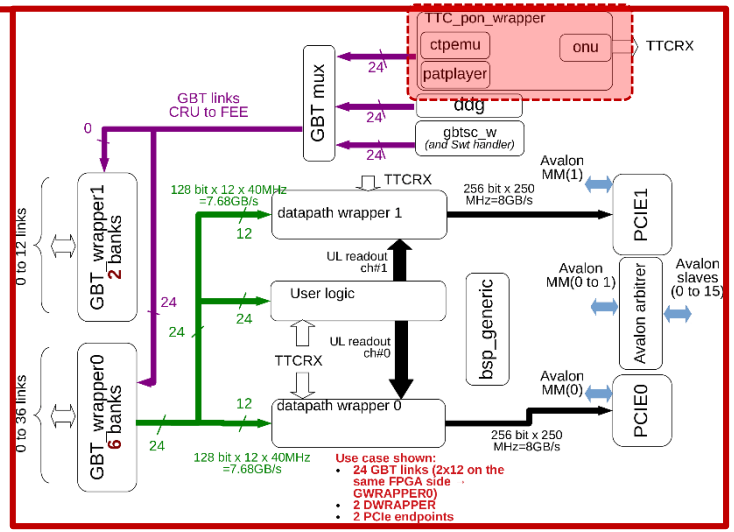
configPlls(clockingScheme)	- configure on-board PLLs to clockingScheme: “local”, “external”
selectGlobal240(clockingScheme)	- select between local osc and recovered PON RX clock (“local” or “external”)
configPonTx(onuAddr)	- configure fPLL 120MHz for PON TX
onuCalibrationStatus()	- check onu calibration status
selectDownstreamData(downData)	- select data downstream: “ctp”, “pattern”, “midtrg”
getDownstreamData()	- get source of TTC downstream data
enableClock(enable)	- enable/disable 240Mhz clock from external jitter cleaner
getClockFreq()	- get TTC clock frequency
getHBTrigFromLTUCount()	- get count of HB received from LTU
getPHYSTrigFromLTUCount()	- get count of PHYS received from LTU
getSOXEOXTrigFromLTUCount()	- get count of SOx/EOx received from LTU
resetEmulator(doReset)	- reset/disable CTP emulator
setEmulatorTrigMode(mode)	- “periodic”, “manual”, “continuous”
setEmulatorContMode()	- put emulator in “continuous” mode
setEmulatorIdleMode()	- put emulator in idle mode (generates EOx if running)
doManualPhysTrig()	- request one PHYS trig
setEmulatorBCMAX(BXMAX)	- wrap bunch crossing counter at BCMAX (12bit)
setEmulatorHBMAX(HBMAX)	- wrap HB at HBMAX (16bit)
setEmulatorHBKEEP(HBKEEP)	- set HB accept frames (16bit)
setEmulatorPHYSDIV(PHYSDIV)	- generate PHYS trigger every PHYSDIV ticks (28bit), must be >7 to activate
setEmulatorCALDIV(CALDIV)	- generate CAL trigger every CALDIV ticks (28bit), must be >18 to activate
setEmulatorHCDIV(HCDIV)	- generate Health Check trigger every HCDIV ticks (28bit), must be >10 to activate
setFBCT(FBCT_array)	- set trigger at fixed bunch crossings

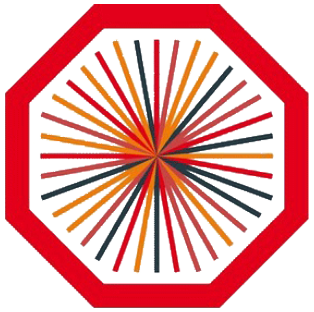


CRU functions – Trigger – 2

setIdlePattern(idlePattern) - pattern player idle pattern
setSyncPattern(syncPattern) - pattern player sync pattern
setResetPattern(resetPattern) - pattern player reset pattern
selectPatternTrig(syncTrig, resetTrig) - select which bit of TTC_DATA to trigger
enableSyncAtStart(enable) - enable sending of SYNC patern when run enable goes high

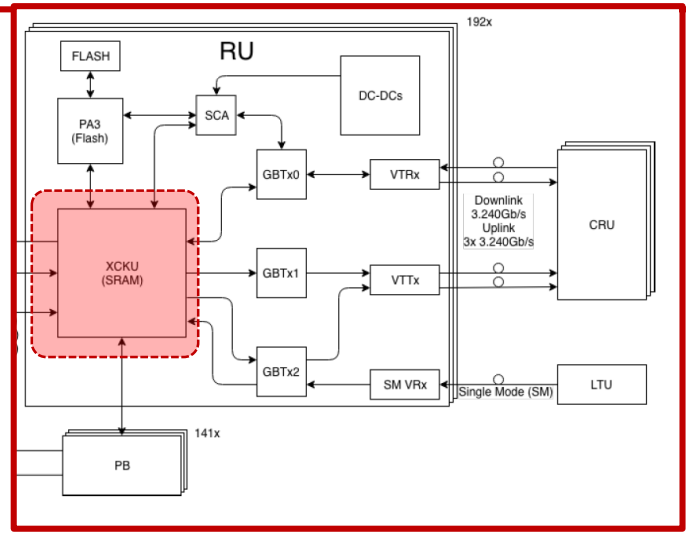
configSync(syncLength, syncDelay) - configure length and delay of SYNC pattern
configReset(resetLength) - configure length of RESET pattern
triggerReset() - manual RESET trigger
triggerSync() - manual SYNC trigger





More details: RU operations functions

RU functions – main RU functions summary – 1



Board control and monitor

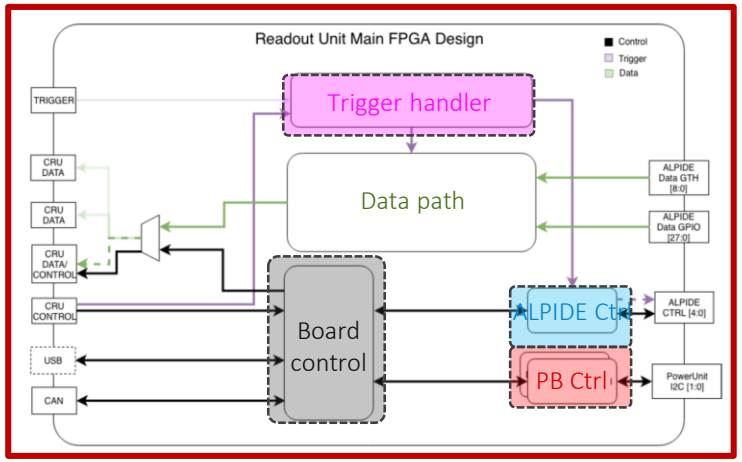
comm	- communication interface (USB, GBT_CRU, SWT)
master_monitor	- GBT wishbone master monitor
master_usb	- USB wishbone master monitor
status	- firmware and chip IDs, dates
radmon	- radiation protection error counters
sysmon	- FPGA status (voltage, temperature, currents)
wb2fifo	- data interface to ProAsic3 for uploading firmware (to be imp.)

Power unit control

powerunit_1, powerunit_2 - interfaces to main & aux I2C bus for each of the two power units

Sensor monitoring/Control

dctrl - ALPIDE DCTRL interface



Trigger

trigger_handler	- configuration of trigger handler
trigger_handler_monitor	- monitor of trigger handler

Misc

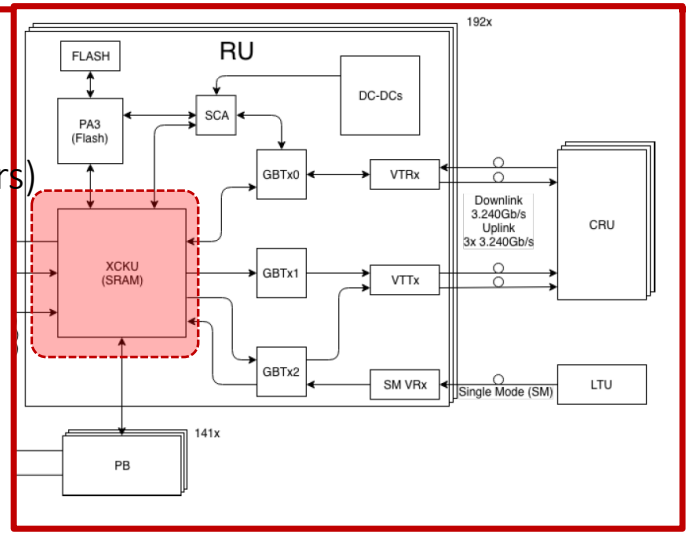
i2c_gbtx	- I2C interface to the 3 GBTx chips
gbtx01, gbtx2	- configuration of GBT controllers for GBTx0, GBTx1 & GBTx2
gbtx_flow_monitor	- monitoring of GBT words (80 + 1 bit each)

RU functions – main RU functions summary – 2

Staves/sensor data path control

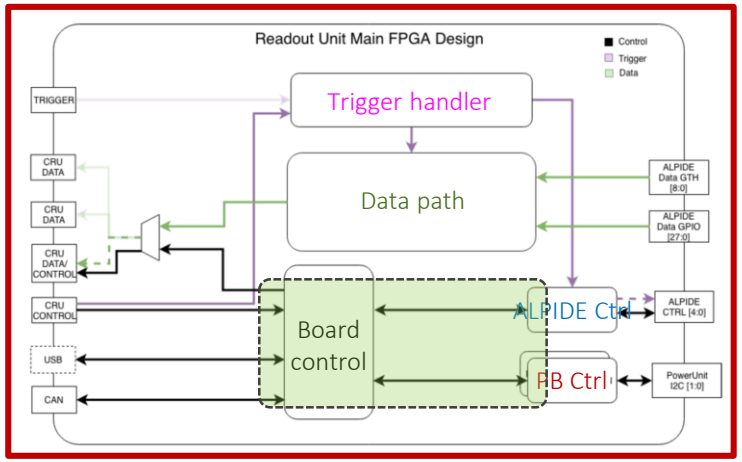
drp_bridge - configuration of GTH transceivers
gth - datapath frontend configuration for the inner barrel (GTH transceivers)
gpio - datapath frontend configuration for the outer barrel (LVDS GPIO)
datapathmon - status of datapath for inner barrel
datapathmon_gpio - status of datapath for outer barrel

gbt_packer_gth - configuration of GBT packer for inner barrel (GTH)
gbt_packer_gpio - configuration of GBT packer for outer barrel (LVDS GPIO)
gbt_packer_monitor_gth - status of GBT packer for inner barrel
gbt_packer_monitor_gpio - status of GBT packer for outer barrel



NOTES:

- Each component provides access to the various class functions of the class associated with this component.
- Depending on the communication class used, this RUv1 class can be accessed from USB, over the GBT from RUv0_CRU, or over the GBT from the O2_CRU.
- All of these components are derived from the Wishbone Slave interface class, which implements basic wishbone transactions.
- The individual classes and their member functions are documented on the Web here.
- The access from the CAN bus interface to this class is under development (at the moment, the access is encapsulated in a separate CAN_HLP class)



RU functions – SCA readout

initialize() - initialize SCA as used on RU (with GPIO, ADC, I2C in

enable_channel(….) - enable various interface channels on SCA

read_adc_channel(channel) - read ADC channel

read_gpio() - read GPIO channel

write_gpio() - write GPIO channel

set_gpio_direction(value) - set the direction for each GPIO pin (bitmask)

set_gpio(index, value) - overwrite value to pin “index”

set_xcku(value) - Ultrascale FPGA reset via SCA

_write_spi(ch, slave, nbits, data…) - write transaction to SPI channel “ch”

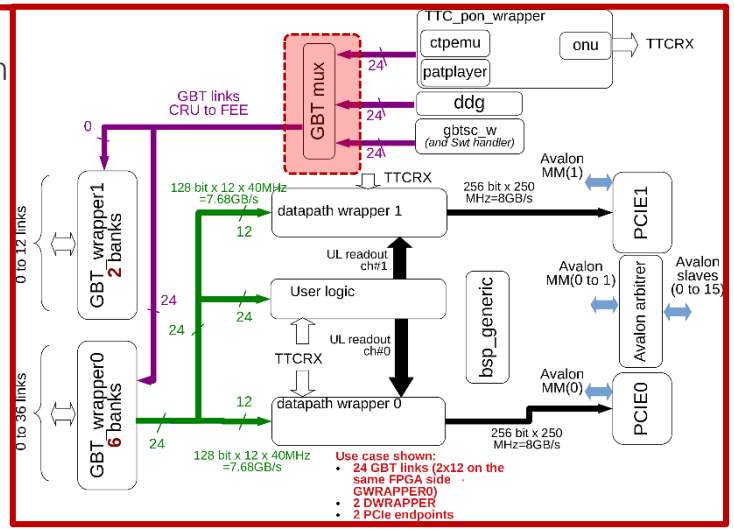
_read_spi(ch, slave, nbits) - read transaction from SPI channel “ch”

_write_i2c(ch, sl_addr, nbytes, data…) - write transaction to I2C channel “ch”

_read_i2c(ch, sl_addr, nbytes) - read transaction from I2C channel “ch”

read_gbtx_register(register, gbtx) - read register in GBTx “gbtx” via I2C

write_gbtx_register(register, value, gbtx) - write “value” to register in GBTx “gbtx”



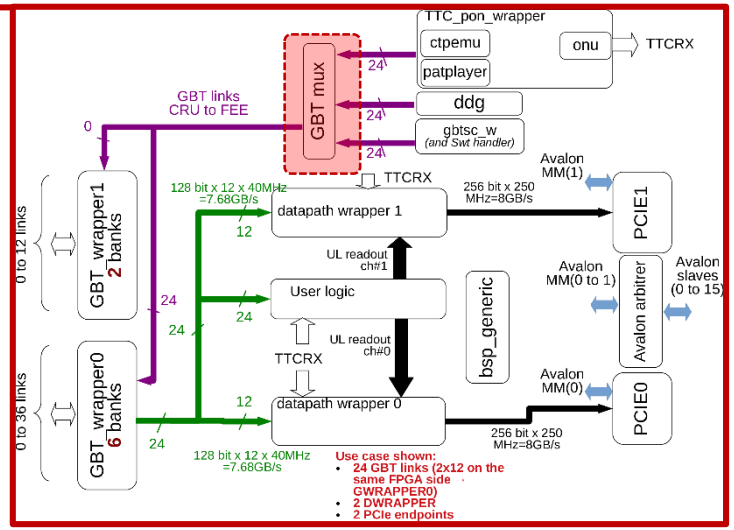
RU functions – SCA communication - EXAMPLE

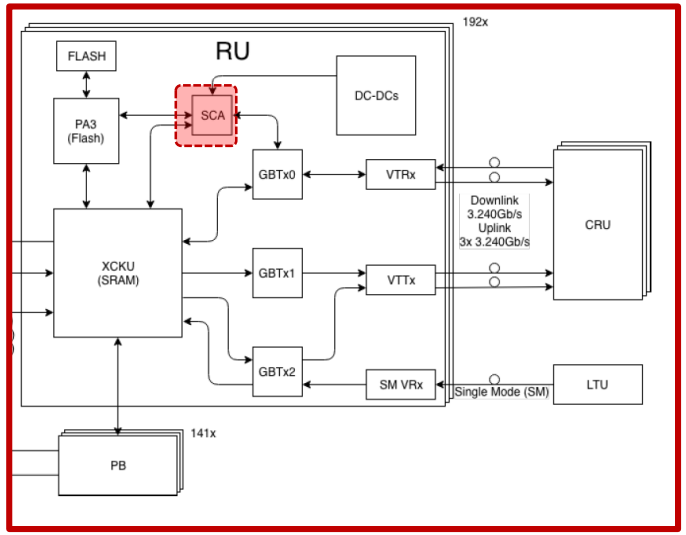
Example interaction: Read RUs current and voltages through theSCA ADCs

[root@gemini py]# ./testbench.py cru initialize --gbt_ch=0

[root@gemini py]# ./testbench.py cru read-adcs-conv

I_MGT: 1.58 - Transceivers current [A]
I_INT: 0.71 - FPGA core current [A]
I_1V2: 0.53
I_1V5: 1.53
I_1V8: 0.76
I_2V5: 0.72
I_3V3: 0.51
I_IN : 1.27 - RU input current from supply [A]
V_MGT: 0.96 - Transceivers voltage [V]
V_INT: 1.01 - FPGA core voltage [V]
V_1V2: 1.21
V_1V5: 1.52
V_1V8: 1.79
V_2V5: 2.49
V_3V3: 3.31
V_IN : 10.21 - RU input voltage from supply [V]
T1 : 37.11 - On Board temperature measurement #1 [C]
T2 : 34.57 - On Board temperature measurement #2 [C]

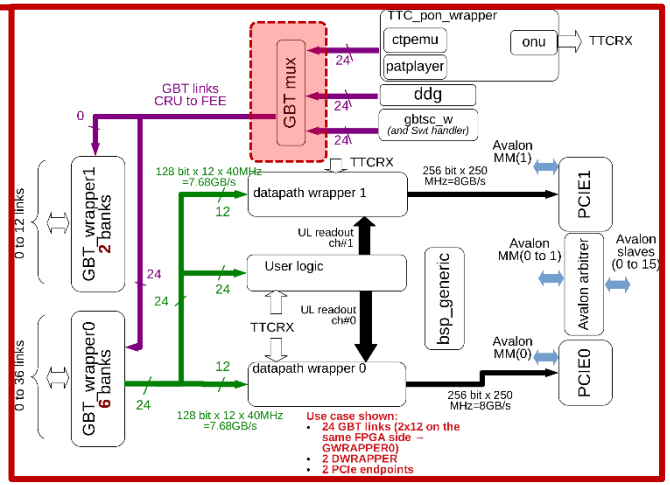




RU functions – SWT communication – EXAMPLE

Preparatory steps

- SWT communication needs to be configured in MUX
- Single GBT channel only, configurable via an Avalon register
- Interface via several Avalon registers, accessed via PCIe BAR
- No Broadcast to RUs capability at the moment



Example: read firmware versions and powerunit enable status, voltages, and currents

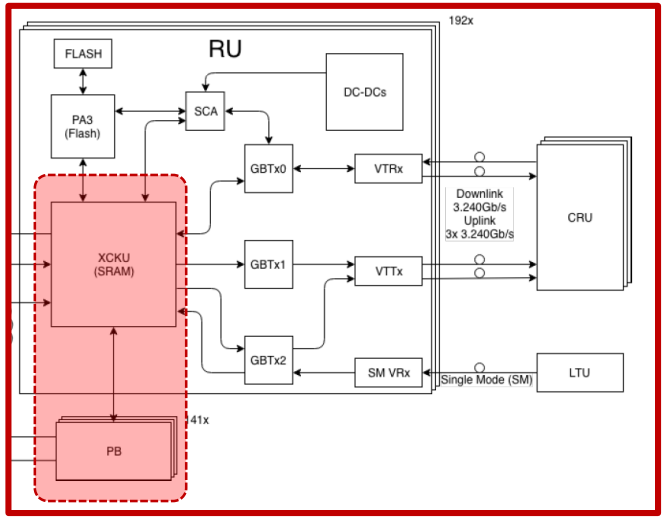
[root@gemini py]# ./testbench.py cru initialize --gbt_ch=0 # initialize CRU SWT for GBT link 0

[root@gemini py]# ./testbench.py cru setGbtTxMux 2 # set TX MUX to "swt"

[root@gemini py]# ./testbench.py rdo read 1 0 # read status module (wb_id=1) register 0
22679

[root@gemini py]# ./testbench.py version
2018-09-24 03:27:15,965 - root - INFO - CRU Version: 6CAAD9B0
2018-09-24 03:27:15,965 - root - INFO - RDO Version: 6FCC5897

[root@gemini py]# ./testbench.py rdo powerunit_1 log-values-IB 0 # PU1, channel 0
2018-09-24 04:08:20,843 - Module Id 6: PowerUnit - INFO - Power enable status: 0x0000
2018-09-24 04:08:20,844 - Module Id 6: PowerUnit - INFO - Bias enable status: 0x00
2018-09-24 04:08:20,844 - Module Id 6: PowerUnit - INFO - Backbias: -4.819 V, 0.0 mA
2018-09-24 04:08:20,845 - Module Id 6: PowerUnit - INFO - Module 0, avdd: 0.011 V, 0.0 mA
2018-09-24 04:08:20,845 - Module Id 6: PowerUnit - INFO - Module 0, dvdd: 0.011 V, 0.0 mA



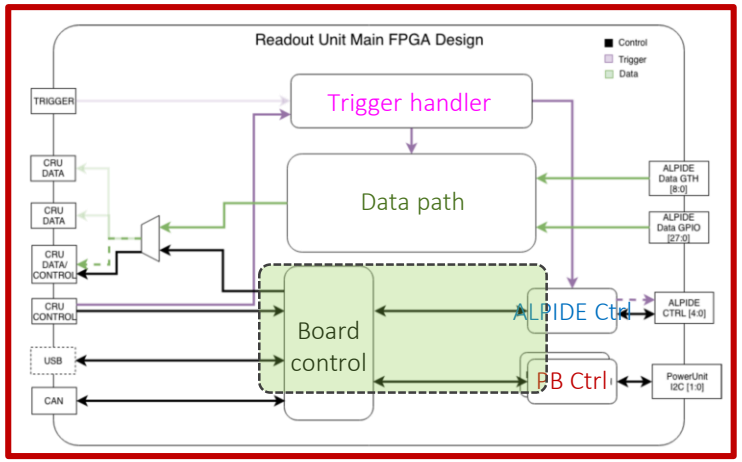
RU functions – setup for data taking – EXAMPLE

First setup the proper CRU configuration:

```
./testbench.py cru setGbtTxMux 0          # choose TTC as TX downstream data
./testbench.py setup-its-readout-test
```

In a separate shell, with the O2 environment setup:

```
roc-setup-hugetlbfs
mkdir /tmp/ramdisk
mount -t tmpfs -o size=2048M tmpfs /tmp/ramdisk
roc-reg-write --id=3b:0.0 --channel=0 --address=0xc00 --value=0x2
readout.exe file://root/ALICE/CRU_ITS/software/config/readout.cfg
```



Then again in the WP10 shell:

```
./testbench.py cru bsp enableRun
./testbench.py cru ttc setEmulatorTrigMode "manual“   # sends SOT
./testbench.py cru ttc doManualPhysTrig               # send a "PHYSICS" trigger
./testbench.py cru ttc doManualPhysTrig               # send a "PHYSICS" trigger
```

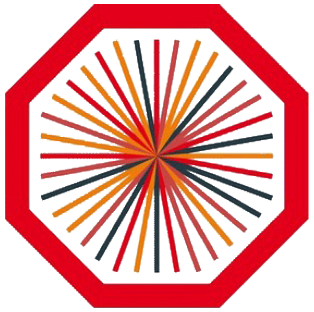
To be improved soon!

RU functions – testbench function "setup-its-readout-test"– EXAMPLE

```
def setup_its_readout_test(self):
    self.cru.gbt.internalDataGenerator(0) # upstream data (no pattern generator)
    self.cru.gbt.txmode("gbt")
    self.cru.gbt.rxmode("gbt")
    self.cru.gbt.loopback(0)                # no loopback
    self.cru.gbt.patternmode("counter") # not really needed, since we are not using test patterns
    self.cru.ttc.selectDownstreamData("ctp") # select TTC data from CTP emulator

    self.cru.ttc.resetEmulator(1) # disable and reset CTP emulator
    self.cru.ttc.setEmulatorIdleMode() # send no trigger
    self.cru.ttc.setEmulatorBCMAX(999)  #  heartbeat every 25us (999+1 * 25ns)
    self.cru.ttc.setEmulatorHBMAX(8) # timeframe every 8 HB frames
    self.cru.ttc.setEmulatorHBKEEP(3) # send 3 Hba, followed by 5 HBr
    self.cru.ttc.setEmulatorPHYSDIV(7) # send trigger every 8 ticks
    self.cru.ttc.resetEmulator(0) # enable CTP emulator

    self.cru.bsp.disableRun() # disable data taking while configuring the wrappers

    self.cru.dwrapper.linkEnableMask(wrapper=0, mask=0) # mask out all GBT links
    #self.cru.dwrapper.linkEnableMask(wrapper=1, mask=0)
    for i in LINK_LIST: # for each link defined in the LINK_LIST, configure to dwrapper 0
       # currently all links in the same wrapper (0), consider dividing between wrappers
       self.cru.dwrapper.linkEnableBit(wrapper=0, link=i)
       self.cru.dwrapper.setGbtDatapathLink(wrap=0, link=i, is_GBT_pkt=1, RAWBYID=0, RAWMAXLEN=0x1EA)

    self.cru.dwrapper.packetmuxPadLen(arbmode=0)
    self.cru.dwrapper.setFlowControl(wrap=0, allowReject=0, forceReject=0)
```

RU functions – setting a module for pulser readout – EXAMPLE

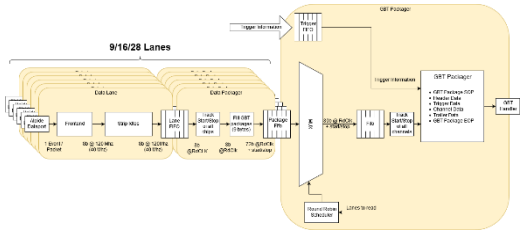
```
/testbench.py cru setGbtTxMux 2 # choose SC as TX downstream data
./testbench.py rdo powerunit-1 initialize
./testbench.py rdo dctrl set_dclk_mask 0
./testbench.py rdo dctrl disable_dclk
./testbench.py powerunit setup_power_IB --bb=0.0
./testbench.py powerunit power-on-IB --module=0 --backbias_en=0
./testbench.py rdo dctrl enable_dclk
./testbench.py rdo dctrl set_dclk_mask 0x1f

# setup for outer barrel:
./testbench.py  test-readout-gpio-setup-sensors 1 True # do calibration of GPIO as well

# alternatively, for inner barrel:
#./testbench.py setup_sensors

./testbench.py setup_readout

./testbench.py rdo gth enable-data 0 # disable GTH frontend
./testbench.py rdo gbt_packer_gth set-settings --enable_data_forward=0 # disable GTH datapath

# also, e.g. enable some pulsers:
./testbench.py clear-pulser
./testbench.py setup-pulser 3 # setup pulser pattern on sensor 3
./testbench.py rdo trigger-handler configure-to-send-pulses
```



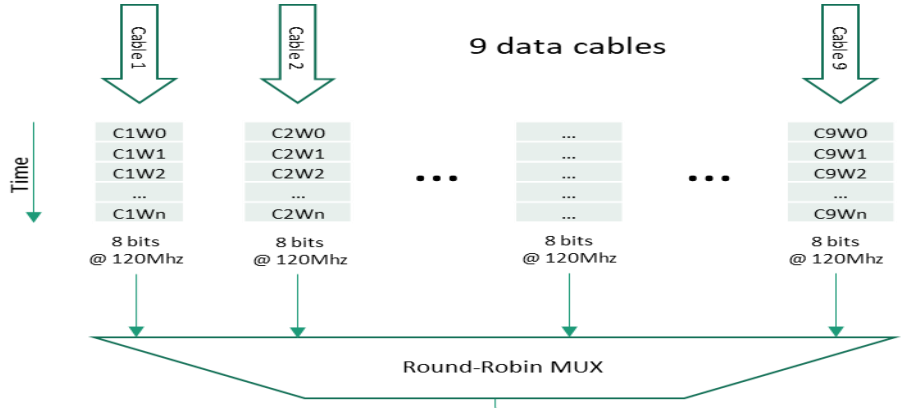
More details: RU operations data readout

RU functions – Data readout firmware blocks

Firmware structure of the data path. Everything is controlled through the data path interface class. The number of lanes depend on the layer: 9 for IB, 16 and 28 for the OB.



RU functions – Data readout – data format IB



Data Valid	72	64	56	48	40	32	24	16	8	0
0	SOP	Length		TTS Busy		0				
1	0		0	Priority Bit	FEE ID		Block Length		Header Size	Header Version
1	0		HB Orbit				TRG Orbit			
1	0		TRG TYPE				0	HB BC	0	TRG BC
1	0		0	Pages Counter		STOP BIT	PAR		Detector Field	
1	St1_Id	Status (e.g. busy)								
1	C1_Id	C1W8							C1W1	C1W0
1	C2_Id	C2W8							C2W1	C2W0
1	...	...							...	...
1	C9_Id	C9W8							C9W1	C9W0
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	St2_ID	Status (e.g. missing data)								
0	EOP	Length		Checksum			End Flag	0		
0	IDLE	0								

Trigger Handler

Data out of the Alpide Readout firmware module in the inner barrel mode arrives with a rate of 8 bits every 120MHz clock period.

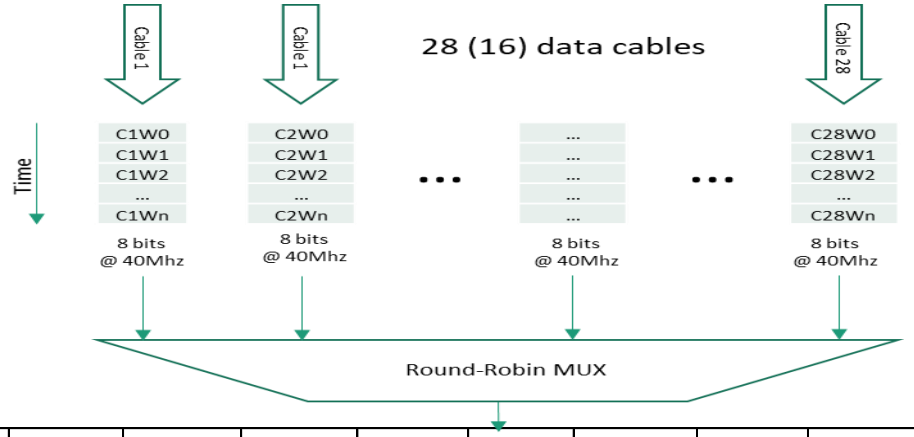
Nine 8-bit data words from one “cable” (copper differential pair) are combined with an 8bit “GBT Header” containing the ID for the FIFO they originate from into an 80 bit GBT word as shown here.

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 as a “Raw Data Header” (RDH) to complete a CRU data packet. Shown here is RDH version 3.

Legend:

C<n>W<m>:	Cable <n>, Word <m>
Cx_Id:	ID of cable in “GBT Header” field
Stx_Id:	ID of status word in “GBT Header” field
SOP:	CRU Start-of-Packet
EOP:	CRU End-of-Packet
BC:	Bunch Crossing ID (12bit)
Orbit:	Orbit ID (32bit)
EndFlag:	Indication if a packet is continued (**bit 27**)

RU functions – Data readout – data format OB



Data Valid	72	64	56	48	40	32	24	16	8	0
0	SOP	Length		TTS Busy		0				
1	0		0	Priority Bit	FEE ID		Block Length		Header Size	Header Version
1	0		HB Orbit				TRG Orbit			
1	0		TRG TYPE				0	HB BC	0	TRG BC
1	0		0	Pages Counter		STOP BIT	PAR		Detector Field	
1	St1_Id	Status (e.g. busy)								
1	C1_Id	C1W8							C1W1	C1W0
1	C2_Id	C2W8							C2W1	C2W0
1	...	...							...	...
1	C28_Id	C28W8							C28W1	C28W0
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	St2_ID	Status (e.g. missing data)								
0	EOP	Length		Checksum			End Flag	0		
0	IDLE	0								

Trigger Handler

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.

Legend:

C<n>W<m>:	Cable <n>, Word <m>
Cx_Id:	ID of cable in “GBT Header” field
SOP:	CRU Start-of-Packet
EOP:	CRU End-of-Packet
BC:	Bunch Crossing ID (12bit)
Orbit:	Orbit ID (32bit)
EndFlag:	If a packet is continued (**bit 27**)

RU functions – Data readout – GBT packet HEADER

The GBT Header field is contained in the eight bits 79:72 of the GBT word:

GBT 80 bits word

Bits	Width	Field	Description
79:72	**8**	**0xe0**	**header indicator (see table below)**
71:44		N/A	not used
43:16	28	active_lanes	Number of lanes eligible for readout
15:0	16	packet_idx	Index of GBT packet within trigger

The first three bit are used to define the meaning of the following 5:

aaa bbbbb

aaa -> 000 -> RDH
001 -> IB
010 -> OB
1xx -> Status Words

	Inner barrel	Outer barrel	Status word
aaa	001	010	111
bb bbb	Lane (0-8)	bb -> Connector (0-3) bbb -> Lane (0-6)	000000 -> Header 000001 -> SW #1 000010 -> … 100000 -> Trailer

RU functions – Data readout – GBT packet TRAILER

111 10000 -> TRAILER

Bits	Width	Field	Description
79:72	8	0xf0	trailer indicator
71:64	8	packet_state	State of GBT_Packer
		4: lane_timeouts	=1 if at least 1 lane timed out
		3: lane_starts_violation	=1 if at least 1 lane had a start violation
		2: packet_overflow	=1 if max number of packets reached
		1: transmission_timeout	=1 if timeout of transmission (lanes)
		0: packet_done	Packet finished
63:60	4		Not used
59:32	28	vlane_timeout	Lane timeouts received
31:28	4		Not used
27:0	28	vlane_stops	Lane stops received

RU functions – Data readout – specification for the FEE ID field

- FEE ID field in RDH is 16-bit wide
- Each RU has 10-bit DIPSWITCH, use bits 9:2 as an 8bit ID field to be mapped to FEE ID field in RDH

Layer	Stave Count	Binary Prefix and Stave Addr	Range of Binary Addresses
L0	12	b 0000 xxxx	0000_0000 : 0000_1011 (0 – 11)
L1	16	b 0001 xxxx	0001_0000 : 0001_1111 (0 – 15)
L2	20	b 001 xxxxx	001_00000 : 001_10011 (0 – 19)
L3	24	b 010 xxxxx	010_00000 : 010_10111 (0 – 23)
L4	30	b 011 xxxxx	011_00000 : 011_11101 (0 – 29)
L5	42	b 10 xxxxxx	10_000000 : 10_101001 (0 – 41)
L6	48	b 11 xxxxxx	11_000000 : 11_101111 (0 – 47)

0	LAYER	0 0	FIBER	0 0	STAVE NUMBER
	L0: 000 L1: 001 L2: 010 L3: 011 L4: 100 L5: 101 L6: 110	Reserv	FIBER UPLINK: b'00 b'01 b'10		STAVE NUMBER L0: 0 – 11 L1: 0 – 15 L2: 0 – 19 L3: 0 – 23 L4: 0 – 29 L5: 0 – 41 L6: 0 – 47

RU functions – Data readout – Additional data format notes

Notes on the Raw Data Header

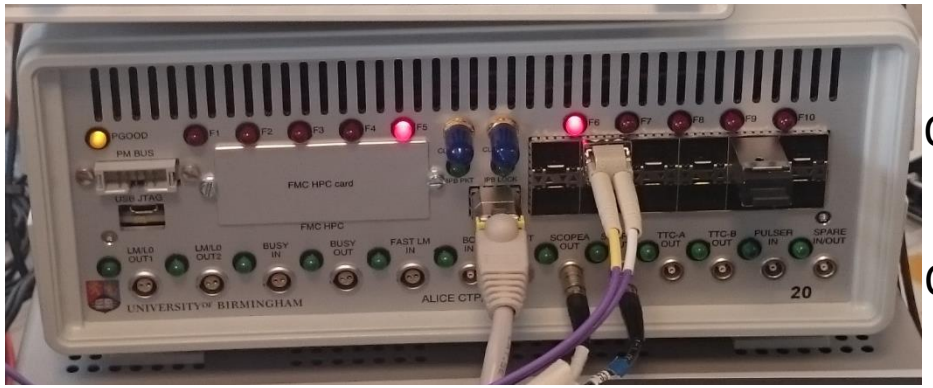
- **Header Size [8bit]:** size of the RDH in bytes: 64 bytes = 0x40
- **Block Length [16bit]:** size of payload (data only, not RDH) in bytes (can be left empty, to be filled in by CRU). What value should be used instead? 0xFFFF?
- **FEE ID [16bit]:** unique ID assigned to FEE (see separate slide)
- **PAR [16bit]:** field used by detector to trigger re-configuration of FEE (are we using this in ITS?)
- **STOP bit [8bit]:** 1 bit to identify the last page (if there are more pages of 8KB belonging to the same trigger). Discussion with F. Costa: This will be left empty in the ITS, since CRU can recognize last page, when pages counter wraps to 0
- **Pages counter [16bit]:** counter to keep track of the different pages belonging to the same trigger
- **Priority Bit [8bit]:** When 0x1 packet is moved forward with higher priority than others (F. Costa: e.g. to report HB frame when the buffers are full)

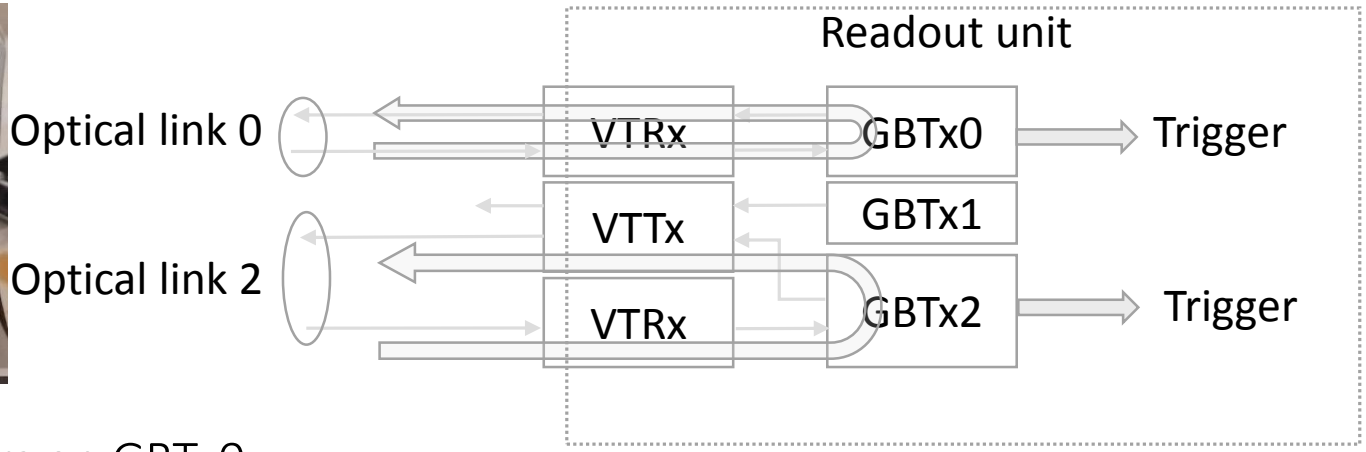
Notes on max GBT packet size and continuation packets

- Each 80 bit GBT packet is mapped to 128 bits in the CRU. The maximum size of each packet can only be 8KB. So this corresponds to **512 GBT** words (HEADER + DATA), since 512 * 128 bit = 8KB.
- It looks like the CRU protocol with the SOP and EOP framing has a redundant mechanism to continue packets via the EndFlag, as is also provided by the Pages Counter and Stop Bit in the RDH. Do we use both to send large packets (> 8KB)?

More details: RU operations trigger management

RU functions – Trigger – hardware connection to LTU

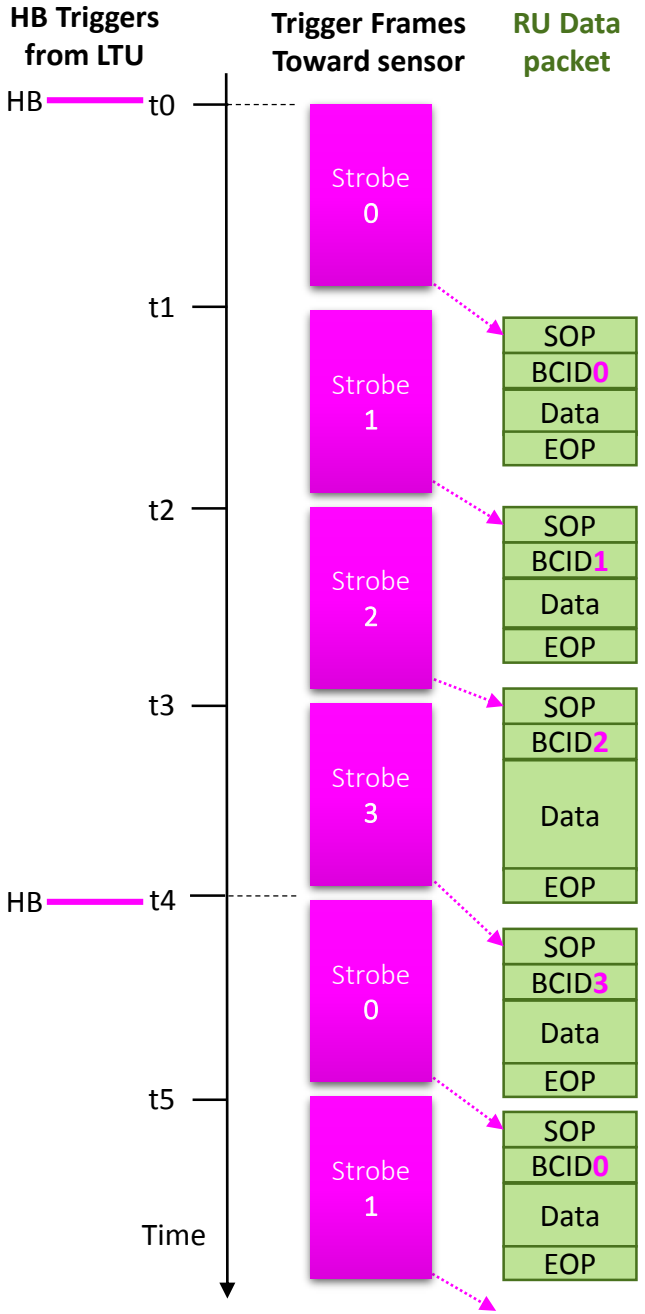




- Present firmware set for receiving triggers on GBTx0
- New firmware developed for receiving on GBTx2
- Testing done with multimode VTRX in place of the single mode for simplicity
- Tested to work with CRU emulator (RUV0) on both links
- Not working with LTU (none of the trigger bits set in the received message)
- Loopback is however working with LTU
- New firmware developed to inspect received trigger data
  - To be tested

Source	Link	Trigger received	Loopback
CRU emulator (RUV0)	0	Y	Y
CRU emulator (RUV0)	2	Y	Y
CRU	0	Y	Y
CRU	2	TBT	TBT
LTU	0	TBT	Y
LTU	2	N	Y

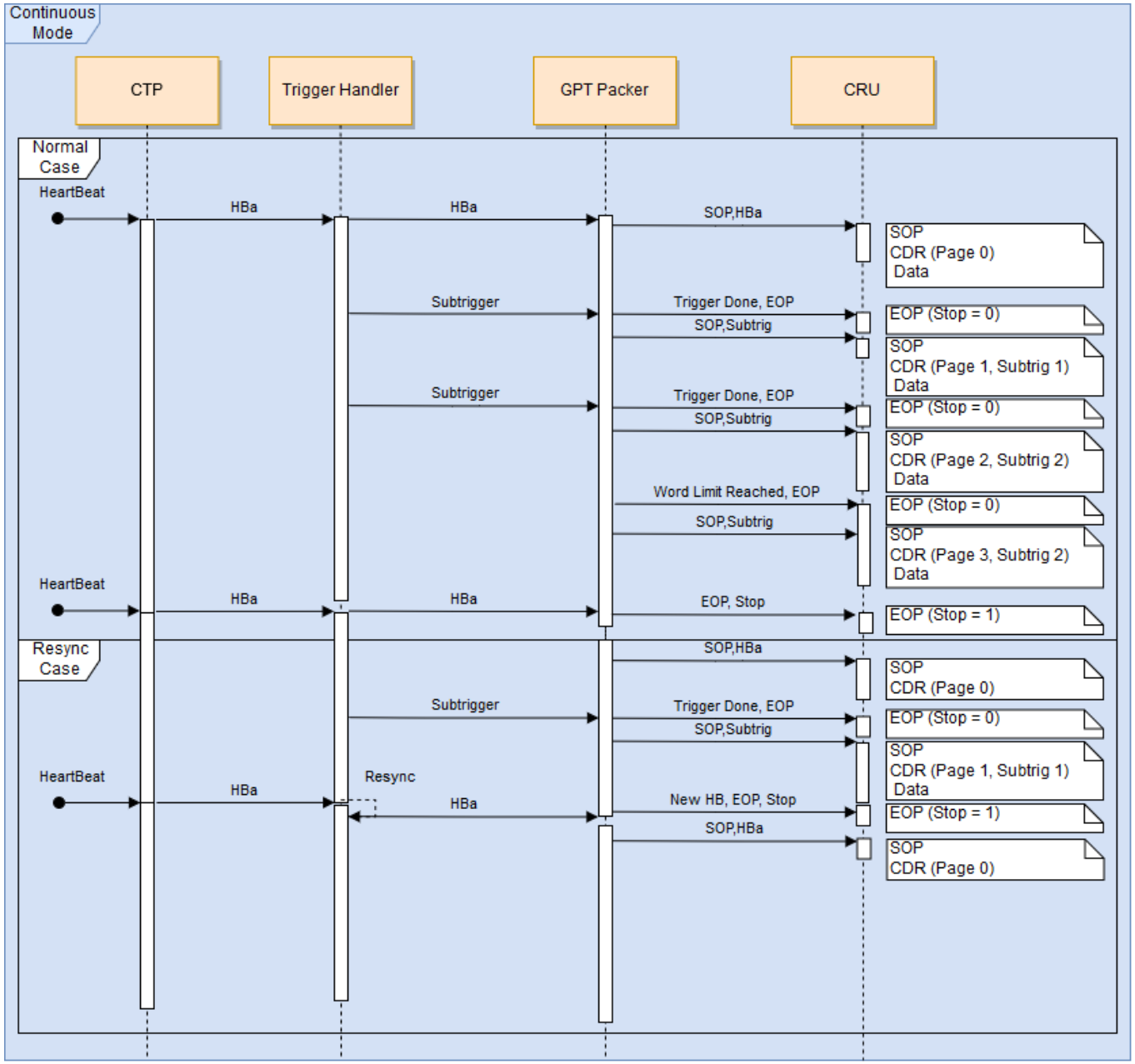
RU functions – Trigger – continuous mode timing information



- "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 frame, 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.

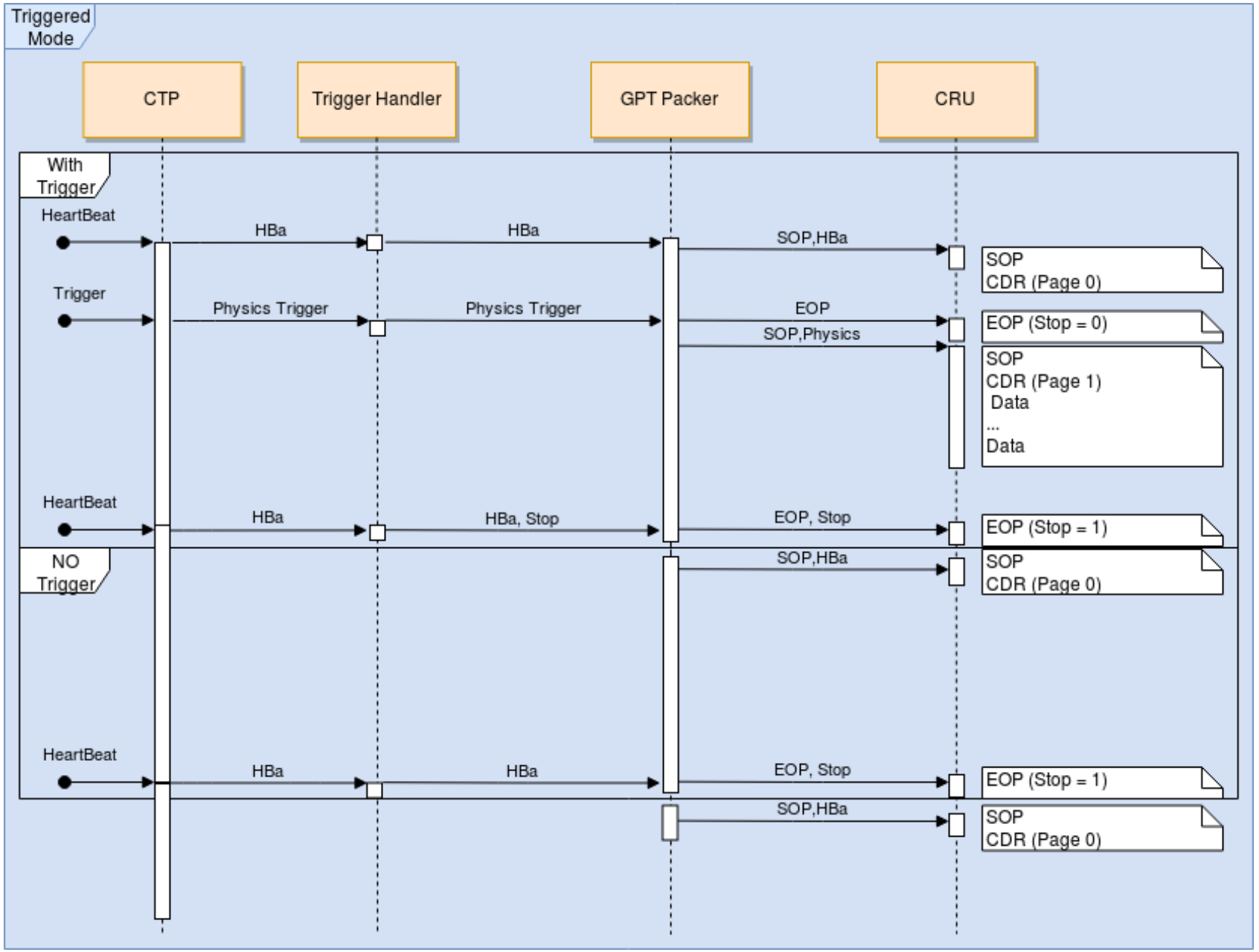
- **HB** = Heart Beat Trigger
- **Strobe<x>** = Trigger strobe to sensor with trigger time = BC+Orbit at time<x>
- **BCID<x>** = BC ID (part of RDH) (with BC+Orbit) at time<x> after HB
- **Data** = Sensor Pixel addresses
- **SOP** = "Start Of Packet" (CRU protocol)
- **EOP** = "End Of Packet" (CRU protocol)

RU functions – Trigger – continuous mode trigger management



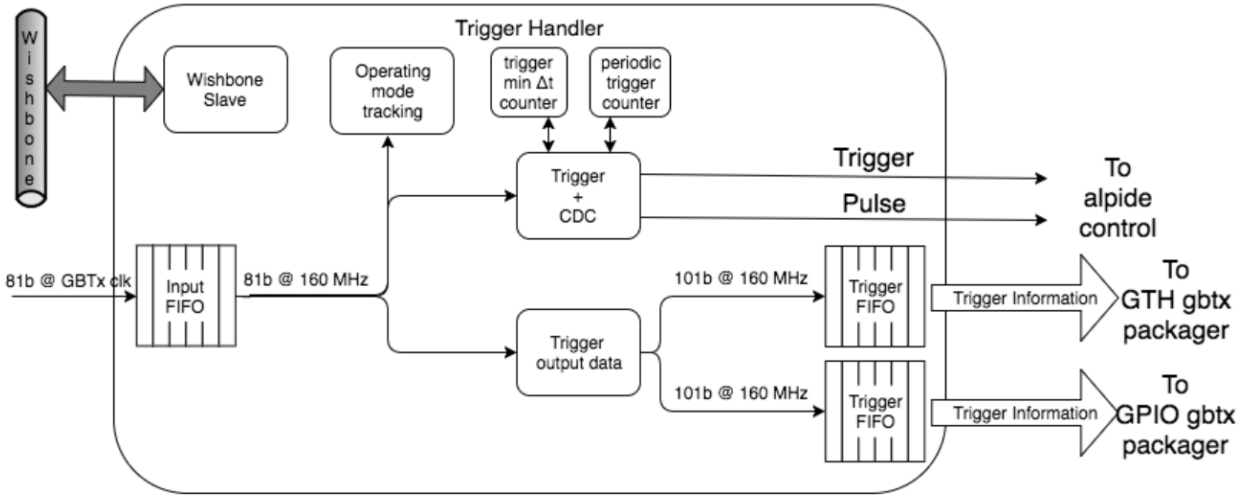
For each sensor strobe (sub-trigger respect to the HB frame, see slide before) we generate a packet with data.

RU functions – Trigger – triggered mode trigger management

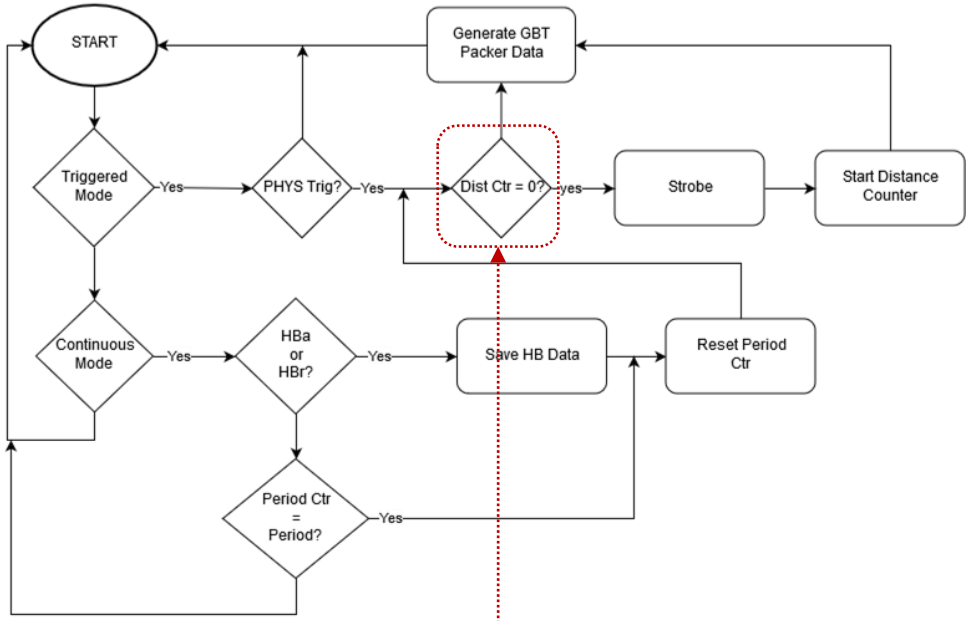


- We always start a new packet at every HB trigger
- In case of a trigger, we fill the current packet with data, then close it and start a new one.
- This repeats for all the triggers we receive.
- At the next HB trigger, we close the current packet (once completed) and open a new one.

RU functions – Trigger – trigger handler flowcharts



Trigger from the CTP (either through the CRU or the LTU)



Ensures too close trigger do not propagate to the sensor, but are recorded for data reconstruction.

More details: RU operations radiation

RU functions – radiation protection highlights

XCKU060

- Current protection level ensures tolerance to the foreseen operating condition.
- Scrubbing is necessary to keep the firmware healthy over long periods (> 10h).
- **Anyway**: if not scrubbing, the firmware is resilient enough to SEE to operate on average for 1300 hours (29 h full IB or 10 h full OB) before experiencing any critical upset.
- Data interruption in case of firmware upset has been measured **and lasts on average one second** for a the specific lane, few seconds if a reset of the FPGA is necessary.
- Errors on clock resources or other key subsystem are negligible (too rare to gather any significant statistics).
- Radiation monitor counters are available across the firmware.

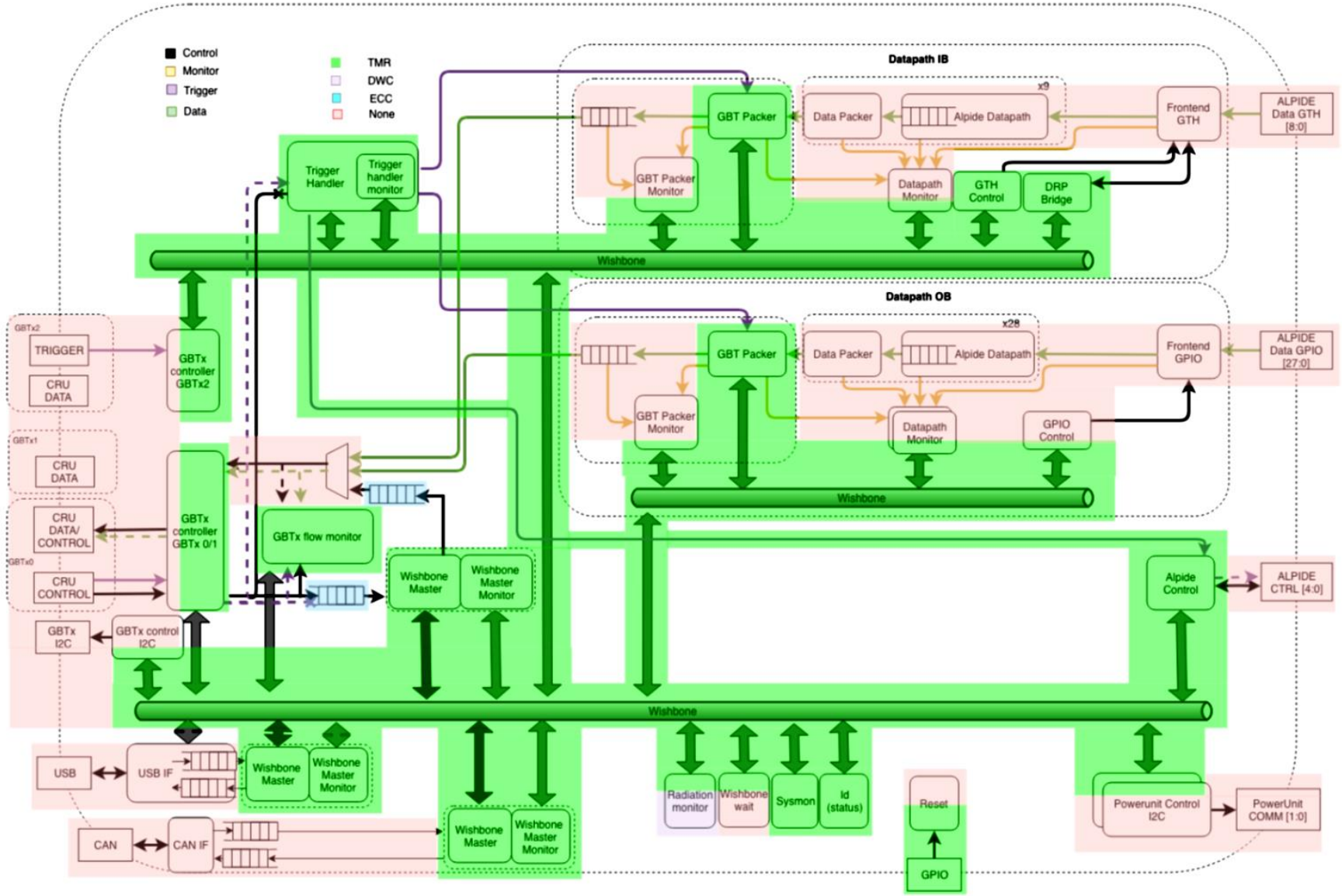
PA3E600M

- Ensures scrubbing and programming facilities for the main FPGA.
- Uses multiple firmware images to avoid FLASH possible corruption.
- Radiation monitor counters are available across the firmware.

Power

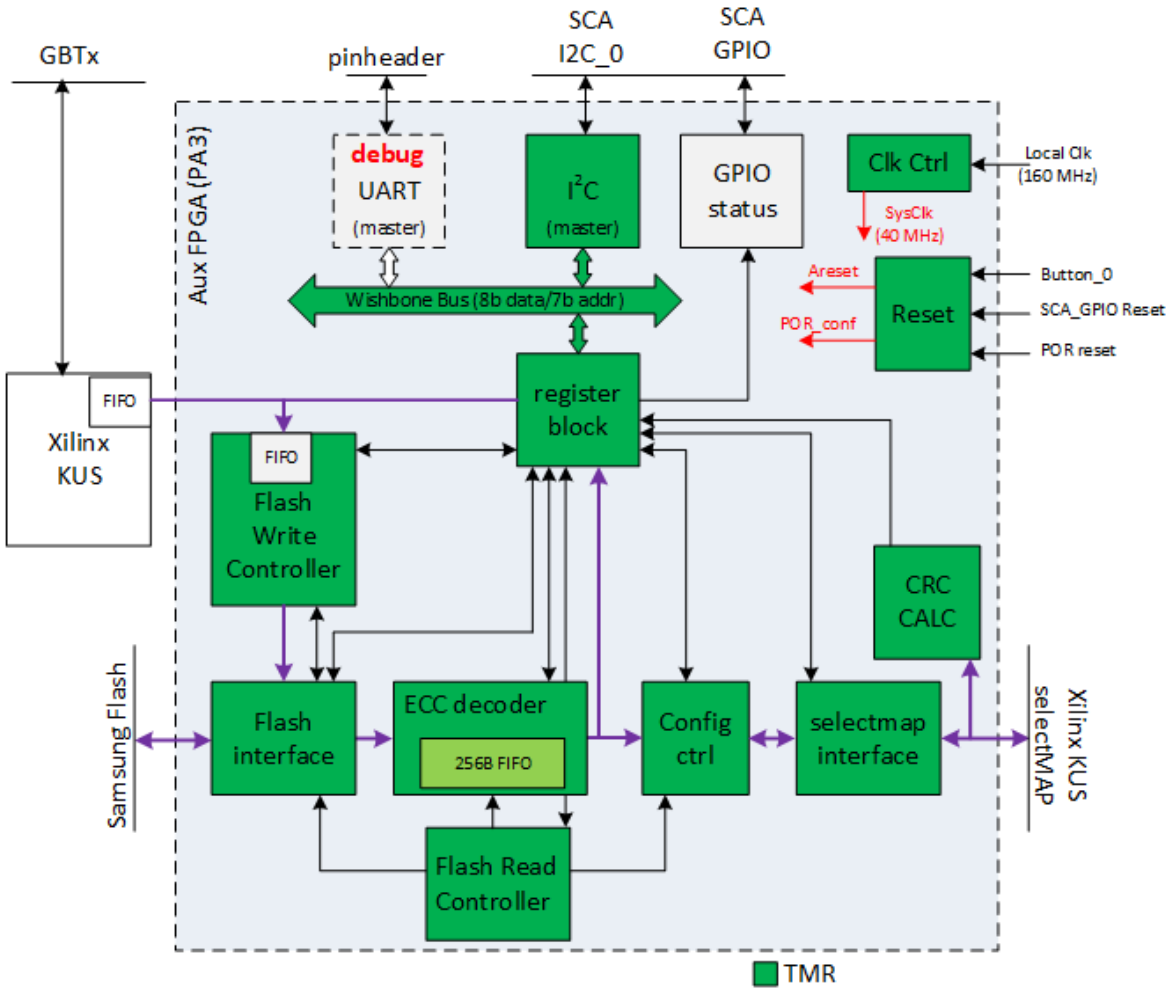
- Commercial DCDC proved compliant with the task, one power glitch every 72 hours foreseen within the whole ITS.

Radiation – KU060 firmware triplication status overview



Radiation – PA3 system firmware block schematic

- Stable release of firmware (**v0206**).



Feature	v0206
Transfer speed to Flash via UART	~50 kBytes/s (if included)
Transfer speed to Flash via I²C (using CRU)	~25 kBytes/s
Transfer speed via Xilinx FIFO (using CRU)	~800 kBytes/s
Initial Programming of Xilinx from Flash	~2 sec
Blind Scrubbing of Xilinx from Flash	~1.7 sec
I²C Single Register Read	~39 us
I²C Single Register Write	~39 us
Resource utilization	63% COREs 3/24 RAMs
Timing (Required: 40 MHz)	~47 MHz

RU functions – SCA control of the PA3 system – 1

Initialization

initialize() - initialize ProAsic3, set I2C channel
set_i2c_channel(ch) - set I2C channel used (0 or 1)

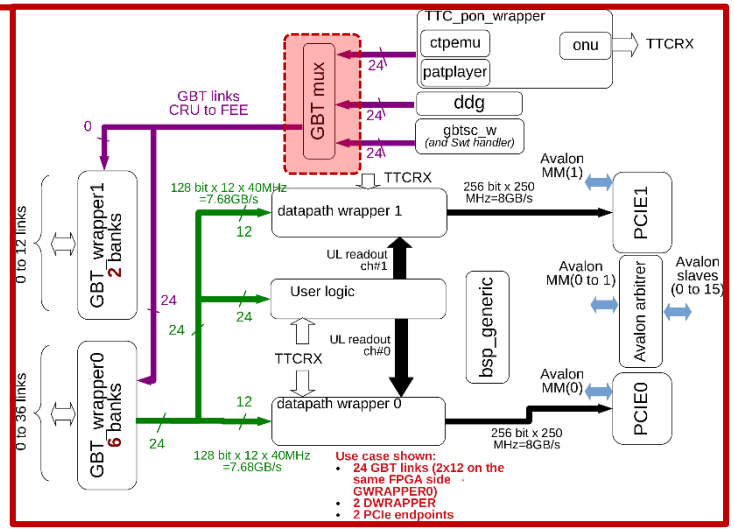
Registers and settings

write_reg(addr, val) - write PA3 register
read_reg(addr) - read PA3 register
write_fifo_reg(addr, data) - write to PA3 in FIFO mode (<16 bytes)
read_fifo_reg(addr, len) - read from PA3 in FIFO mode (<16 bytes)
set_reset(val) - set RESET GPIO pin
set_start_program - set START_PROGRAM GPIO pin
get_clock_status() - clock status
get_clock_config() - clock config

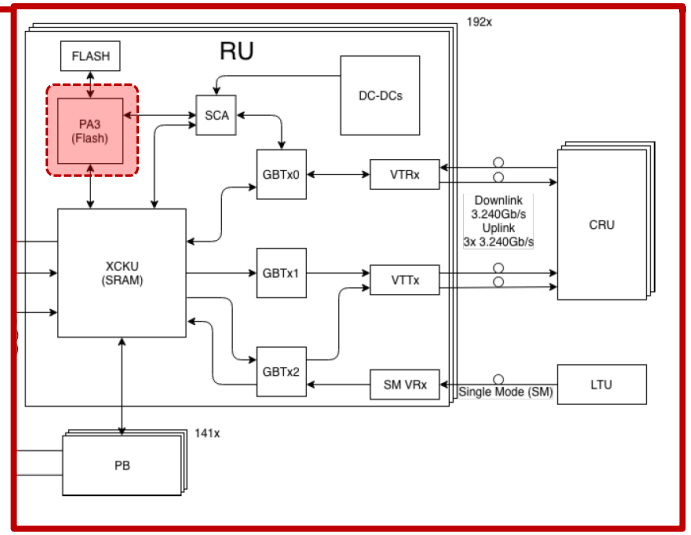
Programming and scrubbing

set_cc_command_register(opcode, execute) -
get_cc_command_register() -
get_program_done() -
reprogram_ultrascale() -
start_blind_scrubbing() -
stop_blind_scrubbing() -
trigger_single_scrub() -
clear_scrubbing_counter() -
get_cc_status() -
get_scrubbing_counter() -
get_crc() -

is_idle() -
is_init_config_done() -
is_scrubbing() -
get_cc_active_state() -
set_rc_command_register(opcode, execute) -
get_rc_command_register() -
get_rc_current_page() -
get_rc_active_state() -



RU functions – SCA control of the PA3 system – 2



SelectMap interface functions

set_smap_command_registers(opcode, execute) -
get_smap_command_register() -
set_smap_byte_tx(byte) -
get_smap_byte_tx() -
get_smap_byte_rx() -
get_smap_status() -

FLASH FIFO functions

get_fifo_rx_data() -
set_fifo_tx_data(data) -
get_fifo_status() -
set_fifo_writer_command_register(op, execute) -
get_fifo_writer_command_register() -
get_fifo_writer_status() -

FLASH memory interface functions

set_flash_command_register(op, execute) -
get_flash_command_register() -
set_flash_address(block, page) -
get_flash_byte_counter() -
set_flash_select_ic(val) -
get_flash_status_word() -
set_pattern_checker(val) -
get_flash_state() -
get_flash_page_size() -
set_flash_page_size() -

RU functions – SCA control of the PA3 system – 3

Error Correction Code functions

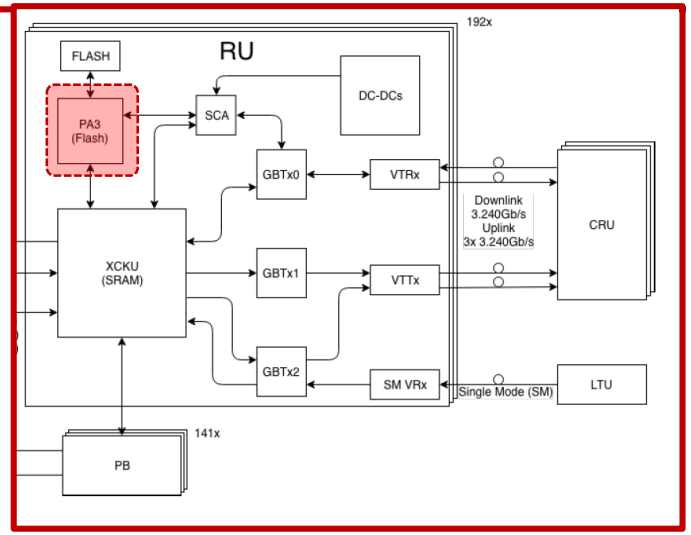
set_ecc_command_register(opcode, execute) -
get_ecc_command_register() -
enable_ecc() -
disable_ecc() -
reset_ecc_counters_and_status() -
clear_ecc_status() -
get_ecc_sb_error_counter() -
get_ecc_fifo_tmr_error() -

Debug

get_dipswitches() -
get_led() -
set_led() -

Utility functions for firmware writing

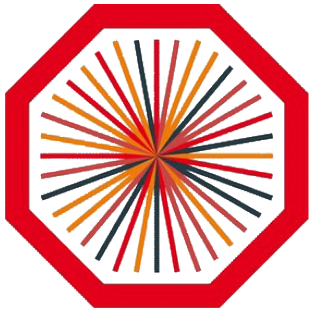
flash_read_page(page) -
flash_write_parameter_page(….) -
flash_write_firmware(filename, …) -



Radiation – expected failure rates

Failure mode	Affected section	Estimated occurrence in ITS operations (average MTBF)			Corrective action	Downtime
		IB	OB	Whole ITS		
Sensor data lane	1 sensor for IB ½ module for OB	22 - 40 h	4 - 6 h	3 - 5 h	Self-repairing	< 1 s >
GBT data*	1 full stave	29 h	10 h	7 h	30% self repairing, 70% reset by slow control	< 5 s >
Clock resources	1 full stave	Negligible	Negligible	Negligible	Reset by slow control	< 5 s >
Transceiver	1 sensor, IB only	> 932 h	–	–	Reset by slow control	< 5 s>
Flash memory	1 full stave	Negligible	Negligible	Negligible	FLASH reprogramming (30 s beam off, 30m beam on)	30 s – 30 m
[1]PA3	1 full stave	172 h	58 h	43 h	Reset by slow control	< 0 s>
[2]DCDC glitch	1 full stave	294 h	98 h	71 h	Power cycle	< 10 s >

* Data for the non-TMR block, final version will use TMR protected block (done). This failure mode also include sensor control and clock failures.
[1] When PA3 get stuck the main FPGA is not compromised, and therefore no downtime occurs. TMR of key block in PA3 firmware will further improve that (done).
[2] Considering 200 RU with 8 DCDC each (20% overestimation)



Summary

Summary – Simplified system readiness matrix

This is from WP10 point of view, we are here to downgrade few/some/many tiles accordingly to other system requirements/considerations/suggestions!

	PB	RU			CRU		DCS		O2
		Hrd	Frmw	Sftw	Frmw	Sftw	Alf	Fred	
RU Programming		100%	50% [1]	70%	100%	100% *	?	?	
RU Ctrl/mon/operation (CRU/DCS/O2)		100% [2]	100%	70%	100%	100% *	?	?	
Power board ctrl/mon/operation	100% [3]	100%	100%	100%	100%	100% *	70%	70%	
Sensor/stave ctrl/mon/operation		100%	70% [4]	70%	70% [5]	80% *	70%	70%	
Data Readout ctrl/mon		100% [2]	80%	80%	100%	80% *			
Trigger management		90%	80% [6]	100%	100%	80% *			
Radiation tolerance/monitor	100%	100%	80%	80%	100%	100% *	?	?	
Backup DCS control (CAN)		100%	70% [7]	70%			?	?	

[1] Missing the CRU→XCKU060- → PA3 fast path
[2] GBT issue still on open investigation
[3] Power-up transient issue being addressed
[4] Ongoing effort to extend ALPIDE ctrl functionalities
[5] RU SWT push operations / broadcast to be discussed
[6] We have input from the commissioning
[7] radiation protection missing (and not trivial)
* Documentation is improving