MTD DAQ and Tamalero lpGBT Power Up

Hayden Swanson, Naomi Gonzalez

BOSTON UNIVERSITY

CMS

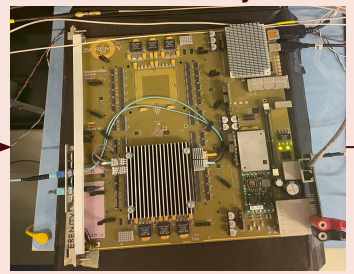
Currently we are unable to communicate with the lpGBT on our readout board version 3 using the MTD DAQ firmware + software.

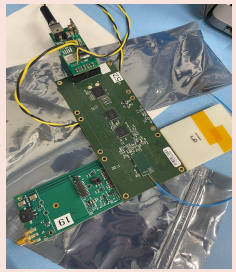
Goal: Compare how we first contact the lpGBT between tamalero and MTD DAQ.

MTD DAQ

Firmware: MTD DAQ firmware
Software: MTD DAQ Software

PC → **DAQ** Serenity → **Front End** RB3 (MUX64 ASIC)

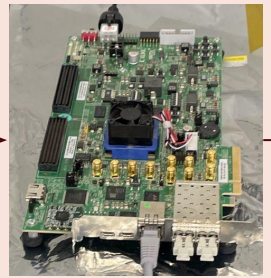


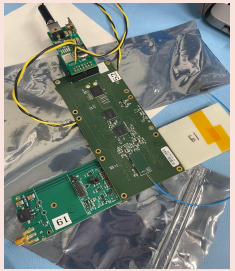


ETL Testing DAQ

Firmware: module_test_fw
Software: module_test_sw / tamalero

PC → **DAQ** KCU105 → **Front End** RB3 (MUX64 ASIC)



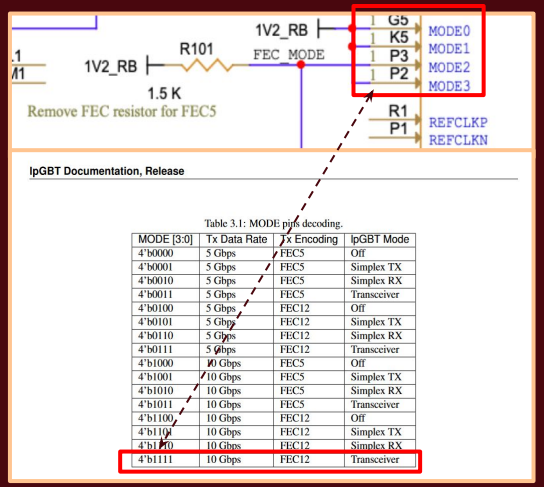


What we have tried so far

Ensured correct uplink configuration in firmware based on the hardware:

- Ran MTD DAQ software routine using firmware with the correct data rates and FEC modes (10Gbps, FEC12)
- Also attempted FEC5 @ 10Gbps just in case

We cannot read any registers on lpGBT using MTD DAQ firmware + software



What are the Configurations Steps that lead to First Read of lpGBT in each FW/SW?

Power Up Sequence

Tamalero

Strategy: Ensure communication by reading the lpGBT the ROMREG

1. **Set the frame format of the KCU depending on the lpGBT version**
2. Toggle Uplink data path test patterns
   a. This replaces the data signals coming from the ePortRx oon the lpGBT with the downlink for 0.1 seconds then returns it back to normal
3. Set highSpeedDataOutInvert to 1
   a. Equivalent to swapping HSOUTP and HSOUTN on the PCB
4. Power Up lpGBT
   a. Sets dllConfigDone and pllConfigDone
      i. DLL = delay locked loop, related to uplink phase alignment
      ii. PLL = Phase locked loop, circuit receives high speed serial data from the downlink
5. Read ROM register

```
self.kcu.write_node("READOUT_BOARD_%d.SC.FRAME_FORMAT" % self.rb, 0)
```

lpGBT v0 and v1 have a different data format that needs be specified on the firmware level to the KCU.

The IC Frame Format tells you the format of the data words from the lpGBT.

See firmware code

```
type READOUT_BOARD_SC_CTRL_t is record
  TX_RESET                :std_logic;     -- Reset TX datapath
  RX_RESET                :std_logic;     -- Reset RX datapath
  TX_START_WRITE          :std_logic;     -- Request a write config to the GBTx (IC)
  TX_START_READ           :std_logic;     -- Request a read config to the GBTx (IC)
  TX_GBTX_ADDR            :std_logic_vector( 7 downto 0);  -- I2C address of the GBTx
  TX_REGISTER_ADDR        :std_logic_vector(15 downto 0);  -- Address of the first register to be accessed
  TX_NUM_BYTES_TO_READ    :std_logic_vector(15 downto 0);  -- Number of words/bytes to be read (only for read transactions)
  FRAME_FORMAT            :std_logic;                      -- IC Frame format: 0 = lpGBT v0; 1 = lpGBT v1
  TX_DATA_TO_GBTX         :std_logic_vector( 7 downto 0);  -- Data to be written into the internal FIFO
  TX_WR                   :std_logic;                      -- Request a write operation into the internal FIFO (Data to GBTx)
  TX_CMD                  :std_logic_vector( 7 downto 0);  -- Command: The Command field is present in the frames received by t
  TX_ADDRESS              :std_logic_vector( 7 downto 0);  -- Command: It represents the packet destination address. The addres
  TX_TRANSID              :std_logic_vector( 7 downto 0);  -- Command: Specifies the message identification number. The reply m
  TX_CHANNEL              :std_logic_vector( 7 downto 0);  -- Command: The channel field specifies the destination interface of
  TX_DATA                 :std_logic_vector(31 downto 0);  -- Command: data field (According to the SCA manual)
  SCA_ENABLE              :std_logic;                      -- Enable flag to select SCAs
  START_RESET             :std_logic;                      -- Send a reset command to the enabled SCAs
  START_CONNECT           :std_logic;                      -- Send a connect command to the enabled SCAs
  START_COMMAND           :std_logic;                      -- Send the command set in input to the enabled SCAs
  INJ_CRC_ERR             :std_logic;                      -- Emulate a CRC error
end record READOUT_BOARD_SC_CTRL_t;
```

Power Up Sequence

Tamalero

Strategy: Ensure communication by reading the lpGBT the ROMREG

1. Set the frame format of the KCU depending on the lpGBT version
2. **Toggle Uplink data path test patterns with downlink for 0.1 seconds**
3. Set highSpeedDataOutInvert to 1
   a. Equivalent to swapping HSOUTP and HSOUTN on the PCB
4. Power Up lpGBT
   a. Sets dllCOnfigDone and pllConfigDone
      i. DLL = delay locked loop, related to uplink phase alignment
      ii. PLL = Phase locked loop, circuit receives high speed serial data from the downlink
5. Read ROM register

```python
# toggle the uplink to and from 40MHz clock, for some reason this is
# needed for the mgt to lock
self.wr_adr(0x118, 0xC0) #
https://lpgbt.web.cern.ch/lpgbt/v0/registermap.html#x118-uldatasource0
sleep(0.01)
self.wr_adr(0x118, 0) # https://lpgbt.web.cern.ch/lpgbt/v0/registermap.html#x118-
uldatasource0
```

14.1. Test pattern generators

The lpGBT offers a possibility to generate patterns and inject them at various places in the data path in order to simplify chip/system debugging. Points at which data can be generated are highlighted on Fig. 14.2.

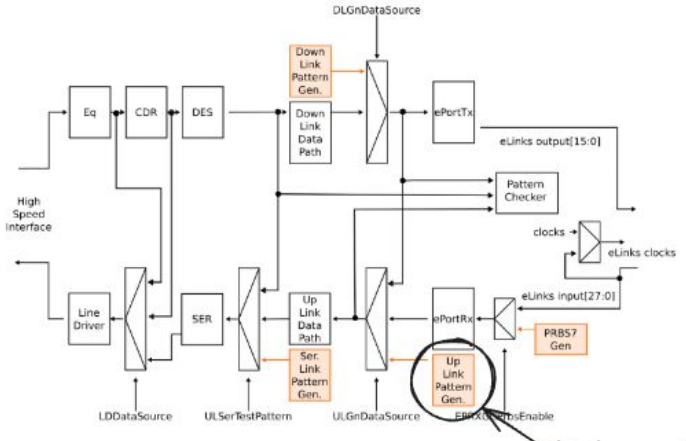


Fig. 14.2 lpGBT data path pattern generators.

Power Up Sequence

Tamalero

Strategy: Ensure communication by reading the lpGBT the ROMREG

1. Set the frame format of the KCU depending on the lpGBT version
2. Toggle Uplink data path test patterns
3. **Set highSpeedDataOutInvert to 1**
4. Power Up lpGBT
   a. Sets dllConfigDone and pllConfigDone
      i. DLL = delay locked loop, related to uplink phase alignment
      ii. PLL = Phase locked loop, circuit receives high speed serial data from the downlink
5. Read ROM register

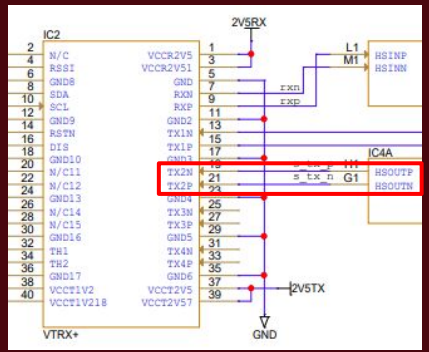
```python
# https://lpgbt.web.cern.ch/lpgbt/v0/registermap.html?highlight=0x036#chip-config
self.wr_adr(0x036, 0x80)
```

15.1.4. CHIP Config

0x036 ChipConfig

- Bit 7 - **highSpeedDataOutInvert** - Inverts high speed data output lines (equivalent to swapping HSOUTP and HSOUTN on the PCB)
- Bit 6 - **highSpeedDataInInvert** - Inverts high speed data input lines (equivalent to swapping HSINP and HSINN on the PCB)
- Bit 2:0 - **ChipAddressBar[2:0]** - Sets most significant bits of the chip address (see Section 3.3).

Since the differential pair lines are backwards on the Readout Board to the lpGBT we need to switch this flag.



LpGBT

Power Up Sequence

Tamalero

Strategy: Ensure communication by reading the lpGBT the ROMREG

1. Set the frame format of the KCU depending on the lpGBT version
2. Toggle Uplink data path test patterns
   a. This replaces the data signals coming from the ePortRx oon the lpGBT with the downlink for 0.1 seconds then returns it back to normal
3. Set highSpeedDataOutInvert to 1
   a. Equivalent to swapping HSOUTP and HSOUTN on the PCB
4. **Power Up lpGBT**
5. Read ROM register

```python
self.wr_adr(0x0ef, 0x6)
sleep(0.01) # sleeps to wait for power up to finish
```

15.1.17. Power Up State Machine

[0x0ef] POWERUP2

Controls behavior of the power up state machine (for more details refer to Power-up state machine)

- **Bit 2** - **dllConfigDone** - When asserted, the power up state machine is allowed to proceed to PLL initialization. Please refer Configuration for more details.
- **Bit 1** - **pllConfigDone** - When asserted, the power up state machine is allowed to proceed to initialization of components containing DLLs (ePortRx, phase-shifter). Please refer Configuration for more details.
- **Bit 0** - **updateEnable** - When asserted, the power up state machine copies the values from fuses to configuration registers during power. Please refer Configuration for more details.

Sets both of these bits to 1, dllCOnfigDone and pllConfigDone on the lpGBT. Both are needed for full power up on lpGBT.

- Dll = delay locked loop, related to uplink phase alignment,
  - Uplink phase alignment
- PLL = Phase locked loop, the Clock and Data Recover and PLL circuit receives high speed serial data from the downlink.
  - Architecture and Functionality Overview

Power Up Sequence

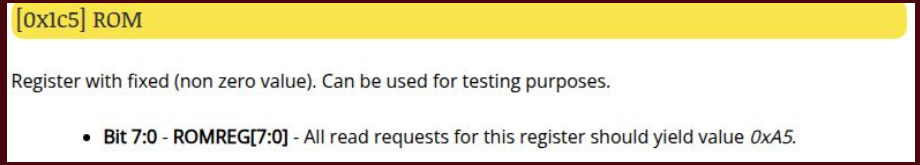
Tamalero

Strategy: Ensure communication by reading the lpGBT the ROMREG

1. Set the frame format of the KCU depending on the lpGBT version
2. Toggle Uplink data path test patterns
3. Set highSpeedDataOutInvert to 1
4. Power Up lpGBT
5. **Read ROM register**

NOTE: Read is attempted 50 times

If you get here you are cooking 😎



[0x1c5] ROM

Register with fixed (non zero value). Can be used for testing purposes.

- **Bit 7:0 - ROMREG[7:0]** - All read requests for this register should yield value *0xA5*.

If it is v0 lpgbt this should read 0xA5 and if it is v1 this should read 0xA6

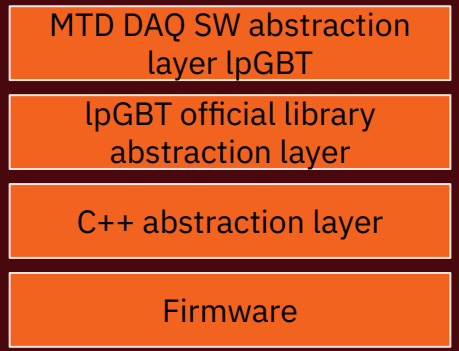
Power Up Sequence

MTD

Strategy: Power up and keep trying to read the status register for the power up state machine

1. **Select the link**
2. Reset the SCC
3. Set highSpeedDataOutInvert to 1
4. Series of setups, writes to 20 registers under these methods from the official lpGBT software
   a. clock_generator_setup
   b. line_driver_setup
   c. equalizer_setup
   d. config_done / power-up
5. Read current state of the power up state machine until it is done

uhal communication here



Example, link = 6, related to the fibers

```
self.emp_cont.getDatapath().selectLink(link)
```

```
void DatapathNode::selectLink(uint32_t link) const
{
  // logger::trace("Selecting link {}", link.value());
  this->selectRegChan(link / 4, link % 4);
}
```

```
void DatapathNode::selectRegChan(uint32_t quad, uint32_t chan) const
{
  getNode("ctrl.quad_sel").write(quad);
  getNode("ctrl.chan_sel").write(chan);
  getClient().dispatch();
}
```

Power Up Sequence

MTD

Strategy: Power up and keep trying to read the status register for the power up state machine

1. Select the link
2. **Reset the SCC**
3. Set highSpeedDataOutInvert to 1
4. Series of setups, writes to 20 registers under these methods from the official lpGBT software
   a. clock_generator_setup
   b. line_driver_setup
   c. equalizer_setup
   d. config_done / power-up
5. Read current state of the power up state machine until it is done

```
self.emp_cont.getSCC().reset()
```

```
getNode("ctrl.gbtsc_rst").write(1);
getClient().dispatch();
```

Power Up Sequence

MTD

Strategy: Power up and keep trying to read the status register for the power up state machine

1. Select the link
2. Reset the SCC
3. **Set highSpeedDataOutInvert to 1**
4. Series of setups, writes to 20 registers under these methods from the official lpGBT software
   a. clock_generator_setup
   b. line_driver_setup
   c. equalizer_setup
   d. config_done / power-up
5. Read current state of the power up state machine until it is done

```
self.lpgbt_cont_.lpgbt_com.set_link(
    reg_addr = reg_addr,
    mask = mask,
    lpgbt_addr = int(self.lpgbt_cont_.lpgbt_addr, 16))
```

0x0036, CHIPCONFIG REG

0x80

```
def set_link(self, reg_addr, mask, lpgbt_addr):
    """
    Inverts uplink, needed for correct communication
    """
    self.SCCIC.icInvertLink(reg_addr, mask, lpgbt_addr)
```

```
void SCCICNode::icInvertLink(unsigned aAddress, unsigned aData, unsigned
aGbtxAddress) const
{
  reset();
  getNode("gbtx_addr").write(aGbtxAddress);
  unsigned lVal = (1 << 20) + ((aAddress & 0xfff) << 8) + (aData & 0xff);
  getNode("txdata_fifo").write(lVal);
  getClient().dispatch();
}
```

Seems to be set!

100000011011010000000

Power Up Sequence

MTD

Strategy: Power up and keep trying to read the status register for the power up state machine

1. Select the link
2. Reset the SCC
3. Set highSpeedDataOutInvert to 1
4. **Series of setups, writes to 20 registers under these methods from the official lpGBT software**
   a. **clock_generator_setup**
   b. **line_driver_setup**
   c. **equalizer_setup**
   d. **config_done / power-up**
5. **Read current state of the power up state machine until it is done**

Need to clarify with Giorgio/Ozgur what part of this was failing. Possibly step 5 because the rest are blind writes.

`clock_generator_setup`

Seems to write to a bunch of registers. Not registers are read yet.

Registers written to:

- `REFCLK`
- `CLKGCONFIG0`
- `CLKGCONFIG1`
- `CLKGPLLINTCUR`
- `CLKGPLLPROPCUR`
- `CLKGPLLRES`
- `CLKGFFCAP`
- `CLKGCDRINTCUR`
- `CLKGFLLINTCUR`
- `CLKGCDRPROPCUR`
- `CLKGCDRFFPROPCUR`
- `CLKGLFCONFIG0`
- `CLKGLFCONFIG1`
- `CLKGWAITTIME`
- `EPRXDLLCONFIG`
  - `self.write_reg(self.EPRXDLLCONFIG, dll_config)`

`line_driver_setup`

Still only writes to registers

- `LDCONFIGH`
- `LDCONFIGL`

`equalizer_setup`

Still only writes to registers

- `EQCONFIG`
- `EQRES`

`config_done`

Still only writes to registers, this starts the power up state machine

- `POWERUP2`

lpGBT Register Comparisons
Clock Generation

Reg Name	MTD*	Tamalero
REFCLK	011	000
CLKGConfig0	1110 1000	1110 1000
CLKGConfig1	0011 1000	0011 1000
CLKGPLLIntCur	1001 1001	1001 1001
CLKGPLLPropCur	1001 1001	1001 1001
CLKGPllRes	0010 0010	0010 0010
CLKGFFCAP	00 011 011	00 011 011
CLKGCDRIntCur	0101 0101	0101 0101
CLKGFLLIntCur	0101 0101	0101 0101

Reg Name	MTD*	Tamalero
CLKGCDRPropCur	0101 0101	0101 0101
CLKGFLLIntCur	0101 0101	0101 0101
CLKGCDRPropCur	0101 0101	0101 0101
CLKGCDRFFPropCur	0110 0110	0110 0110
CLKGLFConfig0	1000 1111	1000 1111
CLKGLFConfig1	1111 1111	1111 1111
CLKGWaitTime	1000 1000	1000 1000
EPRXDLLConfig	01 10 0100	01 01 0000

*Values based on code; MTD Value setup is hardcoded in Python lpgbt Layer

Register Differences

[0x03b] REFCLK

Configuration for the reference clock pad

- **Bit 2** - **REFCLKForceEnable** - Enable the reference clock pad regardless of the operation mode.
- **Bit 1** - **REFCLKAcBias** - Enables the common mode generation for the REFCLK.
- **Bit 0** - **REFCLKTerm** - Enables the 100 Ohm termination for the REFCLK input.

MTD
011

Tamalero
000

Register Differences

MTD
01 10 0100

Tamalero
01 01 0000

[0x0f1] EPRXDllConfig

Configuration register containing settings for EPRX DLLs. This register contains also auxiliary EPRX setting.

- **Bit 7:6 - EPRXDllCurrent[1:0]** - Current for the DLL charge pump (default: 1).

EPRXDllCurrent[1:0]	Current [uA]
2'd0	1
2'd1	2
2'd2	4
2'd3	8

- **Bit 5:4 - EPRXDLLConfirmCount[1:0]** - Number of clock cycles (in the 40 MHz clock domain) to confirm locked state (default: 2).

EPRXDLLConfirmCount[1:0]	Number of clock cycles
2'd0	1
2'd1	4
2'd2	16
2'd3	31

- **Bit 3 - EPRXDLLFSMClkAlwaysOn** - Force clock of ePortRx DLL state machine to be always enabled (disables clock gating)
- **Bit 2 - EPRXDLLCoarseLockDetection** - Use coarse detector for the DLL lock condition
- **Bit 1 - EPRXEnableReInit** - Enables the re-initialization of an ePortRxGroup when the phase-aligner state machine finds the phase-selection to be out of range (default: 0)
- **Bit 0 - EPRXDataGatingDisable** - Disable data gating. When the data gating is enabled (**EPRXDataGatingDisable** bit set to zero) the ePortRx group consumes less power. This is a recommended mode of operation (default: 0)

lpGBT Register Comparisons

Line Driver Setup

Reg Name	MTD*	Tamalero
LDConfigH	0 1000000	0 11111111
LDConfigL	0 0000000	0 0000000

Equalizer Setup

Reg Name	MTD*	Tamalero
EQConfig	11 00	00 00
EQRes	00 00 00 00	00 00 00 00

*Values based on code; MTD Value setup are hardcoded values from Python lpgbt layer

Register Differences

15.1.6. Line Driver

[0x039] LDConfigH

Line driver configuration register

- **Bit 7** - **LDEmphasisEnable** - Enable pre-emphasis in the line driver. The amplitude of the pre-emphasis is controlled by LDEmphasisAmp[6:0] and the duration by LDEmphasisShort.
- **Bit 6:0** - **LDModulationCurrent[6:0]** - Sets high-speed line driver modulation current: I_m = 137 uA * LDModulationCurrent[6:0]

[0x03a] LDConfigL

Line driver configuration register

- **Bit 7** - **LDEmphasisShort** - Sets the duration of the pre-emphasis pulse. Please not that pre-emphasis has to be enabled (LDEmphasisEnable) for this field to have any impact.

- **Bit 6:0** - **LDEmphasisAmp[6:0]** - Sets high-speed line driver pre-emphasis current: I_{pre} = 137 uA * LDEmphasisAmp[6:0]. Please note that pre-emphasis has to be enabled (LDEmphasisEnable) for these registers bits to be active.

  -Note for the LDConfigH and LDConfigL registers: since the high-speed line driver contains an internal 100 Ohm "termination", the currents set by LDModulationCurrent[6:0] and LDEmphasisAmp[6:0] bits are shared between the internal and external load impedances. This needs to be taken into account when computing the output signal amplitude. To calculate the modulation amplitude the user should thus use the equivalent resistor value of 50 Ohm, that is, the internal 100 Ohm resistor in parallel with the external 100 Ohm termination impedance.

Reg Name	MTD*	Tamalero
LDConfigH	0 1000000	0 11111111

Register Differences

15.1.5. Equalizer

[0x037] **EQConfig**

Main equalizer configuration register

- **Bit 4:3 - EQAttenuation[1:0]** - Attenuation of the equalizer. Use a gain setting of *1/1* (*EQAttenuation[1:0]=2'd3*) when VTRX+ is used.

EQAttenuation[1:0]	Gain [V/V]
2'd0	1/3
2'd1	2/3
2'd2	2/3
2'd3	1/1

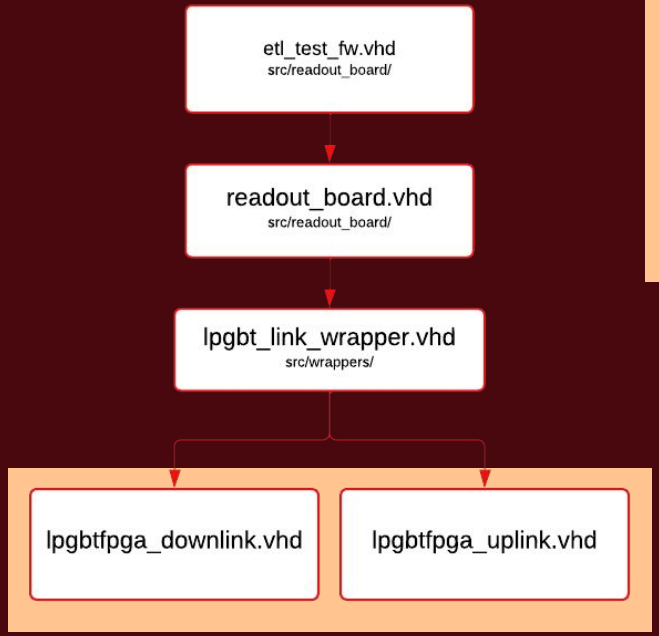
- **Bit 1:0 - EQCap[1:0]** - Capacitance select for the equalizer

EQCap[1:0]	Capacitance [fF]
2'd0	0
2'd1	70
2'd2	70
2'd3	140

Reg Name	MTD*	Tamalero
EQConfig	11 00	00 00

Firmware Overview Links

KCU105



Lpgbtfpga_downlink
+
Lpgbtfpga_uplink

Are common firmware blocks

Serenity

LPGBT Uplink

KCU105

```
uplink_inst : entity lpgbt_fpga.lpgbtfpga_uplink

  generic map (
    datarate                  => g_UPLINK_DATARATE,
    fec                       => J + 1,  -- FEC5 = 1 FEC12 = 2
    c_multicyledelay          => g_UPLINK_MULTICYCLE_DELAY, 4
    c_clockratio              => g_UPLINK_CLOCK_RATIO,8
    c_mgtwordwidth            => g_UPLINK_WORD_WIDTH, 32
    c_allowedfalseheader      => g_UPLINK_ALLOWED_FALSE_HEADER, 5
    c_allowedfalseheaderovern => g_UPLINK_ALLOWED_FALSE_HEADER_OVERN, 64
    c_requiredtrueheader      => g_UPLINK_REQUIRED_TRUE_HEADER, 30
    c_bitslip_mindly          => g_UPLINK_BITSLIP_MINDLY, 1
    c_bitslip_waitdly         => g_UPLINK_BITSLIP_WAITDLY 40
    )

  port map (
    uplinkclk_i         => uplink_clk,
    uplinkrst_n_i       => uplink_reset_n,
    mgt_word_i          => mgt_data,
    bypassinterleaver_i => g_LPGBT_BYPASS_INTERLEAVER,
    bypassfecencoder_i  => g_LPGBT_BYPASS_FEC,
    bypassscrambler_i   => g_LPGBT_BYPASS_SCRAMBLER,

    uplinkclkouten_o           => fec_mux_uplink_data(J).valid,
    userdata_o(223 downto 0)   => fec_mux_uplink_data(J).data,
    userdata_o(229 downto 224) => fec_mux_unused_bits(J * 6 + 5 downto J * 6),
    ecdata_o                   => fec_mux_uplink_data(J).ec,  --external control
    icdata_o                   => fec_mux_uplink_data(J).ic,  --internal control
    mgt_bitslipctrl_o          => fec_mux_bitslip(J),
    datacorrected_o            => fec_mux_datacorrected(J * 230 + 229 downto J * 230),
    iccorrected_o              => fec_mux_iccorrected(J * 2 + 1 downto J * 2),
    eccorrected_o              => fec_mux_eccorrected(J * 2 + 1 downto J * 2),
    rdy_o                      => fec_mux_uplink_ready(J)
    );
end generate;
```

Serenity

```
lpgbtfpga_uplink_inst : entity lpgbt_lib.lpgbtfpga_uplink
  GENERIC MAP(
      -- General configuration
      DATARATE                  => DATARATE,
      FEC                       => FEC,

      -- Expert parameters
      c_multicyleDelay          => 3,
      c_clockRatio              => 8,
      c_mgtWordWidth            => MGT_WORD_WIDTH, 32
      c_allowedFalseHeader      => 5,
      c_allowedFalseHeaderOverN => 64,
      c_requiredTrueHeader      => 30,
      c_bitslip_mindly          => 2,
      c_bitslip_waitdly         => 40
  )
  PORT MAP(
      -- Clock and reset
      uplinkClk_i               => mgt_rx_clk_s,
      uplinkClkOutEn_o          => uplinkStrobe_s,
      uplinkRst_n_i             => mgt_rx_rdy_s,

      -- Input
      mgt_word_i                => uplink_frame_from_mgt_i,

      -- Data
      userData_o                => lpgbtfpga_uplink_data_s(229 downto 0),
      EcData_o                  => lpgbtfpga_uplink_data_s(231 downto 230),
      IcData_o                  => lpgbtfpga_uplink_data_s(233 downto 232),

      -- Control
      bypassInterleaver_i       => uplink_bypass_interleaver_i,
      bypassFECEncoder_i        => uplink_bypass_fec_encoder_i,
      bypassScrambler_i         => uplink_bypass_scrambler_i,

      -- Transceiver control
      mgt_bitslipCtrl_o         => mgt_rx_slide_s,

      -- Status
      dataCorrected_o           => lpgbtfpga_uplink_user_data_corrected_s,
      IcCorrected_o             => lpgbtfpga_uplink_ic_data_corrected_s,
      EcCorrected_o             => lpgbtfpga_uplink_ec_data_corrected_s,
      rdy_o                     => lpgbtfpga_uplink_rdy_s,
      frameAlignerEven_o        => open
  );
```

LPGBT Downlink

KCU105

```
downlink_inst : entity lpgbt_fpga.lpgbtfpga_downlink

  generic map (
    c_multicyleDelay => g_DOWNLINK_MULTICYCLE_DELAY, 4
    c_clockRatio     => g_DOWNLINK_CLOCK_RATIO,  8
    c_outputWidth    => g_DOWNLINK_WORD_WIDTH   32
    )
  port map (
    clk_i               => downlink_clk,
    rst_n_i             => downlink_reset_n,
    clken_i             => downlink_data.valid,
    userdata_i          => downlink_data.data,
    ecdata_i            => downlink_data.ec,
    icdata_i            => downlink_data.ic,
    mgt_word_o          => mgt_data,
    interleaverbypass_i => g_LPGBT_BYPASS_INTERLEAVER,
    encoderbypass_i     => g_LPGBT_BYPASS_FEC,
    scramblerbypass_i   => g_LPGBT_BYPASS_SCRAMBLER,
    rdy_o               => downlink_ready(I)
    );
```

Serenity

```
lpgbtfptga_downlink_inst : ENTITY lpgbt_lib.lpgbtfpga_downlink
  GENERIC map(
      -- Expert parameters
      c_multicyleDelay          => 3,                    --! Multicycle delay: USEd
      c_clockRatio              => 8,                    --! Clock ratio is clock_d
      c_outputWidth             => MGT_WORD_WIDTH 32     --! Transceiver's word siz
  )
  PORT map(
      -- Clocks
      clk_i                     => mgt_tx_clk_s,         --! Downlink datapath cloc
      clkEn_i                   => downlinkStrobe_s,     --! Clock enable (1 over 8
      rst_n_i                   => cdc_tx_ready_s,       --! Downlink reset SIGNAL

      -- Down link
      UserData_i                => downlinkData_s(31 downto 0),
      ECData_i                  => downlinkData_s(33 downto 32),
      ICData_i                  => downlinkData_s(35 downto 34),

      -- Output
      mgt_word_o                => downlink_frame_to_mgt_o,    --! Downlink encoded fr

      -- Configuration
      interleaverBypass_i       => downlink_bypass_interleaver_i,     --! Bypass downl
      encoderBypass_i           => downlink_bypass_fec_encoder_i,     --! Bypass downl
      scramblerBypass_i         => downlink_bypass_scrambler_i,       --! Bypass downl

      -- Status
      rdy_o                     => lpgbtfpga_downlink_rdy_o           --! Downlink rea
  );
```

Thoughts and Next steps

- Firmware on initial inspections seems to have correct FEC mode and data rates for uplink/downlink

Future tests:

1. Reflash serenity with FEC12 firmware and probe VTRX+ with an oscilloscope to confirm if data is being outputted
2. Find MGT declarations in firmware
    - Check MGT rates of 2.56 Gbps for downlink + 10.24 Gbps uplink
    - Flash serenity with single MGT
3. Write C++/Uhal code that directly talks to the firmware to see if we can configure and set a GPIO pin on the lpGBT high/low (check with scope) without the MTD software to isolate this problem as either a firmware or software problem

Thank you!

Let us know your thoughts on the best way forward :)

Contact
ngonzalz@bu.edu
hayden.swanson@cern.ch