Update record

20150518: Figures are updated; section 5 & 6 are rewritten.
20150520: The FSM for automatically Rx alignment is added, some place are revised.

GBT Module for the FELIX Project

Kai Chen, Hucheng Chen, Francesco Lanni
Brookhaven National Laboratory

Updated @ May 31, 2015

Contents

1 Introduction

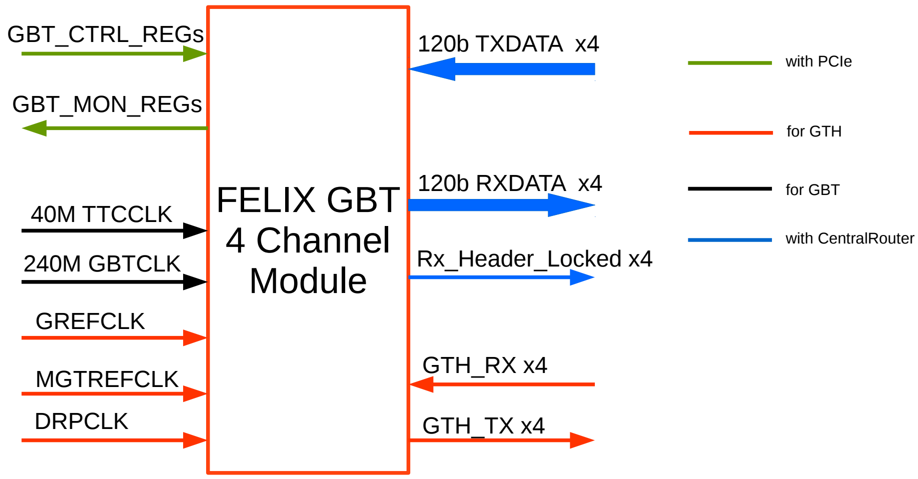

Figure 1: The input/output of the FELIX GBT module (4.8 Gbps version)

There are 2 CXP transceiver modules on the HTG-710 board, each CXP has 12 lanes. he FELIX GBT module is part of the FELIX firmware, it will encode the data from the central router, with normal GBT FEC mode, or Wide-Bus mode, or the FELIX full mode. The encoded GBT data are sent out through the CXP module. The GBT module also decodes the data from the CXP module, and sends them to the central router. The link speed for FEC and Wide-Bus modes are 4.8 Gbps, it's 9.6 Gbps for the full mode. The FELIX GBT module uses the GTH quad (4 channels) as the unit. The control and monitor of the GBT module is realized via the PCIe interface.

The input/output of the GBT module are listed in Figure 1. The reference clock for the GTH can be chosen from GRefClk or MGTRefClk. For the FELIX demonstrator, the GRefClk is used. The 240M GBTClk is one possible source for the GBT encoding and decoding, another selectable source is the TxOutClk and RxOouClk of GTH core. The GRefClk can be used to drive the 240M GBTClk. We can also separate GBTClk to be GBTRxClk and GBTTxClk, and use them for the Tx and Rx side separately, and we can adjust their phase separately. The details about the GBT module and clock distribution will be introduced later.

2 Register for the FELIX GBT module

Table 1: Registers for the GBT module

Regs Array	No.	Definition	Direction	Default Value	Temp Addr
GBT	0	LogicRst	W	ALL 0	0x2400
CTRL	1	Ctrl_Reg1	W	ALL 0	0x2410
Registers	2	Mode_Sel	W	ALL 0	0x2420
	...	...	...	...	
	7	Reserved	W	ALL 0	0x2470
GBT	8	RxSlide	W	ALL 0	0x2480
CTRL	9	TxUsrRdy, RxUsrRdy	W	0x0FFF0FFF0FFF0FFF	0x2490

Continued on next page

Table 1: Registers for the GBT module

Regs Array	No.	Definition	Direction	Default Value	Temp Addr
Registers	A	GTTxRst, GTRxRst, SoftRst	W	ALL 0	0x24A0
	B	CPLL/QPLL Reset, SoftTxRst	W	ALL 0	0x24B0
	C	SoftRxRst	W	ALL 0	0x24C0
	D	Odd/Even, Top/Bot	R/W	ALL 0	0x24D0
	E	DataSel	W	ALL 0	0x24E0
	F	Reserved	W	ALL 0	0x24F0
GBT CTRL Registers	10	CH[11-0] Tx Latency Config.	W	0x0000000000000000	0x2500
	11	CXP[23-12] Tx Latency Config.	W	0x0000000000000000	0x2510
	12	CXP[11-0] Rx Latency Config.	W	0x0000000000000000	0x2520
	13	CXP[23-12] Rx Latency Config.	W	0x0000000000000000	0x2530
	14	Tx Data Format	W	ALL 0	0x2540
	15	Rx Data Format	W	ALL 0	0x2550
	16	GBTTxRst, GBTRxRst	W	ALL 0	0x2560
	17	TxTimeDomainCrossing, OutSel	W	ALL 0	0x2570
	18	Reserved	W	ALL 0	0x2580
	...	...	...	...	
	1F	Reserved	W	ALL 0	0x25F0
GBT MON Registers	0	GBT_Version	R	depends on version	0x2600
	1	Reserved	R		0x2610
	...	...		...	
	7	Reserved	R		0x2670
GBT MON Registers	8	TxRstDone, RxRstDone	R		0x2680
	9	TxFsmRstDone, RxFsmRstDone	R		0x2690
	A	CPLLLock, CPLLFbClkLost	R		0x26A0
	B	RxCdrLock, QPLLLock	R		0x26B0
	C	RxOutClk_Sampled	R		0x26C0
	...	...	...	...	
	F	Reserved	R		0x26F0
GBT MON Registers	10	RxIsHeader	R		0x2700
	11	RxIsData, RxDataValid	R		0x2710
	12	RxAlignmentDone, OutSelCalc	R		0x2720
	13	TopBot_ByFSM,RxDecoderErr	R		0x2730
	...	...	...	...	
	1C	ErrCntGrp A	R		0x27C0
	1D	ErrCntGrp B	R		0x27D0
	1E	ErrCntGrp C	R		0x27E0
	1F	ErrCntGrp D	R		0x27F0

- For control purpose: GBT_CTRL_REG
  - REG_0: LogicRst (0x0), Default Value: All 0
    - reserved for reset of different levels of logic blocks
  - REG_1: Ctrl_Reg1 (0x1), Default Value: All 0
    - bit(0): Rx Alignment Check Reset

    * bit(1): Error check counters reset

- REG_2: Mode_Sel (0x2), Default Value: All 0
    * bit(0): Mode for the descrambler output multiplexer. When '0', use the calculated Sel by firmware (directly use the OutSelCalc of 0x2720 to control OutSel of 0x2570). When '1', use the Sel value written by software via 0x2570.
    * bit(1): Rx Alignment mode sel. When '0', use the FSM of firmware to do it automatically. When '1', use software to do it.
    * bit(2): Control where is the topbot value from. Only when this bit is '0' and Rx Alignment mode sel is '0', the topbot value generated by FSM is used. For others, the topbot value from software (0x24D0) is used.
- REG_8 (0x8), Default Value: All 0
    * bit(11-0): Manual RxSlide for the CH[11-0]
    * bit(27-16): Manual RxSlide for the CH[23-12]
    * bit(43-32): Reserved
    * bit(59-48): Reserved
- REG_9 (0x9), Default Value: All 1 for the used bit
    * bit(11-0): TxUsrRdy for the CH[11-0]
    * bit(27-16): TxUsrRdy for the CH[23-12]
    * bit(43-32): RxUsrRdy for CH[11-0]
    * bit(59-48): RxUsrRdy for CH[23-12]
- REG_A (0xA), Default Value: All 0
    * bit(11-0): GTH Tx reset for the CH[11-0]
    * bit(27-16): GTH Tx reset for the CH[23-12]
    * bit(43-32): GTH Rx reset for CH[11-0]
    * bit(59-48): GTH Rx reset for CH[23-12]
    * bit(14-12): Soft Reset for CH[11-0] (Tx & Rx for the GTH Quads)
    * bit(30-28): Soft Reset for CH[23-12] (Tx & Rx for the GTH Quads)
- REG_B (0xB), Default Value: All 0
    * bit(11-0): CPLL reset for the CH[11-0] by Channel
    * bit(27-16): CPLL reset for the CH[23-12] by Channel
    * bit(14-12): QPLL Reset for CH[11-0] by Quad
    * bit(30-28): QPLL Reset for CH[23-12] by Quad
    * bit(43-32): Soft TxReset for the CH[11-0] by channel
    * bit(59-48): Soft TxReset for the CH[23-12] by channel
    * bit(46-44): Soft TxReset for CH[11-0] by Quad
    * bit(62-60): Soft TxReset for CH[23-12] by Quad
- REG_C (0xC), Default Value: All 0
    * bit(43-32): Soft RxReset for the CH[11-0] by channel
    * bit(59-48): Soft RxReset for the CH[23-12] by channel
    * bit(46-44): Soft RxReset for CH[11-0] by Quad
    * bit(62-60): Soft RxReset for CH[23-12] by Quads

- REG_D (0xD), Default Value: All 0
  - bit(11-0): CH[11-0] Odd/Even (0: Even, 1: Odd): used by the data alignment
  - bit(27-16): CH[23-12] Odd/Even (0: Even, 1: Odd): used by the data alignment
  - bit(59-48): CH[23-12] Top/Bot (0: Top, 1: Bot): used by the data alignment
  - bit(43-42): CH[11-0] Top/Bot (0: Top, 1: Bot): used by the data alignment
- REG_E (0xE), Default Value: All 0
  - bit(47-0): CH[11-0] multiplexer selection: used by the data alignment, 4 bit per channel, reserved
- REG_F (0xF), Default Value: All 0
  - bit(47-0): CH[23-12] multiplexer selection: used by the data alignment, 4 bit per channel, reserved
- REG_10 (0x10), Tx Latency Optimization Configuration
  - bit(23-0): Tx Time Domain Crossing Latency Configuration, defalut 0x000
  - bit(47-24): Tx Scrambler Latency Configuration, default 0x000
- REG_11 (0x11), Tx Latency Optimization Configuration
  - reserved
- REG_12 (0x12), Rx Latency Optimization Configuration
  - bit(23-0): Rx Descrambler Latency Configuration, defalut 0x000
  - bit(47-24): reserved
- REG_13 (0x13), Rx Latency Optimization Configuration
  - reserved
- REG_14 (0x14), GBT Tx Data Format, default 0x0000000000000000
  - bit(23-0): CH[11-0], 2 bit per channel. 00: FEC, 01: Wide-Bus
  - bit(55-32): CH[23-12], 2 bit per channel. 00: FEC, 01: Wide-Bus
- REG_15 (0x15), GBT Rx Data Format, default 0x0000000000000000
  - bit(23-0): CH[11-0], 2 bit per channel. 00: FEC, 01: Wide-Bus
  - bit(55-32): CH[23-12], 2 bit per channel. 00: FEC, 01: Wide-Bus
- REG_16 (0x16), default: All 0
  - bit(11-0): GBT TxReset for CH[11-0]
  - bit(27-16): GBT TxReset for CH[23-12]
  - bit(43-32): GBT RxReset for CH[11-0]
  - bit(59-48): GBT RxReset for CH[23-12]
- REG_17 (0x16), default: All 0
  - bit(11-0): Tx Time Domain Crossing selection for CH[11-0]. 0: align one time; 1: continuous align.
  - bit(27-16): Tx Time Domain Crossing selection for CH[23-12]
  - bit(43-32): Descrambler output selection for CH[11-0]
  - bit(59-48): Descrambler output selection for CH[23-12]

- For monitor purpose: GBT_MON_REG

- REG_0: GBT_Version (0x0), Default Value: depends on the GBT logic version
  * bit(0): PLL_SEL
  * bit(1): RX_CLK_SEL
  * bit(2): GTHREFCLK_SEL
  * bit(15-3): reserved
  * bit(31-16): GTH IP version
  * bit(47-32): GBT version
  * bit(63-48): Date
- REG_8 (0x8), Right Value: 0x0FFF0FFF0FFF0FFF
  * bit(11-0): TxResetDone for the CH[11-0]
  * bit(27-16): TxResetDone for the CH[23-12]
  * bit(43-32): RxResetDone for CH[11-0]
  * bit(59-48): RxResetDone for CH[23-12]
- REG_9 (0x9), Right Value: 0x0FFF0FFF0FFF0FFF, or 0x000000000FFF0FFF
  * bit(11-0): TxFsmResetDone for the CH[11-0]
  * bit(27-16): TxFsmResetDone for the CH[23-12]
  * bit(43-32): RxFsmResetDone for CH[11-0]
  * bit(59-48): RxFsmResetDone for CH[23-12]
- REG_A (0xA), Right Value: 0x0FFF0FFF00000000
  * bit(11-0): CPLLFbClkLost for the CH[11-0]
  * bit(27-16): CPLLFbClkLost for the CH[23-12]
  * bit(43-32): CPLLLock for CH[11-0]
  * bit(59-48): CPLLLock for CH[23-12]
- REG_B (0xB) , Right Value: 0xFFF0FFF0FFF0FFF
  * bit(11-0): RxCdrLock for the CH[11-0]
  * bit(14-12): QPLLLock for the CH[11-0] by Quad
  * bit(27-16): RxCdrLock for CH[23-12]
  * bit(30-28): QPLLLock for CH[23-12] by Quad
- REG_C (0xC) , Value will change during configuration
  * bit(11-0): The sampled RxOutClk, for CH[11-0]
  * bit(27-16): The sampled RxOutClk, for CH[23-12]
- REG_10 (0x10). Reserved.
  * bit(2-0): CH[11-0] QPLL locked. Reserved.
  * bit(18-16): CH[23-12] QPLL locked. Reserved.
  * bit(15 downto 4): CH[11-0] CPLL locked. Reserved.
  * bit(31 downto 20): CH[23-12] CPLL locked. Reserved.
  * bit(43 downto 32): CH[11-0] Rx Word Is Header. Reserved
  * bit(59 downto 48): CH[23-12] Rx Word Is Header. Reserved.
- REG_11 (0x11)
  * bit(11 downto 0): CH[11-0] Rx_Is_Data. 1: Data, 0: Idle.

    * bit(27 downto 16): CH[23-12] Rx_Is_Data. 1: Data, 0: Idle.
    * bit(43 downto 32): CH[11-0] Rx_Data_Valid. Keep 1 when link is stable.
    * bit(59 downto 48): CH[23-12] Rx_Data_Valid. Keep 1 when link is stable.
  - REG_12 (0x12)
    * bit(11 downto 0): CH[11-0] Rx Alignment is Done.
    * bit(27 downto 16): CH[23-12] Rx Alignment is Done.
    * bit(43 downto 32): CH[11-0] Out Sel recommended value.
    * bit(59 downto 48): CH[23-12] Out Sel recommended value.
  - REG_13 (0x13)
    * bit(11 downto 0): CH[11-0] Rx FEC decoder error flag.
    * bit(27 downto 16): CH[23-12] Rx FEC decoder error flag.
    * bit(43 downto 32): CH[11-0] The TopBot value from FSM.
    * bit(59 downto 48): CH[23-12] The TopBot value from FSM.
  - REG_1C (0x1C): For debugging
    * bit(9 downto 0): GBT Channel 0 error number
    * bit(19 downto 10): GBT Channel 1 error number
    * bit(29 downto 20): GBT Channel 2 error number
    * bit(39 downto 30): GBT Channel 3 error number
    * bit(49 downto 40): GBT Channel 4 error number
    * bit(59 downto 50): GBT Channel 5 error number
  - REG_1D (0x1D): For debugging
    * bit(9 downto 0): GBT Channel 6 error number
    * bit(19 downto 10): GBT Channel 7 error number
    * bit(29 downto 20): GBT Channel 8 error number
    * bit(39 downto 30): GBT Channel 9 error number
    * bit(49 downto 40): GBT Channel 10 error number
    * bit(59 downto 50): GBT Channel 11 error number
  - REG_1E (0x1E): For debugging
    * bit(9 downto 0): GBT Channel 0 error number
    * bit(19 downto 10): GBT Channel 12 error number
    * bit(29 downto 20): GBT Channel 13 error number
    * bit(39 downto 30): GBT Channel 14 error number
    * bit(49 downto 40): GBT Channel 15 error number
    * bit(59 downto 50): GBT Channel 16 error number
  - REG_1F (0x1F): For debugging
    * bit(9 downto 0): GBT Channel 0 error number
    * bit(19 downto 10): GBT Channel 17 error number
    * bit(29 downto 20): GBT Channel 18 error number
    * bit(39 downto 30): GBT Channel 19 error number
    * bit(49 downto 40): GBT Channel 20 error number
    * bit(59 downto 50): GBT Channel 21 error number

3 Interface with CentralRouter

- From GBT to CentralRouter
  - x24 FRAME_LOCKED_O: means the header "0101" or "0110" is obtained for the 120bit frame data.
  - x24 120bit GBT data
    - FEC mode
      - bit(111-32): x5 e-Link
      - bit(115-112): IC & EC
      - bit(119-116): Header
      - bit(31-0): not used
    - Wide_Bus mode (means the Front-End GBTx is Wide_Bus mode)
      - bit(111-32): x5 e-Link
      - bit(115-112): IC & EC
      - bit(119-116): Header
      - bit(31-0): x2 e-Link
- From CentralRouter to GBT
  - x24 120bit GBT data
    - FEC mode
      - bit(111-32): x5 e-Link
      - bit(115-112): IC & EC
      - bit(119-116): Header
      - bit(31-0): not used, can be connected to '0'
    - Wide_Bus mode (Table 8 of GBTx manual)
      - bit(111-64): x3 e-Link
      - bit(63-0): not used, can be connected to '0'
      - bit(115-112): IC & EC
      - bit(119-116): Header

4 Configuration of the FELIX GBT package file

The package file is *FELIX_gbt_pckage.vhd*.

- GBT_VERSION: used to indetify the GBT Module version, and the GTH Core version
- GTHREFCLK_SEL: choose the source of the GTH reference clock
  - '0': use the MGTREFCLK
  - '1': use GREFCLK
- LINERATE: The TX_LINERATE and RX_LINERATE are reserved. Only 4.8G is supported now.

- WORD_WIDTH: only support 20 now, 20 means the GTH data width is 20. For the 4.8G version, the UsrClk is 240 MHz, so the data width is 4800/240.
- DYNAMIC_DATA_MODE_EN:
  - This function is not supported by the CERN GBT-FPGA. It added for a more convenient use.
  - '0': disable the dynamic GBT mode change. The GBT_DATA_TXFORMAT_PACKAGE and GBT_DATA_RXFORMAT_PACKAGE are used to set the default GBT mode for Tx and Rx of each channel. 2 bit are used for each channel. "00" means normal FEC mode, "01" means Wide-Bus mode.
  - '1': enable the dynamic GBT mode change. The 0x2540 and 0x2550 are used to configure the mode for Tx and Rx of each channel.
- DATA_MODE: when we disable the dynamic encoding mode change, this is used to choose the encoding mode. It configure all the 24 channels, the channel by chennel configuration will be added in the future.
- DYNAMIC_LATENCY_OPT: enable or disable the dynamic change of latency optimization register.
  - When it is '0', the TX_LATOPT_SCRAMBLER, TX_LATOPT_TIMECROSSING, and RX_LATOPT_DES can be used to set the latency optimization.
  - When it is '1', registers 0x2500, 0x2510, 0x2520, 0x2530 are used to configure the register dynamically.
- PLL_SEL: choose to use the QPLL or CPLL for the GTH. When GTHREFCLK_SEL is from GREFCLK, QPLL should be used.
- RX_CLK_SEL: choose the RXUSRCLK source, LOCAL_GBTRXCLK is advised.
  - MASTER_RXOUTCLK: use RxOutClk of the master channel to drive the 4 RxUsrClk of the GTH quad. It also use as the GBT Decoding blocks: RxGearBox and Descrambler.
  - LOCAL_GBTRXCLK: use external 240MHz clock to drive the 4 RxUsrClk. The 4 channels are equal, none of them is master.
- TX_CLK_SEL: choose the TXUSRCLK source, TxOutClk is advised.
  - MASTER_TXOUTCLK: use TxOutClk of the master channel to drive the 4 TxUsrClk of the GTH quad. It also used as the GBT Encoding blocks: Scrambler and TxGearBox.
  - LOCAL_GBTTXClk: use external 240MHz clock to drive the 4 RxUsrClk, and GBT encoding blocks. One advantage is we can adjust the phase between this external clock and the 40M clock. To make the time domain crossing from this 40M to this 240M has a fixed latency.
- RX_DESCR_MUX_EN: enable the multiplexer for the descrambler output data. Before the data are sent to the 40MHz clock domain.
- SAME_LAT_FOR_WB_FEC: when it is '1', for Wide-Bus mode, the descrambler start will be delayed by an extra 2 cycles to make it same with FEC mode.
- TX_TC_DYNAMIC_EN: back up. Enable the dynamic change of method for the time domain crossing in scrambler

- TX_SCRAMBLER_SEL: back up. Choose the method for the time domain crossing in scrambler when dynamic configuration is disabled.
  - 0: GBTTXRESET trigger the one-time alignment
  - 1: Continuous alignment
- RXGEARBOX_MODE: back up, reserved for the optimization test of RxGearBox.
- DECODER_MODE: choose the mode for the GBT decoder.
  - SYNC: use pipeline synchronized logic to realize the FEC decoder. Not finished yet.
  - COMB: use the default combinational logic.
- GF_DELAY3: backup for test purpose. Extra GBT-FRAME decoder latency configuration. '1' is advised.
- TX_TEST_CLK: reserved for test purpose.
- PHASE_ADJUST: this is for the debugging, it will shift the phase at the TX side. Different phase is choosed for different channels.

5 Details about the FELIX GBT module

5.1 Organization of the FELIX GBT core

The block diagram of the GBT module is shown as Figure 2. It contains two parts: the GBT IP, and the GTH IP.

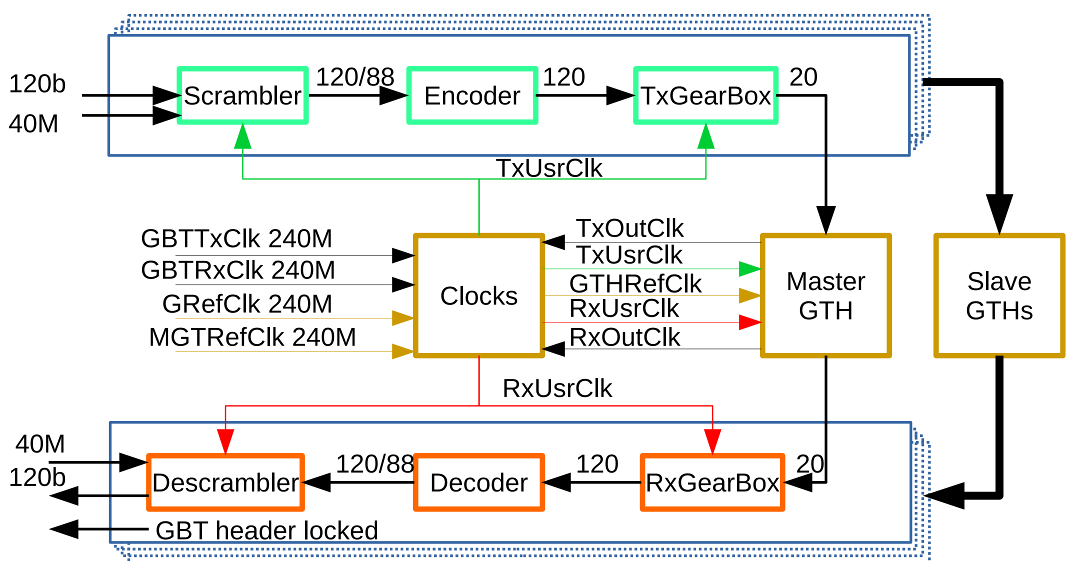

Figure 2: The block diagram of the FELIX 4-channel GBT module

The GTH IP is generated by Xilinx VIVADO. Some modifications are done:

- The low-level GTH configuration file is modified to decrease the latency. For example, RXCOMMADETEN is disabled.
- The TxOutClk of the master channel or external 240 MHz clock can be used to drive the TxUsrClk.
- The RxOutClk of the master channel or external 240 MHz clock can be used to drive the RxUsrClk.
- The bitslip is needed to do with the GTH output data. When the bitslip is finished, a stable GBT frame header is locked. At this time the phase between RxOutClk and RXUSRCLK is uncertain. A multiplexer is added to the GBT RxGearBox, to solve the time domain crossing problem for the 20 bit data transferred from RxOutClk to RxUsrClk in the GTH hard core.
- The GTH IP based on both of CPLL and QPLL are provided.

The FELIX GBT are based on the CERN GBT-FPGA verison 3.0.2. The main modifications to it are listed below:

- The GTX IP is removed, to make the GBT has nothing to do with the transceiver.
- The dynamic change of the GBT mode between normal FEC mode and Wide-Bus mode is provided.
- A simpler RxGearBox with Rx alignment function are designed to replace the RxGearBox and framealigner provided by CERN GBT-FPGA. The alignment operation can be automatically done by firmware or controlled by software.
- The scrambler and descrambler are changed from 40 MHz clock domain to 240 MHz domain. Enable signal are added to control the descrambler and scrambler.
- The new time domain crossing between 40 MHz and 240 MHz are added in scrambler and descrambler.

5.2 The clock distribution in the FELIX GBT module

The clock distribution for the GBT and GTH in the FELIX demonstrator is shown in Figure 3:

- The GTHRefClk can be from GRefClk or MGTRefClk. For the FELIX demonstrator, the GRefClk is selected.
- The RxUsrClk is used by GTH, descrambler and RxGearBox. It can be from the RxOutClk of the master channel, or from the GBTRxClk. The RxOutClk has a bad quality, the jitter is several hundred picoseconds.
- The TxUsrClk is used by GTH, scrambler, and TxGearBox. It can be from TxOutClk of the master channel, or from the GBTTxClk. We use TxOutClk now for the previous testing.
- We can directly use the GRefClk to drive the GBTTxClk and GBTRxClk when using the GBT module, but when they are from different sources, the phase of them can be adjusted separately.

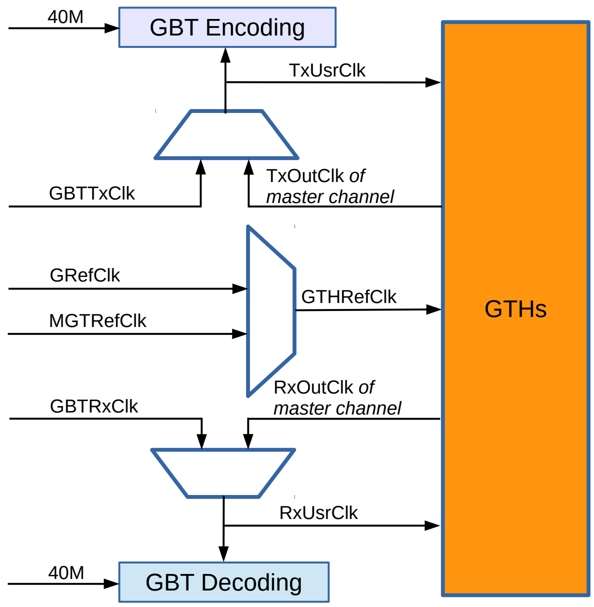

Figure 3: The clock distribution for the FELIX GBT module

5.3 The latency of the FELIX GBT module

To benchmark the latency, we need to define some time unit.

- The UI (Unit Interval) is 1/4.8GHz=208.3 ps.
- The Cycle is 1/240MHz=4.167 ns.
- The BC is about 1/40MHz=25 ns.

We also need to define the start point and stop point for the latency test. The definition of the latency we use is different with the measurement did by CERN GBT-FPGA project.

For the FELIX GBT Tx side, the latency are split to two parts: the delay of the GBT encoding and the delay of the GTH transmitter. Figure 4 shows the definition of the GBT encoding part:

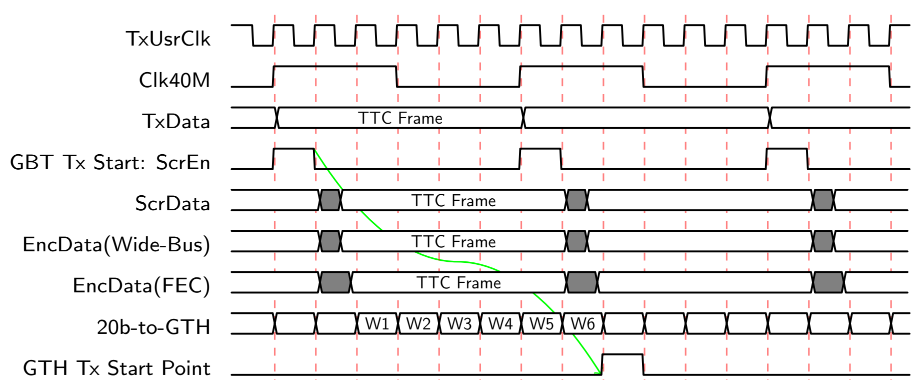

Figure 4: The latency definition for the GBT Tx side

The start point of the GBT encoding is the falling edge of the Scrambler enable pulse. At this point, the scrambler starts to work. The end point of the GBT encoding is the rising edge of the TxUsrClk which samples the last 20 bit word of this TTC frame. It's also the start point of the GTH transmitter. The GTH transmitter end point is when the first serial bit of the last word W6 is transmitted out from the FPGA GTH transmitter.

For the GBT encoding, the scrambler has a fast speed. For the FEC mode, the FEC encoder will spend some time. But the 32 bit generated by the FEC is in the words W5 and W6, so we can start to transfer W1 just one Cycle after the scrambler is enabled, even if the FEC encoder needs more Cycles to finish.

The calculation for the latency of the FELIX GBT Tx side is:

- The GBT Tx encoding: 7 Cycle: 29.167 ns.
- The GTH transmitter: 4 Cycle + 33.73125 UI = 23.7 ns.
- Totally: 52.9 ns.

For the FELIX GBT Rx side, the latency are also split to two parts: the delay of the GTH receiver and the delay of the GBT decoding. Figure 5 shows the definition of the GBT decoding part:

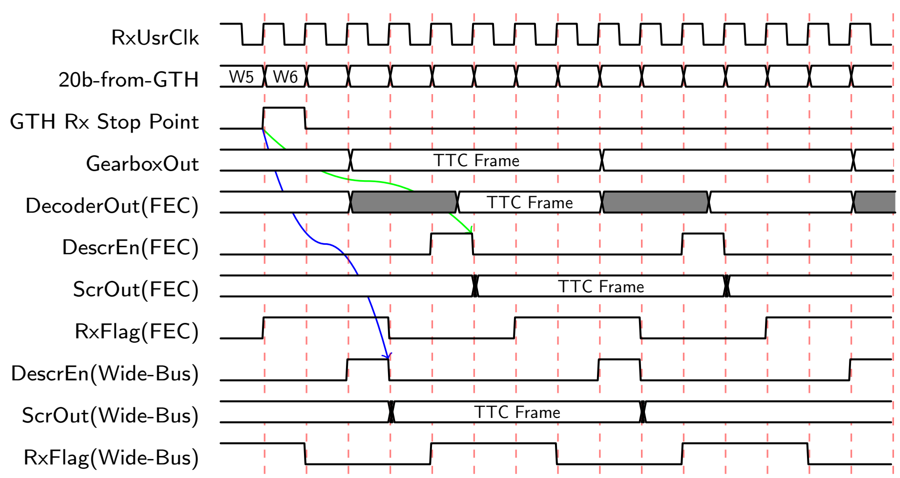

Figure 5: The latency definition for the GBT Rx side

The GTH Rx start point is when the first serial bit of W6 is received by the FPGA GTH receiver. The start point of the GBT decoding is the rising edge when the last 20-bit W6 are update on the 20-bit output port of GTH. The end point of the GBT RX is the falling edge of the descrambler enable pulse.

The calculation for latency of the FELIX GBT Rx side is:

- The GTH receiver: 3 Cycle + 123 UI = 38.125 ns.
- The GBT Rx decoding (Wide-Bus): 3 Cycles: 12.5 ns.
- The GBT Rx decoding (FEC): 5 Cycles: 20.833 ns.
- Totally: 59 ns for FEC, 50.7 ns for Wide-Bus.

For the loopback, total latency is about 112 ns for FEC mode, and 104 ns for Wide-Bus mode. Our tests use the lowest bit of TxData as start test point, and use the lowest bit of descrambler output as stop test point. The loopback latency is about 108 ns for the Wide-Bus mode, and 116 ns for the FEC mode. They are 4 ns bigger than the calculation above, because the TxData is updated before the scrambler start point for about one cycle. The scrambler enable signal can also be configured to be delayed by one cycle, to give more setup time for the data transfer from 40 MHz to 240 MHz domain. This increases the latency for the time domain crossing, but make it more robust when the update time of the TxData is later than the 40 MHz's rising edge.

The FEC mode needs two cycles more than the Wide-Bus mode to do the FEC decoding. We can also delay the Wide-Bus mode with two cycles, to make them have the same latency. It can be configured in the GBT package file.

5.4 The time domain crossing in the FELIX GBT module

There are several time domain crossing in the FELIX GBT module. The setup time and hold time should be guaranteed for the data transferred between them.

5.4.1 The scrambler input data from 40 MHz to TxUsrClk

As shown in Figure 4, the TxUsrClk will sample the 40 MHz clock, and generate the scrambler enable signal according to the sampling results. We have two methods to generate the pulse.

a. When GBTTxReset is given, we generate scrambler enable pulse based on the sampled result. When the GBTTxReset is cleared, we simply keep giving the pulse every 6 Cycles.

b. The GBTTxReset is not used, we use TxUsrClk to sample the 40 MHz, and keep generating the scrambler enable pulse by the sampling results.

For the method b, to assure the interval of the generated pulse is even, the sampled point can not be close to the edges of the 40 MHz. If this requirement is not guaranteed, errors may happen. For method a, no error will happen, but the latency may change by 1 Cycle, when we do the GBTTxReset.

Both of the two methods are supported, we can change it dynamically via PCIe interface. Method b is recommended, since we can adjust the phase between the 40 MHz and the TxUsrClk, to make no errors happen, and make the latency is fixed. We can also configure the Tx latency optimization register, to decide how many Cycles we wait after the 40 MHz rising edge, to generate the scrambler enable pulse.

5.4.2 The GTH data transfer from RxOutClk to RxUsrClk

Inside the GTH, at first the 20 bit data is aligned to the RxOutClk for every channel. This data needs to be transfered to the RxUsrClk domain. For single channel project, we can use the RxOutClk to drive the RxUsrClk, there is no time domain crossing problem. For a project with big number channels, in order to save clock resources, we must share RxUsrClk for some channels. Two methods can be used to save the clock resources:

- Method 1: A quad share the RxUsrClk, it's driven by the RxOutClk of the master channel. This method is supported by the FELIX GBT module, we can change the package file to convert to this mode.
  Advantage of this method: we can use the link to recover the TTC clock, if without the TTCfx module. For FELIX, we don't need this function.
  Drawbacks of this method:
  - The master channel must always be connected, and keep working.
  - The quality of this RxUsrClk is bad, the jitter is several hundred picoseconds.
- Method 2: Use external GBTRxClk to drive RxUsrClk of all channels. For FELIX, all the clocks are synchronized, so we use it as default.

For all the channels of method 2 and slave channels of method 1, we need to solve the time domain crossing problem. Method 2 is used by FELIX demonstrator, so we will take it as an example, to explain the Rx alignment process, and how to solve the time domain crossing problem.

The RxGearBox of GBT will check the received 20b data, when the reversed GBT header is always at b3-b0 every 6 cycles, that means the GBT header is locked, and the Rx alignment is successful. Inside the GTH, we can use the RXSLIDE to do the bitslip of the 20b data, the step is 2b. When we shift 2b, the RxOutClk of this channel will also be shifted by 2 UI. So for a link, when we do the bitslip until the GBT header is locked, the RxOutClk will always has a same phase, this is why we can obtain a fixed latency.

But it's actually not fixed, because the GTH data is 4.8 Gbps, the CDR of GTH treats the data as DDR, which cause the phase of the recovered clock may change 1 UI when we do the Rx side reset. Meanwhile the 20b data will also has a 1b shift. Bitslip can only shift data with step of 2b, so the GBT frame header can be on b4-b1, or b2-b0 & b19 of the last 20b data. We name it Odd/Even problem for convenient. It results in a RxOutClk phase shift of 1 UI, and 20b data shift of 1b. If we enable the RXCOMMADETEN of the GTH channel, the 20b data can be shifted with step of 1b, but it increases the latency for 1/240M, and the RxOutClk still has 1 UI shift. We disable it, and do it outside the GTH, and solve the time domain crossing problem in the same time.

The external sourced RxUsrClk is synchronized with the TTC clock, and has a fixed phase. So the phase between the RxUsrClk and RxOutClk is also almost fixed, with 1 UI uncertainty. The problem is the phase between them are uncertain, any cable length difference may change it.

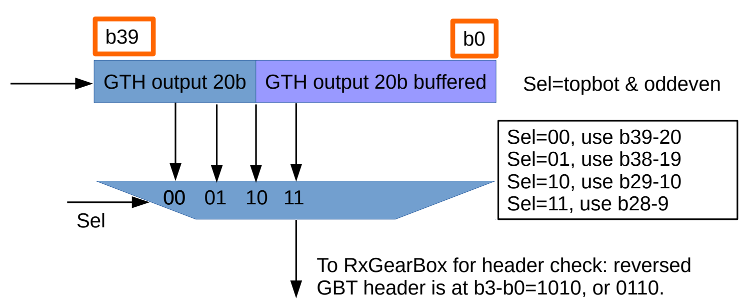

Figure 6: The multiplexer for RxGearBox

In order to make sure the data transferred from RxOutClk to RxUsrClk is right, we add a multiplexer in the GBT RxGearBox. It is shown in Figure 6. The GTH output data is buffered. We use the topbot and oddeven registers to choose the data to RxGearBox. The oddeven here is used to solve the Odd/Even problem, just do what GTH does when we enable the RXCOMMADETEN. It makes the bitslip step be 1. The oddeven here is controlled by the PCIe, but the value it should be is determined by the GTH itself, the GTH receiver reset may change the value. What we can do is to try with a 0 & 1 to check which one is needed to lock stable GBT header locked.

The topbot here is used to solve the time crossing problem. As introduced above, if the phase between RxUsrClk and RxOutClk is not good, the data transfer between them may has error. What we can do is to change the topbot value from 0 to 1, then 20b data to RxGearBox is shifted by 10b. In order to get the GBT header locked, extra bitslip is needed to shift the GTH output by 10b, meanwhile the RxOutClk will be shifted by 10 UI. Then the phase between RxUsrClk and RxOutClk is changed, to solve the time domain crossing problem.

- If both of the topbot=0, and topbot=1 has no good phase which locks a stable GBT header, then there must be some problems about the link.
- If only one has a good phase, then we should use it.
- If both of them have a good phase, then we need to choose one to use. What we do is: shift the phase of RxOutClk, use RxUsrClk to sample the RxOutClk. For the 20 UI, the shift step is 2, we can get a 10 bit result. From the result, we can know the phase between them. Based on this we can choose the topbot value which has a better phase for the time domain crossing.

As described above, due to the Odd/Even problem, phase of the RxOutClk may change 1 UI when we do the GTH receiver reset. This may change our selection to set topbot to be 0 or 1, when both of them have a good phase. It has the possibility to change the descrambler operation time by 1/240M. Because when the 1 UI change our selection, it's at one of the two boundary points, where both top=0 and topbot=1 have good phases. One method is to record the topbot value we used, and keep use the same value all the time, to get a fixed latency.

The Rx alignment process is automatically done by the firmware, but we can also use the software to do the whole process, or just to set the topbot value. When we record the topbot value to a database, we can use the topbot calculated by the firmware, which is register 0x2730. We can also use software to do the alignment, to obtain the topbot value.

5.4.3 The descrambler output data from RxUsrClk to the 40 MHz

The descrambler output data will be transferred to the 40 MHz clock domain. When the external GBTRxClk is used as the RxUsrClk, they can be phase aligned. Since the descrambler speed is much faster that 1/240M. So the 40 MHz clock can sample the descrambler output directly.

If the RxOutClk of the master channel is used as the RxUsrClk, the phase between RxUsrClk and the 40 MHz is uncertain. Shown in Figure 7, a multiplexer is added. One input is the descrambler output, the other input is buffered by the falling edge of the 40 MHz. As shown in Figure 5, the RxFlag has fixed relationship with descrambler enable signal. The 40 MHz samples the RxFlag, and use the output to select the multiplexer output.

Any uncertainty latency of the path from Front-End to the descrambler output may change which cycle the GBT decoding start, then may change the recommended Sel value. To obtain a fixed latency, we can

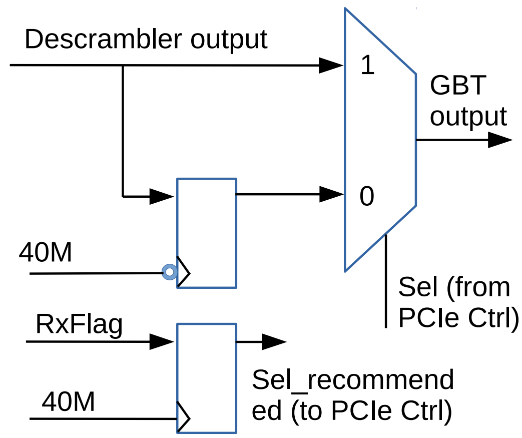

Figure 7: Multiplexer for the descrambler output

record the Sel value for each channel to a database, and keep use it to control the multiplexer. Both of the FEC mode and Wide-Bus modes should be recorded if they are configured to use different decoding time. If we record the value and use it, then the latency will be fixed, and it can tolerate an uncertainty of several nanoseconds for the whole data path before the descrambler. When GBTRxClk is used as RxUsrClk, we can also keep this multiplexer, use this method to obtain a fixed latency for the whole data path. The package file of GBT can be configured to enable or bypass this multiplexer, it's enabled default. The recommended Sel value is used default, but we can also use software to set the value.

5.4.4 Conclusion

- If for the FELIX Rx link, the fixed latency is not required, then we don't need to consider record the Sel value for the descrambler output multiplexer or topbot value for the multiplexer in the RxGearBox.
- If a fixed latency is required, then we only need to record the Sel value for the descrambler output. The topbot value's change has the possibility to results in a 1/240MHz latency change for the descrambler output, but it can be calibrated when we use a fixed Sel value for the descrambler output multiplexer.
- If the latency value is also not important, we can even use FIFO for the time domain crossing from RxOutClk to RxUsrClk, and from RxUsrClk to the 40 MHz clock.

6 FELIX GBT Configuration Steps

An example with Python script is in hostSoftware/felix_gbt_config. Attention should be paid to the sleep between some reset operations.

6.1 Initialization of the GBT

See gbt_config_top_tx.py. When the 240M and 40M clocks to GBT are stable, we can run it.

1. Set TxUsrRdy & RxUsrRdy, write 0x0FFF0FFF0FFF0FFF to register 0x2490.
2. SoftReset of the Quads. Bit[14-12] & Bit[28-26] of register 0x24A0 are for the softreset of the 6 Quads. We can do the reset to any Quads, write 0x70007000 followed by a 0x00000000 will reset all the 6 Quads.

6.2 Configuration of the GBT TX

The TX reset are also included in gbt_config_top_tx.py.

1. SoftTxReset for the Quads. Write 0x70007000 followed by 0x00000000 to 0x24B4 will reset all the Quads. See Section 2 for the details about the register definition.
2. Write 0x2540 to set the GBT Tx encoding mode for all the 24 channels.
3. Configure the Tx Latency Optimization register. Write to register 0x2500.
4. TX time domain crossing selection: Write register REG_TXTC_SEL (0x2570).
5. GBTTxRst. Write 0x0FFF0FFF followed by 0x00000000 to 0x2560 will do the GBT TX Reset to all the 24 channels. See Section 2 for the details about the register definition.

6.3 Configuration of the GBT RX

The GBT RX reset can be done by gbt_config_top_rx.py.

1. Configure the Rx Latency Optimization register. Write to register 0x2520.
2. GTHRxRst: Write 0x0FFF0FFF followed by 0x00000000 to 0x24A4 will do the GTH RX Reset to all the 24 channels. (We can also reset only the channel we want to configure.)
3. Write 0x2550 to set the GBT Rx encoding mode for all the 24 channels.
4. If the register 0x2420 is configured to use software to do the Rx alignment, then we need to do the Rx alignment for the channels we want to configure (see 6.4 for the Rx Alignment details). If it's configured to use the FSM in firmware, then we don't need to do anything.
5. Do the GBT Rx Reset for the channel we want to configure. Write 0x0FFF0FFF followed by 0x00000000 to 0x2464 will reset all the channels.
6. If the multiplexer for descrambler output is enabled, and the register 0x2420 is configured to control it by software, then we need to read the calculated recommended selection for the descrambler multiplexer, that is REG_OUTSEL_CALC (0x2724). We can use the recommended value, and write it to REG_OUT_SEL (0x2574).

   AS introduced in part 3 of section 5.4, if some latency in the whole data path from Front-End to FELIX change a little, the recommended value may change from 0 to 1, or from 1 to 0. This may make the latency change for 1 cycle of 25 ns. One method to avoid this is record the recommended value when the system run for the first time, and use it in the future. This will help us to get a fixed latency.

7. Check the alignment status again. Write 0x1 followed by 0x0 0x2410 (REG_ALIGN_CHK_RST). To reset the error flag. Wait for some time. Then readback the 0x2720 (REG_ALIGNMENT_DONE), check the bits for this channels.

8. When the GBT mode is FEC, we can also check the register 0x2730. It's the error status for the FEC decoding. If no error happens during the transmission for some channel, the bit for it should be '0'. If it's '1', means some bit are error, but the FEC decoder can recover the data depends on how many bits has error.

6.4 Rx Alignment

The RxUsrClk has two possible source, one is RxOutClk, another one is the local 240M. The Rx Alignment method for them are different. We use local 240M for the FELIX demonstrator, the steps are listed as below, see gbt_config.py.
We will use REG_TOPBOT and REG_ODDEVEN, they can be '0' or '1', each has 10 phases, totally we have 4 groups, 40 phases. The REG_ODDEVEN can be changed by us, but its right value is determined by the GTH reset, we need to scan the phases to check it should be '0' or '1'. The REG_TOPBOT is controlled by us. We need to find the good phase which make the GBT header is locked.
The good phase number could be 0, 1, 2. 0 means the alignment is failed for all the 40 phases. 1 means there is only one phase which lock the GBT header, we will directly use this phase. 2 means there are two good phases, one with topbot=0, one with topbot=1, and they has same oddeven value. We will have to choose a better one.

1. Set REG_TOPBOT and REG_ODDEVEN to '0' & '0'. This is group 0.
   Check for these 10 phases, whether there is a good phase.
   How to check: Write 0x1 followed by 0x0 0x2410 (REG_ALIGN_CHK_RST). To reset the error flag. Wait for some time. Then readback the 0x2720 (REG_ALIGNMENT_DONE), check the bit for this channel. If it's '1' then the alignment is succeed. If not, then it's fail for some reason.
   How to shift the phase: Write '1' followed by '0' to the bit for this channel, in register REG_RXSLIDE (0x2480). Do it twice, then the phase will shift 1.

   - Check the 10 phases of group 0, if a good phase is found (Alignment succeed), then stop, now top_find=1 means we found a good phase for topbot=0. Go to step 5, to judge whether a good phase also exist for topbot=1, and to choose which phase to use.
   - If after we scan all the 10 phases, none of them are good phase, then we go to step 2.

2. Set Set REG_TOPBOT and REG_ODDEVEN to '0' & '1'. DO similar operation with step 0, for 10 phases of this group, group 1.

   - If a good phase is found (Alignment succeed), then stop, now top_find=1 means we found a good phase for topbot=0. Go to step 5, to judge whether a good phase also exist for topbot=1, and to choose which phase to use.
   - If after we scan all the 10 phases, none of them are good phase, then we failed to get a goodphase for topbot=0. Go to step 3.

3. Set REG_TOPBOT and REG_ODDEVEN to '1' & '0'. DO similar operation with step 0, for 10 phases of this group, group 2.

- If a good phase is found (Alignment succeed), then stop, now let bot_find=1, means we found a good phase for topbot=1. Also let topbot_final=1, means we use the good phase of topbot=1.
- If after we scan all the 10 phases, none of them are good phase, then we go to step 4.

4. Set REG_TOPBOT and REG_ODDEVEN to '1' & '1'. DO similar operation with step 0, for 10 phases of this group, group 4.
   - If a good phase is found (Alignment succeed), then stop, now let bot_find=1, means we found a good phase for topbot=1. Also let topbot_final=1, means we use the good phase of topbot=1.
   - If after we scan all the 10 phases, none of them are good phase, then we failed for all the 40 phases. Let topbot_final=-1.
5. When we jump to this step, means we find a good phase for topbot=0, and it's possible that there is also a good phase for topbot=1, we want to choose the better (stable) one. At this time we don't need to change the oddeven value anymore. The method:
   - When we jump to this step, at this time, topbot=0, oddeven=0/1. And this a a good phase (Rx alignment is OK, GBT header is locked). We assume this phase to be phase 0. And we shift the phase for 10 (the phase after phase 9 will go back to phase 0). For the phase 0-9, we read back the REG_CLK_SAMPLED (0x26C0), and pick the bit for this channel. For these 10 phases, we get a vector phase[9:0].
   - We use these vector, to calculate: If the first bit '1' is at phase[1] phase[5], then topbot_final=1. Else topbot_final=0.

   After the operation, go to step 6.
6.
   - If topbot_final=0, then the operation is over. We use the phase 0.
   - If topbot_final=1, then we use the good phase with topbot=1. We need to switch to that phase.

   6.1. Keep REG_ODDEVEN unchanged, change REG_TOBOT to 1.

   6.2. Shift the phase 5 times to phase 5.

7 Mapping of the GBT links

To use the CXP module, we must pull up the RST_L signal of the CXP module, to disable the reset. For the demonstrator, we will implement 16 links. The mapping between them and the CXP channels are listed as below.

Table 2: Mapping of the GBT channels

GTH Bank	GBT No. in logic	CXP Channel	CXP No.
111.0	1	2	1
111.1	3	4	1
111.2	0	1	1
111.3	2	3	1

Continued on next page

Table 2: Mapping of the GBT channels

GTH Bank	GBT No. in logic	CXP Channel	CXP No.
112.0	5	6	1
112.1	4	5	1
112.2	7	8	1
112.3	6	7	1
113.0	-	9	1
113.1	-	11	1
113.2	-	10	1
113.3	-	12	1
117.0	9	2	2
117.1	11	4	2
117.2	8	1	2
117.3	10	3	2
118.0	13	6	2
118.1	12	5	2
118.2	15	8	2
118.3	14	7	2
119.0	-	9	2
119.1	-	11	2
119.2	-	10	2
119.3	-	12	2

8 Configuration of the IDT clock generators for the GBT testing

In order to do the GBT test, we need to provide the 240MHz reference clocks for the GTH links.

8.1 Enable the clock generators

At first, we need to enable these two clock generators ICS8N4Q001L, and the REFCLK generator U35. The corresponding 3 switches of S1 should be OFF.

8.2 Configuration of the I2C bus switch chip PCA9548APW

The device address for U15 is 1110000, when writing 0x40, the SDA and SCL are connected to U5, for CXP1. When writing 0x80, they are connected to U4, for CXP2. The RST_N of the I2C switch should be disabled, this FPGA should pull down I2C_MAIN_RST_F.

8.3 Configuration of the IDT chips U4 & U5

The device address of ICS8N4Q001L is 1101110.

- To get a 240MHz output, we need to:

  - Write 0xA8 to 0x12
  - Write 0x2A to 0x00
  - Write 0x00 to 0x04
  - Write 0x11 to 0x08
  - Write 0x0A to 0x0C
  - Write 0x1F to 0x14
  - Write 0xA0 to 0x12

- To get a 120MHz output, we need to:
  - Write 0xA8 to 0x12
  - Write 0x2A to 0x00
  - Write 0x00 to 0x04
  - Write 0x11 to 0x08
  - Write 0x14 to 0x0C
  - Write 0x1F to 0x14
  - Write 0xA0 to 0x12

The *IDT_CER-5X7-PRG-GD_MAU_20120302.pdf* from IDT website can be referred when calculating the register values for the desired output frequency.

8.4 The *.vhd* files.

The file *clkcfg.vhd* is used to configure U4 & U5 to output a 240M/120M clock. The *i2c_clk_gen.vhd* is used to generate a slow clock for i2c master. The *i2c_master.vhd* is the I2C master.

The configuration of the I2C bus switch is slightly different from usual I2C chips. In line 92 of i2c_master.vhd, a different state change is applied to it. For high-level write operation, the register address should use same value with register data, just as line 119 & 120 of clkcfg.vhd.

The *clkcfg.vhd* is used by the GBT test, to easily generate a 240M/120M GTH reference clock. A new top-level vhdl file can be designed to do any single read/write operation to these I2C slaves. The PCIe interface or Chipscope can be used to control the read/write operations.