DUNE WIB firmware

B. Land, A. Madorsky, A. Nikolica

Table of Contents

[1 Introduction 2](#_Toc80697272)

[2 WIB firmware location 3](#_Toc80697273)

[3 Building WIB firmware 4](#_Toc80697274)

[4 Firmware structure 4](#_Toc80697275)

[4.1 COLDATA receivers 5](#_Toc80697276)

[4.2 COLDATA Frame decoders 6](#_Toc80697277)

[4.3 DAQ Frame Builders 6](#_Toc80697278)

[4.4 FELIX transmitters 6](#_Toc80697279)

[4.5 ZYNQ CPU module 6](#_Toc80697280)

[4.6 COLDATA I2C interface 7](#_Toc80697281)

[4.6.1 How to use I2C interface: 7](#_Toc80697282)

[4.6.2 COLDATA v1 I2C bugs 8](#_Toc80697283)

[4.7 COLDATA FAST command generator 8](#_Toc80697284)

[4.7.1 How to use FAST command generator 9](#_Toc80697285)

[4.8 I2C interface for WIB on-board devices 9](#_Toc80697286)

[4.9 Control and status registers 9](#_Toc80697287)

[4.10 Timing Endpoint 12](#_Toc80697288)

[4.11 DAQ spy memory 13](#_Toc80697289)

[4.11.1 Preliminary steps 13](#_Toc80697290)

[4.11.2 Operation with the trigger distributed by TLU 13](#_Toc80697291)

[4.11.3 Operation with the software trigger 13](#_Toc80697292)

[5 Clock regions 14](#_Toc80697293)

[6 References 16](#_Toc80697294)

[7 Revision table 16](#_Toc80697295)

# Introduction

This document describes the structure of the firmware used in the DUNE Warm Interface Board (WIB). WIB functions include:

* Reception of data from Front-End Motherboards (FEMBs)
* Decoding the data frames sent by FEMBs
* Forming the data blocks for DAQ system
* Providing power conversion for FEMBs
* Providing slow control and monitoring for FEMBs

The simplified schematics of WIB connections to the outside world is shown in Figure 1.

  
Figure 1: WIB board connections in the DUNE experiment

# WIB firmware location

The DUNE WIB firmware source code is located in this repository:  
<https://github.com/madorskya/wib_sim>

This location may change in the future; this document will be modified to reflect that.

# Building WIB firmware

The **wib\_zu6cg** subfolder in the repository contains the complete Xilinx Vivado project that can be directly opened and built in Vivado. The version of Vivado used for development is 2020.1.

The firmware project contains two synthesis and implementation runs configured for two different but pin-compatible ZYNQ [9] devices:

* XCZU6CG-FFVB1156-1-E
* XCZU9EG-FFVB1156-1-E

This was necessary because some of the WIB prototype boards have been built using XCZU9EG devices. However, production WIB boards are expected to use only XCZU6CG devices.

# Firmware structure

WIB firmware has been designed following the requirement document [1]. Figure 2 shows the simplified structure of the WIB firmware. Red connections signify the data acquisition (DAQ) path, purple connections are timing signals, AXI bus that the CPU is using to communicate with multiple firmware blocks is shown in green, and control and status signals are shown in gray color. Sections below describe each of the firmware blocks in details. Figure 3 contains a screenshot from Vivado firmware project.



Figure . WIB firmware structure.

Diagram, schematic

Description automatically generated

Figure 3. Block design screenshot from Vivado firmware project

## COLDATA receivers

The COLDATA receivers are serial receivers operating at 1.28 Gbps bit rate. WIB receives data from 4 FEMBs, each of them has 2 COLDATA chips, and each COLDATA chip is sending data via 2 serial links. Therefore, the total count of COLDATA receivers is 4\*2\*2 = 16. These receivers are implemented using hard GTH IPs [9] available in the ZYNQ device. The output of each receiver is a 16-bit bus carrying the deserialized data.

## COLDATA Frame decoders

COLDATA chips can transmit data in one of the several formats detailed in [5]. The particular format is selected by programming an I2C register. The COLDATA Frame decoder module is capable of decoding all of these formats. The format type is determined automatically from the received data stream.

## DAQ Frame Builders

The DAQ Frame Builder module is responsible for preparing the data for transmission to the DAQ system. The DAQ system expects the data to arrive in a format detailed in [4]. There are two DAQ serial links: Link 0 carries data from FEMBs 0,1, and link 1 carries data from FEMBs 2,3. Each of the two Frame Builder modules prepares data for one of the DAQ links, by combining the deframed data arriving from corresponding COLDATA Frame decoders. The outputs of each Frame Builder are a 32-bit data bus and an additional 4-bit bus carrying K-symbol flags.

## FELIX transmitters

These modules are responsible for sending data prepared by Frame Builders to the FELIX boards [8] via 9.6192 Gpbs serial links. DUNE experiment is using FELIX boards to receive DAQ data from WIBs and transfer these data to storage. The FELIX transmitter modules are implemented using hard GTH IPs [9] available in the ZYNQ device.

## ZYNQ CPU module

This module is a hard CPU IP core provided in ZYNQ devices. Its main purpose is control and monitoring of the firmware modules. Access to all modules in WIB firmware is provided via standard ZYNQ AXI bus interface.

List of currently implemented modules with AXI registers is shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Base address, hex** | **Module type** | **FEMB**  **#** | **COLDATA**  **chip #** | **Description** |
| A0030000 | coldata\_fast\_cmd | All | All | COLDATA FAST command generator for all COLDATA chips |
| A0010000 | coldata\_i2c | 0 | 0 | I2C interface for COLDATA chip |
| A0040000 | coldata\_i2c | 0 | 1 | I2C interface for COLDATA chip |
| A0050000 | coldata\_i2c | 1 | 0 | I2C interface for COLDATA chip |
| A0060000 | coldata\_i2c | 1 | 1 | I2C interface for COLDATA chip |
| A0070000 | coldata\_i2c | 2 | 0 | I2C interface for COLDATA chip |
| A0080000 | coldata\_i2c | 2 | 1 | I2C interface for COLDATA chip |
| A0090000 | coldata\_i2c | 3 | 0 | I2C interface for COLDATA chip |
| A00A0000 | coldata\_i2c | 3 | 1 | I2C interface for COLDATA chip |
| A00B0000 | axi\_iic | n/a | n/a | I2C interface for WIB on-board devices |
| A00C0000 | Register bank | n/a | n/a | Control and status registers |
| n/a | Timing endpoint | n/a | n/a | Timing endpoint, programmed via control register |
| A0100000 | DAQ spy memory | 0,1 | All | 1MB DAQ spy memory for FEMBs 0,1. Capable of recording DAQ output data sent to FELIX. |
| A0200000 | DAQ spy memory | 2,3 | All | 1MB DAQ spy memory for FEMBs 2,3. Capable of recording DAQ output data sent to FELIX. |

Table 1. WIB firmware modules and their base addresses

All registers are 32 bits wide.

## COLDATA I2C interface

Register list is shown below. Note that there are 8 such modules, so the register addresses are shown as offsets from the base address of each unit. See base addresses in Table 1.

|  |  |  |
| --- | --- | --- |
| **Offset, hex**  **Relative to base** | **Register name** | **Description** |
| 0 | start\_transfer | Writing 1 into this register starts I2C transfer programmed into addr\_data register. Write 0 immediately after writing 1. |
| 4 | addr\_data | Combined address and data register. See format description below. |

Table 2. I2C interface registers

Format of addr\_data register is shown below:

|  |  |
| --- | --- |
| **Bits** | **Function** |
| 31:27 | Unused |
| 26:23 | Chip address |
| 22:20 | Register page address |
| 19 | Read = 1, write = 0 |
| 18 | Placeholder for Acknowledge bit (ACK) |
| 17:10 | Register address |
| 9 | Placeholder for Acknowledge bit (ACK) |
| 8:1 | Write operation: data to write to register, Read operation: data read from register |
| 0 | Placeholder for Acknowledge bit (ACK) |

Table 3. Format of addr\_data register

### How to use I2C interface:

Write operation:

1. Write the parameters into addr\_data register as needed. Leave ACK bit placeholders = 0
2. Set start\_transfer register to 1, then immediately set it to 0
3. Wait for 60 uS or longer before attempting the next transfer

Read operation:

1. Write the parameters into addr\_data register as needed. Leave ACK bit placeholders = 0
2. Set start\_transfer register to 1, then immediately set it to 0
3. Wait for 60 uS or longer
4. Read addr\_data register back. Bits 8:1 in that register contain data that were read from the register. Bits 0, 9, and 18 should contain values of 1 in each. These are recorded states of ACK bits, received after transfer of each of 3 bytes.

Note that two COLDATA I2C interfaces that serve the same FEMB are sharing clock lines. That means that you can use only one of these interfaces at any given time. Example:

* Program I2C interface for FEMB #0 COLDATA #0 to read or write a register
* You can use I2C interfaces for FEMB #1,2,3 at this time, don’t have to wait until transaction in step 1 is finished.
* However, to use any of the I2C interfaces for FEMB #0 for the next transaction, wait for 60 uS or more until the transaction in step 1 is finished

### COLDATA v1 I2C bugs

**Bug #1** leads to missing ACK bits after first byte in certain conditions. That ACK bit is recorded in bit 18. It should be ignored.

**Bug #2** leads to incorrect latching of the chip ID inside COLDATA. Workaround: When trying to read/write any register in COLDATA or any of the COLDADC chips, first issue **two** read commands to that chip, any register. Then proceed to read/write registers that you actually intended to access. Issuing two read commands is only necessary when switching from one chip to another.

The COLDATA chip has a number of I2C registers, and additionally it provides a channel for I2C access to COLDADC chips connected to it. The details of COLDATA and COLDADC I2C registers can be found in [5] and [6].

## COLDATA FAST command generator

Register list:

|  |  |  |
| --- | --- | --- |
| **Address, hex** | **Register name** | **Description** |
| A0030000 | fast\_cmd\_code | Writing a command code into this register immediately generates the corresponding FAST command. See the list of valid command codes below. Command codes **cannot** be combined using OR operation; only one command at a time can be issued. Invalid command codes are ignored. |
| A0030004 | edge\_to\_act\_delay | Delay between EDGE and ACT command for correct COLDADC reset procedure. Write a value of 19 into this register. |

Table 4. FAST command generator registers

FAST command codes:

|  |  |  |
| --- | --- | --- |
| **Command code, bin** | **Command name** | **Description** |
| 000001 | RESET | COLDATA chip reset |
| 000010 | ACT | Performs command stored in ACT command register |
| 000100 | SYNC | Zero time stamp |
| 001000 | EDGE | Move edge of 2 MHz clock to next rising edge of 64 MHz clock |
| 010000 | IDLE | No description for this command in COLDATA datasheet, need to check with COLDATA team |
| 100000 | EDGE\_ACT | COLDADC rev 1 requires the following reset procedure:  First EDGE command, then after a delay, ACT command programmed as COLDADC reset. COLDADC reset must end (rise) between 62.5 and 125 ns after 2M clock rising edge. This command code makes FAST command unit to issue EDGE and ACT commands with precise timing needed for correct reset procedure. Note that before using this command, ACT command must be programmed with FASTACT\_COLDADC\_RESET\_COMMAND code. |

Table 5. FAST command codes

### How to use FAST command generator

1. If you are intending to use ACT command, first program the desired ACT command into COLDATA **ACTCOMMANDREG** register via I2C.
2. If you are intending to use EDGE\_ACT command, first program COLDATA **ACTCOMMANDREG** register with **FASTACT\_COLDADC\_RESET\_COMMAND** code (via I2C), and WIB **edge\_to\_act\_delay** register with the value of 19.
3. Write command code into **fast\_cmd\_code**. The command will be immediately executed.

The COLDATA register addresses and command codes as well as additional details about COLDATA FAST commands can be found in [5].

## I2C interface for WIB on-board devices

This is a standard Xilinx IIC IP. It’s connected via a multiplexer to all on-board devices that require I2C programming. Please use the driver provided by Xilinx to talk to this module.

Before accessing a particular device, please make sure to select it by using i2c\_select control register. See “Control and status registers” section below for information about control registers. More details about the on-board devices can be found in [2].

## Control and status registers

WIB firmware implements 32 control registers and 32 status registers. Each register is 32-bit wide. Control registers are readable and writable; status registers are read-only. Default values for all control registers are zeros.

Control registers (read/write) are listed in Table 6:

|  |  |  |  |
| --- | --- | --- | --- |
| **Address, hex** | **Bits in register** | **Parameter name** | **Description** |
| A00C0000 | 7:0 | ts\_addr | Timing point address |
| A00C0000 | 9:8 | ts\_tgrp | Timing point group code |
| A00C0000 | 28 | ts\_srst | Timing point reset |
| A00C0004 | 3:0 | i2c\_select | On-board I2C bus selector. Devices are selected according to the table below:   |  |  | | --- | --- | | **Value** | **Selected device** | | 0 | SI5344 | | 1 | SI5342 | | 2 | QSFP | | 3 | PL\_FEMB\_PWR | | 4 | PL\_FEMB\_EN | | 5 | SENSOR\_I2C | | 6 | PL\_FEMB\_PWR2 | | 7 | LTC2977 | | 8 | PL\_FEMB\_PWR3 | | 9 | FLASH | | 10 | ADN2814 | |
| A00C0004 | 4 | fp\_sfp\_sel | P15 SFP connection selector 0=CDR 1=GTH (schematic [3] page 14) |
| A00C0004 | 5 | rx\_timing\_sel | U1 input selector 0=backplane 1=SFP (schematic [3] page 15) |
| A00C0004 | 6 | daq\_spy\_reset[0] | DAQ spy FSM reset for FELIX link 0, FEMB 0,1 |
| A00C0004 | 7 | daq\_spy\_reset[1] | DAQ spy FSM reset for FELIX link 1, FEMB 2,3 |
| A00C0004 | 11:8 | PRBS selection | PRBS pattern selection for COLDATA RX and FELIX TX links   |  |  | | --- | --- | | **Value** | **Pattern** | | 0 | Normal operation | | 1 | PRBS-7 | |
| A00C0004 | 12 | fb\_reset | Frame builder reset. Write 1 then 0 to reset. |
| A00C0004 | 13 | coldata\_rx\_reset | Reset of the serial receivers. Write 1 then 0 to reset. |
| A00C0008 | 15:0 | link\_mask | There are 16 input serial links, 4 per FEMB. The link\_mask bits, when set to 1, are telling firmware that the corresponding serial link is not working. These bits are also passed in the DAQ data header [4], so the unpacker knows which data to ignore. The bit assignment is shown below:   |  |  |  | | --- | --- | --- | | **Link\_mask bit** | **FEMB** | **Link** | | 0 | 0 | 0 | | 1 | 0 | 1 | | 2 | 0 | 2 | | 3 | 0 | 3 | | 4 | 1 | 0 | | 5 | 1 | 1 | | 6 | 1 | 2 | | 7 | 1 | 3 | | 8 | 2 | 0 | | 9 | 2 | 1 | | 10 | 2 | 2 | | 11 | 2 | 3 | | 12 | 3 | 0 | | 13 | 3 | 1 | | 14 | 3 | 2 | | 15 | 3 | 3 | |
| A00C000C | 0 | ts\_edge\_sel | Timing endpoint clock edge selector |
| A00C000C | 1 | fake\_time\_stamp\_en | Enable fake time stamp. Fake time stamp is generated by a local firmware counter and replaces the timing endpoint time stamp. |
| A00C0010 | 7:0 | cmd\_code\_idle | Timing system command code for IDLE |
| A00C0010 | 15:8 | cmd\_code\_edge | Timing system command code for EDGE |
| A00C0010 | 23:16 | cmd\_code\_sync | Timing system command code for SYNC |
| A00C0010 | 31:24 | cmd\_code\_act | Timing system command code for ACT |
| A00C0014 | 7:0 | cmd\_code\_reset | Timing system command code for RESET |
| A00C0014 | 15:8 | cmd\_code\_adc\_reset | Timing system command code for ADC RESET |
| A00C0014 | 23:16 | cmd\_code\_trigger | Timing system command code for Trigger |
| A00C0018 | 31:0 | fake\_time\_stamp\_init[31:0] | Initial value for fake time stamp, bits 31:0 |
| A00C001C | 31:0 | fake\_time\_stamp\_init[63:32] | Initial value for fake time stamp, bits 63:32 |
| A00C0020 | 0 | fake\_daq\_stream | Enables sending fake data to FELIX even without FEMBs attached. |
| A00C0024 | 17:0 | spy\_rec\_time | Spy ring buffer recording time, in 32-bit words, based on FELIX fabric clock, 240.48MHz |
| A00C002C | 31:0 | felix\_txctrl0\_in | Contents of txctrl0 bus for FELIX TX. Assign 0 for normal operation |
| A00C0030 | 31:0 | felix\_txctrl1\_in | Contents of txctrl1 bus for FELIX TX. Assign 0 for normal operation |
| A00C0038 | 0 | felix\_reset\_pll\_datapath | FELIX TX reset PLL and datapath |
| A00C0038 | 4 | felix\_reset\_datapath | FELIX TX reset datapath only |
| A00C0038 | 9:8 | felix\_tx\_8b10b\_en | FELIX TX enable 8b/10b encoding. These bits should always be set to ‘b11 for normal operation |

Table 6. Control registers

**How to set initial value for fake time stamp (FTS):**

1. Disable FTS by setting **fake\_time\_stamp\_en = 0**
2. Write the new initial value into **fake\_time\_stamp\_init** register
3. Enable FTS by setting **fake\_time\_stamp\_en = 1**

**How to use Timing system command codes:**

Each of the FAST commands, including the 2MHz clock sync command, can be triggered by a timing system command sent from TLU. The exact codes of TLU commands corresponding to each FAST command are programmable using the following registers:

* **cmd\_code\_idle**
* **cmd\_code\_edge**
* **cmd\_code\_sync**
* **cmd\_code\_act**
* **cmd\_code\_reset**
* **cmd\_code\_adc\_reset**
* **cmd\_code\_trigger**

These registers support 8-bit command codes. However, the current timing endpoint has 4-bit commands, so only 4 lower bits are used in each of the registers at this time. Default for all of them is 0, which is considered invalid. Unless reprogrammed to non-zero value, the corresponding command is disabled.

Status registers (read-only):

|  |  |  |  |
| --- | --- | --- | --- |
| **Address, hex** | **Bits in register** | **Parameter name** | **Description** |
| A00C0080 | 0 | daq\_spy\_full[0] | “full” flag for DAQ spy memory, FELIX link 0, FEMB 0,1 |
| A00C0080 | 1 | daq\_spy\_full[1] | “full” flag for DAQ spy memory, FELIX link 1, FEMB 2,3 |
| A00C0084 | 15:0 | rxprbserr | PRBS error detection bits, one for each of the 16 input serial links. |
| A00C0088 | 5:0 | fw\_second | Firmware generation time stamp, second |
| A00C0088 | 11:6 | fw\_minute | Firmware generation time stamp, minute |
| A00C0088 | 16:12 | fw\_hour | Firmware generation time stamp, hour |
| A00C0088 | 22:17 | fw\_year | Firmware generation time stamp, year - 2000 |
| A00C0088 | 26:23 | fw\_month | Firmware generation time stamp, month |
| A00C0088 | 31:27 | fw\_day | Firmware generation time stamp, day |
| A00C008c | 3:0 | bp\_slot\_addr | Slot address read from the crate backplane |
| A00C008c | 7:4 | bp\_crate\_addr | Crate address read from the crate backplane |
| A00C0090 | 3:0 | ts\_stat | ts\_stat signal from timing endpoint |
| A00C0090 | 4 | ts\_rst | ts\_rst signal from timing endpoint |
| A00C0090 | 8 | ts\_rdy | ts\_rdy signal from timing endpoint |
| A00C0090 | 15:12 | ts\_sync | ts\_sync signal from timing endpoint |
| A00C0090 | 16 | ts\_sync\_v | ts\_sync\_v signal from timing endpoint |
| A00C0090 | 17 | adn2814\_lol | LOL signal from CDR |
| A00C0090 | 18 | and2814\_los | LOS signal from CDR |
| A00C0090 | 27:20 | 0xff | fixed pattern |
| A00C0094 | 19:0 | spy\_addr [0] | Current memory address for spy memory 0, in bytes |
| A00C0098 | 19:0 | spy\_addr [1] | Current memory address for spy memory 1, in bytes |
| A00C00A0 | 31:0 | ts\_tstamp[31:0] | Timing point time stamp bits 31:0 |
| A00C00B0 | 63:32 | ts\_tstamp[63:32] | Timing point time stamp bits 63:32 |
| A00C00BC | 31:0 | test\_pattern | Hardwired value = 0xbabeface |
| A00C00C0 | 0 | felix\_rst\_tx\_done | FELIX TX reset done signal |
| A00C00C0 | 4 | felix\_tx\_active | FELIX TX active signal |
| A00C00C0 | 8 | felix\_pwrgood | FELIX TX power good signal |

Table 7. Status registers

## Timing Endpoint

This is a timing endpoint module. The only parameters that need programming are Address and Group codes. These parameters are programmed via control registers. See **ts\_addr** and **ts\_tgrp** registers in Control and Status registers section above for details. Use **ts\_srst** register to reset the endpoint, by writing 1 and then 0. The endpoint must only be reset after the external clock (from the hardware PLL on WIB) is stable. There are additional requirements on the PLL configuration found in [11], and the PLL configuration is implemented by a series of I2C register writes to the PLL from the WIB software. The source code for this module is provided by the timing system developers. More details can be found in [7].

## DAQ spy memory

There are two independent DAQ spy memory modules, one for FELIX link 0 (FEMBs 0,1), another for FELIX link 1 (FEMBs 2,3). Each memory unit is 1MB in size. They operate in 32-bit words, so the size is 256K 32-bit words. The spy memory controllers implement a “ring” buffer functionality, with the possibility to trigger using a signal distributed by the timing master (TLU). How to operate:

### Preliminary steps

* Program **cmd\_code\_trigger** register with the TLU command for external trigger
* Program **spy\_rec\_time** parameter with the number of 32-bit words to record after trigger. That number should be less or equal to **256K – trigger\_latency**. Trigger latency time here is represented in terms of FELIX fabric clock cycles, which is  
  **1 / 240.48 MHz = ~ 4.158 ns.**

### Operation with the trigger distributed by TLU

1. Disable external triggers
2. Reset the spy memory, by using driving **daq\_spy\_reset=1** and then **=0**. At the time of reset, the spy memory starts continuously recording data transmitted via FELIX links. The memory address overwraps at maximum address back to 0.
3. Wait for at least the trigger latency time or longer to let the memory record the data before trigger.
4. Enable or issue a trigger command. At the time of the trigger command, the spy memory will start counting words stored in it. It will store **spy\_rec\_time** 32-bit words, then stop recording, and set **daq\_spy\_full** bit to 1.
5. Wait until **daq\_spy\_full** bit from that module is set to 1.
6. Read out and store the entire memory array. The size of the array is 1 MB = 256K 32-bit words.
7. Read out the current spy memory address, from this register: **spy\_addr[N],** where **N = 0,1** is the spy memory/FELIX link number. This is the address where the recording stopped.
8. Calculate the address where you need to start decoding memory data:   
     
   **decoding\_start\_addr = spy\_addr[19:2]–spy\_rec\_time–trigger\_latency**  
     
   All arithmetic here is 18-bit, unsigned. The **decoding\_start\_addr** result overwraps to maximum value of **0x3ffff** if it goes below 0. The result represents the 32-bit word index.  
   Note that the resulting address may not necessarily point at the first word of a FELIX data block. Start searching from that address until you detect the SOF symbol.
9. Unpack the following number of 32-bit words:  
   **spy\_rec\_time + trigger\_latency**  
   rounded down to the FELIX frame size.
10. Go to step 1.

### Operation with the software trigger

1. Reset the memory by driving **daq\_spy\_reset=1** and then **=0**. The memory is now recording permanently.
2. Wait for ~1.1 ms or longer, to overwrite the entire memory.
3. Stop recording by setting **daq\_spy\_reset=1**, and keep it **=1**. The recording is now stopped at arbitrary address. There is no way to obtain that address, unfortunately.
4. Read out the entire spy memory.
5. Search for valid DAQ data block header, with the smallest time stamp value. This is the first block that you should be decoding. Start decoding from that block onward, and wrap the address around to 0 when it reaches the maximum value of 1 MB.

The data in the memory array are stored in the same DAQ format [4] as transmitted to FELIX. Each frame is 120 32-bit words in length. The K bits are not stored, so the format decoder should be using specific 32-bit data patterns at the beginning and end of each frame to decode the data (words 0, 118, 119 in [4]).

# Clock regions

WIB firmware uses several different clocks. Some of them are synchronous to each other, others are entirely asynchronous. Figure 4 shows the clock schematics of the WIB data path.

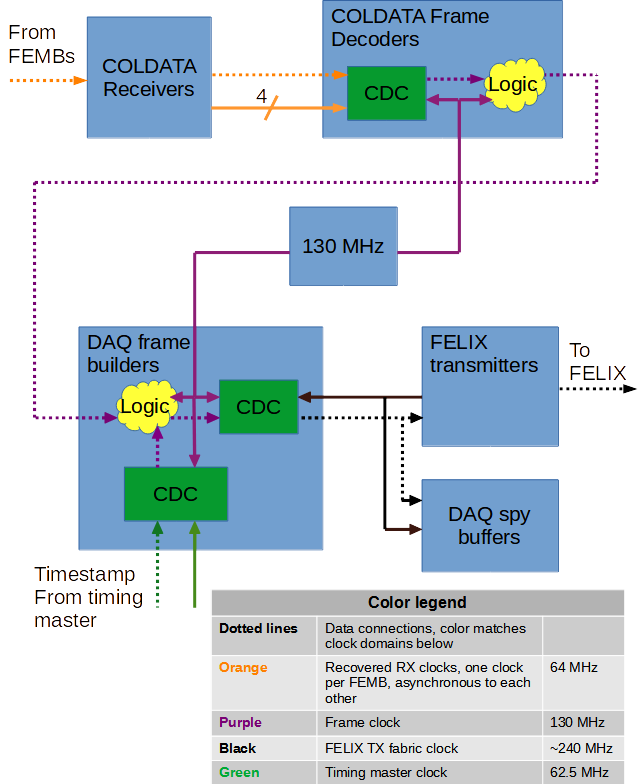


Figure 4. WIB data path clock schematics.

The data arrive from COLDATA chips over 1.28 Gbps serial links. The reference clocks for these links have an option to be generated by standalone oscillators on each FEMB, so WIB firmware assumes that FEMB reference clocks may be asynchronous to each other. Therefore, the recovered RX clocks from CODATA serial receivers are used as inputs to the Clock Domain Crossing (CDC) modules inside COLDATA Frame Decoders. The data from all COLDATA receivers are brought to the domain of the single Frame clock, running at 130 MHz. This frequency is selected to be slightly above the doubled frequency of the recovered RX clocks (64 MHz \* 2 = 128 MHz). The doubling requirement is related to the fact that the Frame decoder processes the data from each COLDATA receiver one byte in each clock period. This greatly simplifies the Frame decoder logic but requires double processing frequency. The requirement of the Frame decoder clock frequency to be slightly higher than doubled recovered RX clock is to guarantee that the FIFOs in the CDC module never overflow.

The deframed data are passed to DAQ Frame builder, which builds the data frames for DAQ transmission using the same Frame clock. The prepared DAQ data frames are then transferred into FELIX transmitter clock domain using another CDC module. The FELIX transmitter clock is taken from the “fabric clock” output of the GTH hard IP. Note that the frequency of the FELIX transmitter clock is a multiple of LHC machine frequency: 40.079 MHz \* 6 = 240.474 MHz.

The 64-bit time stamp arriving from the timing master via timing endpoint must be brought into the Frame clock domain before it can be used by DAQ Frame builder. This is achieved by using another CDC module.

# References

1. WIB firmware requirements (Josh’s document)
2. WIB hardware manual
3. WIB schematics (https://docs.dunescience.org/cgi-bin/private/ShowDocument?docid=17849)
4. File “WIB-DAQ\_Format\_2021-02-18.xlsx” in git repository, doc directory.
5. COLDATA datasheet
6. COLDADC datasheet
7. DUNE Timing System – Single Phase Firmware
8. FELIX manual
9. Ultrascale + ZYNQ manual
10. Ultrascale + GTH manual
11. D. Cussans, D. Newbold, A. Thea, “Dune Timing System Integration Guide”, Google document, 2021. Available: <https://docs.google.com/document/d/1A9LnkR_0Z2bDIFv0G0GBI7O04allCHZTYcISBaYXr-o/edit> [Accessed: 2 September, 2021]

# Revision table

|  |  |
| --- | --- |
| **Date** | **Changes** |
| 2020-07-23 | First draft. Will add more modules and registers as the design is progressing. |
| 2020-08-05 | Reworked table of implemented modules. Only one FAST command module is needed, since FAST command output is fanned out to all FEMBs in hardware |
| 2020-08-10 | Added more modules |
| 2020-10-10 | Added WIB configuration sequence, DAQ spy memory modules, DAQ spy memory control and status bits, control bits for clock selection. |
| 2020-10-19 | Added PRBS pattern selection and error detection, Frame Bulder reset bit. Moved coldata\_rx\_reset bit since it was conflicting with daq\_spy\_reset bits. Added description of DAQ spy memory modules. |
| 2020-10-30 | Added link\_mask register |
| 2021-02-05 | Added a lot of missing registers, removed configuration section |
| 2021-03-17 | Reworked the spy memory into ring buffer. Added corresponding control and status registers, and reworked instructions.  Added TLU time stamp reclocking FIFO, but this did not require changes in this document.  Replaced reference [4] with the updated document. |
| 2021-03-22 | Added section on using spy memory with software trigger |
| 2021-06-01 | Added missing ts\_srst register bit |
| 2021-08-08 | Renamed the document to reflect that it’s now a more or less complete WIB firmware manual, not just register list. Added all typically needed sections for such document. Added missing FELIX control and status registers |
| 2021-08-24 | Added Clock regions section |
| 2021-09-02 | Add reference to PLL configuration in timing system integration guide, and reset requirement |