DUNE WIB firmware

P. Keener, J. Klein, B. Land, A. Madorsky, A. Nikolica

Updated: 12 Nov. 2022

Table of Contents

[1 Introduction 3](#_Toc119228987)

[2 WIB firmware location 4](#_Toc119228988)

[3 Building WIB firmware 5](#_Toc119228989)

[4 Firmware structure 5](#_Toc119228990)

[4.1 COLDATA receivers 7](#_Toc119228991)

[4.2 COLDATA Frame decoders 8](#_Toc119228992)

[4.2.1 COLDATA CRC error flags 8](#_Toc119228993)

[4.3 DAQ Frame Builders 8](#_Toc119228994)

[4.4 FELIX transmitters 8](#_Toc119228995)

[4.5 ZYNQ CPU module 8](#_Toc119228996)

[4.6 COLDATA I2C interface 9](#_Toc119228997)

[4.6.1 I2C Write operation 10](#_Toc119228998)

[4.6.2 I2C Read operation 10](#_Toc119228999)

[4.6.3 COLDATA and COLDADC I2C fine timing requirements 10](#_Toc119229000)

[4.6.4 COLDATA and COLDADC I2C fine clock phase adjustment 11](#_Toc119229001)

[4.6.5 Data cable latency measurement 11](#_Toc119229002)

[4.7 COLDATA FAST command generator 12](#_Toc119229003)

[4.7.1 How to use FAST command generator 13](#_Toc119229004)

[4.8 I2C interface for WIB on-board devices 13](#_Toc119229005)

[4.9 Control and status registers 13](#_Toc119229006)

[4.9.1 Control registers (read/write) are listed in Table 6: 13](#_Toc119229007)

[4.9.2 How to set initial value for fake time stamp (FTS): 16](#_Toc119229008)

[4.9.3 Using Timing system command codes 16](#_Toc119229009)

[4.9.4 Using fake\_daq\_stream mode 17](#_Toc119229010)

[4.9.5 Status registers (read-only): 17](#_Toc119229011)

[4.10 Timing Endpoint 18](#_Toc119229012)

[4.11 DAQ spy memory 19](#_Toc119229013)

[4.11.1 Preliminary steps 19](#_Toc119229014)

[4.11.2 Operation with the trigger distributed by DTS 19](#_Toc119229015)

[4.11.3 Operation with the software trigger 19](#_Toc119229016)

[4.12 Monitoring ADCs 20](#_Toc119229017)

[4.13 10 MHz reference clock on P12 connector 20](#_Toc119229018)

[5 Clock regions 20](#_Toc119229019)

[6 DAQ readout format 22](#_Toc119229020)

[6.1 Introduction 22](#_Toc119229021)

[6.2 Metadata Header 22](#_Toc119229022)

[6.2.1 FELIX Start of Frame Word 23](#_Toc119229023)

[6.2.2 Word 0 23](#_Toc119229024)

[6.2.3 Words 1 and 2 23](#_Toc119229025)

[6.2.4 Word 3 23](#_Toc119229026)

[6.2.5 Word 4 24](#_Toc119229027)

[6.3 Metadata Trailer 24](#_Toc119229028)

[6.3.1 Word 117 24](#_Toc119229029)

[6.3.2 FELIX trailer word 0 25](#_Toc119229030)

[7 Migration from COLDATA p2 to p3 chips 25](#_Toc119229031)

[8 Migration from CDR-based to DCSK timing endpoint 26](#_Toc119229032)

[9 COLDATA data time stamp synchronization 27](#_Toc119229033)

[9.1 Firmware time stamp alignment logic 28](#_Toc119229034)

[9.1.1 Time stamp synchronization procedure details 29](#_Toc119229035)

[9.1.2 Time stamp processing in WIB 29](#_Toc119229036)

[9.1.3 Alignment setup procedure 30](#_Toc119229037)

[10 References 31](#_Toc119229038)

[11 Revision table 32](#_Toc119229039)

# 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 . Block design screenshot from Vivado firmware project

## COLDATA receivers

The COLDATA receivers are serial receivers operating at 1.28 Gbps (COLDATA p2) or 1.25 Gbps (COLDATA p3) 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.

### COLDATA CRC error flags

Each COLDATA link receiver is calculating CRC from received data and compares the calculated CRC with the CRC bytes received with the data. Errors are flagged in **crc\_err** sticky bits. How to use:

1. Read **crc\_err** flags
2. Set **crc\_err\_reset = 1** and back **= 0** to reset the sticky bits.

## 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 . 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** | **Operation** | **Description** |
| 0x0 | start\_transfer | W | Writing 1 into this register starts I2C transfer programmed into addr\_data register. Write 0 immediately after writing 1. |
| 0x0 | busy | R | Bit [0] contains busy flag. While this bit reads as 1 the I2C logic is busy and cannot be used. |
| 0x4 | addr\_data | W/R | Combined address and data register. See format description below. |
| 0x8 | stimulus | W | Writing 1 into bit [0] generates stimulus for cable latency measurement. Please write 0 into this register when latency measurement is completed, otherwise the SDA line will remain at zero level and I2C interface will not work. |
| 0x8 | latency | R | Bits [7:0] contain measured latency value in units of system clock (16 ns). This is the roundtrip latency from WIB via the cable to COLDATA and back to WIB. |

Table . 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

### I2C 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

### I2C 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

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 and COLDADC I2C fine timing requirements

This is a typical I2C operation in WIB:



Figure . Typical WIB <--> COLDATA or COLDADC I2C transaction

The **sda\_out** signal is WIB to COLDATA, **sda\_in** is COLDATA to WIB. Timing for read and write operations is identical. Shown below is blown-up view of one data bit:



Figure . Blown-up view of one data bit timing.

The time scale on top of Figure 5 is in units of system clock (62.5 MHz).  All transitions (both on **scl** and **sda\_out**) are always happening only on the falling edges of 62.5 MHz system clock.

Additionally, **scl** and **sda\_out** signals are locked in the IO block Flops when they are leaving FPGA. This guarantees very consistent signal timing relative to 62.5 MHz system clock which does not change with any future firmware versions. This precise timing of the **scl** and **sda\_out** signals seems to be absolutely critical for the COLDADC I2C circuitry to work properly.

The **sda\_in** signal is also locked into IO block Flop using falling edge of the 62.5 MHz system clock when it enters FPGA. The value of **sda\_in** signal is analyzed and recorded by I2C state machine at the falling edge of **scl**.

### COLDATA and COLDADC I2C fine clock phase adjustment

When very long high-speed cables are used for connecting WIB with FEMBs, the relative clock phase between system clock and I2C clock may shift by a few ns. This is sufficient to induce I2C read/write failures. WIB firmware can adjust the relative clock phase in very small steps (~15 ps) to accommodate cables of any reasonable length.

The initial relative phase between system clock and I2C clock is set at the time of WIB reboot and is equal to 0. The total count of phase adjustment steps is 1064. These steps cover the entire duration of one system clock: 16ns/1064 = ~15.04ps. The relative phase can be reset to initial value using these two methods:

1. Reboot WIB.
2. Perform the adjustment 1064 times from the initial phase. The relative phase will return to the initial phase = 0. Note that this method requires keeping track of the total count of adjustment steps performed since last WIB reboot.

To change the relative phase by one step of ~15 ps, write 1 and then 0 into **ps\_en\_in** register. Repeat these steps until I2C interface is working without errors with all FEMBs.

Example script that performs a user-defined count of phase adjustment steps is available here:

**soft\_debug\_p3/i2c\_phase.sh**

It accepts one parameter that is the adjustment step count.

The optimal adjustment step counts determined experimentally for some cable lengths are shown below:

|  |  |
| --- | --- |
| **Total cable length, m** | **Adjustment steps** |
| 8 | 300 |
| 22 | 300 |
| 29 | 400 |

Note that in each particular system these adjustments may need to be re-optimized.

### Data cable latency measurement

COLDATA P3 chip added a provision that allows to measure latency of the data cable connecting WIB to COLDATA. This latency is important for the time stamp synchronization procedure described later in this manual. This procedure only works on TOP COLDATA chip on each FEMB. The steps for latency measurement are listed below:

1. Program COLDATA I2C registers 0x2B, 0x2C, 0x2D to the value of 0xB2.
2. Program COLDATA I2C ACTCOMMANDREG (0x20) to the value of 9.
3. Issue FAST ACT command to enable loopback connection.
4. Wait for FAST ACT command to execute. Since the loopback connection is only active for 64 uS, the usual Linux delay techniques such as usleep are not sufficiently precise to use here. The easiest way to wait for FAST command to execute is to issue 6 dummy write commands into I2C stimulus register with the value of 0.
5. Issue measurement stimulus pulse by writing 1 into stimulus register
6. Wait for 10 us to allow the loopback pulse to arrive
7. Read measured latency value from latency register
8. Write 0 into stimulus register to release SDA line.

Note that the latency value obtained after this measurement represents the round-trip latency, from WIB to COLDATA and back to WIB. The extra latency added by WIB and COLDATA logic is negligible. The latency value is measured in the units of system clock (16 ns).

An example cable latency measurement code is provided in the git repository, at the following path:

soft\_debug\_p3/sw/src/cable\_latency.cxx

## 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 . 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 . 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 | 15:0 | | ts\_addr | | Timing point address | |
| 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. | |
| A00C0004 | | 14 | | coldata\_rxbufreset | Reserved, don’t use, write 0 | |
| A00C0004 | | 15 | | csd\_reset | Reserved, don’t use, write 0 | |
| A00C0004 | | 16 | | ts\_clk\_sel | System clock source selection: 0 = CDR recovered clock(default) 1 = PLL clock synchronized with CDR or running independently if CDR clock is missing. PLL clock should only be used on test stand when timing master is not available. | |
| A00C0004 | | 18 | | ps\_en\_in | Fine clock phase adjustment for COLDATA I2C interfaces. One step = ~15 ps. | |
| A00C0004 | | 19 | | mon\_adc\_start | Set this bit to 1 and then to 0 to start monitoring ADC conversion and serial readout. All 4 ADCs are converting simultaneously. | |
| A00C0004 | | 20 | | crc\_err\_reset | Reset of COLDATA CRC sticky error flags | |
| A00C0004 | | 21 | | raw\_channel\_map | 0 = UVX map, 1 = raw ADC channel map | |
| 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, 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 | 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. | |
| A00C000C | 2 | | cmd\_stamp\_sync\_en | | Enable issuing SYNC FAST command when bits [14:0] of the DTS time stamp match cmd\_stamp\_sync register value | |
| A00C000C | 3 | | align\_en | | Enable automatic alignment of the COLDATA data to the DTS time stamp | |
| A00C000C | 15:8 | | dts\_time\_delay | | DTS time stamp delay. This delay is measured in 8 ns units. It should be longer than maximum COLDATA RX link latency (see RXL parameter). | |
| A00C000C | 30:16 | | cmd\_stamp\_sync | | If cmd\_stamp\_sync\_en == 1, the SYNC FAST command will be issued when bits [14:0] of the DTS time stamp match cmd\_stamp\_sync register value | |
| 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 | |
| A00C0014 | | 24 | | cmd\_en\_idle | | Timing system command enable for IDLE |
| A00C0014 | | 25 | | cmd\_en\_edge | | Timing system command enable for EDGE |
| A00C0014 | | 26 | | cmd\_en\_sync | | Timing system command enable for SYNC |
| A00C0014 | | 27 | | cmd\_en\_act | | Timing system command enable for ACT |
| A00C0014 | | 28 | | cmd\_en\_reset | | Timing system command enable for RESET |
| A00C0014 | | 29 | | cmd\_en\_adc\_reset | | Timing system command enable for ADC RESET |
| A00C0014 | | 30 | | cmd\_en\_trigger | | Timing system command enable 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. | |
| A00C0020 | 8:1 | | context\_fld | | Context ID field in DAQ readout. | |
| A00C0024 | 17:0 | | spy\_rec\_time | | Spy ring buffer recording time, in 32-bit words, based on FELIX fabric clock, 240.48MHz | |
| A00C0028 | 15:0 | | flex | | Flex field in DAQ readout | |
| A00C0028 | 29:22 | | femb\_pulser\_in\_frame | | Femb\_pulser\_in\_frame field in DAQ readout | |
| A00C0028 | 30 | | ready | | Ready field in DAQ readout | |
| A00C0028 | 31 | | ws | | Ws field in DAQ readout | |
| 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 | |
| A00C0034 | 5:0 | | link[0] | | Link[0] field in DAQ readout | |
| A00C0034 | 11:6 | | link[1] | | Link[1] field in DAQ readout | |
| A00C0034 | 21:12 | | crate\_id | | Crate\_ID field in DAQ readout | |
| A00C0034 | 27:22 | | det\_id | | Det\_ID field in DAQ readout | |
| A00C0034 | 31:28 | | psr\_cal | | Psr\_cal field in DAQ readout | |
| A00C0038 | 0 | | felix\_reset\_pll\_datapath | | Reserved, don’t use | |
| A00C0038 | 4 | | felix\_reset\_tx\_datapath | | Reserved, don’t use | |
| A00C0038 | | 5 | | felix\_reset\_rx\_datapath | | Reserved, don’t use |
| A00C0038 | | 6 | | felix\_rx\_reset | | Reset all FELIX TX+RX logic |
| A00C0038 | 9:8 | | felix\_tx\_8b10b\_en | | FELIX TX enable 8b/10b encoding. These bits should always be set to ‘b11 for normal operation | |
| A00C003C | 0 | | dac\_src\_sel[0] | | DAC0 source selector. 0 = external 1 = internal | |
| A00C003C | 1 | | dac\_src\_sel[1] | | DAC1 source selector. 0 = external 1 = internal | |
| A00C003C | 2 | | dac\_src\_sel[2] | | DAC2 source selector. 0 = external 1 = internal | |
| A00C003C | 3 | | dac\_src\_sel[3] | | DAC3 source selector. 0 = external 1 = internal | |
| A00C003C | 4 | | mon\_vs\_pulse\_sel | | Monitor vs pulse selector.  0 = Monitor, 1 = Pulse | |
| A00C003C | 5 | | inj\_cal\_pulse | | Inject Calibration pulse switch.  Set to 1 and then to 0 to generate pulse | |

Table . 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**

### Using 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 DTS. The exact codes of DTS commands corresponding to each FAST command are programmable using the following registers:

|  |  |
| --- | --- |
| **Command code** | **Enable flag** |
| cmd\_code\_idle | cmd\_en\_idle |
| cmd\_code\_edge | cmd\_en\_edge |
| cmd\_code\_sync | cmd\_en\_sync |
| cmd\_code\_act | cmd\_en\_act |
| cmd\_code\_reset | cmd\_en\_reset |
| cmd\_code\_adc\_reset | cmd\_en\_adc\_reset |
| cmd\_code\_trigger | cmd\_en\_trigger |

Table . Timing system command codes and enable flags

These registers support 8-bit command codes. Default values for all of them are zeros. Each command is disabled by default. To enable any of the commands, perform the following steps:

* Write 8-bit command code into Command code register
* Write 1 into corresponding Enable flag register

### Using fake\_daq\_stream mode

In the fake\_daq\_stream mode, the data frames are generated one after another immediately, without any consideration of proper timing. Each WIB is doing it independently. Therefore, the timing between the DAQ frames in fake\_daq\_stream mode does not carry any significance and should not be used to make any conclusions. It may be different from WIB to WIB. This mode should only be used for FELIX link validation in the absence of FEMBs attached to WIB.

### 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 | 5 | ts\_rdy | ts\_rdy signal from timing endpoint |
| A00C0090 | 15:8 | 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 | 19 | mon\_adc\_busy | 1 = Monitoring ADCs busy. Start conversion only when this flag = 0. |
| 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 |
| A00C00A4 | 63:32 | ts\_tstamp[63:32] | Timing point time stamp bits 63:32 |
| A00C00A8 | 7:0 | align\_delay[0] | alignment delay for link 0, automatically calculated |
| A00C00A8 | 15:8 | align\_delay[1] | alignment delay for link 1, automatically calculated |
| A00C00A8 | 23:16 | align\_delay[2] | alignment delay for link 2, automatically calculated |
| A00C00A8 | 31:24 | align\_delay[3] | alignment delay for link 3, automatically calculated |
| A00C00AC | 7:0 | align\_delay[4] | alignment delay for link 4, automatically calculated |
| A00C00AC | 15:8 | align\_delay[5] | alignment delay for link 5, automatically calculated |
| A00C00AC | 23:16 | align\_delay[6] | alignment delay for link 6, automatically calculated |
| A00C00AC | 31:24 | align\_delay[7] | alignment delay for link 7, automatically calculated |
| A00C00B0 | 7:0 | align\_delay[8] | alignment delay for link 8, automatically calculated |
| A00C00B0 | 15:8 | align\_delay[9] | alignment delay for link 9, automatically calculated |
| A00C00B0 | 23:16 | align\_delay[10] | alignment delay for link 10, automatically calculated |
| A00C00B0 | 31:24 | align\_delay[11] | alignment delay for link 11, automatically calculated |
| A00C00B4 | 7:0 | align\_delay[12] | alignment delay for link 12, automatically calculated |
| A00C00B4 | 15:8 | align\_delay[13] | alignment delay for link 13, automatically calculated |
| A00C00B4 | 23:16 | align\_delay[14] | alignment delay for link 14, automatically calculated |
| A00C00B4 | 31:24 | align\_delay[15] | alignment delay for link 15, automatically calculated |
| A00C00B8 | 1:0 | crc\_err[0] | CRC error sticky flags for COLDATA link 0, ADC[1:0] |
| A00C00B8 | 3:2 | crc\_err[1] | CRC error sticky flags for COLDATA link 1, ADC[1:0] |
| A00C00B8 | 5:4 | crc\_err[2] | CRC error sticky flags for COLDATA link 2, ADC[1:0] |
| A00C00B8 | 7:6 | crc\_err[3] | CRC error sticky flags for COLDATA link 3, ADC[1:0] |
| A00C00B8 | 9:8 | crc\_err[4] | CRC error sticky flags for COLDATA link 4, ADC[1:0] |
| A00C00B8 | 11:10 | crc\_err[5] | CRC error sticky flags for COLDATA link 5, ADC[1:0] |
| A00C00B8 | 13:11 | crc\_err[6] | CRC error sticky flags for COLDATA link 6, ADC[1:0] |
| A00C00B8 | 15:14 | crc\_err[7] | CRC error sticky flags for COLDATA link 7, ADC[1:0] |
| A00C00B8 | 17:16 | crc\_err[8] | CRC error sticky flags for COLDATA link 8, ADC[1:0] |
| A00C00B8 | 19:18 | crc\_err[9] | CRC error sticky flags for COLDATA link 9, ADC[1:0] |
| A00C00B8 | 21:20 | crc\_err[10] | CRC error sticky flags for COLDATA link 10, ADC[1:0] |
| A00C00B8 | 23:22 | crc\_err[11] | CRC error sticky flags for COLDATA link 11, ADC[1:0] |
| A00C00B8 | 25:24 | crc\_err[12] | CRC error sticky flags for COLDATA link 12, ADC[1:0] |
| A00C00B8 | 27:26 | crc\_err[13] | CRC error sticky flags for COLDATA link 13, ADC[1:0] |
| A00C00B8 | 29:28 | crc\_err[14] | CRC error sticky flags for COLDATA link 14, ADC[1:0] |
| A00C00B8 | 31:30 | crc\_err[15] | CRC error sticky flags for COLDATA link 15, ADC[1:0] |
| 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 |
| A00C00C4 | 13:0 | mon\_adc\_val[0] | Monitoring ADC measurement result, chip U100 |
| A00C00C4 | 29:16 | mon\_adc\_val[1] | Monitoring ADC measurement result, chip U103 |
| A00C00C8 | 13:0 | mon\_adc\_val[2] | Monitoring ADC measurement result, chip U104 |
| A00C00C8 | 29:16 | mon\_adc\_val[3] | Monitoring ADC measurement result, chip U105 |

Table . Status registers

## Timing Endpoint

This is a timing endpoint module. The only parameter that needs programming is the Address. See **ts\_addr** register 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 SFP 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 (DTS). How to operate:

### Preliminary steps

* Program **cmd\_code\_trigger** register with the DTS 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 DTS

1. Disable external triggers
2. Reset the spy memory, by 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 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 DAQ metadata header and trailer).

## Monitoring ADCs

Using monitoring ADCs:

1. Start conversion by setting **mon\_adc\_start = 1** and then **= 0**
2. Wait until **mon\_adc\_busy == 0**
3. Read out measured values from **mon\_adc\_val[3:0]** registers

An example script for reading monitoring ADCs is available here:

soft\_debug\_p3/mon\_adc\_read.sh

## 10 MHz reference clock on P12 connector

The firmware is using PL\_CLK1 CPU IP output to generate 10MHz clock for P12 connector. Normally, PL clocks are configured by First Stage Boot Loader (FSBL). However, Petalinux 2020.1 used for WIB project generates invalid FSBL. To work around this issue, we are using FSBL generated by Petalinux 2019. That 2019 FSBL was generated from a skeleton firmware that does not have PL\_CLK1 configured. Therefore, the PL\_CLK1 output must be configured manually at this time, using CPU configuration register. The operation is very simple:

Write value of 0x1033200 into register 0xff5e00c4

An example script that performs this operation is available here:

soft\_debug\_p3/pl\_clk1\_en.sh

# 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.

Diagram

Description automatically generated with medium confidence

Figure . WIB data path clock schematics.

The data arrive from COLDATA via 1.25 Gbps (COLDATA p3) serial links. The reference clocks for these links are generated from the system clock in the COLDATA chips. Therefore, the data from the serial receivers can be read using the 62.5 MHz system clock. The data is then transferred into the 125MHz clock domain, which is double the frequency of the system clock. 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 deframed data are passed to DAQ Frame builder, which builds the data frames for DAQ transmission using the same 125 MHz clock. The prepared DAQ data frames are then transferred into FELIX transmitter clock domain using a FIFO-based 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 125 MHz clock domain before it can be used by DAQ Frame builder. This is achieved by using a simple register-based CDC module.

# DAQ readout format

This section describes updates to the metadata in the DUNE Cold Electronics data format. The metadata is filled by the Frame Builder firmware module. The metadata definitions in this document replace any in previous documents.

## Introduction

A data frame from the DUNE Front-End 3ASIC electronics includes 2 FEMB’s worth of 14-bit ADC values (128 channels from each FEMB), as well as 15 bits of time stamp from each COLDATA chip (2 per FEMB). A 16th bit in each time stamp from the COLDATA chips is set whenever a calibration pulse is issued, which serves to mark a particular frame as “including a pulse.” Metadata that accompanies the ADC values measured by the front-end motherboards of the DUNE 3ASIC cold electronics must do the following things:

* Allow unpacking and interpretation of the data by online and offline analyses. It must therefore include the nominal DUNE 64-bit time stamp for each frame, local status and error codes, and data integrity checks (e.g., CRC20).
* Report on any *dynamic* configuration changes that do not come from the DAQ system’s Control, Configuration, and Monitoring (CCM) service. These include, for example, DAC settings during a calibration run. These are effectively “conditions” metadata.
* Allow for higher-level checks of data quality, through, for example, comparisons of local and global time stamps.

Note that *static* configuration information---what the gain settings are for FEMB channels, or shaper settings, etc.---do not need to be included in the metadata because they already will be stored in the DAQ configuration data base which can be accessed by both online and offline analysis code.

The Cold Electronics metadata falls into two classes: fixed-definition bits, like the 64-bit time stamp, and “flex” bits, whose definition is context-dependent.

## Metadata Header

The header for the Cold Electronics Metadata is shown below:





Figure . Metadata header. FELIX Start of Frame Word is shown separately on top

Tables below detail each section of the header metadata.

### FELIX Start of Frame Word

|  |  |
| --- | --- |
| **Bits** | **Description** |
| 7:0 | “Start of Frame” K28.1 character |
| 31:8 | Zeros |

Table . Start of Frame Word bit fields

### Word 0

|  |  |  |
| --- | --- | --- |
| **Bits** | **Description** | **Register name** |
| 7:0 | DAQ format version, hardcoded | version |
| 11:6 | Det ID, programmable via register | det\_id |
| 21:12 | Crate ID, programmable via register | crate\_id |
| 25:22 | Slot number, read out from backplane pins |  |
| 31:26 | Link number, programmable via register | link[0] or link[1] |

### Words 1 and 2

These words hold the full DUNE 64-bit time stamp. This time stamp comes directly from the timing endpoint on the WIB itself, and thus represents the time the frame arrived at the WIB (thus distinct from the COLDATA time stamp, which is when the sample was taken by the ADCs on the FEMB). The lower-order 32 bits are in Word 1 and the upper bits are in Word 2.

### Word 3

|  |  |
| --- | --- |
| **Bits** | **Description** |
| 12:0 | Reserved, set to zero |
| 15:13 | COLDATA Time Stamp ID shows which COLDATA’s local time stamp is transmitted in this frame. |
| 17:16 | **Currently not implemented**  FEMB valid flags indicate which of the FEMB’s data are valid in this frame. Each DAQ output link carries data from two FEMBs:   |  |  |  | | --- | --- | --- | | **DAQ link** | **FEMBs transmitted** | **FEMB valid flag bit** | | 0 | 0 | 16 | | 0 | 1 | 17 | | 1 | 2 | 16 | | 1 | 3 | 17 | |
| 25:18 | Link mask, indicates which FEMB serial links are valid. Bits [3:0]=FEMB0, bits [7:4]=FEMB1 |
| 26 | Loss of Lock bit from the on-board PLL. |
| 31:27 | Reserved, set to zero |

Table . Word 3 bit fields.

### Word 4

|  |  |  |
| --- | --- | --- |
| **Bits** | **Description** | **Register name** |
| 7:0 | Pulser in Frame bits, programmable via register | femb\_pulser\_in\_frame |
| 15:8 | FEMB synchronization flags. These flags indicate which of the data links (if any) are receiving data with the local time stamp not matching the DTS time stamp bits[14:0] |  |
| 30:16 | COLDATA time stamp. As all the COLDATA chips should be synchronized (and, if not, the synchronization flags will be set high) we need only one. This time stamp can be different than the WIB time stamp because of the round trip delay time to the FEMB. It will also be re-synchronized by using the DUNE Timing System’s SYNC pulses, which occur every 1 s. The number of the link from which the time stamp is taken rotates from frame to frame, and can be found in Word 3, bits [15:13]. |  |
| 31 | Reserved, currently zero |  |

Table . Word 1 bit fields.

## Metadata Trailer

The trailer for the Cold Electronics Metadata is shown below





Figure . Metadata trailer. Word with index 117 is the last word in data frame, word with index 0 is FELIX frame trailer. Word with IDLE symbol is a filler that’s transmitted between frames.

Below we detail each section of the trailer metadata.

### Word 117

|  |  |  |
| --- | --- | --- |
| **Bits** | **Description** | **Register name** |
| 15:0 | “Flex bits”, context-dependent. For example, during a pulser calibration run, these bits can include the setting of the DAC used to generate the pulse height. Programmable via register. | flex |
| 17:16 | Reserved, filled with zeros |  |
| 18 | “WIB Synchronization” bit reports whether the WIB’s timing endpoint is synchronized with the master DUNE Timing System clock, beyond the PLL lock which is already included in the LOL bit of the header. |  |
| 22:19 | These bits indicate that pulser calibration is ongoing. There is one bit for each COLDATA chip (2 per FEMB). These bits are distinct from the “FEMB Pulser-In-Frame” bits of Word 2 because they persist for as long as the pulse calibration lasts. They are set when the WIB server initiates a pulse calibration (as directed by DAQ CCM) and then unset when the calibration is completed. Programmable via register. | psr\_cal |
| 23 | “WIB not ready” bit. This bit is set to 1 when the WIB is undergoing a configuration transition, including power-up, transition to calibrations, etc. When not in transition, the bit is zero. Programmable via register. | ready |
| 31:24 | These 8 bits hold a “Context Code” that includes any dynamic re-configurations (thus, condition changes) that may happen on the WIB. So, for example, particular calibration modes will be encoded here, and then that context can be used to cast the flex bits to include information like the current DAC setting. For static configurations (ie, normal running), the Configuration ID may be redundant with information in the configuration data base. Programmable via register. | context\_fld |

Table . Word 117 bit fields.

### FELIX trailer word 0

|  |  |
| --- | --- |
| **Bits** | **Description** |
| 7:0 | K.28.6 code used as “End of Frame” by FELIX. |
| 27:8 | CRC-20 checksum |
| 31:28 | Reserved, filled with zeros |

Table . Felix trailer word 0 bit fields.

# Migration from COLDATA p2 to p3 chips

COLDATA p3 chips are very similar to p2 but have certain differences. Additionally, FEMB design (a.k.a. “monolithic FEMB”) for p3 chips is different. The following changes are necessary to accommodate p3 chips:

1. I2C links’ “C2W” and “W2C” pins are swapped to accommodate new FEMB cable pin assignment. This change is transparent for software. The WIB firmware project now has new synthesis and implementation runs, named synth\_zu6\_mono and impl\_zu6\_mono respectively. These runs implement firmware for COLDATA p3 chips and corresponding monolithic FEMB design.
2. The COLDATA p3 chips are transmitting data to WIB at 1.25 Gbps bit rate, while p2 chips used 1.28 Gbps rate. To accommodate the 1.25 Gbps, the WIB MGT reference clock was programmed to 125 MHz (was 128 MHz for p2 chips). The firmware did not need to be changed.
3. Modular FEMBs with p2 chips had separate I2C lines to and from each COLDATA chip. Monolithic FEMBs with p3 chips have a single I2C line for both COLDATA chips. The software has to be modified accordingly. See table below for examples.

|  |  |  |
| --- | --- | --- |
| **Operation** | **COLDATA p2** | **COLDATA p3** |
| Write to chip TOP | femb->i2c\_write (0,2,0,1,0); | femb->i2c\_write (0,3,0,1,0); |
| Write to chip BOTTOM | femb->i2c\_write (1,2,0,1,0); | femb->i2c\_write (0,2,0,1,0); |

Note that for p2 chips, the chip is selected by passing first argument (chip index) to i2c\_write and i2c\_read functions, and the second argument (chip address) is always =2. For p3 chips, chip index is always =0 because all communication is performed via chip 0. Chip 0 (TOP) is selected by passing second argument (chip address) = 3, and Chip 1 (BOTTOM) is selected by passing chip address = 2.  
These changes have to be implemented in software. The examples given in the table above are showing i2c functions already implemented in WIB software. Only calls to i2c\_write function are shown; calls to i2c\_read functions have to be modified identically.

1. Similarly, access to ADC chips has to be modified as well. All exchange with ADC chips is now done via a single I2C interface. Chip index parameter is always =0, and ADC chips are selected using the following chip address parameters:

|  |  |
| --- | --- |
| **COLDATA chip** | **Addresses of ADC chips** |
| TOP | 8, 9, 10, 11 |
| BOTTOM | 4, 5, 6, 7 |

1. On the modular FEMB design for COLDATA p2 chips one of the “C2W” I2C lines has been inverted by swapping wires in the differential pair. This inversion was reversed using software.   
   This error was corrected in the monolithic FEMB design for p3 chips. Correspondingly, the reverse inversion has to be removed in software. Please see i2c\_read function for details.
2. COLDATA p2 chips required programming a number of registers controlling internal PLL to Warm or Cold settings in order to be correctly initialized. PLL in p3 chips works in cold and warm conditions with the default settings, so PLL registers don’t have to be programmed anymore.

The firmware repository contains two directories with the debugging software:

|  |  |
| --- | --- |
| **Directory** | **Description** |
| soft\_debug\_p2 | Debugging software for COLDATA p2 chips and modular FEMB |
| soft\_debug\_p3 | Debugging software for COLDATA p3 chips and monolithic FEMB  Changes relative to p2:   * femb\_test.cxx: the PLL register programming is commented out * femb\_text.cxx: the chips indexes and addresses are modified as shown in step 3 above. * femb.cc: in i2c\_read function, the data inversion is removed * adc\_test.cxx: ADC chip addresses modified as shown in step 4 above. * The external PLL configuration script in clock directory is modified to use PLL setup with 125 MHz reference clock for COLDATA RX MGTs |

# Migration from CDR-based to DCSK timing endpoint

The timing team has provided a firmware module for interfacing the DCSK timing master. That module has been integrated into firmware starting with the following firmware time stamp:

2022-10-25 14:56:42

The implementation is shown in the figure below:



The following changes have to be implemented in order to accommodate the DCSK timing:

1. The timing endpoint address (ts\_addr register) is now a 16-bit value, and ts\_tgrp register was removed. Rework software accordingly.
2. Rework timing command handling code as shown in “Using Timing system command codes” section
3. DTS master must be reprogrammed to send DCSK data stream
4. On-board PLL configuration must be changed according to the following file (path in repository is shown):  
   clock/coldata\_p3\_DCSK/Si5344-RevD-wib62.5\_DCSK-Registers.h

An example PLL programming script is available here:

soft\_debug\_p3/clock/si5345\_config.c

# COLDATA data time stamp synchronization

The internal logic of each COLDATA chip is synchronized using two reset signals:

1. The “2MHz” clock reset is performed by using EDGE fast command. This is necessary so all ADCs start digitization sequence at precisely the same system clock period
2. The internal 15-bit time stamp in each COLDATA chip is reset using SYNC fast command. This operation should guarantee that all ADC measurements performed at the same time are marked with the same 15-bit time stamp. These time stamps increment on each system clock and should stay synchronized to each other after SYNC fast command.

WIB receives ADC data from COLDATA chips via 1.25 Gbps serial links. There are two such links per COLDATA chip. Receivers for these links are implemented in WIB using Multi-Gigabit Transceivers (MGTs) in Xilinx FPGA. One feature of the MGTs is that the precise latency of data passing via the MGTs is not guaranteed. The latency of each MGT may vary by a few system clocks from one initialization to the next. Differences in cable lengths also lead to changes in latency from one WIB to another. Even temperature changes can affect that latency, due to slight delay changes in semiconductors and cables.

As a result, the data frames received from each serial link may come at slightly different times.

The obvious problem is:

* The 64-bit DTS time stamp is wide enough to represent a very long period of time (~9300 years or so), but it does not match the digitization moment precisely
* The 15-bit COLDATA time stamps do match the precise digitization moment, but they wrap around every ~524 uS.

It is very desirable to have a single time stamp that is long enough to cover the duration of the experiment, and at the same time is precise enough to carry exact information about digitization time of every ADC sample. The section below describes the time stamp alignment logic implemented in firmware.

## Firmware time stamp alignment logic

Listed below are critical latency parameters that affect the time stamp synchronization logic:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Location** | **Description** | **Value (in units of system clocks, 16 ns)** |
| CAB | High-speed cable | WIB-to-FEMB high-speed cable latency | 0..8 clocks, must be measured |
| FL\_WIB | WIB | FAST command logic latency in WIB | 19 clocks |
| RX\_WIB | WIB | WIB serial receivers’ maximum latency | Compensated automatically by alignment |
| TS\_CLD | COLDATA chip | COLDATA time stamp reset logic latency | 1 clock (?) |
| TX\_CLD | COLDATA chip | COLDATA serial transmitters latency | Compensated automatically by alignment |

It could be good to know the values of all these parameters individually, but in reality, we only need to know certain combinations:

|  |  |  |
| --- | --- | --- |
| **Name** | **Details** | **Description** |
| FASTL | FL\_WIB + CAB + TS\_CLD | Total latency of the propagation of SYNC FAST command from WIB to COLDATA |
| RXL | TX\_CLD + CAB + RX\_WIB | Total maximum latency of the data transmission from COLDATA to WIB |

FASTL parameter can be measured as detailed in COLDATA P3 datasheet, last page. The logic for this measurement is implemented in firmware. See section “Data cable latency measurement” above for the details of this procedure.

RXL delay considerations (Dave Christian’s email 2022-05-27):  
“The maximum length cables that we will have will be the cables for FEMBs located at the bottom of the Vertical Drift Far Detector (& for the Vertical Drift ProtoDUNE-2). These will consist of a 25m length of Samtec twinax and a 2.5m length of 3M miniSAS twinax. I don’t know the exact propagation speed of signals on either type of cable, but I think 2/3 of the speed of light is a safe guess (it could be a bit faster). I will use 8 inches per nsec and assume this is the same for the two types of cable. 27.5 m = 1083 inches 🡪 cable delay (1 way) = 135 ns = ~8 system clocks.

The shortest cables that will be used in DUNE are the 9m long Samtec cables that will be used for the FEMBs located at the top of Far Detector 1. However, we will probably use 2m miniSAS cables in ICEBERG and we might use even shorter cables for bench tests. I estimate that the cable delay (1 way) for a 2m miniSAS cable is = 78.74 inches / 8 inches/ns = 9.8 ns = 0 system clocks”

All the above parameters should be measured in system clock units (16 ns) to be used in the WIB logic.

### Time stamp synchronization procedure details

1. Send SYNC fast command when DTS time stamp bits [14:0] == -FASTL (in 2’s complement format). That makes the COLDATA internal time counter reset exactly when DTS time stamp bits [14:0] == 0. Therefore, the COLDATA 15-bit time stamp should now be synchronous with lower bits of DTS time stamp.
2. Since DTS time stamp bits [14:0] wrap around every ~524 uS, the SYNC command will be sent with that period, constantly reinforcing the synchronization between DTS and COLDATA time stamps.

### Time stamp processing in WIB

The 15-bit COLDATA time stamps arriving with the data frames will necessarily be delayed relative to DTS time stamp’s lower bits, due to the RXL latency. Moreover, the COLDATA data frames may arrive not exactly at the same time from different data links, due to latency variations. These data frames should carry identical 15-bit time stamps though. The steps listed below show the procedure of aligning the data frames with the correct DTS 64-bit time stamp:

1. Delay the DTS time stamp in firmware by a number of system clocks slightly exceeding RXL
2. Delay each COLDATA frame by the number of clocks calculated individually for each link so that their 15-bit time stamps match exactly the lower 15 bits of the delayed DTS time stamp. The delay calculation for each links is done in firmware, automatically.
3. Compare each delayed COLDATA 15-bit time stamp with delayed DTS time stamp and make sure they match. If they don’t match, flag an error.
4. The delayed COLDATA frames now can be sent to Frame Builder along with the delayed DTS time stamp. This makes sure that the DTS time stamp in each DAQ data frame carries the exact digitization time of the data in that frame.

This procedure effectively cancels the latency of the transmission of the COLDATA frames from COLDATA chips to WIB, relative to DTS time stamp. Note that the above logic is specifically using data delays even for DTS time stamp (even though it behaves like a simple counter) instead of simpler logic based on addition and/or bitwise operations, to avoid errors or even any perceived appearance of the direct manipulation of the time stamps. The logic only uses original time stamps from DTS and COLDATA by delaying and comparing them.

### Alignment setup procedure

1. Enable 2 MHz clock edge synchronization by setting cmd\_code\_edge = 0x10. This will configure WIB to issue EDGE FAST command on each time stamp update command, that comes once per second.
2. Measure CAB latency as detailed in “Data cable latency measurement” section. Note that the result you get is CAB\*2 (round trip). Calculate FASTL.
3. Calculate 15-bit 2’s complement of FASTL value. Simple way to do it in C:  
   fastl\_compl = (-fastl) & 0x7fff;
4. Write that value into cmd\_stamp\_sync register.
5. Enable SYNC command by setting cmd\_stamp\_sync\_en = 1. The firmware is now sending SYNC command every time the DTS time stamp bits [14:0] == cmd\_stamp\_sync. This ensures that the COLDATA time stamps become zeros at the same system clock cycle when DTS time stamp bits [14:0] are zeros.
6. Set DTS time stamp delay for alignment by writing the delay value into dts\_time\_delay register. Preliminarily, the value of 0x50 seems to be optimal, even though that may need to be adjusted.
7. Enable automatic COLDATA data alignment by setting align\_en = 1.
8. Read automatic alignment delays from align\_delay[\*] registers. Make sure each of them is less or equal to 0x1f. If any of them exceed that value, the dts\_time\_delay parameter should be increased. Do not analyze the delays from the links connected to missing FEMBs, those can show arbitrary values.

An example alignment setup code is provided in the git repository, at the following path:

soft\_debug\_p3/align.sh

Diagram

Description automatically generated

Figure . Data latency and time stamp alignment diagram.

Graphical user interface, text

Description automatically generated

Figure . Time stamp alignment logic structure.

# 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. COLDATA datasheet
5. COLDADC datasheet
6. DUNE Timing System – Single Phase Firmware
7. FELIX manual
8. Ultrascale + ZYNQ manual
9. Ultrascale + GTH manual
10. 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 DTS time stamp reclocking FIFO, but this did not require changes in this document.  Replaced reference to DAQ format 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 |
| 2021-10-23 | Added DAQ format section from Josh’s document, reformatted, fixed a few errors.  To do:   * summary of the complete DAQ frame * self-synchronization procedure |
| 2022-02-22 | Reworked DAQ format according to JK’s message from 2021-12-06  Added registers for programming the new DAQ format fields |
| 2022-04-05 | Added section on COLDATA p3 and monolithic FEMB migration  Reworked text to show differences between p2 and p3 chips. |
| 2022-05-16 | Added section on using fake DAQ stream mode. Converted some of the sub-headers into proper format. |
| 2022-06-04 | * Clock domain diagram is reworked to show changes for COLDATA P3 chips. * The clock network description is also reworked to match the diagram. * Removed section about bugs in COLDATA P1 I2C logic – not relevant for P3 chips * Replaced all references to TLU with DTS (Detector Timing System) * Added section on data cable latency measurement * Added section on COLDATA data time stamp synchronization logic and procedure * Moved ts\_tstamp[63:32] status register to different address * Added a number of new registers supporting the new firmware features |
| 2022-06-06 | * Updated ADC channel mapping in firmware for monolithic FEMB. No document changes are needed for this. * Removed “version” register. The DAQ version is now hardcoded to 4, for the version with the monolithic FEMB channel mapping |
| 2022-06-23 | Fixed mistake in instructions for cable latency measurement |
| 2022-07-15 | * Added missing felix\_rx\_reset register * Reworked description for other FELIX registers * Added system clock source selection control (ts\_clk\_sel) |
| 2022-08-09 | Added section about precise I2C timing |
| 2022-08-23 | New additions:   * 10MHz clock output on connector P12 * Switches for DAC source selector * Monitor vs pulse selector * Calibration pulse injection switch * Monitor ADC readout * Masked unused high bit in the slot number reported to DAQ * COLDATA serial links CRC flags |
| 2022-10-18 | Expanded description of the FEMB valid bits in DAQ frames |
| 2022-10-25 | Updated for the DCSK timing endpoint. In particular, command codes are now 8-bit long, so needed to add enable flags for each command. Added section on migrating to DCSK |
| 2022-10-26 | Added DCSK timing implementation diagram, removed deprecated registers. Added note about ts\_addr into DCSK migration section. |
| 2022-11-11 | Implemented fine phase adjustment between system 62.5MHz clock and COLDATA I2C clock. Tested with cables of various lengths. Added section on how to use it. |
| 2022-11-12 | Added ADC channel mapping switching, controlled by “raw\_channel\_map” parameter |