DUNE WIB firmware

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

Updated: 2024-01-25

Table of Contents

[1 Introduction 3](#_Toc145270207)

[2 WIB firmware location 5](#_Toc145270208)

[3 Building WIB firmware 5](#_Toc145270209)

[4 Firmware structure 5](#_Toc145270210)

[4.1 COLDATA receivers 6](#_Toc145270211)

[4.2 COLDATA Frame decoders 7](#_Toc145270212)

[4.2.1 COLDATA CRC error flags 7](#_Toc145270213)

[4.3 DAQ Frame Builders 7](#_Toc145270214)

[4.4 10Gps ethernet transmitters 7](#_Toc145270215)

[4.5 ZYNQ CPU module 7](#_Toc145270216)

[4.6 COLDATA I2C interface 8](#_Toc145270217)

[4.6.1 I2C Write operation 9](#_Toc145270218)

[4.6.2 I2C Read operation 9](#_Toc145270219)

[4.6.3 COLDATA and COLDADC I2C fine timing requirements 9](#_Toc145270220)

[4.6.4 Using COLDATA and COLDADC I2C fine clock phase adjustment 10](#_Toc145270221)

[4.6.5 Data cable latency measurement 11](#_Toc145270222)

[4.7 COLDATA FAST command generator 11](#_Toc145270223)

[4.7.1 How to use FAST command generator 12](#_Toc145270224)

[4.8 I2C interface for WIB on-board devices 12](#_Toc145270225)

[4.9 PTC I2C access 12](#_Toc145270226)

[4.9.1 Handling I2C arbitration on the multi-master buses 13](#_Toc145270227)

[4.9.2 How to use PTC I2C access to WIB 14](#_Toc145270228)

[4.9.3 WIB software modifications needed for multi-master I2C buses 14](#_Toc145270229)

[4.10 Control and status registers 14](#_Toc145270230)

[4.10.1 Control registers (read/write) are listed in Table 6: 14](#_Toc145270231)

[4.10.2 How to set initial value for fake time stamp (FTS): 18](#_Toc145270232)

[4.10.3 Using Timing system command codes 18](#_Toc145270233)

[4.10.4 Using fake\_daq\_stream mode 18](#_Toc145270234)

[4.10.5 Status registers (read-only): 18](#_Toc145270235)

[4.11 Timing Endpoint 20](#_Toc145270236)

[4.12 DAQ spy memory 20](#_Toc145270237)

[4.12.1 Preliminary steps 21](#_Toc145270238)

[4.12.2 Operation with the trigger distributed by DTS or supplied via P11 connector 21](#_Toc145270239)

[4.12.3 Operation with the software trigger 22](#_Toc145270240)

[4.13 Monitoring ADCs 22](#_Toc145270241)

[4.14 System Monitor 22](#_Toc145270242)

[4.15 Calibration logic 23](#_Toc145270243)

[4.15.1 Calibration DAC 23](#_Toc145270244)

[4.15.2 Calibration Pulse generator 23](#_Toc145270245)

[4.16 10 MHz reference clock on P12 connector 23](#_Toc145270246)

[5 Clock regions 24](#_Toc145270247)

[6 DAQ readout format 25](#_Toc145270248)

[6.1 Introduction 25](#_Toc145270249)

[6.2 HERMES firmware module 26](#_Toc145270250)

[6.2.1 HERMES reset input logic 26](#_Toc145270251)

[6.3 Data frames 27](#_Toc145270252)

[6.4 Header considerations 27](#_Toc145270253)

[6.5 Data samples packing 28](#_Toc145270254)

[6.6 Data block format 29](#_Toc145270255)

[6.7 Header Word 1 29](#_Toc145270256)

[6.8 Header Word 2 30](#_Toc145270257)

[6.9 Header Word 3 30](#_Toc145270258)

[6.10 Bandwidth calculation 30](#_Toc145270259)

[7 Migration from COLDATA p2 to p3 chips 31](#_Toc145270260)

[8 Migration from CDR-based to DCSK timing endpoint 32](#_Toc145270261)

[9 COLDATA data time stamp synchronization 33](#_Toc145270262)

[9.1 Firmware time stamp alignment logic 33](#_Toc145270263)

[9.1.1 Time stamp synchronization procedure details 34](#_Toc145270264)

[9.1.2 Time stamp processing in WIB 34](#_Toc145270265)

[9.1.3 Alignment setup procedure 35](#_Toc145270266)

[10 References 36](#_Toc145270267)

[11 Revision table 37](#_Toc145270268)

# 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.

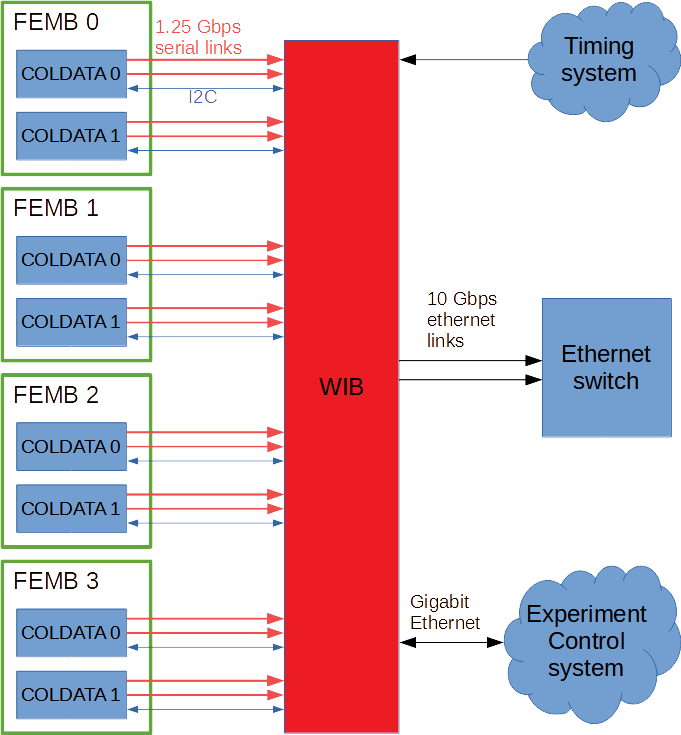


Figure . WIB connections in the DUNE system.

# 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 2022.2.

# 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.

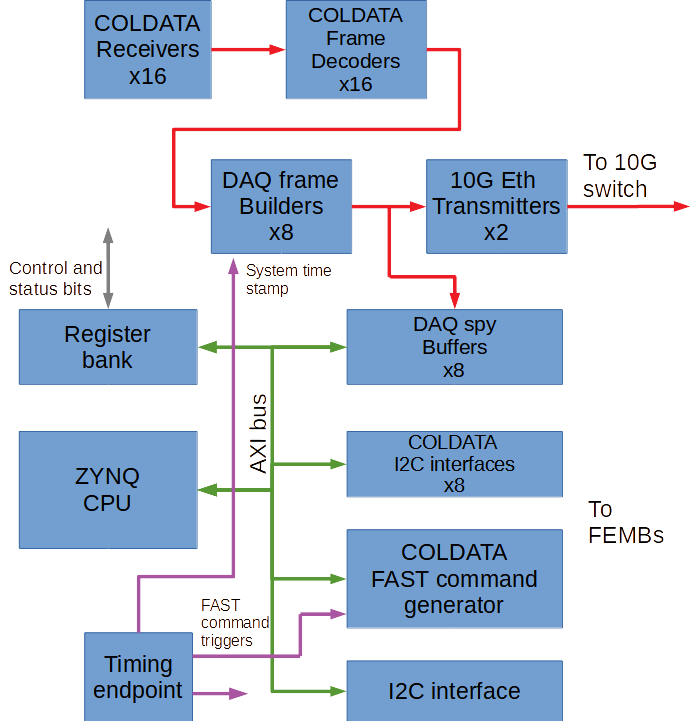


Figure . WIB firmware structure.

## 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 eight Frame Builder modules prepares data arriving from one COLDATA chip. The outputs of each Frame Builder are a 64-bit data bus, a valid flag, and a “last word” flag.

## 10Gps ethernet transmitters

These modules are responsible for sending data prepared by Frame Builders to the DAQ system via 10 Gpbs ethernet links. The firmware module that implements 10Gbps ethernet protocol is provided by DAQ team, and it’s called “HERMES”.

## 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 | All | I2C interface for COLDATA chip |
| A0050000 | coldata\_i2c | 1 | All | I2C interface for COLDATA chip |
| A0070000 | coldata\_i2c | 2 | All | I2C interface for COLDATA chip |
| A0090000 | coldata\_i2c | 3 | All | 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 |
| 440000000 | DAQ spy memory | 0 | 0 | 256KB DAQ spy memory. Capable of recording DAQ output data sent to HERMES. |
| 440100000 | DAQ spy memory | 0 | 1 | 256KB DAQ spy memory. Capable of recording DAQ output data sent to HERMES. |
| 440200000 | DAQ spy memory | 1 | 0 | 256KB DAQ spy memory. Capable of recording DAQ output data sent to HERMES. |
| 440300000 | DAQ spy memory | 1 | 1 | 256KB DAQ spy memory. Capable of recording DAQ output data sent to HERMES. |
| 440400000 | DAQ spy memory | 2 | 0 | 256KB DAQ spy memory. Capable of recording DAQ output data sent to HERMES. |
| 440500000 | DAQ spy memory | 2 | 1 | 256KB DAQ spy memory. Capable of recording DAQ output data sent to HERMES. |
| 440600000 | DAQ spy memory | 3 | 0 | 256KB DAQ spy memory. Capable of recording DAQ output data sent to HERMES. |
| 440700000 | DAQ spy memory | 3 | 1 | 256KB DAQ spy memory. Capable of recording DAQ output data sent to HERMES. |

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 . 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**.

### Using COLDATA and COLDADC I2C fine clock phase adjustment

With very long high-speed cables 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 is able to adjust the relative clock phase in very small steps (~15 ps) to accommodate cables of any reasonable length.

The initial relative phase between system and I2C clocks is set at the time of WIB reboot, and is equal to 0. The total count of phase adjustment steps is 1064. 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 adjustments is available here:

soft\_debug\_p3/i2c\_phase.sh

It accepts one parameter that is the adjustment step count.

The optimal adjustment 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].

## PTC I2C access

The **PL\_FEMB\_PWR** and **SENSOR\_I2C** buses contain sensors that must be accessed independently by WIB ZYNQ CPU and by PTC via the backplane. The PTC access must work independently of the ZYNQ CPU status, so that the sensors can be accessed even when the OS on WIB ZYNQ CPU has failed. To achieve that, these two buses are connected to MIO I2C interfaces and also to the PL GPIO pins. That way, ZYNQ CPU and PTC can access these buses concurrently, using I2C multi-master capability. Figure 5 shows the block schematics of the firmware implementation of the PTC I2C access. The I2C follower module keeps track of the current state of I2C transfer driven by PTC and controls the I2C buffers accordingly. The Activation and Bus Selection (ABS) block is an I2C slave module that PTC uses to activate one WIB at a time, and to select the bus to address.

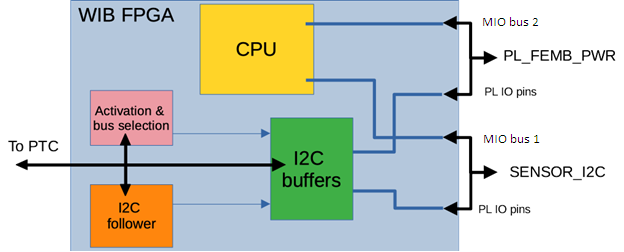


Figure . Multi-master I2C access to sensor buses.

The ABS module 7-bit I2C address is constructed as follows:

**ABS\_module\_address = 0x38 + slot;**

where **slot** is a crate slot number that WIB reads from backplane. The ABS module’s registers are listed in Table 6:

|  |  |
| --- | --- |
| **Register address** | **Description** |
| 0 | Configuration register  Bit 0: 1 = activate this WIB for PTC I2C access to sensors, 0 = disable access to sensors  Bit 1: 0 = PL\_FEMB\_PWR bus, 1 = SENSOR\_I2C bus |
| 4 | Readback register  Read only register that can be used to read Crate and Slot numbers. Bit locations are: **{crate\_id[3:0], slot[2:0]}** |

Table . Activation and Bus Selection (ABS) address map.

### Handling I2C arbitration on the multi-master buses

The I2C standard fully supports multi-master buses. However, some software cooperation is required. In case of bus collision, one or more masters may fail to complete the transaction. In such case, the typical behavior of an I2C library is to return an error code. The top-level software must analyze the error codes returned by each transaction and retry the failed transactions. This must be done in PTC and in WIB software.

### How to use PTC I2C access to WIB

1. WIB I2C logic uses 10 MHz clock provided by PS. Make sure to enable that clock as detailed in “10 MHz reference clock on P12 connector” section. This only must be done once after reboot or power up.
2. Activate WIB and select bus to access by writing ABS Configuration register on the WIB that you want to access
3. Read sensors from the selected bus
4. If necessary, program ABS Configuration register to access the other bus. Keep activation bit = 1
5. Read the sensors from the other bus
6. Set activation bit = 0
7. Repeat the sequence for the next WIB

### WIB software modifications needed for multi-master I2C buses

Previously, access to sensors in **PL\_FEMB\_PWR** and **SENSOR\_I2C** buses was performed by first programming **i2c\_select** register and then using I2C bus 0 (WIB Petalinux bus index). In the current firmware version these buses are no longer accessible via **i2c\_select** + bus 0. Instead, please use the following buses:

|  |  |
| --- | --- |
| **I2C bus name** | **I2C bus index in WIB Petalinux** |
| **PL\_FEMB\_PWR** | 2 |
| **SENSOR\_I2C** | 1 |

Programming **i2c\_select** register is not needed for accessing these two buses.

Every transaction on these two buses must be checked for errors. If any error is detected, the transaction must be retried.

## 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 | 17:16 | | prio\_enc\_descramble | | Descrambles BP\_IO pins from WIB->PTB->PTC depending on version of hardware used:  2'b00 -> "new" PTB with PTCv4 (default)  2'b01 -> "new" PTB with PTCv3B  2'b10 -> "old" PTB with PTCv3B  2'b11 -> not a legal value | |
| 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 | Unused | | 4 | PL\_FEMB\_EN | | 5 | Unused | | 6 | PL\_FEMB\_PWR2 | | 7 | LTC2977 | | 8 | PL\_FEMB\_PWR3 | | 9 | FLASH | | 10 | ADN2814 |   I2C buses PL\_FEMB\_PWR and SENSOR\_I2C are accessed via separate I2C interfaces, see section “PTC I2C access” above | |
| 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 | | DAQ spy FSM reset | |
| A00C0004 | 7 | | reserved | |  | |
| A00C0004 | 11:8 | | PRBS selection | | PRBS pattern selection for COLDATA RX   |  |  | | --- | --- | | **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 | 1 = use raw ADC channel map, 0 = use hardcoded channel map. This option is available only in FELIX version. DEIMOS version uses only raw ADC map. | |
| A00C0004 | | 22 | | cal\_dac\_start | Pulse this bit to 1 and then 0 to start DAC programming | |
| A00C0004 | | 23 | | smon\_reset | System monitor reset | |
| A00C0004 | | 24 | | hermes\_reset | HERMES reset, manual | |
| A00C0004 | | 25 | | p11\_enable | Enable spy memory trigger from P11 connector | |
| 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 DAQ even without FEMBs attached. | |
| A00C0020 | 8:1 | | context\_fld | | Context ID field in DAQ readout. | |
| A00C0020 | 14:9 | | edge\_delay | | EDGE FAST command delay, in 62.5MHz system clocks | |
| A00C0024 | 14:0 | | spy\_rec\_time | | Spy ring buffer recording time, in 64-bit words, based on system clock, 16 ns | |
| 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 | |
| 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 | |
| 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 | |
| A00C003C | 10:6 | | cp\_phase | | Calibration pulse phase relative to 2 MHz ADC clock. Units: system clock, 16 ns | |
| A00C003C | 14:11 | | cp\_femb\_en | | Calibration pulse enable per FEMB (0..3) | |
| A00C003C | 31:16 | | cal\_dac\_data | | Calibration DAC data | |
| A00C0040 | 20:0 | | cp\_period | | Calibration pulse period. Units: ADC clock, 16 ns \* 32 = 512 ns | |
| A00C0044 | 26:0 | | cp\_high\_time | | Calibration pulse high time, units: system clock, 16 ns | |
| A00C0048 | 31:0 | | ext\_mac\_addr[0] bits [31:0] | | Hermes link 0 MAC address lower 32 bits | |
| A00C004C | 15:0 | | ext\_mac\_addr[0] bits [47:32] | | Hermes link 0 MAC address higher 16 bits | |
| A00C004C | 31:16 | | ext\_port\_addr[0] | | Hermes link 0 port address | |
| A00C0050 | 31:0 | | ext\_ip\_addr[0] | | Hermes link 0 IP address | |
| A00C0054 | 31:0 | | ext\_mac\_addr[1] bits [31:0] | | Hermes link 1 MAC address lower 32 bits | |
| A00C0058 | 15:0 | | ext\_mac\_addr[1] bits [47:32] | | Hermes link 1 MAC address higher 16 bits | |
| A00C0058 | 31:16 | | ext\_port\_addr[1] | | Hermes link 1 port address | |
| A00C005C | 31:0 | | ext\_ip\_addr[1] | | Hermes link 1 IP address | |

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 0 |
| A00C0080 | 1 | daq\_spy\_full[1] | “full” flag for DAQ spy memory 1 |
| A00C0080 | 2 | reserved |  |
| A00C0080 | 3 | daq\_spy\_full[2] | “full” flag for DAQ spy memory 2 |
| A00C0080 | 4 | daq\_spy\_full[3] | “full” flag for DAQ spy memory 3 |
| A00C0080 | 5 | daq\_spy\_full[4] | “full” flag for DAQ spy memory 4 |
| A00C0080 | 6 | daq\_spy\_full[5] | “full” flag for DAQ spy memory 5 |
| A00C0080 | 7 | daq\_spy\_full[6] | “full” flag for DAQ spy memory 6 |
| A00C0080 | 8 | daq\_spy\_full[7] | “full” flag for DAQ spy memory 7 |
| 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. |
| A00C0090 | 20 | cal\_dac\_busy | 1 = Calibration DAC FSM is busy programming DAC |
| A00C0094 | 14:0 | spy\_addr [0] | Current memory address for spy memory 0, in bytes |
| A00C0094 | 30:16 | spy\_addr [1] | Current memory address for spy memory 1, in bytes |
| A00C0098 | 14:0 | spy\_addr [2] | Current memory address for spy memory 2, in bytes |
| A00C0098 | 30:16 | spy\_addr [3] | Current memory address for spy memory 3, 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 |
| 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 |
| A00C00CC | 14:0 | spy\_addr [4] | Current memory address for spy memory 4, in bytes |
| A00C00CC | 30:16 | spy\_addr [5] | Current memory address for spy memory 5, in bytes |
| A00C00D0 | 14:0 | spy\_addr [6] | Current memory address for spy memory 6, in bytes |
| A00C00D0 | 30:16 | spy\_addr [7] | Current memory address for spy memory 7, in bytes |
| A00C00D4 | 15:0 | sm\_temp | System Monitor, ADC code for temperature |
| A00C00D4 | 31:16 | sm\_vccint | System Monitor, ADC code for VCCINT |
| A00C00D8 | 15:0 | sm\_vccaux | System Monitor, ADC code for VCCAUX |
| A00C00D8 | 31:16 | sm\_vccbram | System Monitor, ADC code for VCCBRAM |

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 eight DAQ spy memory modules, one for each COLDATA chip. Each memory unit is 256KB in size. They operate in 64-bit words, so the size is 32K 64-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

For trigger distributed by DTS:

* Program **cmd\_code\_trigger** and **cmd\_en\_trigger** registers with the DTS command for external trigger
* Program **spy\_rec\_time** parameter with the number of 64-bit words to record after trigger. That number should be less or equal to **32K – trigger\_latency**. Trigger latency time here is represented in terms of system clock cycles, which is  
  **1 / 62.5 MHz = 16 ns.**

For trigger supplied via P11 connector:

* Program **p11\_enable = 1**
* Program **spy\_rec\_time** parameter with the number of 64-bit words to record after trigger. That number should be less or equal to **32K – trigger\_latency**. Trigger latency time here is represented in terms of system clock cycles, which is  
  **1 / 62.5 MHz = 16 ns.**

### Operation with the trigger distributed by DTS or supplied via P11 connector

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 HERMES. 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 or pulse. At the time of the trigger command or rising edge on P11, the spy memory will start counting words stored in it. It will store **spy\_rec\_time** 64-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 256KB = 32K 64-bit words.
7. Read out the current spy memory address, from this register: **spy\_addr[N],** where **N = 0..7** is the spy memory/COLDATA chip 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–spy\_rec\_time–trigger\_latency**  
     
   All arithmetics here is 15-bit, unsigned. The **decoding\_start\_addr** result overwraps to maximum value of **0x7fff** if it goes below 0. The result represents the 64-bit word index.  
   Note that the resulting address may not necessarily point at the first word of a HERMES data block. Start searching from that address until you detect the start of frame.
9. Unpack the following number of 64-bit words:  
   **spy\_rec\_time + trigger\_latency**  
   rounded down to the HERMES frame size (899 words).
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 ~0.6 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 256 KB.

The data in the memory array are stored in the same DAQ format as transmitted via HERMES. Each frame is 899 64-bit words in length. The format decoder should be using specific 64-bit data patterns at the beginning of each frame to decode the data. Unfortunately, the HERMES header format does not contain a very distinctive data pattern that can be used to search for the start of the frame. A workaround that can be employed should use the following facts:

* Header word 1 contains 64-bit DTS time stamp
* Time stamp difference from one DAQ frame to the next is always 0x800
* Length of each DAQ frame is 899 words

Alternatively, frame start can be found using the fact that Header word 3 currently contains all zeros. However, note that this will change in the future.

Example scripts that reset and read out spy memory can be found here:

soft\_debug\_p3/daq\_spy\_reset.sh

soft\_debug\_p3/daq\_spy\_read.sh

The **daq\_spy\_read.sh** script takes one argument that is the spy memory index, 0..7.

## 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

## System Monitor

The System Monitor is implemented in firmware with its own state machine that constantly reads the ADC values for Temperature, VCCINT, VCCAUX and VCCBRAM, and places them into registers. See Table 9 for the register addresses. An example script that reads ADC values and converts them into °C and V is located here:

**soft\_debug\_p3/read\_sysmon.sh**

## Calibration logic

### Calibration DAC

Using calibration DAC:

1. Write 16-bit data to be programmed into DAC into register **cal\_dac\_data**
2. Wait until **cal\_dac\_busy == 0**
3. Program DAC data by setting **cal\_dac\_start = 1** and then **= 0**

Example scripts for writing and testing calibration DAC are available here:

**soft\_debug\_p3/cal\_dac\_set.sh**

**soft\_debug\_p3/cal\_dac\_test.sh**

### Calibration Pulse generator

Using Calibration Pulse generator:

1. Reset ADC digitization clock phase by issuing EDGE FAST command. WIB firmware is using this command to recreate the ADC digitization clock phase so that the Calibration pulse phase can be fixed relative to it. The EDGE FAST command does not need to be repeated every time you program the Calibration Pulse generator. If it was issued at least once after timing endpoint reset, the recreated ADC clock will stay in phase with the actual ADC clock.
2. Program **cp\_period** register with the calibration pulse period. Note that since the calibration pulse is always synchronized with ADC digitization clock (sysclk/32), the **cp\_period** units are also sysclk/32 = 512 ns. Value of 0 corresponds to cal pulse period of 1 ADC clock, value of 1 = period of 2 ADC clocks, etc.
3. Program **cp\_phase**. This value shifts the calibration pulse relative to ADC digitization clock. Since there are only 32 system clocks in one ADC digitization clock, this register accommodates values from 0 to 31.
4. Program **cp\_high\_time**. This value defines how long the calibration pulse stays =1. Units are sysclk period, 16 ns.
5. Enable calibration pulse for some or all FEMBs using **cp\_femb\_en**.

## 10 MHz reference clock on P12 connector

The firmware uses 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 a 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.

Timeline

Description automatically generated

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 is passed to DAQ Frame builder, which builds the data frames for DAQ transmission using the 62.5 MHz system clock. The prepared DAQ data frames are then transferred into HERMES module. The HERMES module takes an external 125MHz reference clock and uses it to generate 10Gbps ethernet packets. It also transfers the data into ethernet clock domain internally.

# 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.

## HERMES firmware module

A screenshot of a computer program

Description automatically generated

The firmware module provided by DAQ team is called HERMES. The picture above shows the graphical wrapper for that module. HERMES can be configured to accept arbitrary number of the input streams. In WIB firmware it’s configured for 8 streams.

Each stream has three signals:

|  |  |
| --- | --- |
| **Name** | **Description** |
| d[63:0] | Input data word |
| d\_valid | This flag should be set when data word is valid |
| d\_last | This flag should be set when last word in a frame is transmitted |

### HERMES reset input logic

If **ts\_clk\_sel == 0** then HERMES is held in reset when Timing Endpoint is not ready.  
If **ts\_clk\_sel == 1** then Timing Endpoint status does not affect HERMES reset input.  
If **ts\_clk\_sel == 0** and Timing Endpoint is ready, or **ts\_clk\_sel == 1**, then HERMES can be manually reset by toggling **hermes\_reset** register **=1** and then **=0**.

## Data frames

HERMES accepts frames of arbitrary length on each data stream input. The content of the frame is not important and is not changed by HERMES (Exception: Channel ID, Version, and Time stamp fields are required, see Data Block Format section below). HERMES adds its own header and trailer words to each frame, but the user should not be concerned about that.

WIB DAQ format for UDP-based protocol is designed with the following limitations (messages from D. Christian and D. Newbold, 2022-09-21):

|  |  |  |
| --- | --- | --- |
| **Item** | **Description** | **Size** |
| ADC sample | Each sample must be represented by 14 data bits. If the original data was collected in 12-bit samples, they should be padded by zeros in lower bits | 14 bits |
| Data blocks | * 64 time ticks of packed 14-bit data for 64 channels * ordered with 64 channels of time tick 0, then 64 channels of time tick 1, and so on * each time tick = 32 ticks of the 62.5 MHz clock | 64 x 64 x 14 bits = 57344 bits = 896 64b words (samples only) |
| Total ADC channel count | 512 ADC channels per WIB |  |

The count of channels per data block (64) neatly corresponds to one COLDATA chip. Therefore, each data block contains data from one COLDATA chip.

A graphical view of the data frame format can be found in this spreadsheet:

WIB-DAQ-format.xlsx, tab: “DEIMOS”

## Header considerations

Information that needs to be transmitted with each data block:

|  |  |  |
| --- | --- | --- |
| **Field** | **Description** | **Size, bits** |
| Common header fields.  These fields are added in a single 64-bit word by HERMES firmware module. Their values are programmed via registers in HERMES firmware module. | | |
| Version | DAQ format version | 6 |
| Det ID | Det ID | 6 |
| Crate | Crate ID | 10 |
| Slot | Crate Slot | 4 |
| Stream | Stream number | 8 |
| Sequence # | Sequence number | 12 |
| Block length | Block length | 12 |
| Common trailer fields.  Added by HERMES firmware module | | |
| CRC | CRC for the entire DAQ block | ?? |
| WIB-generated header | | |
| Channel ID | Required by DAQ specs. Constructed as follows:   |  |  | | --- | --- | | **Bits** | **Data** | | 0 | COLDATA Index | | 2:1 | FEMB index | | 7:3 | Reserved | | 8 |
| Version | Required by DAQ specs, purpose unclear at this time. | 4 |
| COLDATA time 0 | COLDATA time stamp link 0 | 15 |
| COLDATA time 1 | COLDATA time stamp link 1 | 15 |
| CRC error | Indicates that CRC error happened on the data link | 2 |
| Link valid | Indicates that data was received from a valid link | 2 |
| LOL | Loss of lock from on-board PLL | 1 |
| WIB sync | Indicates that the timing endpoint is synchronized | 1 |
| FEMB sync | Indicates that FEMB is synchronized with global time stamp, one bit per COLDATA link | 2 |
| Pulser | FEMB pulser was active | 1 |
| Calibration | Calibration is ongoing | 1 |
| Ready | Indicates that WIB is ready (not in the process of being configured) | 1 |
| Context | Context code | 8 |
| Time stamp | Required by DAQ specs Global 64-bit time stamp | 64 |

## Data samples packing

None of the ways of packing 14-bit data samples into 64-bit words are particularly “nice”. Some of the data samples must be broken into fragments with any packing strategy. We’ve selected the packing pattern shown below because it requires reassembling the minimal number of 14-bit words from a maximum of 2 fragments. 64 14-bit samples from 4 ADCs are packed into 14 64-bit words. This data block represents data for one system clock cycle of 16 ns.

Graphical user interface, application, timeline

Description automatically generated

Figure . ADC samples packaged into 64-bit words

Explanation for the picture:

* Top line shows bit numbers
* Colored sections correspond to 14-bit data samples
* Grey data samples are split into 2 fragments of various lengths
* Note that the packing pattern repeats every 7 64-bit words but contains data from different ADCs.

## Data block format

|  |  |  |  |
| --- | --- | --- | --- |
| **Words, 64-bit** | **Clock tick** | **Channels** | **Description** |
| 1 |  |  | Header word 1, see format below |
| 2 |  |  | Header word 2, see format below |
| 3 |  |  | Header word 3, see format below |
| 4..17 | 0 | 0..63 | ADC Data samples as shown in Figure 1 |
| 18..31 | 1 | 0..63 | ADC Data samples as shown in Figure 1 |
| …More words with ADC samples… | | | |
| 886..899 | 63 | 0..63 | ADC Data samples as shown in Figure 1 |

Each of the “ADC data sample” sections are composed of two 7-word blocks as shown in “Data Samples packing” chapter, carrying 64 14-bit ADC samples total.

## Header Word 1

|  |  |
| --- | --- |
| **Field** | **Bits** |
| Global Time stamp | 63:0 |

## 

## Header Word 2

|  |  |  |
| --- | --- | --- |
| **Field** | **Acronym in spreadsheet** | **Bits** |
| COLDATA Link 0 time stamp | COLDATA Link 0 time stamp | 14:0 |
| COLDATA Link 1 time stamp | COLDATA Link 1 time stamp | 30:16 |
| CRC error | CRC err | 34:33 |
| Link valid | Link val | 36:35 |
| Loss of Lock | LOL | 37 |
| WIB sync | WS | 38 |
| FEMB sync | FS | 40:39 |
| Pulser | Pls | 41 |
| Calibration | Cal | 42 |
| Ready | Rdy | 43 |
| Context | Context | 51:44 |
| Version | Version | 55:52 |
| Channel ID | Channel ID | 63:56 |
| Reserved | Rsv | 15, 32:31 |

## Header Word 3

|  |  |
| --- | --- |
| **Field** | **Bits** |
| Reserved | 63:0 |

## Bandwidth calculation



# 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 transmit 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 = 0 and cmd\_en\_edge = 1. 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 as shown below:  
   fastl = 19 + cab;
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 0x7f. 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 may 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. Ultrascale + ZYNQ manual
8. Ultrascale + GTH manual
9. 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. |
| 2022-11-23 | Fixed alignment setup procedure to take into account update to DCSK timing system. |
| 2023-02-21 | Added programmable delay for EDGE FAST command, register edge\_delay |
| 2023-02-24 | In DAQ frame, replaced FLEX bits with CRC error bits from COLDATA links |
| 2023-04-10 | Added registers for calibration DAC control, section on how to use DAC |
| 2023-04-13 | Reworked for Ethernet readout, including spy memory |
| 2023-04-26 | Added prio\_enc\_descramble parameter to accommodate all versions of PTC |
| 2023-08-08 | Added section on PTC I2C access to WIB sensors |
| 2023-08-24 | Fixed error in data alignment section, max valid alignment value was wrong |
| 2023-08-28 | Added sections on Calibration Pulse Generator and System Monitor |
| 2023-09-10 | Added sections on P11 connector use as spy memory trigger, HERMES reset logic |
| 2024-01-25 | Added registers for programming Hermes MAC, IP addresses, and port numbers for both Eth links. Updated Hermes screenshot in DAQ section. Firmware compiled with Hermes 1.9 IP. |