DUNE WIB registers

## A. Madorsky, UF

DRAFT, 2020-10-19

Access to all modules in WIB firmware is provided via standard ZYNQ AXI bus interface.

List of currently implemented and planned modules with AXI registers is shown below. Note that this list may be expanded in the future.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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. |

All registers are 32 bits wide.

Sections below contain details of the registers of each module.

# COLDATA I2C interface

Register list:

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

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) |

### How to use I2C interface:

Write operation:

1. Write the parameters into addr\_data register as needed. Leave ACK bit placeholders = 0
2. Set start\_transfer register to 1, then immediately set it to 0
3. Wait for 27 uS or longer before attempting the next transfer

Read operation:

1. Write the parameters into addr\_data register as needed. Leave ACK bit placeholders = 0
2. Set start\_transfer register to 1, then immediately set it to 0
3. Wait for 27 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 27 uS or more until the transaction in step 1 is finished

### COLDATA v1 I2C bugs:

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

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

# COLDATA FAST command generator

Register list:

|  |  |  |
| --- | --- | --- |
| **Address, hex**  **Relative to base** | **Register name** | **Description** |
| 0 | 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. |
| 4 | edge\_to\_act\_delay | Delay between EDGE and ACT command for correct COLDADC reset procedure. Write a value of 19 into this register. |

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. |

### 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.

# I2C interface for WIB on-board devices

This is a standard Xilinx IIC module. 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 details.

# 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.

Control registers (read/write) are listed below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset relative to base** | **Bits in register** | **Parameter name** | **Description** |
| 0 | 7:0 | ts\_addr | Timing point address |
| 0 | 9:8 | ts\_tgrp | Timing point group code |
| 4 | 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 | |
| 4 | 4 | fp\_sfp\_sel | P15 SFP connection selector 0=CDR 1=GTH (schematic page 14) |
| 4 | 5 | rx\_timing\_sel | U1 input selector 0=backplane 1=SFP (schematic page 15) |
| 4 | 6 | daq\_spy\_reset[0] | DAQ spy FSM reset for FELIX link 0, FEMB 0,1 |
| 4 | 7 | daq\_spy\_reset[1] | DAQ spy FSM reset for FELIX link 1, FEMB 2,3 |
| 4 | 11:8 | PRBS selection | PRBS pattern selection for COLDATA RX links   |  |  | | --- | --- | | **Value** | **Pattern** | | 0 | Normal operation | | 1 | PRBS-7 | |
| 4 | 12 | fb\_reset | Frame builder reset. Write 1 then 0 to reset. |
| 4 | 13 | coldata\_rx\_reset | Reset of the serial receivers. Write 1 then 0 to reset. |

Status registers (read-only):

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset relative to base** | **Bits in register** | **Parameter name** | **Description** |
| 0x80 | 0 | daq\_spy\_full[0] | “full” flag for DAQ spy memory, FELIX link 0, FEMB 0,1 |
| 0x80 | 1 | daq\_spy\_full[1] | “full” flag for DAQ spy memory, FELIX link 1, FEMB 2,3 |
| 0x84 | 15:0 | rxprbserr | PRBS error detection bits, one for each of the 16 input serial links. |
| 0xbc | 31:0 | test\_pattern | Hardwired value = 0xbabeface |

# Timing Endpoint

This is a timing endpoint module. The only parameters that need programming are Address and Group codes. These parameters are programmed via control registers. See Control and Status registers section above for details.

# 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 of the modules should be operated independently. How to operate:

1. Reset the module, by using daq\_spy\_reset bit
2. Wait until daq\_spy\_full bit from that module is set to 1.
3. Read out and store the entire memory array. The size of the array is 1 MB.
4. Go to step 1.

The data in the memory array are stored in the same DAQ format [1] 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 [1]).

# WIB configuration sequence

The table below shows configuration sequence that needs to be followed to bring WIB up after power-up. The reference software will be provided as an archived file. This sequence is incomplete and will be expanded.

|  |  |  |
| --- | --- | --- |
| **Step name** | **Reference code** | **Description** |
| eth0 enable | eth0\_start.sh | Bring up eth0 network interface. Currently this is done manually by using the script. The eth0 should be set for automatic startup in the petalinux, immediately after booting. Also, need to work out a way to assign unique IP addresses to all WIBs in the system |
| Clock synthesizer configuration | clocks/si5345\_config.c | Clock synthesizer programming. The reference program, even though written in C, actually calls standard Linux i2c package tools. Recommend replacing these calls with proper i2c library functions. |
| FEMB power voltages configuration | voltages/wib\_voltages.c | DC-DC converters and LDO programming. The reference program, even though written in C, actually calls standard Linux i2c package tools and devmem utility. Recommend replacing these calls with proper i2c library functions and memory accesses. |
| FEMB power enable | coldata\_power\_on.sh | Power enable signals for all DC-DC converters and LDOs set to enable all power outputs. |
| FEMB serial receivers reset |  | Write 1 then 0 into coldata\_rx\_reset bit. |

# References

1. File “WIB-DAQ\_Format\_Aug10\_2020\_AM\_additions.xlsx” in git repository, doc directory.

# 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. |