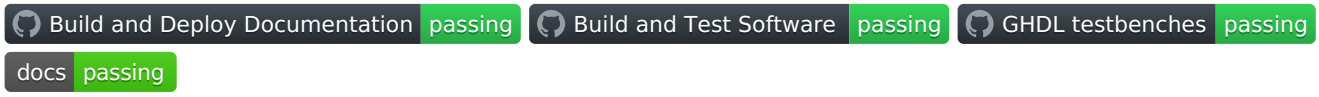
musip-midas Documentation

None

None

None

Table of contents



1. musip-midas

This is a MIDAS-based frontend for the Mupix11 based quad moduels.

1.1 Project Structure

- `analyzer/` – Analysis utilities
- `custom/` – Custom pages for MIDAS
- `docs/` – Documentation folder
- `midas_fe/` – Core frontend source code
- `tests/` – Unit tests for the project
- `tools/` – Helper tools and scripts (mainly for the analyzer)
- `.clang-format` – Project-wide code style config
- `mkdocs.yml` – Setup for the documentation

1.2 Build Instructions

For a full installation guide go to: Setup

1.2.1 Prerequisites

- CMake ≥ 3.15
- C++17-compatible compiler (e.g. `clang++`, `g++`)
- MIDAS
- Python ≥ 3.7 (for scripts)

1.2.2 Build

```
mkdir build cd build cmake .. make
```

1.2.3 Linting

Install clang-format and cpplint. The clang-format will clean the changed files while cpplint also gives some more static analysis.

```
cd build make clangformat make cpplint
```

1.2.4 Testing

CMake will download googletests to run the tests.

```
cd build ./tests/sample_test ./tests/bits_utils_test ./tests/mutrig_config_test
```

There are also some test managed with pytest.

```
cd tests pytest
```

1.2.5 Docs

We use mkdocs and doxygen for generating the documentation. You should have the doxygen package installed on your system, then:

```
pip install mkdocs pip install mkdocs-material pip install mkdoxy pip install mkdocs-with-pdf
```

Once all these are installed one can generate the documentation via:

```
make doc_mkdocs make serve_mkdocs
```

2. Setup

2.1 Prerequisites

- openSUSE Leap 15.6 (recommended)
- Root access required
- Minimum:
- 32 GB RAM (required for Arria 10 firmware compilation)
- SSD storage
- PCIe slot with at least x8 electrical connectivity
- Arria 10 (A10) board installed with:
- PCIe power connected
- USB connected
- Clock source connected (or SMA loopback for testing)

2.2 OS Installation

2.2.1 Recommended Settings

- Filesystem: **ext4**
- Install on SSD
- Disable snapshots
- Do not create swap
- Create user: `mu3e`
- Configure root password

2.3 Repository Setup

Add repositories:

```
sudo zypper addrepo <repo-url> <alias> sudo zypper refresh sudo zypper dup --allow-vendor-change
```

Recommended repositories:

- science
- network
- vscode

2.4 Required Packages

Install all required packages before proceeding.

2.4.1 General

```
sudo zypper install git cmake kernel-devel htop tmux gcc12 gcc12-c++ python
```

2.4.2 Additional Components

Install packages required for:

- ROOT
- MIDAS
- Kernel driver
- Geant4
- Quartus
- VS Code

(Refer to package lists in the full documentation if dependencies are missing.)

2.5 Clone Repositories and install

2.5.1 MIDAS

```
git clone git@bitbucket.org:tmidas/midas.git cd midas git submodule update --init --recursive mkdir build cd build cmake .. make install
```

2.5.2 Musip

```
git clone git@github.com:makoeppel/musip.git cd musip
```

2.6 ROOT Installation

```
git clone --branch latest-stable --depth=1 https://github.com/root-project/root.git ~/root_src mkdir -p ~/compiled_software/root_build cd ~/compiled_softw
```

Configure and build:

```
cmake \ -DCMAKE_INSTALL_PREFIX=/opt/root \ -DCMAKE_CXX_STANDARD=17 \ -DLLVM_CXX_STD=c++17 \ -Dxrootd=OFF \ -DCMAKE_C_COMPILER=/usr/bin/gcc-12 \
~/root_src make -j12
```

Verify:

```
root root-config --cflags
```

Expected:

```
-std=c++17
```

2.7 Configure `.bashrc`

Required sections:

2.7.1 Quartus

```
export ALTERAPATH="$HOME/programs/intelFPGA/18.1" export QUARTUS_ROOTDIR=${ALTERAPATH}/quartus export PATH=$PATH:${ALTERAPATH}/quartus/bin
```

Set license server:

```
export LM_LICENSE_FILE="<license-server>"
```

2.7.2 ROOT

```
source /opt/root/bin/thisroot.sh
```

2.7.3 MIDAS

```
export MIDASSYS=$HOME/midas export MIDAS_EXPTAB=$HOME/musip/online/exptab export MIDAS_EXPT_NAME=Mu3e export PATH=$PATH:$MIDASSYS/bin
```

2.7.4 Online

Reload:

```
source ~/.bashrc
```

2.8 Geant4 (Optional)

Install only if required for simulation work.

Typical workflow:

```
mkdir ~/geant4_src mkdir ~/compiled_software/geant4_build ccmake <geant4-source> make -j<N> make install
```

Enable:

```
source geant4.sh
```

Test using example `B1`.

2.9 Quartus Installation

Recommended version:

- Quartus Prime Pro 18.1

Requirements:

- ≥32 GB RAM
- ≥32 GB free disk space

Install:

```
mkdir ~/programs cd ~/programs tar -xvf Quartus-18.1*.tar ./setup.sh
```

Install support for:

- Arria 10
- Arria V
- MAX10

Verify:

```
quartus
```

2.10 Udev Rules

2.10.1 USB Blaster

Create:

```
/etc/udev/rules.d/51-usbblaster.rules
```

with:

```
SUBSYSTEM=="usb", ATTR{idVendor}=="09fb", ATTR{idProduct}=="6001", MODE="0666" SUBSYSTEM=="usb", ATTR{idVendor}=="09fb", ATTR{idProduct}=="6002", MODE="06
```

Create:

```
sudo nano 99-mudaq.rules
```

with:

```
KERNEL=="mudaq*", OWNER="root", GROUP="users", MODE="0666"
```

Reload:

```
sudo udevadm control --reload-rules sudo udevadm trigger
```

Verify:

```
jtagconfig
```

Ff your a10 dev board is properly connected you should see something like this:

```
 1) PCIe40 [1-13] 02E660DD 10AX115H1(.|E2|ES)/10AX115H2/.. 020A40DD 5M(1270ZF324|2210Z)/EPM2210
```

2.11 A10 Firmware Build

Select board:

```
cd ~/musip/firmware/a10_board
```

Build:

```
make make flow make app
```

Upload:

```
make pgm make app_upload
```

Open terminal:

```
make terminal
```

You should see the FPGA menu.

Note

Some systems need to be rebooted to get the PCIe working. Therefore, reboot the system and run `sudo ./recover_pcie.sh`.

2.12 Build Kernel Driver

```
cd ~/musip/midas_fe/kerneldriver make
```

Load:

```
sudo ./recover_pcie.sh
```

Expected:

```
loaded 'mudaq'
```

2.13 Verify PCIe Communication

```
cd ~/musip/build/tools ./rw rr 0x1
```

Then:

```
./dmatest 2 0 1 0x1 5 0
```

And then press `1` in the menu followed by `q`.

Inspect:

```
less memory_content.txt
```

Non-zero data confirms successful communication.

2.14 DAQ Startup Procedure

2.14.1 First-Time Setup

```
odbinit -s 100MB
```

2.14.2 Load Firmware

```
cd ~/musip/firmware/a10_board make pgm make app_upload
```

2.14.3 Load Driver

```
cd ~/musip/midas_fe/kerneldriver sudo ./recover_pcie.sh
```

2.15 Essential Frontends

Start in this order:

1. `mhttpd`
2. `mlogger`
3. `quads_config_fe`
4. `readout_fe`
5. `quadana`

2.16 Quick Validation Checklist

- [ ] ROOT starts successfully
- [ ] `root-config --cflags` reports C++17
- [ ] Quartus launches
- [ ] `jtagconfig` detects hardware
- [ ] FPGA menu appears via `make terminal`
- [ ] `lspci` shows Altera device
- [ ] `recover_pcie.sh` loads `mudaq`
- [ ] `rw rr 0x1` returns firmware hash
- [ ] DMA test produces valid data
- [ ] MIDAS available at `localhost:8080`

Once all checks pass, the DAQ machine is operational.

3. Frontends

3.1 File quads_config_fe.cpp

FileList > **midas_fe** > **quads_config_fe.cpp**

Go to the source code of this file

MIDAS frontend for configuring and controlling MuPix quads. More...

- `#include <stdio.h>`
- `#include <stdlib.h>`
- `#include <sys/mman.h>`
- `#include <unistd.h>`
- `#include <iostream>`
- `#include <list>`
- `#include "midas.h"`
- `#include "DummyFEBSlowcontrolInterface.h"`
- `#include "FEBSlowcontrolInterface.h"`
- `#include "Mutrig3Config.h"`
- `#include "mcstd.h"`
- `#include "mfe.h"`
- `#include "missing_hardware.h"`
- `#include "msystem.h"`
- `#include "mudaq_device.h"`
- `#include "odb_setup.h"`
- `#include "odbxx.h"`
- `#include "utils.h"`

3.1.1 Public Attributes

Type	Name
std::vector< uint32_t >	**adc_banks** = {}
uint8_t	**bitpattern_mupix** = {}
uint8_t	**bitpattern_mutrig** = {}
std::vector< uint32_t >	**counters_XXCE** = {}
std::vector< uint32_t >	**counters_XXCF** = {}
std::vector< uint32_t >	**counters_XXCH** = {}
std::vector< uint32_t >	**counters_XXCP** = {}
std::vector< uint32_t >	**counters_XXCR** = {}
EQUIPMENT	**equipment** = /* multi line expression */
BOOL	**equipment_common_overwrite** = TRUE
std::vector< uint32_t >	**feb_hits** = {0,0,0,0}
std::vector< uint32_t >	**feb_hits_last** = {0,0,0,0}
FEBSlowcontrolInterface *	**feb_sc**
const char *	**frontend_file_name** = __FILE__
const char *	**frontend_name** = "Quads"
std::vector< uint32_t >	**lvds_banks** = {}
midas::odb	**m_settings**
std::vector< uint32_t >	**matrix_banks** = {}
mudaq::DmaMudaqDevice *	**mup** = nullptr
std::vector< uint32_t >	**readout_banks** = {}
reset	**reset_protocol**
std::vector< uint32_t >	**values_XXSM** = {}
std::vector< float >	**values_XXTM** = {}

3.1.2 Public Functions

Type	Name
int	**begin_of_run** ()
int	**end_of_run** ()
int	**frontend_exit_user** ()
int	**frontend_init** ()
void	**init_banks** ()
int	**init_mudaq** (**mudaq::MudaqDevice** & mu)
int	**quad_loop** ()
int	**read_sc_event** (char * pevent, int off)
void	**sc_settings_changed** (midas::odb o)
int	**write_command_by_id** (uint8_t command, uint32_t payload, bool has_payload)
int	**write_command_by_name** (const char * name, uint32_t payload=0, uint16_t address=0)

3.1.3 Detailed Description

This frontend is part of the MIDAS data acquisition system. It handles initialization, configuration, and control of MuPix devices using either a real or dummy FEB (Front-End Board) slow control interface. It communicates with the hardware via a `mudaq::MudaqDevice`, optionally utilizing DMA.

Key responsibilities of this frontend include: * Opening and verifying the MuPix hardware device. * Selecting the appropriate FEB slow control implementation based on compilation flags. * Initializing LVDS bank configurations and bit patterns. * Interfacing with the MIDAS Online Database (ODB) to fetch run-time configuration. * Registering itself with MIDAS using appropriate frontend metadata.

Preprocessor directives like `NO_A10_BOARD` are used to determine whether to instantiate a dummy or real FEB control interface, enabling testing on systems without hardware attached.

Note:

Ensure all dependencies (mudaq library, MIDAS, system headers) are present and properly configured.

Author:

Marius Snella Köppel

Date:

2025-07-04

3.1.4 Public Attributes Documentation

variable adc_banks

```
std::vector<uint32_t> adc_banks;
```

---

variable bitpattern_mupix

```
uint8_t bitpattern_mupix[N_BYTES_MUPIX];
```

---

variable bitpattern_mutrig

```
uint8_t bitpattern_mutrig[N_BYTES_MUTRIG];
```

---

variable counters_XXCE

```
std::vector<uint32_t> counters_XXCE;
```

---

variable counters_XXCF

```
std::vector<uint32_t> counters_XXCF;
```

---

variable counters_XXCH

```
std::vector<uint32_t> counters_XXCH;
```

---

variable counters_XXCP

```
std::vector<uint32_t> counters_XXCP;
```

---

variable counters_XXCR

```
std::vector<uint32_t> counters_XXCR;
```

---

variable equipment

```
EQUIPMENT equipment[];
```

---

variable equipment_common_overwrite

```
BOOL equipment_common_overwrite;
```

---

variable feb_hits

```
std::vector<uint32_t> feb_hits;
```

---

variable feb_hits_last

```
std::vector<uint32_t> feb_hits_last;
```

---

variable feb_sc

```
FEBSlowcontrolInterface* feb_sc;
```

---

variable frontend_file_name

```
const char* frontend_file_name;
```

---

variable frontend_name

```
const char* frontend_name;
```

---

variable lvds_banks

```
std::vector<uint32_t> lvds_banks;
```

---

variable m_settings

```
midas::odb m_settings;
```

---

variable matrix_banks

```
std::vector<uint32_t> matrix_banks;
```

---

variable mup

```
mudaq::DmaMudaqDevice* mup;
```

---

variable readout_banks

```
std::vector<uint32_t> readout_banks;
```

---

variable reset_protocol

```
reset reset_protocol;
```

---

variable values_XXSM

```
std::vector<uint32_t> values_XXSM;
```

---

variable values_XXTM

```
std::vector<float> values_XXTM;
```

3.1.5 Public Functions Documentation

function begin_of_run

```
int begin_of_run ()
```

---

function end_of_run

```
int end_of_run ()
```

---

function frontend_exit_user

```
int frontend_exit_user ()
```

---

function frontend_init

```
int frontend_init ()
```

---

function init_banks

```
void init_banks ()
```

---

function init_mudaq

```
int init_mudaq ( mudaq::MudaqDevice & mu )
```

---

function quad_loop

```
int quad_loop ()
```

---

function read_sc_event

```
int read_sc_event ( char * pevent, int off )
```

---

function sc_settings_changed

```
void sc_settings_changed ( midas::odb o )
```

---

function write_command_by_id

```
int write_command_by_id ( uint8_t command, uint32_t payload, bool has_payload )
```

---

function write_command_by_name

```
int write_command_by_name ( const char * name, uint32_t payload=0, uint16_t address=0 )
```

The documentation for this class was generated from the following file `midas_fe/quads_config_fe.cpp`

4. Utils

4.1 File bits_utils.h

FileList > **midas_fe** > **bits_utils.h**

Go to the source code of this file

Utility for bitwise parameter encoding in bit-pattern buffers. More...

- `#include <stdlib.h>`
- `#include <cstdint>`
- `#include <vector>`

4.1.1 Public Functions

Type	Name
uint32_t	**setParameter** (uint8_t * bitpattern_w, uint32_t value, uint32_t offset, uint32_t nbits, bool inverted) *Set a value into a bit pattern buffer at a specific bit offset.*

4.1.2 Detailed Description

This header defines a function for writing integer values into a bitfield buffer, which is useful in applications like hardware register configuration or slow control settings for DAQ systems.

The `setParameter` function encodes a given value into a byte array at a specific bit offset, handling arbitrary bit lengths and bit order inversion.

4.1.3 Public Functions Documentation

function setParameter

Set a value into a bit pattern buffer at a specific bit offset.

```
uint32_t setParameter ( uint8_t * bitpattern_w, uint32_t value, uint32_t offset, uint32_t nbits, bool inverted )
```

This function encodes a `value` of length `nbits` into the buffer `bitpattern_w` starting at the given `offset`. If `inverted` is true, the bits are written in reverse order (MSB first).

Parameters:

- `bitpattern_w` Target buffer (must be pre-allocated).
- `value` Integer value to encode.
- `offset` Starting bit offset in the buffer.
- `nbits` Number of bits used to encode the value.
- `inverted` Whether to reverse bit order (MSB-first).

Returns:

uint32_t The new bit offset after writing.

The documentation for this class was generated from the following file `midas_fe/bits_utils.h`

4.2 File constants.h

FileList > **midas_fe** > **constants.h**

Go to the source code of this file

Definitions of project-wide constants and event data structures. More...

- `#include <array>`
- `#include <map>`
- `#include <string>`
- `#include <vector>`
- `#include "registers.h"`

4.2.1 Classes

Type	Name
struct	**mevent_t**
struct	**dsin_t**
struct	**reset**
struct	**resetcommand**

4.2.2 Public Types

Type	Name
enum uint8_t	**ADC_Command**
enum uint8_t	**ADC_Mode**
enum uint8_t	**ADC_Mux_Address**

4.2.3 Public Attributes

Type	Name
constexpr uint16_t	**CMD_MUTRIG_ASIC_CFG** = `0x0110`
constexpr uint16_t	**CMD_MUTRIG_ASIC_OFF** = `0x0130`
constexpr uint16_t	**CMD_MUTRIG_CNT_RESET** = `0x0160`
constexpr uint16_t	**CMD_TILE_ASIC_PWR** = `0x0210`
constexpr uint16_t	**CMD_TILE_ASIC_PWROR** = `0x0220`
constexpr uint16_t	**CMD_TILE_INJECTION_SETTING** = `0x0260`
constexpr uint16_t	**CMD_TILE_POWERMONITORS_READ** = `0x0240`
constexpr uint16_t	**CMD_TILE_TEMPERATURES_READ** = `0x0230`
constexpr uint16_t	**CMD_TILE_TEMPERATURES_READ_IDS** = `0x0270`
constexpr uint16_t	**CMD_TILE_TMB_INIT** = `0x0200`
constexpr uint16_t	**CMD_TILE_TMB_STATUS** = `0x0250`
constexpr uint16_t	**FEB_REPLY_ERROR** = `0x0001`
constexpr uint16_t	**FEB_REPLY_SUCCESS** = `0x0000`
constexpr uint32_t	**MAX_LVDS_LINKS_PER_FEB** = `36`
constexpr uint32_t	**MAX_SLOWCONTROL_MESSAGE_SIZE** = `100 - 4`
constexpr uint32_t	**MAX_SLOWCONTROL_WRITE_MESSAGE_SIZE** = `(1 << 16) - 1`
constexpr uint32_t	**NMUTRIGCHANNELS** = `32`
constexpr uint32_t	**N_BITS_MUTRIG** = `2662`
constexpr uint32_t	**N_BYTES_MUPIX** = `48`
constexpr uint32_t	**N_BYTES_MUTRIG** = `333`
constexpr uint32_t	**N_CHIPS** = `8`
constexpr uint32_t	**N_CHIPS_MAX** = `12`
constexpr uint32_t	**N_FEBS** = `4`
constexpr uint32_t	**N_FEBS_QUAD** = `4`
constexpr uint32_t	**N_MUTRIGS_PER_FEB** = `4`
constexpr size_t	**N_TMB_MATRIX_TEMPERATURES** = `26`
constexpr size_t	**N_TMB_STATUS_VALUES** = `4`
constexpr size_t	**N_TMB_TEMPERATURES_VALUES** = `26+1`
constexpr size_t	**N_TMB_TEMPERATURE_IDS_VALUES** = `26\*2+1`
constexpr size_t	**N_TxTM_VALUES** = `26`
constexpr float	**TMB_TEMPERATURE_FACTOR** = `0.0078125`
const std::array< const std::string, 13 >	**adcnames** = `/* multi line expression */`
constexpr uint32_t	**dma_buf_nwords** = `dma\_buf\_size / sizeof(uint32\_t)`
constexpr size_t	**dma_buf_size** = `MUDAQ\_DMABUF\_DATA\_LEN`
constexpr uint32_t	**length** = `length\_32bits \* 4`
constexpr uint32_t	**length_32bits** = `12`
constexpr uint32_t	**max_requested_words** = `0x80000`
constexpr uint32_t	**nadcvals** = `13`

4.2.4 Detailed Description

This header centralizes configuration constants used across the MuPix MIDAS frontend, including parameters for DMA buffering, FEB and chip limits, slow control protocol constraints, and data format definitions.

It also declares the `mevent_t` structure, which defines the format for decoded readout events from MuPix hardware.

Constants include: * Maximum LVDS links per FEB * Number of FEBs and chips * DMA buffer sizing and usage * Maximum message sizes for slow control * Configuration payload lengths

These constants provide a consistent interface across the data and control subsystems of the DAQ framework.

4.2.5 Public Types Documentation

enum ADC_Command

```
enum ADC_Command { Reset = 0b1000, Configure = 0b0100, Measure = 0b0010 };
```

---

enum ADC_Mode

```
enum ADC_Mode { Single = 0b0100, Sequence = 0b0010, All = 0b0001 };
```

---

enum ADC_Mux_Address

```
enum ADC_Mux_Address { ref_vssa = 0, Baseline = 1, blpix = 2, thpix = 3, blpix_2 = 4, ThLow = 5, ThHigh = 6, TEST_OUT = 7, vssa = 8, thpix_2 = 9, VCAL = 10, VTemp1 = 11, VTemp2 = 12 };
```

4.2.6 Public Attributes Documentation

variable CMD_MUTRIG_ASIC_CFG

```
constexpr uint16_t CMD_MUTRIG_ASIC_CFG;
```

---

variable CMD_MUTRIG_ASIC_OFF

```
constexpr uint16_t CMD_MUTRIG_ASIC_OFF;
```

variable CMD_MUTRIG_CNT_RESET

```
constexpr uint16_t CMD_MUTRIG_CNT_RESET;
```

variable CMD_TILE_ASIC_PWR

```
constexpr uint16_t CMD_TILE_ASIC_PWR;
```

variable CMD_TILE_ASIC_PWROR

```
constexpr uint16_t CMD_TILE_ASIC_PWROR;
```

variable CMD_TILE_INJECTION_SETTING

```
constexpr uint16_t CMD_TILE_INJECTION_SETTING;
```

variable CMD_TILE_POWERMONITORS_READ

```
constexpr uint16_t CMD_TILE_POWERMONITORS_READ;
```

variable CMD_TILE_TEMPERATURES_READ

```
constexpr uint16_t CMD_TILE_TEMPERATURES_READ;
```

variable CMD_TILE_TEMPERATURES_READ_IDS

```
constexpr uint16_t CMD_TILE_TEMPERATURES_READ_IDS;
```

variable CMD_TILE_TMB_INIT

```
constexpr uint16_t CMD_TILE_TMB_INIT;
```

variable CMD_TILE_TMB_STATUS

```
constexpr uint16_t CMD_TILE_TMB_STATUS;
```

variable FEB_REPLY_ERROR

```
constexpr uint16_t FEB_REPLY_ERROR;
```

variable FEB_REPLY_SUCCESS

```
constexpr uint16_t FEB_REPLY_SUCCESS;
```

variable MAX_LVDS_LINKS_PER_FEB

```
constexpr uint32_t MAX_LVDS_LINKS_PER_FEB;
```

variable MAX_SLOWCONTROL_MESSAGE_SIZE

```
constexpr uint32_t MAX_SLOWCONTROL_MESSAGE_SIZE;
```

variable MAX_SLOWCONTROL_WRITE_MESSAGE_SIZE

```
constexpr uint32_t MAX_SLOWCONTROL_WRITE_MESSAGE_SIZE;
```

variable NMUTRIGCHANNELS

```
constexpr uint32_t NMUTRIGCHANNELS;
```

variable N_BITS_MUTRIG

```
constexpr uint32_t N_BITS_MUTRIG;
```

variable N_BYTES_MUPIX

```
constexpr uint32_t N_BYTES_MUPIX;
```

variable N_BYTES_MUTRIG

```
constexpr uint32_t N_BYTES_MUTRIG;
```

variable N_CHIPS

```
constexpr uint32_t N_CHIPS;
```

variable N_CHIPS_MAX

```
constexpr uint32_t N_CHIPS_MAX;
```

variable N_FEBS

```
constexpr uint32_t N_FEBS;
```

variable N_FEBS_QUAD

```
constexpr uint32_t N_FEBS_QUAD;
```

variable N_MUTRIGS_PER_FEB

```
constexpr uint32_t N_MUTRIGS_PER_FEB;
```

variable N_TMB_MATRIX_TEMPERATURES

```
constexpr size_t N_TMB_MATRIX_TEMPERATURES;
```

variable N_TMB_STATUS_VALUES

```
constexpr size_t N_TMB_STATUS_VALUES;
```

variable N_TMB_TEMPERATURES_VALUES

```
constexpr size_t N_TMB_TEMPERATURES_VALUES;
```

variable N_TMB_TEMPERATURE_IDS_VALUES

```
constexpr size_t N_TMB_TEMPERATURE_IDS_VALUES;
```

variable N_TxTM_VALUES

```
constexpr size_t N_TxTM_VALUES;
```

variable TMB_TEMPERATURE_FACTOR

```
constexpr float TMB_TEMPERATURE_FACTOR;
```

variable adcnames

```
const std::array<const std::string, 13> adcnames;
```

variable dma_buf_nwords

```
constexpr uint32_t dma_buf_nwords;
```

variable dma_buf_size

```
constexpr size_t dma_buf_size;
```

variable length

```
constexpr uint32_t length;
```

variable length_32bits

```
constexpr uint32_t length_32bits;
```

variable max_requested_words

```
constexpr uint32_t max_requested_words;
```

variable nadcvals

```
constexpr uint32_t nadcvals;
```

The documentation for this class was generated from the following file `midas_fe/constants.h`

4.3 File missing_hardware.h

FileList > **midas_fe** > **missing_hardware.h**

Go to the source code of this file

Hardware abstraction flags for simulation or emulation builds. More...

4.3.1 Detailed Description

This header defines macros used to compile the project in environments without physical hardware access. These flags help isolate and test the software stack in simulated or development environments.

- `NO_A10_BOARD`: Disables hardware-specific initialization and enables dummy interfaces.
- `EMULATE_HARDWARE_ERRORS`: Could be used to inject artificial errors for testing robustness.

This file is typically included early in the frontend logic and affects conditional compilation of hardware interfaces.

The documentation for this class was generated from the following file `midas_fe/missing_hardware.h`

4.4 File odb_setup.h

FileList > **midas_fe** > **odb_setup.h**

Go to the source code of this file

Default MIDAS ODB structure and initialization for Quads frontend. More...

- `#include "midas.h"`
- `#include "odbxx.h"`
- `#include "registers.h"`

4.4.1 Public Attributes

Type	Name
midas::odb	**settings**

4.4.2 Public Functions

Type	Name
constexpr std::array< T, N >	**filled_array** (const T & value)

4.4.3 Detailed Description

This header defines the structure of the Online Database (ODB) used by the Quads DAQ frontend, including readout configuration and DAQ command parameters.

The file maps the following ODB sections: * `/Equipment/Quads/Settings`: Contains readout options (e.g., datagen control, dummy mode, ASIC reset, maximum data words, etc.). * `/Equipment/Quads/DAQ`: Includes DAQ-related commands such as firmware loading.

It uses a `midas::odb` map for structured and type-safe access to parameters, and provides utility functions like `filled_array` for initializing default values.

Note:

This file must be included before using any ODB access in the frontend.

4.4.4 Public Attributes Documentation

variable settings

```
midas::odb settings;
```

4.4.5 Public Functions Documentation

function filled_array

```
template<typename T, std::size_t N> constexpr std::array< T, N > filled_array ( const T & value )
```

The documentation for this class was generated from the following file `midas_fe/odb_setup.h`

4.5 File utils.h

FileList > **midas_fe** > **utils.h**

Go to the source code of this file

Utility functions for bit-level parameter setting and ODB indexing. More...

- `#include "FEBSlowcontrolInterface.h"`
- `#include "mutrig_config.h"`
- `#include "Mutrig3Config.h"`
- `#include "bits_utils.h"`

4.5.1 Public Functions

Type	Name
int	**ChangeTDCTest** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
int	**ConfigureASICs** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings, uint8_t * bitpattern_w) *Configures all enabled ASICs across active FEBs using bit patterns.*
int	**ConfigureInjectASICs** (**FEBSlowcontrolInterface** & feb_sc, const std::vector< uint8_t > & inj_col, const std::vector< uint8_t > & inj_row) *Configure injection. Structure of injection 1.) Set VCal(a VDAC) in a different interface: chips for which VCal is set to none Zero will experience the injection This can however also be accomodated in this interface at some point, for now Chip Injection Selection via VCal 2.) Select the pixel you want to inject: DISCLAIMER: Due to chip design choices, always two neighbouring pixels will be injected at the same time. (Condition; int(col)/2=int(col+1)/2 [Integer Division] The selection is achieved via a crosshair selection: row and col adresses are set and all combinations of rows and cols will be anabled for injection. In this first approach we will only select one pixel at a time, ie. one row, one column. This should probably remain like this, due to the limited driving power of the Injection circuit.*
int	**ConfigureMuTRiGASICs** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings, uint8_t * bitpattern_w) *Configures all enabled ASICs across active MuTRiG FEBs using bit patterns.*
int	**ConfigureTDACs** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings) *Write TDACs to all enabled ASICs across active FEBs.*
int	**FullChipInjection** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings, uint8_t min_columns, uint8_t max_columns, uint8_t min_rows, uint8_t max_rows, uint32_t injection_pulse_duration, uint32_t num_repetitions, uint32_t wait_between_pulses)
int	**InitFEBs** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings) *Initializes all active Front-End Boards (FEBs) using slow control.*
int	**InjectASICs** (**FEBSlowcontrolInterface** & feb_sc, uint32_t injection_pulse_duration)
int	**InjectASICsInLoop** (**FEBSlowcontrolInterface** & feb_sc, uint32_t injection_pulse_duration, uint32_t num_repetitions, uint32_t wait_between_pulses)
int	**MuTRiGResetLVDSAddr** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
int	**MuTRiG_reset_asics** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
int	**MuTRiG_reset_counters** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
int	**MuTRiG_reset_datapath** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings) *Trigger Reset on MuTRiG FEB.*
int	**MuTRiG_reset_lvds** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
int	**TBinit** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
int	**UpdatePower** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
int	**UpdatePowerOverride** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
void	**adcContinuousReadout** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings)
uint64_t	**calculateADCCommand** (ADC_Command adcCommand, uint16_t adcDivisionFactor, ADC_Mode adcMode)
int	**create_dummy_event** (uint32_t * dma_buf_dummy, size_t eventSize, int nEvents, int serial_number) *Creates one or more dummy MIDAS events in memory.*
uint32_t	**generate_random_pixel_hit_swb** (uint32_t time_stamp) *Generates a simulated pixel hit for test data streams.*
uint32_t	**getODBIdx** (midas::odb odb, std::string name) *Gets the index of a named key in a MIDAS ODB object.*
uint32_t	**getOffset** (midas::odb odb, std::string name) *Computes the bit offset of a named field in a MIDAS ODB structure.*

Type	Name
void	**get_BiasDACs_from_odb** (midas::odb m_config, uint8_t * bitpattern_w, uint32_t asicIDx) *Extracts and writes Bias DAC configuration bits for a given ASIC.*
void	**get_ConfDACs_from_odb** (midas::odb m_config, uint8_t * bitpattern_w, uint32_t asicIDx) *Extracts and writes Configuration DAC bits for a given ASIC.*
void	**get_DACs_from_odb** (midas::odb m_nbits, midas::odb m_config, uint8_t * bitpattern_w, uint32_t asicIDx, std::string firstWord, std::string lastWord, bool inverted) *Writes DAC configuration bits from ODB to the bit pattern buffer.*
void	**get_VDACs_from_odb** (midas::odb m_config, uint8_t * bitpattern_w, uint32_t asicIDx) *Extracts and writes Voltage DAC bits for a given ASIC.*
int	**read_tdac_file** (vector< uint32_t > & vec, std::string path) *Read TDAC file from a given path.*
int	**resetASICs** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings) *Sends a reset to all MuPix chips.*
void	**sendCommand** (**FEBSlowcontrolInterface** & feb_sc, midas::odb m_settings, uint64_t command)
int	**to_signed_12b** (uint32_t i)
int	**to_signed_16b** (uint32_t i)

4.5.2 Detailed Description

This header provides common utility routines used in frontend configuration and data preparation tasks. These functions support bitfield encoding and MIDAS Online Database (ODB) access.

Included Functions: * `getODBIdx`: Retrieves the index of a named subkey in a `midas::odb` structure. * `getOffset`: Computes a bit offset for a named subkey in the ODB. * `get_DACs_from_odb`: Writes DAC configuration bits from ODB to the bit pattern buffer. * `get_BiasDACs_from_odb`: Extracts and writes Bias DAC configuration bits for a given ASIC. * `get_ConfDACs_from_odb`: Extracts and writes Configuration DAC bits for a given ASIC. * `get_VDACs_from_odb`: Extracts and writes Voltage DAC bits for a given ASIC. * `InitFEBs`: Initializes all active Front-End Boards (FEBs) using slow control. * `ConfigureASICs`: Configures all enabled ASICs across active FEBs using bit patterns. * `generate_random_pixel_hit_swb`: Generates a simulated pixel hit for test data streams. * `create_dummy_event`: Creates one or more dummy MIDAS events in memory.

These utilities simplify frontend implementation by abstracting repetitive logic related to bit encoding and dynamic configuration.

Note:

These functions are performance-sensitive and should be used with care in high-rate paths such as configuration loops.

4.5.3 Public Functions Documentation

function ChangeTDCTest

```
int ChangeTDCTest ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function ConfigureASICs

Configures all enabled ASICs across active FEBs using bit patterns.

```
int ConfigureASICs ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings, uint8_t * bitpattern_w )
```

For each ASIC enabled by the ASIC mask, retrieves DAC configurations from the ODB, encodes them into a payload, and sends them to the corresponding FEB.

Parameters:

- `feb_sc` Reference to the FEB slow control interface.
- `m_settings` MIDAS ODB object containing DAQ and config sections.
- `bitpattern_w` Temporary buffer used to build the configuration bitstream.

Returns:

FE_SUCCESS if all ASICs configured successfully; error code otherwise.

---

function ConfigureInjectASICs

Configure injection. Structure of injection 1.) Set VCal(a VDAC) in a different interface: chips for which VCal is set to none Zero will experience the injection This can however also be accomodated in this interface at some point, for now Chip Injection Selection via VCal 2.) Select the pixel you want to inject: DISCLAIMER: Due to chip design choices, always two neighbouring pixels will be injected at the same time. (Condition; int(col)/2=int(col+1)/2 [Integer Division] The selection is achieved via a crosshair selection: row and col adresses are set and all combinations of rows and cols will be anabled for injection. In this first approach we will only select one pixel at a time, ie. one row, one column. This should probably remain like this, due to the limited driving power of the Injection circuit.

```
int ConfigureInjectASICs ( FEBSlowcontrolInterface & feb_sc, const std::vector< uint8_t > & inj_col, const std::vector< uint8_t > & inj_row )
```

128x7bit long register [Enable Injection Row<0>,Enable Injection Row<1>,NCNC,AmpOut,Enable Injection Column, HitBusEnable]-->(Shift direction)--> The implementation for the characterisation setup can be found here: https://bitbucket.org/mu3e/mupix8_daq/src/master/library/sensors/MuPix11.cpp

3.) Send Injection Command to trigger injection [4 bit Chip Address | 6 bit Inject Command | 10 bit Duration | zeroes/ones/random]

Parameters:

- `feb_sc` Reference to the FEB slow control interface.
- `inj_col` Reference to the FEB slow control interface.
- `inj_row` MIDAS ODB object containing DAQ and config sections.

Returns:

FE_SUCCESS if all ASICs configured successfully; error code otherwise.

---

function ConfigureMuTRiGASICs

Configures all enabled ASICs across active MuTRiG FEBs using bit patterns.

```
int ConfigureMuTRiGASICs ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings, uint8_t * bitpattern_w )
```

For each ASIC enabled by the ASIC mask, retrieves DAC configurations from the ODB, encodes them into a payload, and sends them to the corresponding FEB.

Parameters:

- `feb_sc` Reference to the FEB slow control interface.
- `m_settings` MIDAS ODB object containing DAQ and config sections.
- `bitpattern_w` Temporary buffer used to build the configuration bitstream.

Returns:

FE_SUCCESS if all ASICs configured successfully; error code otherwise.

---

function ConfigureTDACs

Write TDACs to all enabled ASICs across active FEBs.

```
int ConfigureTDACs ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

For each ASIC enabled by the ASIC mask, retrieves TDAC configurations from a TDAC file, encodes them into a payload, and sends them to the corresponding FEB.

Parameters:

- `feb_sc` Reference to the FEB slow control interface.
- `m_settings` MIDAS ODB object containing DAQ and config sections.

Returns:

FE_SUCCESS if all ASICs configured successfully; error code otherwise.

---

function FullChipInjection

```
int FullChipInjection ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings, uint8_t min_columns, uint8_t max_columns, uint8_t min_rows, uint8_t max_rows, uint32_t injection_pulse_duration, uint32_t num_repetitions, uint32_t wait_between_pulses )
```

---

function InitFEBs

Initializes all active Front-End Boards (FEBs) using slow control.

```
int InitFEBs ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

Writes unique FPGA IDs and configures LVDS link settings for each active FEB.

Parameters:

- `feb_sc` Reference to the FEB slow control interface.
- `m_settings` MIDAS ODB object containing DAQ link configuration.

Returns:

FE_SUCCESS on success.

---

function InjectASICs

```
int InjectASICs ( FEBSlowcontrolInterface & feb_sc, uint32_t injection_pulse_duration )
```

---

function InjectASICsInLoop

```
int InjectASICsInLoop ( FEBSlowcontrolInterface & feb_sc, uint32_t injection_pulse_duration, uint32_t num_repetitions,
uint32_t wait_between_pulses )
```

---

function MuTRiGResetLVDSAddr

```
int MuTRiGResetLVDSAddr ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function MuTRiG_reset_asics

```
int MuTRiG_reset_asics ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function MuTRiG_reset_counters

```
int MuTRiG_reset_counters ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function MuTRiG_reset_datapath

Trigger Reset on MuTRiG FEB.

```
int MuTRiG_reset_datapath ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

Parameters:

- `feb_sc` : slow control interface

Returns:

FE_SUCCESS

---

function MuTRiG_reset_lvds

```
int MuTRiG_reset_lvds ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function TBinit

```
int TBinit ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function UpdatePower

```
int UpdatePower ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function UpdatePowerOverride

```
int UpdatePowerOverride ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function adcContinuousReadout

```
void adcContinuousReadout ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

---

function calculateADCCommand

```
uint64_t calculateADCCommand ( ADC_Command adcCommand, uint16_t adcDivisionFactor, ADC_Mode adcMode )
```

---

function create_dummy_event

Creates one or more dummy MIDAS events in memory.

```
int create_dummy_event ( uint32_t * dma_buf_dummy, size_t eventSize, int nEvents, int serial_number )
```

Fills a provided DMA buffer with synthetic event headers and pixel hits. This simulates what a real event would look like for testing purposes.

Parameters:

- `dma_buf_dummy` Pointer to the target DMA buffer.
- `eventSize` Size of a single event (in words).
- `nEvents` Number of events to create.
- `serial_number` Starting event serial number (incremented internally).

Returns:

Updated serial number after all events are written.

---

function generate_random_pixel_hit_swb

Generates a simulated pixel hit for test data streams.

```
uint32_t generate_random_pixel_hit_swb ( uint32_t time_stamp )
```

Encodes a random chip, column, row, and time-over-threshold (ToT) value into a 32-bit hit word.

Parameters:

- `time_stamp` Timestamp to embed in the hit word.

Returns:

uint32_t Encoded hit data.

---

function getODBIdx

Gets the index of a named key in a MIDAS ODB object.

```
uint32_t getODBIdx ( midas::odb odb, std::string name )
```

Searches the top-level keys in a `midas::odb` structure and returns the index of the key matching the provided name.

Parameters:

- `odb` The `midas::odb` object representing the configuration section.
- `name` The name of the subkey to find.

Returns:

uint32_t The index of the key if found, otherwise the total count.

---

function getOffset

Computes the bit offset of a named field in a MIDAS ODB structure.

```
uint32_t getOffset ( midas::odb odb, std::string name )
```

Iterates through subkeys in an ODB group to determine the bit offset of a given field, assuming each key represents a sequential bit region.

Parameters:

- `odb` The `midas::odb` group containing configuration fields.
- `name` The name of the target field.

Returns:

uint32_t The cumulative bit offset of the named field.

---

function get_BiasDACs_from_odb

Extracts and writes Bias DAC configuration bits for a given ASIC.

```
void get_BiasDACs_from_odb ( midas::odb m_config, uint8_t * bitpattern_w, uint32_t asicIDx )
```

Calls `get_DACs_from_odb()` with hardcoded range for Bias DAC fields.

Parameters:

- `m_config` ODB root configuration node.
- `bitpattern_w` Target buffer for encoded configuration bits.
- `asicIDx` Index of the ASIC to configure.

---

function get_ConfDACs_from_odb

Extracts and writes Configuration DAC bits for a given ASIC.

```
void get_ConfDACs_from_odb ( midas::odb m_config, uint8_t * bitpattern_w, uint32_t asicIDx )
```

Calls `get_DACs_from_odb()` with hardcoded range for Conf DAC fields.

Parameters:

- `m_config` ODB root configuration node.
- `bitpattern_w` Target buffer for encoded configuration bits.
- `asicIDx` Index of the ASIC to configure.

---

function get_DACs_from_odb

Writes DAC configuration bits from ODB to the bit pattern buffer.

```
void get_DACs_from_odb ( midas::odb m_nbits, midas::odb m_config, uint8_t * bitpattern_w, uint32_t asicIDx, std::string firstWord,
std::string lastWord, bool inverted )
```

This function reads DAC values from an ODB configuration and writes them into a byte buffer as bit-encoded values, using the `setParameter` function.

Parameters:

- `m_nbits` ODB subkey describing bit widths for each DAC parameter.
- `m_config` ODB subkey containing DAC values per ASIC.
- `bitpattern_w` Target buffer for encoded configuration bits.
- `asicIDx` Index of the ASIC to configure.
- `firstWord` Name of the first DAC parameter to include.
- `lastWord` Name of the last DAC parameter to include.
- `inverted` Whether to encode bits in MSB-first (inverted) order.

---

function get_VDACs_from_odb

Extracts and writes Voltage DAC bits for a given ASIC.

```
void get_VDACs_from_odb ( midas::odb m_config, uint8_t * bitpattern_w, uint32_t asicIDx )
```

Calls `get_DACs_from_odb()` with hardcoded range for VDAC fields.

Parameters:

- `m_config` ODB root configuration node.
- `bitpattern_w` Target buffer for encoded configuration bits.
- `asicIDx` Index of the ASIC to configure.

---

function read_tdac_file

Read TDAC file from a given path.

```
int read_tdac_file ( vector< uint32_t > & vec, std::string path )
```

Tries to read a TDAC file from a given path. If the path is not present an empty TDAC configuration (no pixel is masked) is created.

Parameters:

- `vec` Reference to the bitpattern vector.
- `path` Path to th file.

Returns:

FE_SUCCESS if the file is read; error code otherwise.

---

function resetASICs

Sends a reset to all MuPix chips.

```
int resetASICs ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings )
```

Sends a reset to all MuPix chips.

Parameters:

- `feb_sc` Reference to the FEB slow control interface.
- `m_settings` MIDAS ODB object containing DAQ and config sections.

Returns:

FE_SUCCESS on success.

---

function sendCommand

```
void sendCommand ( FEBSlowcontrolInterface & feb_sc, midas::odb m_settings, uint64_t command )
```

---

function to_signed_12b

```
int to_signed_12b ( uint32_t i )
```

---

function to_signed_16b

```
int to_signed_16b ( uint32_t i )
```

The documentation for this class was generated from the following file `midas_fe/utils.h`

5. Registers

RegName	Reg/Bit/Index	Doc	Board	TYPE

RegName	Reg/Bit/ Index	Doc	Board	TYPE
LED_REGISTER_W	REG: 0x00	USED TO CHANGE LEDS ON THE BOARDS	FARM	WRITE-REG
RESET_REGISTER_W	REG: 0x01	RESET REGISTER	ALL	WRITE-REG
RESET_BIT_ALL	BIT: 0	RESET BIT TO RESET ALL	ALL	-
RESET_BIT_DATAGEN	BIT: 1	RESET BIT FOR THE DATAGENERATOR WHICH IS GENERATING THE LINK DATA FROM FEBS	SWB	-
RESET_BIT_SWB_STREAM_MERGER	BIT: 2	RESET ROUND ROBIN MERGER	SWB	-
RESET_BIT_SWB_TIME_MERGER	BIT: 3	RESET TIME MERGER	SWB	-
RESET_BIT_RECEIVER	BIT: 4	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_DATAFIFO	BIT: 5	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_FIFOPLL	BIT: 6	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_SC_SECONDARY	BIT: 7	RESET BIT FOR THE SLOWCONTROL SECONDARY	SWB	-
RESET_BIT_SC_MAIN	BIT: 8	RESET BIT FOR THE SLOWCONTROL MAIN	SWB	-
RESET_BIT_PCIE_LOCAL	BIT: 9	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_TOP_PROC	BIT: 10	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_PCIE_APPL	BIT: 12	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_EVENT_COUNTER	BIT: 13	RESET BIT FOR THE SWB_DATA_DEMERGER	SWB	-
RESET_BIT_DMA_EVAL	BIT: 14	RESET BIT FOR DMA EVALUATIONG / MONITORING FOR PCIE 0	ALL	-
RESET_BIT_LINK_TEST	BIT: 15	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_RUN_START_ACK	BIT: 16	REST BIT FOR SEEING THE RUN ACK IN RUN_CONTROL	SWB	-
RESET_BIT_RUN_END_ACK	BIT: 17	REST BIT FOR SEEING THE RUN END IN RUN_CONTROL	SWB	-
RESET_BIT_NIOS	BIT: 18	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_DDR	BIT: 19	RESET BIT FOR DDR CONTROL ENTITIE	FARM	-
RESET_BIT_DATAFLOW	BIT: 20	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_LINK_MERGER	BIT: 21	NOT USED AT THE MOMENT	ALL	-
RESET_BIT_DATA_PATH	BIT: 22	RESET BIT FOR THE DATA PATH	SWB	-
RESET_BIT_FARM_DATA_PATH	BIT: 23	RESET BIT FOR THE DATA PATH	FARM	-
RESET_BIT_FARM_STREAM_MERGER	BIT: 24	RESET BIT FOR FARM STREAM MERGER	FARM	-
RESET_BIT_FARM_TIME_MERGER	BIT: 25	RESET BIT FOR FARM TIME MERGER	FARM	-
RESET_BIT_LINK_LOCKED	BIT: 26	RESET BIT FOR LINK LOCKED BIT	SWB/ FARM	-
RESET_BIT_GLOBAL_TS	BIT: 27	RESET BIT FOR GLOBAL TIME COUNTER	SWB/ FARM	-
RESET_BIT_FARM_BLOCK	BIT: 28	RESET BIT FOR THE DATA PATH	FARM	-
RESET_BIT_SWB_COUNTERS	BIT: 29	RESET BIT FOR THE COUNTERS ON THE SWB	SWB	-

RegName	Reg/Bit/ Index	Doc	Board	TYPE
RESET_BIT_PCIE	BIT: 31	NOT USED AT THE MOMENT	ALL	-
DATAGENERATOR_REGISTER_W	REG: 0x02	REGISTER TO CONTROL THE DATAGENERATOR WHICH IS GENERATING THE LINK DATA FROM FEBS	SWB	WRITE-REG
DATAGENERATOR_BIT_ENABLE	BIT: 0	NOT USED AT THE MOMENT	SWB	-
DATAGENERATOR_BIT_ENABLE_PIXEL	BIT: 1	BIT TO ENABLE PIXEL DATA	SWB	-
DATAGENERATOR_BIT_ENABLE_FIBRE	BIT: 2	BIT TO ENABLE FIBRE DATA	SWB	-
DATAGENERATOR_BIT_ENABLE_TILE	BIT: 3	BIT TO ENABLE TILE DATA	SWB	-
DATAGENERATOR_BIT_ENABLE_TEST	BIT: 4	NOT USED AT THE MOMENT	SWB	-
DATAGENERATOR_BIT_DMA_HALFFUL_MODE	BIT: 5	NOT USED AT THE MOMENT	SWB	-
DATAGENERATOR_FRACCOUNT_RANGE	RANGE: 15 DOWNTO 8	NOT USED AT THE MOMENT	SWB	-
DATAGENERATOR_NPIXEL_RANGE	RANGE: 15 DOWNTO 8	NOT USED AT THE MOMENT	SWB	-
DATAGENERATOR_NFIBRE_RANGE	RANGE: 23 DOWNTO 16	NOT USED AT THE MOMENT	SWB	-
DATAGENERATOR_NTILE_RANGE	RANGE: 31 DOWNTO 24	NOT USED AT THE MOMENT	SWB	-
DATAGENERATOR_DIVIDER_REGISTER_W	REG: 0x03	REGISTER TO SLOW DOWN THE DATAGENERATOR WHICH IS GENERATING THE LINK DATA FROM FEBS	SWB	WRITE-REG
SWB_DATA_TYPE_REGISTER_W	REG: 0x04	SET THE DIFFERENT DATA TYPES (6BITS) FOR THE DATA PATHS ON THE SWB BOARD	SWB	WRITE-REG
DMA_CONTROL_COUNTER_RANGE	RANGE: 15 DOWNTO 0	NOT USED AT THE MOMENT	ALL	-
DMA_SLOW_DOWN_REGISTER_W	REG: 0x06	NOT USED AT THE MOMENT	ALL	WRITE-REG
LINK_TEST_REGISTER_W	REG: 0x07	NOT USED AT THE MOMENT	ALL	WRITE-REG
LINK_TEST_BIT_ENABLE	BIT: 0	NOT USED AT THE MOMENT	ALL	-
RUN_NR_REGISTER_W	REG: 0x08	REGISTER TO WRITE THE CURRENT RUN NUMBER	SWB	WRITE-REG
RUN_NR_ADDR_REGISTER_W	REG: 0x09	ASK FOR RUN NUMBER OF FEB WITH THIS ADDR	SWB	WRITE-REG
FEB_ENABLE_REGISTER_W	REG: 0x0A	ENABLE REGISTER FOR FEBS FOR RUN CONTROL AND SLOWCONTROL	SWB	WRITE-REG
DATA_LINK_MASK_REGISTER_W	REG: 0x0B	NOT USED AT THE MOMENT	SWB	WRITE-REG
GET_N_DMA_WORDS_REGISTER_W	REG: 0x0C	NUMBER OF REQUESTED WORDS WHICH WILL BE SEND VIA DMA	SWB & FARM	WRITE-REG

RegName	Reg/Bit/ Index	Doc	Board	TYPE
SC_MAIN_ENABLE_REGISTER_W	REG: 0x0D	REG THAT THE SLOWCONTROL MAIN ONLY STARTS ON A 0->1 TRANSITION	SWB	WRITE-REG
SC_MAIN_LENGTH_REGISTER_W	REG: 0x0E	LENGHT (15 DOWNTO 0) FOR SLOWCONTROL PACKAGE	SWB	WRITE-REG
SWB_GENERIC_MASK_REGISTER_W	REG: 0x0F	MASK GENERIC LINKS FOR DEV SETUP	SWB	WRITE-REG
SWB_LINK_MASK_PIXEL_REGISTER_W	REG: 0x10	MASK PIXEL LINKS	SWB	WRITE-REG
SWB_LINK_MASK_SCIFI_REGISTER_W	REG: 0x11	MASK SCIFI LINKS	SWB	WRITE-REG
SWB_LINK_MASK_TILES_REGISTER_W	REG: 0x12	MASK TILE LINKS	SWB	WRITE-REG
SWB_READOUT_STATE_REGISTER_W	REG: 0x13	READOUT STATE	SWB	WRITE-REG
USE_BIT_GEN_LINK	BIT: 0	READOUT STATE WHERE LINK DATA IS GENERATED (FOR DEBUGGING)	SWB & FARM	-
USE_BIT_STREAM	BIT: 1	READOUT STATE WHERE LINK DATA IS READ OUT IN ROUND-ROBIN	SWB & FARM	-
USE_BIT_MERGER	BIT: 2	READOUT STATE WHERE LINK DATA IS TIME ALIGNED (TREE)	SWB & FARM	-
USE_BIT_GEN_MERGER	BIT: 4	READOUT STATE WHERE THE TIME ALIGNED DATA IS GENERATED (FOR DEBUGGING)	SWB & FARM	-
USE_BIT_FARM	BIT: 5	READOUT STATE WHERE THE DATA IS SEND TO THE FARM (1) ELSE (0) IT IS READOUT VIA DMA ON THE SWB	SWB & FARM	-
USE_BIT_TEST	BIT: 6	NOT USED AT THE MOMENT	SWB & FARM	-
USE_BIT_PIXEL_US	BIT: 7	READOUT STATE TO ONLY READOUT US PIXEL DATA VIA DMA (FOR DEBUGGING)	SWB & FARM	-
USE_BIT_PIXEL_DS	BIT: 8	READOUT STATE TO ONLY READOUT DS PIXEL DATA VIA DMA (FOR DEBUGGING)	SWB & FARM	-
USE_BIT_SCIFI	BIT: 9	READOUT STATE TO ONLY READOUT SCIFI DATA VIA DMA (FOR DEBUGGING)	SWB & FARM	-
USE_BIT_TEST_ERROR	BIT: 10	READOUT STATE FOR TESTING AN ERROR HANDLING IN THE TIME MERGER USING GENERATED DATA	SWB & FARM	-
USE_BIT_DDR	BIT: 11	READOUT STATE FOR USING THE DDR MEMORY	FARM	-
USE_BIT_ALL	BIT: 12	READOUT STATE TO ONLY READOUT ALL DATA IN ROUND ROBIN VIA DMA (FOR DEBUGGING)	SWB	-

RegName	Reg/Bit/ Index	Doc	Board	TYPE
USE_BIT_TEST_DATA	BIT: 13	READOUT STATE WHERE LINK DATA IS REAL DATA FROM MEM FILE (FOR SIMULATION	SWB	-
USE_BIT_SUBHDR_SUPPRESS	BIT: 14	READOUT STATE WHERE SUBHEADERS ARE SUPPRESSED	SWB	-
USE_BIT_HEAD_SUPPRESS	BIT: 15	READOUT STATE WHERE HEADERS ARE SUPPRESSED	SWB	-
USE_BIT_INJECTION	BIT: 16	READOUT STATE TO INJECTED DATA	SWB & FARM	-
USE_BIT_WRITE_BUFFER_INJECTION	BIT: 17	READOUT STATE TO INJECTED DATA	SWB & FARM	WRITE-REG
USE_BIT_GENERIC	BIT: 18	READOUT STATE TO ONLY READOUT GENERIC DATA VIA DMA (FOR DEBUGGING)	SWB	-
USE_BIT_FEB_SYNC	BIT: 19	SETTING THIS BIT TO ONE ENABLES THE SYNC OF THE FEB DATA	SWB	-
SWB_READOUT_LINK_REGISTER_W	REG: 0x14	NOT USED AT THE MOMENT	SWB & FARM	WRITE-REG
SWB_COUNTER_REGISTER_W	REG: 0x15	ADDR REGISTER TO READOUT COUNTER VALUES FROM THE SWB, TO HAVE MORE INFORMATION ABOUT THE COUNTER LOOK AT A10_COUNTER.MD	SWB	WRITE-REG
FARM_READOUT_STATE_REGISTER_W	REG: 0x;	READOUT STATE	FARM	WRITE-REG
FARM_DATA_TYPE_REGISTER_W	REG: 0x17	DATA TYPE FOR READOUT	FARM	WRITE-REG
FARM_DATA_TYPE_ADDR_RANGE	RANGE: 1 DOWNTO 0	DATA TYPE: "00" = PIXEL, "01" = SCIFI, "10" = TILES	FARM	-
FARM_EVENT_ID_ADDR_RANGE	RANGE: 17 DOWNTO 2	MIDAS EVENT ID	FARM	-
SWB_SUBHEAD_SUPPRESS_REGISTER_W	REG: 0x19	SUBHEADER SUPPRESSION PER LINK	SWB	WRITE-REG
SWB_HEAD_SUPPRESS_REGISTER_W	REG: 0x1A	HEADER SUPPRESSION PER LINK	SWB	WRITE-REG
SWB_ZERO_HISTOS_REGISTER_W	REG: 0x1B	TODO	SWB	WRITE-REG
SWB_HISTO_ADDR_REGISTER_W	REG: 0x1C	TODO	SWB	WRITE-REG
SWB_HISTO_CHIP_SELECT_REGISTER_W	REG: 0x1D	TODO	SWB	WRITE-REG
SWB_HISTO_LINK_SELECT_REGISTER_W	REG: 0x1E	TODO	SWB	WRITE-REG
SWB_LOOKUP_CTRL_REGISTER_W	REG: 0x1F	CTRL REGISTER TO WRITE CHIP LOOKUP	SWB	WRITE-REG

RegName	Reg/Bit/Index	Doc	Board	TYPE
SWB_LOOKUP_DS_CTRL_REGISTER_W	REG: 0x20	CTRL REGISTER TO WRITE CHIP LOOKUP	SWB	WRITE-REG
SWB_LOOKUP_CTRL_ADDR_RANGE	RANGE: 6 DOWNTO 0	ADDR FOR THE RAM	SWB	-
SWB_LOOKUP_CTRL_COMMAND_RANGE	RANGE: 8 DOWNTO 7	COMMAND FOR THE STATE MACHINE	SWB	-
SWB_LOOKUP_CTRL_VALUE_RANGE	RANGE: 22 DOWNTO 9	LOOKUP VALUE	SWB	-
DDR_BIT_ENABLE_A	BIT: 0	ENABLE STATEMACHINE OF DDR-A	FARM	-
DDR_BIT_COUNTERTEST_A	BIT: 1	ENABLE COUNTER TEST (1) OR DATAFLOW (0) OF DDR-A	FARM	-
DDR_COUNTERSEL_RANGE_A	RANGE: 15 DOWNTO 14	"01" -> GET POSERR_REG, "10" -> GET COUNTERR_REG ELSE CUR TIME COUNTER WRITTEN TO DDR	FARM	-
DDR_RANGE_A	RANGE: 15 DOWNTO 0	RANGE FOR DDR A	FARM	-
DDR_BIT_ENABLE_B	BIT: 16	ENABLE STATEMACHINE OF DDR-B	FARM	-
DDR_BIT_COUNTERTEST_B	BIT: 17	ENABLE COUNTER TEST (1) OR DATAFLOW (0) OF DDR-B	FARM	-
DDR_COUNTERSEL_RANGE_B	RANGE: 31 DOWNTO 30	"01" -> GET POSERR_REG, "10" -> GET COUNTERR_REG ELSE CUR TIME COUNTER WRITTEN TO DDR	FARM	-
DDR_RANGE_B	RANGE: 31 DOWNTO 16	RANGE FOR DDR B	FARM	-
FARM_LINK_MASK_REGISTER_W	REG: 0x22	MASK FARM LINKS	FARM	WRITE-REG
FARM_ID_REGISTER_W	REG: 0x26	FARM ID WRITTEN TO THE RESERVED FILED OF THE MIDAS BANK	FARM	WRITE-REG
FARM_EVENT_ID_REGISTER_W	REG: 0x2B	SET MIDAS EVENT ID (15 DOWNTO 0) AND TRIGGER MASK (31 DOWNTO 16)	FARM	WRITE-REG
XCVR_CTRL_REGISTER_W	REG: 0x2F	SELECT INTERNAL XCVR-REGISTER	SWB	WRITE-REG
XCVR_CTRL_CH_RANGE	RANGE: 21 DOWNTO 16	XCVR CHANNEL SELECTOR	SWB	-
XCVR_CTRL_REG_RANGE	RANGE: 7 DOWNTO 0	XCVR REGISTER SELECTOR	SWB	-
GET_N_GPU_EVENTS_REGISTER_W	REG: 0x35	NUMBER OF REQUESTED GPU EVENTS WHICH WILL BE SEND VIA DMA	FARM	WRITE-REG

RegName	Reg/Bit/Index	Doc	Board	TYPE
STATUS_REGISTER_R	REG: 0xFC00	NOT IN USE	FEB_ALL	-
GIT_HASH_REGISTER_R	REG: 0xFC01	CONTAINS THE GIT HASH OF USED FIRMWARE	FEB_ALL	-
FPGA_TYPE_REGISTER_R	REG: 0xFC02	CONTAINS FPGA TYPE 111010: MUPIX, 111000 : MUTRIG	FEB_ALL	-
FPGA_ID_REGISTER_RW	REG: 0xFC03	FGPA ID	FEB_ALL	-
CMD_LEN_REGISTER_RW	REG: 0xFC04	LENGTH OF DATA TO READ IN A NIOS RPC CALL	FEB_ALL	-
CMD_OFFSET_REGISTER_RW	REG: 0xFC05	POSITON OF THE DATA TO READ IN A NIOS RPC CALL	FEB_ALL	-
RUN_STATE_RESET_BYPASS_REGISTER_RW	REG: 0xFC06	USED TO BYPASS THE RESET SYSTEM AND RUN WITHOUT A CLOCK BOX	FEB_ALL	-
RUN_STATE_RANGE	RANGE: 31 DOWNTO 16	NOT IN USE	FEB_ALL	-
RESET_BYPASS_RANGE	RANGE: 7 DOWNTO 0	NOT IN USE	FEB_ALL	-
RESET_BYPASS_BIT_REQUEST	BIT: 8	SENDS A BYPASS RESET COMMAND	FEB_ALL	-
RESET_BYPASS_BIT_ENABLE	BIT: 9	ENABLES THE USE OF THE RESET BYPASS	FEB_ALL	-
RESET_PAYLOAD_REGISTER_RW	REG: 0xFC07	PAYLOAD FOR THE RESET BYPASS COMMANDS	FEB_ALL	-
RESET_OPTICAL_LINKS_REGISTER_RW	REG: 0xFC08	RESET FIREFLY	FEB_ALL	-
RESET_PHASE_REGISTER_R	REG: 0xFC09	PHASE BETWEEN RESET RX CLOCK AND GLOBAL CLK	FEB_ALL	-
MERGER_RATE_REGISTER_R	REG: 0xFC0A	OUTPUT RATE OF THE DATA MERGER	FEB_ALL	-
ARRIA_TEMP_REGISTER_RW	REG: 0xFC10	ARRIAV INTERNAL TEMP SENSE	FEB_ALL	-
MAX10_ADC_0_1_REGISTER_R	REG: 0xFC11	MAX10 ADC DATA	FEB_ALL	-
NONINCREMENTING_TEST_REGISTER_RW	REG: 0xFC24	TESTING NONINCREMENTING READS/WRITES TO FIFO (NOT IN USE)	FEB_ALL	-
MAX10_VERSION_REGISTER_R	REG: 0xFC25	GIT HASH OF THE MAX10 FIRMWARE	FEB_ALL	-
RUN_START_DENIAL_REGISTER_R	REG: 0xFC2F	REASON FOR DENIED RUN START ACK	FEB_ALL	-
RUN_NUMBER_REGISTER_R	REG: 0xFC2F	RUN NUMBER ON THE FEB	FEB_ALL	-

RegName	Reg/Bit/ Index	Doc	Board	TYPE
LVDS_STATUS_REGISTER_R	REG: 0x4101	NONINCREMENTING READ OF LVDS STATUS REGISTER BLOCK, 1 WORD FOR EACH LVDS LINK FROM HERE ON	FEB MUTRIG	-
LVDS_STATUS_START_REGISTER_W	REG: 0x1100	START OF LVDS STATUS REGISTER BLOCK, 1 WORD FOR EACH LVDS LINK FROM HERE ON	FEB	WRITE-REG
LVDS_STATUS_ALIGN_CNT_RANGE	RANGE: 27 DOWNTO 22	COUNTS HOW OFTEN THE ALIGNER HAS SHIFTED THE LINK	FEB	-
LVDS_STATUS_ARRIVAL_PHASE_RANGE	RANGE: 21 DOWNTO 20	PHASE OF HIT ARRIVAL	FEB	-
LVDS_STATUS_OUTOF_PHASE_RANGE	RANGE: 15 DOWNTO 0	COUNTS HOW OFTEN THE HIT ARRIVAL TIME IS NOT IN THE SAME 4 CYCLE PHASE	FEB	-

RegName	Reg/Bit/Index	Doc	Board	TYPE
MP_CTRL_COMBINED_START_REGISTER_W	REG: 0x0400	USED TO SEND COMBINED CONFIG DATA TO THE MUPIX, CHIP ID ENCODED IN ADDR AS REGADDR + CHIPID	MP_FEB	WRITE-REG
MP_CTRL_TDAC_START_REGISTER_W	REG: 0x0430	USED TO SEND TDAC TO THE MUPIX, CHIP ID ENCODED IN ADDR AS REGADDR + CHIPID	MP_FEB	WRITE-REG
MP_CTRL_CHIP_SELECT1_REGISTER_W	REG: 0x0460	USED TO SPECIFY THE CHIP ID IN CASE OF DIRECT SPI OR DIRECT REGISTER CONFIGURATION	MP_FEB	WRITE-REG
MP_CTRL_CHIP_SELECT2_REGISTER_W	REG: 0x0461	USED TO SPECIFY THE CHIP ID IN CASE OF DIRECT SPI OR DIRECT REGISTER CONFIGURATION	MP_FEB	WRITE-REG
MP_CTRL_BIAS_REGISTER_W	REG: 0x0462	IF YOU WANT TO WRITE THE MUPIX BIAS REG ONLY, SEND DATA HERE	MP_FEB	WRITE-REG
MP_CTRL_CONF_REGISTER_W	REG: 0x0463	IF YOU WANT TO WRITE THE MUPIX CONF REG ONLY, SEND DATA HERE	MP_FEB	WRITE-REG
MP_CTRL_VDAC_REGISTER_W	REG: 0x0464	IF YOU WANT TO WRITE THE MUPIX VDAC REG ONLY, SEND DATA HERE	MP_FEB	WRITE-REG
MP_CTRL_SLOW_DOWN_REGISTER_W	REG: 0x0465	DIVISION FACTOR FOR THE MUPIX SPI CLK	MP_FEB	WRITE-REG
MP_CTRL_SPI_BUSY_REGISTER_R	REG: 0x0466	INDICATES IF THE MUPIX SPI IS BUSY, DO NOT SEND NEW DATA	MP_FEB	-
MP_CTRL_DIRECT_SPI_ENABLE_REGISTER_W	REG: 0x0467	ENABLE DIRECT SPI CONFIGURATION MODE FOR MUPIX	MP_FEB	WRITE-REG
MP_CTRL_SPI_ENABLE_REGISTER_W	REG: 0x0468	ENABLE SPI CONFIGURATION MODE FOR MUPIX (DIRECT SPI NEEDS TO BE DISABLED IN THIS CASE)	MP_FEB	WRITE-REG
MP_CTRL_DIRECT_SPI_BUSY_REGISTER_R	REG: 0x0469	CONTAINS 1 BIT FOR EACH SPI BUS, 1 IF BUSY	MP_FEB	-
MP_CTRL_DIRECT_SPI_START_REGISTER_W	REG: 0x046A	REGISTER FOR DIRECT SPI CONFIGURATION MODE, NEEDS TO BE ENABLED FIRST	MP_FEB	WRITE-REG
MP_CTRL_DIRECT_SPI_CHIP_M_LOW_REGISTER_W	REG: 0x0480	REGISTER TO SET THE CHIP MASK FOR THE LOWER 32 LINKS WHEN USING DIRECT SPI MODE	MP_FEB	WRITE-REG
MP_CTRL_DIRECT_SPI_CHIP_M_HIGH_REGISTER_W	REG: 0x0481	REGISTER TO SET THE CHIP MASK FOR THE HIGHER 32 LINKS WHEN USING DIRECT SPI MODE	MP_FEB	WRITE-REG
MP_CTRL_RESET_REGISTER_W	REG: 0x04A0	WRITE TO THIS REG TRIGGERS A 1 CYCLE MP CTRL RESET	MP_FEB	WRITE-REG
MP_CTRL_RUN_TEST_REGISTER_W	REG: 0x04A1	WRITE TO THIS REG TRIGGERS WRITE OF A TDAC TEST PATTERN	MP_FEB	WRITE-REG

RegName	Reg/Bit/Index	Doc	Board	TYPE
MP_CTRL_N_FREE_PAGES_REGISTER_R	REG: 0x04A2	NUMBER OF FREE TDAC PAGES	MP_FEB	-
MP_CTRL_TESTRAM_RDATA_REGISTER_R	REG: 0x04A3	NUMBER OF FREE TDAC PAGES	MP_FEB	-
MP_CTRL_TESTRAM_WADDR_REGISTER_W	REG: 0x04A4	NUMBER OF FREE TDAC PAGES	MP_FEB	WRITE-REG
MP_CTRL_TESTRAM_RADDR_REGISTER_W	REG: 0x04A5	NUMBER OF FREE TDAC PAGES	MP_FEB	WRITE-REG
MP_CTRL_TESTRAM_WDATA_REGISTER_W	REG: 0x04A6	NUMBER OF FREE TDAC PAGES	MP_FEB	WRITE-REG
MP_CTRL_SLOW_CLK_SHIFT_REGISTER_W	REG: 0x04A7	SHIFT OF THE SLOW CLOCK VS RESET	MP_FEB	WRITE-REG
MP_CTRL_SIN_INVERT_REGISTER_W	REG: 0x04A8	INVERT SIN	MP_FEB	WRITE-REG
MP_CTRL_EXT_CMD_START_REGISTER_W	REG: 0x0800	START OF REGISTERS TO SEND DIRECT COMMANDS TO MUPIX SENSORS, MAINLY FOR READBACK FUNCTIONALITY. 2 ADDRESSES FOR EACH CHIP SINCE A COMMAND HAS 64 BITS, COMMAND TRIGGERS AUTOMATICALLY ONCE WRITTEN HERE	MP_FEB	WRITE-REG
MP_READOUT_MODE_REGISTER_W	REG: 0x1300	TO BE REMOVED	MP_FEB	WRITE-REG
CHIP_ID_MODE_RANGE	RANGE: 5 DOWNTO 4	BITS TO SELECT DIFFERENT CHIP ID NUMBERING MODES (NOT IN USE)	MP_FEB	-
TOT_MODE_RANGE	RANGE: 8 DOWNTO 6	BITS TO SELECT DIFFERENT TOT CALCULATION MODES (DEFAULT IS TO SEND TS2 AS TOT, NOT IN USE)	MP_FEB	-
MP_LVDS_LINK_MASK_REGISTER_W	REG: 0x1301	MASKING OF LVDS CONNECTIONS	FEB	WRITE-REG
MP_LVDS_LINK_MASK2_REGISTER_W	REG: 0x1302	MASKING OF LVDS CONNECTIONS	FEB	WRITE-REG
MP_DATA_GEN_CONTROL_REGISTER_W	REG: 0x1303	CONTROLS THE MUPIX DATA GENERATOR	MP_FEB	WRITE-REG
MP_DATA_GEN_HIT_P_RANGE	RANGE: 3 DOWNTO 0	GENERATOR HIT OUTPUT PROBABILITY, 1/(2^(MP_DATA_GEN_HIT_P_RANGE+1)) FOR EACH CYCLE WHERE A HIT COULD BE SEND	MP_FEB	-
MP_DATA_BYPASS_SELECT_REGISTER_W	REG: 0x1305	BYPASS THE MUPIX SOTER AND PUT INPUT TO_INTEGER(THISREG) DIRECTLY ON OPTICAL LINK (IMPLEMENTED BUT NOT CONNECTED IN TOP)	MP_FEB	WRITE-REG

RegName	Reg/Bit/Index	Doc	Board	TYPE
MP_TS_HISTO_SELECT_REGISTER_W	REG: 0x1306	NOT IN USE	MP_FEB	WRITE-REG
MP_TS_HISTO_LINK_SELECT_RANGE	RANGE: 15 DOWNTO 0	NOT IN USE	MP_FEB	-
MP_TS_HISTO_N_SAMPLE_RANGE	RANGE: 31 DOWNTO 16	NOT IN USE	MP_FEB	-
MP_LAST_SORTER_HIT_REGISTER_R	REG: 0x1307	REGISTER THAT CONTAINS THE LAST MUPIX HIT OF THE SORTER OUTPUT	MP_FEB	-
MP_SORTER_INJECT_REGISTER_W	REG: 0x1308	USED TO INJECT SINGLE HITS AT THE SORTER INPUTS	MP_FEB	WRITE-REG
MP_SORTER_INJECT_SELECT_RANGE	RANGE: 7 DOWNTO 4	INPUT OF THE SORTER TO INJECT TO	MP_FEB	-
MP_CHIP_UNPACKER_CNT_REGISTER_R	REG: 0x1309	COUNTER OUTPUT FROM THE UNPACKER	MP_FEB	-
MP_CHIP_UNPACKER_CNT_REGISTER_W	REG: 0x130A	REGISTER TO SELECT THE CHIP/LINK OF THE UNPACKER COUNTER AND THE COUNTER TO READ FROM	MP_FEB	WRITE-REG
MP_CHIP_UNPACKER_CNT_SELECT_RANGE	RANGE: 7 DOWNTO 0	SELECT THE LVDS LINKT TO READOUT (0-35)	MP_FEB	-
MP_CNT_UNPACKER_SELECT_RANGE	RANGE: 10 DOWNTO 8	SELECT THE COUNTER FROM THIS LINK TO READOUT (0-7)	MP_FEB	-
MP_HIT_ENA_CNT_SORTER_IN_REGISTER_R	REG: 0x130B	HIT ENABLE COUNTER AT THE SORTER INPUT	MP_FEB	-
MP_HIT_ENA_CNT_SORTER_SELECT_REGISTER_W	REG: 0x130C	REGISTER TO SELECT THE LINK FOR THE SORTER INPUT HIN ENA COUNTER	MP_FEB	WRITE-REG
MP_HIT_ENA_CNT_SORTER_OUT_REGISTER_R	REG: 0x130D	HIT COUNTER AT SORTER OUTPUT	MP_FEB	-
MP_RESET_LVDS_N_REGISTER_W	REG: 0x130F	RESET REGISTER FOR MUPIX LVDS RX	MP_FEB	WRITE-REG
MP_USE_ARRIVAL_TIME1_REGISTER_W	REG: 0x1310	USE HIT ARRIVAL TIME INSTEAD OF TIMESTAMP FROM MUPIX (LOWER 32 CHIPS)	MP_FEB	WRITE-REG
MP_USE_ARRIVAL_TIME2_REGISTER_W	REG: 0x1311	USE HIT ARRIVAL TIME INSTEAD OF TIMESTAMP FROM MUPIX (UPPDER 4 CHIPS)	MP_FEB	WRITE-REG

RegName	Reg/Bit/ Index	Doc	Board	TYPE
MP_TRIGGER0_REGISTER_R	REG: 0x1312	TRIGGER0	MP_FEB	-
MP_TRIGGER1_REGISTER_R	REG: 0x1313	TRIGGER1	MP_FEB	-
MP_TRIGGER0_REG_REGISTER_R	REG: 0x1314	PREV-TRIGGER0	MP_FEB	-
MP_TRIGGER1_REG_REGISTER_R	REG: 0x1315	PREV-TRIGGER1	MP_FEB	-
MP_LVDS_INVERT_0_REGISTER_W	REG: 0x;	INVERTING LVDS LINES	FEB	WRITE-REG
MP_LVDS_INVERT_1_REGISTER_W	REG: 0x1317	INVERTING LVDS LINES	FEB	WRITE-REG
MP_IS_A_0_REGISTER_R	REG: 0x1318	CHECK IF LINK IS AN A LINK	FEB	-
MP_IS_A_1_REGISTER_R	REG: 0x1319	CHECK IF LINK IS AN A LINK	FEB	-
MP_IS_B_0_REGISTER_R	REG: 0x1320	CHECK IF LINK IS AN B LINK	FEB	-
MP_IS_B_1_REGISTER_R	REG: 0x1321	CHECK IF LINK IS AN B LINK	FEB	-
MP_IS_C_0_REGISTER_R	REG: 0x1322	CHECK IF LINK IS AN C LINK	FEB	-
MP_IS_C_1_REGISTER_R	REG: 0x1323	CHECK IF LINK IS AN C LINK	FEB	-
MP_READBACK_FIFOS_START_REGISTER_R	REG: 0x2000	FIFOS WHERE SLOWCONTROL FROM THE MUPIX SENSORS CAN BE READ BACK, INCLUDES ALSO THE ERROR SIGNALS, ADC VALUES ARE IN READBACK MEMS	MP_FEB	-
MP_READBACK_MEMS_START_REGISTER_R	REG: 0x3000	MEMORIES WHERE ADC MEASUREMENTS FROM THE MUPIX SENSORS CAN BE READ	MP_FEB	-
MP_HIT_ARRIVAL_START_REGISTER_R	REG: 0x1200	START OF PLL LOCK MONITOR BLOCK, 4 WORDS FOR EACH CHIP, HISTOGRAM LOWER BITS OF MUPIX ARRIVAL TIMESTAMP	MP_FEB	-

RegName	Reg/Bit/ Index	Doc	Board	TYPE
MP_SORTER_COUNTER_REGISTER_R	REG: 0x1000	HIT COUNTERS IN THE SORTER, 40 32 BIT COUNTERS IN TOTAL. FOR THE INNER PIXEL FEBS: 12 COUNTERS WITH IN-TIME HITS PER CHIP, 12 COUNTERS WITH OUT-OF-TIME HITS PER CHIP, 12 COUNTERS WITH OVERFLOWS PER CHIP, A COUNTER WITH THE NUMBER OF OUTPUT HITS AND THE CURRENT CREDIT VALUE. THE LAST TWO COUNTERS ARE CURRENTLY RESERVED FOR FUTURE USE	MP_FEB	-
MP_SORTER_DELAY_REGISTER_W	REG: 0x1028	MINIMUM ROUND-TRIP DELAY FROM SYNC RESET GOING OFF TO HIT WITH TS > 0 APPEARING AT SORTER INPUT IN 8 NS STEPS	MP_FEB	WRITE-REG

RegName	Reg/Bit/Index	Doc	Board	TYPE

RegName	Reg/Bit/ Index	Doc	Board	TYPE
SORTER_COUNTER_REGISTER_R	REG: 0x1000	DIAGNOSTIC COUNTERS IN THE SORTER HIT COUNTERS IN THE SORTER	MP_FEB	-

6. MIDAS

For the data from the FPGA we create MIDAS events using the 32-bit banks 64-bit aligned format: https://daq00.triumf.ca/MidasWiki/index.php/Event_Structure. Each bank contains time sorted 64-bit hits from the MuPix or the MuTRiG. Overall one event can hold up to 100 of these banks with the bank name HT00-HT99. The hit formats have following structure:

6.0.1 MuPix hit format

Bits	Width	Field	C++ accessor equivalent	Description
63	1	indicator	`is_pixel()`	`true` for MuPix
62–58	5	chipid	`chipid()`	Global chip ID
57–50	8	col	`col()`	Column index
49–42	8	row	`row()`	Row index
41–37	5	tot / t2	`tot()` / `t2()`	Time-over-threshold
36–16	21	ts_high	`time()` (part)	Timestamp high
15–11	5	ts_low	`time()` (part)	Timestamp mid
10–4	7	subheader_time	`time()` (part)	Subheader timing
3–0	4	ts_sorterhit	`time()` (part)	Hit timestamp sorter

6.0.2 MuTRiG hit format

Bits	Width	Field	C++ accessor equivalent	Description
63	1	indicator	`is_mutrig()`	`true` for MuTRiG
62–61	2	chipid	`asic()`	ASIC ID
60–56	5	channel	`channel()`	Channel ID
55–47	9	e-t	`tot()` / `eflag()`	Energy or short-hit flag
46–44	3	time_remainder	—	1.6 ns remainder bits
43–39	5	fine_time	—	Fine timestamp
38–16	23	ts_high	`time()` (part)	Timestamp high
15–12	4	ts_low	`time()` (part)	Timestamp mid
11–4	8	subheader_time	`time()` (part)	Subheader timing
3–0	4	ts_sorterhit	`time()` (part)	Hit timestamp sorter

And in structs:

```
struct pixelhit { pixelhit() noexcept : hitdata(0x0) {} pixelhit(uint64_t h) noexcept : hitdata(h) {} uint64_t hitdata;
[[nodiscard]] bool is_pixel() const { return ((hitdata >> 63) & 0x1) == 0; }
[[nodiscard]] uint8_t chipid() const { return (hitdata >> 58) & 0x1F; } [[nodiscard]] uint8_t col() const { return (hitdata >> 50) & 0xFF; }
[[nodiscard]] uint8_t row() const { return (hitdata >> 42) & 0xFF; } [[nodiscard]] uint8_t tot() const { return (hitdata >> 37) & 0x1F; }
[[nodiscard]] uint8_t t2() const { return tot(); } [[nodiscard]] uint32_t ts_high() const { return (hitdata >> 16) & 0x1FFFFF; }
[[nodiscard]] uint8_t ts_low() const { return (hitdata >> 11) & 0x1F; }
[[nodiscard]] uint8_t subheader_time() const { return (hitdata >> 4) & 0x7F; }
[[nodiscard]] uint8_t ts_sorterhit() const { return hitdata & 0xF; } [[nodiscard]] uint64_t time() const { return hitdata & 0x1FFFFFFFFFULL; }
void Print() const { std::printf( "x64:%016llx chipid:%02x col:%u row:%u tot:%u time:%010llx\n", (unsigned long long)hitdata, chipid(), col(),
row(), tot(), (unsigned long long)time() ); } }; struct mutrighit { mutrighit() noexcept : hitdata(0x0) {}
mutrighit(uint64_t h) noexcept : hitdata(h) {} uint64_t hitdata; [[nodiscard]] bool is_mutrig() const { return ((hitdata >> 63) & 0x1) == 1; }
[[nodiscard]] uint8_t chipid() const { return (hitdata >> 61) & 0x3; } [[nodiscard]] uint8_t asic() const { return chipid(); }
[[nodiscard]] uint8_t channel() const { return (hitdata >> 56) & 0x1F; } [[nodiscard]] uint16_t et() const { return (hitdata >> 47) & 0x1FF; }
[[nodiscard]] bool eflag() const { return et() == 0x1FF; } [[nodiscard]] uint16_t tot() const { return et(); }
[[nodiscard]] uint8_t time_remainder() const { return (hitdata >> 44) & 0x7; }
[[nodiscard]] uint8_t fine_time() const { return (hitdata >> 39) & 0x1F; }
[[nodiscard]] uint32_t ts_high() const { return (hitdata >> 16) & 0x7FFFFF; }
[[nodiscard]] uint8_t ts_low() const { return (hitdata >> 12) & 0xF; }
[[nodiscard]] uint8_t subheader_time() const { return (hitdata >> 4) & 0xFF; }
[[nodiscard]] uint8_t ts_sorterhit() const { return hitdata & 0xF; } [[nodiscard]] uint64_t time() const { return hitdata & 0x7FFFFFFFFFULL; }
void Print() const { std::printf( "x64:%016llx chipid:%u channel:%u et:%u rem:%u fine:%u time:%010llx\n", (unsigned long long)hitdata, chipid(),
channel(), et(), time_remainder(), fine_time(), (unsigned long long)time() ); } }; struct hit { hit() noexcept : hitdata(0x0) {}
explicit hit(uint64_t h) noexcept : hitdata(h) {} uint64_t hitdata; // Common discriminator
[[nodiscard]] bool is_pixel() const { return ((hitdata >> 63) & 0x1) == 0; }
[[nodiscard]] bool is_mutrig() const { return ((hitdata >> 63) & 0x1) == 1; } // Convert to typed views
[[nodiscard]] pixelhit as_pixel() const { return pixelhit(hitdata); }
[[nodiscard]] mutrighit as_mutrig() const { return mutrighit(hitdata); } // helper [[nodiscard]] uint64_t raw() const { return hitdata; }
void Print() const { if(is_pixel()) { as_pixel().Print(); } else { as_mutrig().Print(); } } };
```

To extract the banks from the MIDAS event the following analyze function can be used:

```
TAFlowEvent* AnaFillHits::Analyze(TARunInfo* runinfo, TMEvent* event, TAFlags* flags, TAFlowEvent* flow) { // If this module is disabled, don't do anythin
if(!enabled_) { *flags |= TAFlag_SKIP_PROFILE; return flow; } // Only process readout events
if(event->event_id != 301) return flow; // ---------------------------------------- // Build event header // ----------------------------------------
eventheader h; h.event_id = event->event_id; h.trigger_mask = event->trigger_mask; h.serial_number = event->serial_number;
h.midas_time_stamp = event->time_stamp; h.data_size = event->data_size; const uint32_t sr_num = h.serial_number;
SrNo->Fill(static_cast<double>(sr_num)); // ---------------------------------------- // Scan MIDAS banks // ----------------------------------------
event->FindAllBanks(); for(const auto& bank : event->banks) { const std::string firstTwoChars = bank.name.substr(0, 2);
const char* rawData = event->GetBankData(&bank); // ---------------------------------------- // HTxx bank: mixed hit bank // ----------
if(firstTwoChars == "HT") { const hit* dataStart = reinterpret_cast<const hit*>(rawData);
const hit* dataEnd = reinterpret_cast<const hit*>(rawData + bank.data_size); hits_.reserve(hits_.size() + (dataEnd - dataStart));
for(const hit* current = dataStart; current != dataEnd; ++current) { // Optional validation / monitoring can be added here // Example: // if(current->is_pi
hits_.emplace_back(*current); } } } // ---------------------------------------- // Optional monitoring of mixed hits // ----------
for(const auto& hhit : hits_) { if(hhit.is_pixel()) { const auto p = hhit.as_pixel(); timestamp->Fill(static_cast<double>(p.time()));
SrNo_ts->Fill(static_cast<double>(sr_num), static_cast<double>(p.time())); } else { const auto m = hhit.as_mutrig();
SrNo_ts_mutrig->Fill(static_cast<double>(sr_num), static_cast<double>(m.time())); } } // ---------------------------------------- // Package hits into flow
flow = new HitVectorFlowEvent(flow, h, std::move(hits_)); // After std::move, reset to default state hits_ = std::vector<hit>(); return flow; }
```

7. Configuration

7.1 pixel tuning and masking

```
export MIDAS_PATH=/PathToMidas export MIDAS_PYTHON_PATH=$MIDAS_PATH/python export MUSIP_PATH=/PathToMusip export SCRIPT_PATH=$MUSIP_PATH/scripts PYTHONPATH
```

Example:

```
mu3e@localhost:~/musip $ export / mu3e@localhost:~/musip $ export MIDAS_PATH=/home/mu3e/midas mu3e@localhost:~/musip $ export MIDAS_PYTHON_PATH=$MIDAS_PATH
```

7.2 tdac writing

Tdac writing for a chip would work like this: write 32 bit words, 4* 8bit tdac in each word order: col 0 -> 255, starting with col 0, physical col addr. row 0 -> row 255 for each col, starting with row0, physical row addr.

```
example: start with col 0 .. (32 bit) : [8 bit tdac, 8 bit tdac, 8 bit tdac, 8 bit tdac] word 0 : [row 3 , row 2 , row 1 , row 0 ] word 1 : [row 7 , row 6
```