QtHardMon User Guide

v00.16.01

08.03.2016

Contents

Chapter 1

Using QtHardMon

QtHardMon lets you to read in and modify register values of individual boards / cards on your crate. The QtHardMon GUI can list out all registers available for a card along with the current content in these registers. It lets you write custom values to these registers as well (where applicable), thus letting you customize functionality in your cards.

1.1 Specifying The Crate Through The .dmap File

Before the software can be used, the user has to create a .dmap file. The contents of this file lets the software know about the current set of cards plugged in to the crate. Once a correctly written .dmap file is loaded, the software can list out the cards and their internal registers.

The .dmap file is essentially made up of lines with the following syntax:

```
<device_alias>  <sdm_identifier> <path_to_map_file>
```

Each such line in the .dmap file describes a device/card on a crate. <device_alias> is an alternative name the user specifies for a device. The GUI will list the device using this alias name. <sdm_identifier> is the standard device model identifier. This describes the attributes of the device. These attributes can include information on the connection interface, connection address and protocols supported. <path_to_map_file> is the register mapping file for the device. This mapping file is generated by the firmware and contains the list of registers along with their properties such as register size.

An example .dmap file could be as below:

```
# filename:example_crate.dmap
PCIE0   sdm://./pci:mtcadummys0;         pcie_device.map
REBOT0  sdm://./rebot=192.168.10.2,5001  rebot_device.map
```

The first line beginning with the # indicates a comment.

The `example_crates.dmap` file indicates that the user wants to configure a crate with 2 devices on it. `sdm://./pci:mtcadummys0` and `sdm://./rebot=192.168.10.2,5001` are the sdm identifiers for these devices. `PCIE0` is a pcie device with the device file identifier `/dev/mtcadummys0`. `REBOT0` is a tcp ip enabled device reachable at ip 192.168.10.2, port 5001 and uses rebot as the communication protocol. In our example the alias names for cards are `PCIE0` and `REBOT0`. Once `example_crates.dmap` is loaded, the cards will be referred to as `PCIE0` and `REBOT0` in the GUI. pcie_device.map and rebot_device.map describe the addressable registers in devices `PCIE0` and `REBOT0` respectively.

1.2 Loading The Crate Configuration /.dmap File

Once the cards on your crate have been specified through the .dmap file, the next step is loading this configuration on to the GUI. This is done using the "Load Boards" button on the GUI or using the menu option under *File > Load Boards*: (Fig 1.1)

The boards may also be loaded by invoking QtHardMon GUI from the command prompt:

```
$ QtHardMon ./example_crates.dmap &
```

1.3 The QtHardMon Interface

Information on the individual devices become available in the GUI once the .dmap is loaded (Fig 1.2). The GUI is split into sections that display

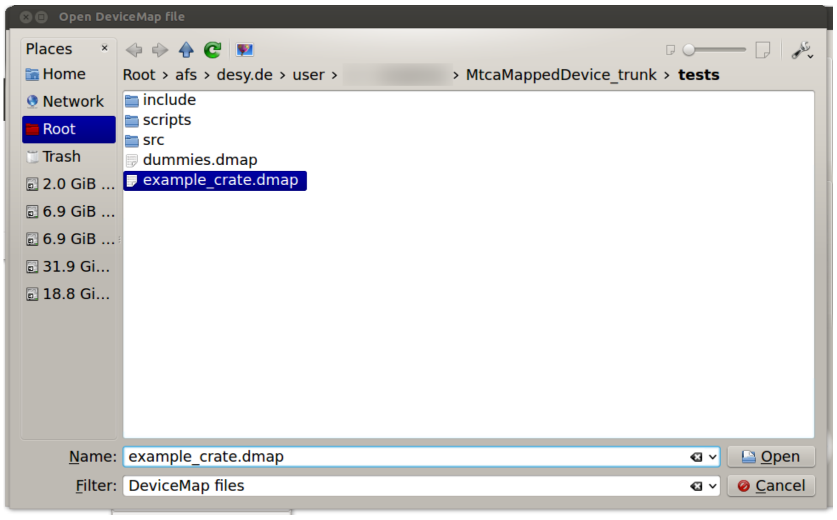
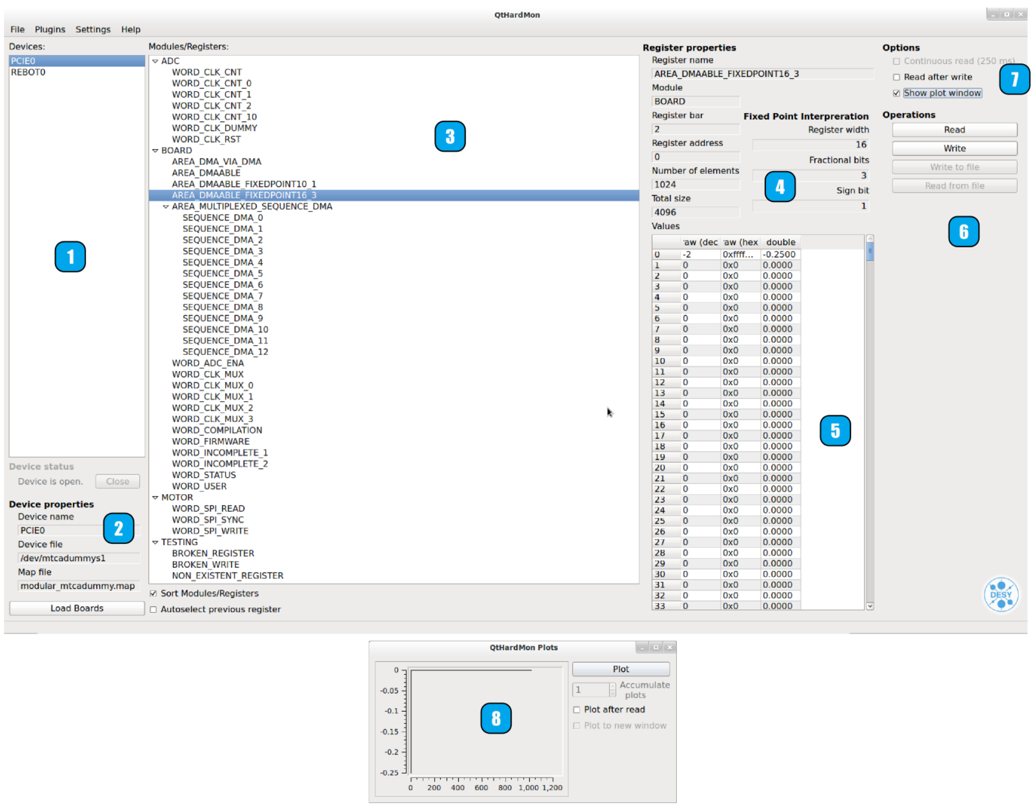


Figure 1.1: Loading the .dmap file



Devices:

PCIE0
REBOT0

Device status

Device is open. Close

Device properties

Device name: PCIE0

Device file: /dev/mtcadummys1

Map file: modular_mtcadummy.map

Load Boards

Modules/Registers:

```
▿ ADC
    WORD_CLK_CNT
    WORD_CLK_CNT_0
    WORD_CLK_CNT_1
    WORD_CLK_CNT_2
    WORD_CLK_CNT_10
    WORD_CLK_DUMMY
    WORD_CLK_RST
▿ BOARD
    AREA_DMA_VIA_DMA
    AREA_DMAABLE
    AREA_DMAABLE_FIXEDPOINT10_1
    AREA_DMAABLE_FIXEDPOINT16_3
  ▿ AREA_MULTIPLEXED_SEQUENCE_DMA
      SEQUENCE_DMA_0
      SEQUENCE_DMA_1
      SEQUENCE_DMA_2
      SEQUENCE_DMA_3
      SEQUENCE_DMA_4
      SEQUENCE_DMA_5
      SEQUENCE_DMA_6
      SEQUENCE_DMA_7
      SEQUENCE_DMA_8
      SEQUENCE_DMA_9
      SEQUENCE_DMA_10
      SEQUENCE_DMA_11
      SEQUENCE_DMA_12
    WORD_ADC_ENA
    WORD_CLK_MUX
    WORD_CLK_MUX_0
    WORD_CLK_MUX_1
    WORD_CLK_MUX_2
    WORD_CLK_MUX_3
    WORD_COMPILATION
    WORD_FIRMWARE
    WORD_INCOMPLETE_1
    WORD_INCOMPLETE_2
    WORD_STATUS
    WORD_USER
▿ MOTOR
    WORD_SPI_READ
    WORD_SPI_SYNC
    WORD_SPI_WRITE
▿ TESTING
    BROKEN_REGISTER
    BROKEN_WRITE
    NON_EXISTENT_REGISTER
```

☑ Sort Modules/Registers
☐ Autoselect previous register

Register properties

Register name: AREA_DMAABLE_FIXEDPOINT16_3

Module: BOARD

Register bar: 2

Register address: 0

Number of elements: 1024

Total size: 4096

Fixed Point Interpreration

Register width: 16

Fractional bits: 3

Sign bit: 1

Values

	raw (dec	raw (hex	double
0	-2	0xffff...	-0.2500
1	0	0x0	0.0000
2	0	0x0	0.0000
3	0	0x0	0.0000
4	0	0x0	0.0000
5	0	0x0	0.0000
6	0	0x0	0.0000
7	0	0x0	0.0000
8	0	0x0	0.0000
9	0	0x0	0.0000
10	0	0x0	0.0000
11	0	0x0	0.0000
12	0	0x0	0.0000
13	0	0x0	0.0000
14	0	0x0	0.0000
15	0	0x0	0.0000
16	0	0x0	0.0000
17	0	0x0	0.0000
18	0	0x0	0.0000
19	0	0x0	0.0000
20	0	0x0	0.0000
21	0	0x0	0.0000
22	0	0x0	0.0000
23	0	0x0	0.0000
24	0	0x0	0.0000
25	0	0x0	0.0000
26	0	0x0	0.0000
27	0	0x0	0.0000
28	0	0x0	0.0000
29	0	0x0	0.0000
30	0	0x0	0.0000
31	0	0x0	0.0000
32	0	0x0	0.0000
33	0	0x0	0.0000

Options

☐ Continuous read (250 ms)
☐ Read after write
☑ Show plot window

Operations

Read
Write
Write to file
Read from file

Figure 1.2: The QtHardMon Interface

information on the devices and these are numbered as in Fig 1.2. The numbered components of the QtHardMon Interface are as below:

1. **Devices List:** This section displays the list of cards that were specified in the .dmap file. The user can click on the listed alias names to select individual cards and then modify/display properties of that particular card.

2. **Device Status and Properties:** Gives an option to open or close the card selected in the `Devices List`. The information on the device identifier and register mapping file of this card is also displayed here. Basically all the information on the card from the .dmap file is shown here.

3. **Register List:** This part of the GUI displays all registers of the card that are accessible to the user. The GUI can be configured to perform a read on a register when its name is clicked on in the `Register List`. Successive clicks on the same register name leads to successive reads from the card, each read corresponding to each click. This behaviour is active, if the '`Read on click in register list`' checkbox is selected in the preferences menu (Section 1.4.3). Selecting the register name using the arrow keys on the keyboard can also trigger a read (in case '`Automatically read when selecting a register`' is selected in the preferences).

   Enabling the `Auto select previous register` check box lets the GUI automatically reselect the previously selected register when a device is reopened (For example when browsing between the devices on the Device List of the GUI). The `Sort Modules/Registers` check box can be used to enable or disable sorting the entries displayed on the Register list.

   The Register List can display registers grouped under modules. It can also display content of individual sequences/channels from a `AREA_MULTIPLEXED_SEQUENCE_<>`. This is a feature introduced by the `deviceacess` library. Please see Appendix A for a description of `AREA_MULTIPLEXED_SEQUENCE_<>` type entries.

4. **Register Properties:** The properties of the currently selected register is shown here. These include the register name, the PCI express **Base Address Range** (Register **bar**), the register address, the number of 32 bit words in the register and the size of the register in bytes.

5. **Register Values:** This displays the current value in the register. Both

decimal and hexadecimal representation of the raw value are displayed here. The double field displays the fixed point converted version of this raw value.

The Register Values section can also be used to input custom values (into writeable registers). Values may be entered as its raw decimal representation, raw hex representation or its fixed point converted decimal representation. This can be done by setting the intended value in the decimal, hex or double field respectively and pressing the write button in the `Register Operations` section of the GUI.

Note: The number of decimal places to be displayed for the double field is customizable in the preferences menu

6. **Register Operations:** This part has two working buttons, the first is the `Read` and the other is the `Write` button. The default behaviour of the GUI is to read in the register values from the card when the register name is selected or clicked on. The read button is useful when the implicit read functionality has been disabled in the preferences menu(1.4.3). In this situation the user has to explicitly ask the GUI to perform a read from the card by pressing the `Read` button.

   The `Write` button is needed because writing to a register using the GUI, is a two step process. The first step is selecting the register on the card and setting the desired value in the `Register Values` field and the second part is actually triggering the write to the card by pressing the `Write` button. The value is written only if the `Write` button is pressed, else it gets discarded when the user moves on to select another register/card or presses the `Read` button.

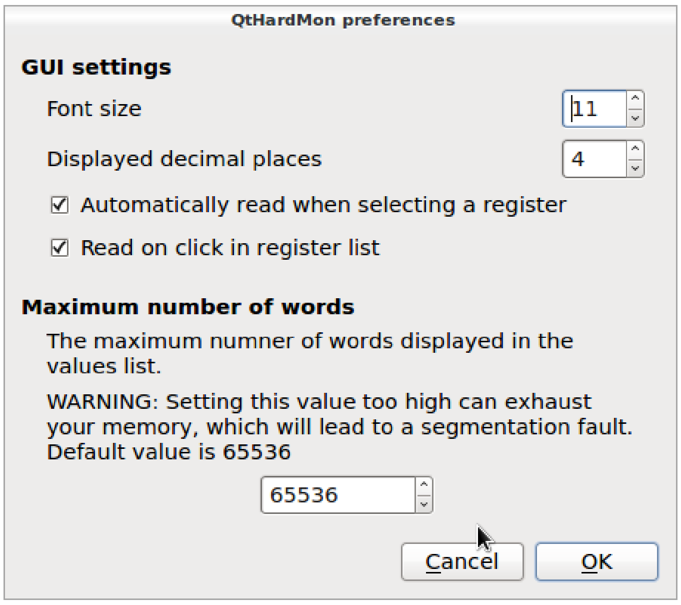
7. **GUI options:** If the check-box titled `‘Read after Write’ is selected`, the GUI actually performs a read from the register and refreshes the display with the new value once the `Write` button is clicked. The second check-box named `‘Show plot window’` opens a plot display window when checked. The function of this plot window is explained below.

8. **Plot Window:** The plot window can be used to draw graphs, using the data contained in the registers. The x-axis of the plot is the number of data points/32 bit elements that has been read in from the register and the y-axis is the value of those elements. It is also possible to customize the number of data points on the x-axis by limiting the number of read in values from the register (See section 1.4.4).

The GUI by default generates a plot (for the selected register) only after the `Plot` button is clicked. This default behaviour can be modified by setting the '`Plot after read`' check-box in the `Plot Window`. Selecting this option triggers plotting of the register values, once the register name is clicked on or selected using the arrow keys. (Note: This behaviour works as described only when '`Automatically read when selecting a register`' and '`Read on click in register list`' options are selected in the preferences menu 1.4.3).

1.4 The Preferences Menu

The QtHardMon preferences menu can be accessed under *Settings* > *Preferences* of the main menu. (Fig 1.3)


QtHardMon preferences

GUI settings

Font size: 11

Displayed decimal places: 4

☑ Automatically read when selecting a register

☑ Read on click in register list

Maximum number of words

The maximum numner of words displayed in the values list.

WARNING: Setting this value too high can exhaust your memory, which will lead to a segmentation fault. Default value is 65536

65536

Figure 1.3: The QtHardMon Preferences

1.4.1 Changing The Font Size

The font size of the GUI can be specified here in the preferences menu, by setting required font size in the provided field.

1.4.2 Double field precision in Register Values

The “Displayed decimal places” spinbox can be used change the number of decimal places displayed for the double representation of the register value. The GUI can display a double value with a maximum precision of 10 decimal places.

1.4.3 Implicit read of Register Values

The two check-boxes - `‘Automatically read when selecting a register’` and `‘Read on click in register list’` are enabled by default. These options trigger an implicit read of the selected register, as soon as the register name is selected from the Register list using the arrow keys on the keyboard or when the name is clicked on (respectively). These check boxes lets the user disable implicit read when selected or clicked. This is useful in case of large registers, where implicitly reading in these slows down the work flow. The `Read` button in the “Register Operations” section of the GUI can be used for reading in values, when both check boxes remain unchecked.

1.4.4 Limiting The Number Of Read In Values

There can be problems when registers have a large number of elements/32-bit words and when all these words are read in one go. The `‘Maximum number of words’` lets the user limit the number of elements read in from the register. The software reads in only the first ‘N’ elements specified in the `‘Maximum number of words’` field, though the register may contain even more elements. The default value of this field is set as 65536, which means that the maximum number of elements that will be read in from any register is 65536. Setting a higher value here and thus reading in larger amounts of data from a register can potentially exhaust the system memory, leading to a crash.

1.5 Saving and Restoring Preferences

As seen, the preferences menu (Section 1.4) lets the user customize font size and behaviour of the GUI. The software lets the user save these customizations as a configuration file (.cfg), which can later be loaded back into a fresh session to restore the user defined settings if needed.

The option for saving and loading the GUI configuration file can be found under *File > Save config / File > Save config as* and *File > Load config* menu items respectively. The configuration file stores/retrieves the following information:

- The loaded .dmap file.
- The selected card/device.
- The selected registers on all cards.
- The user selected settings in the prefrences menu
- The user selected settings in the `GUI Options`
- The user selected settings in the `Plot Window`

Appendices

Appendix A

Multiplexed Sequences

Multiplexed Sequences are currently used to model data access from multi channel ADC's. Samples over all n ADC channels, taken at the same point in time T_0 are placed consecutively in memory. The next set of samples taken at T_1 for the n ADC channels follow the previous data in memory and so on (see Figure A.1). Such an area in memory which contain data in this described order is represented by the `AREA_MULTIPLEXED_SEQUENCE_<area_name>` keyword. Each ADC channel is described by the `SEQUENCE_<area_name>_<sequence_number>` keyword.

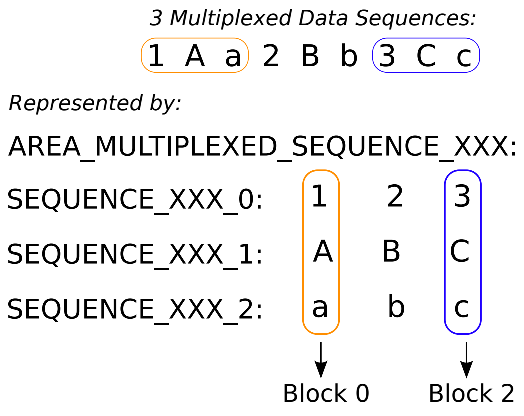


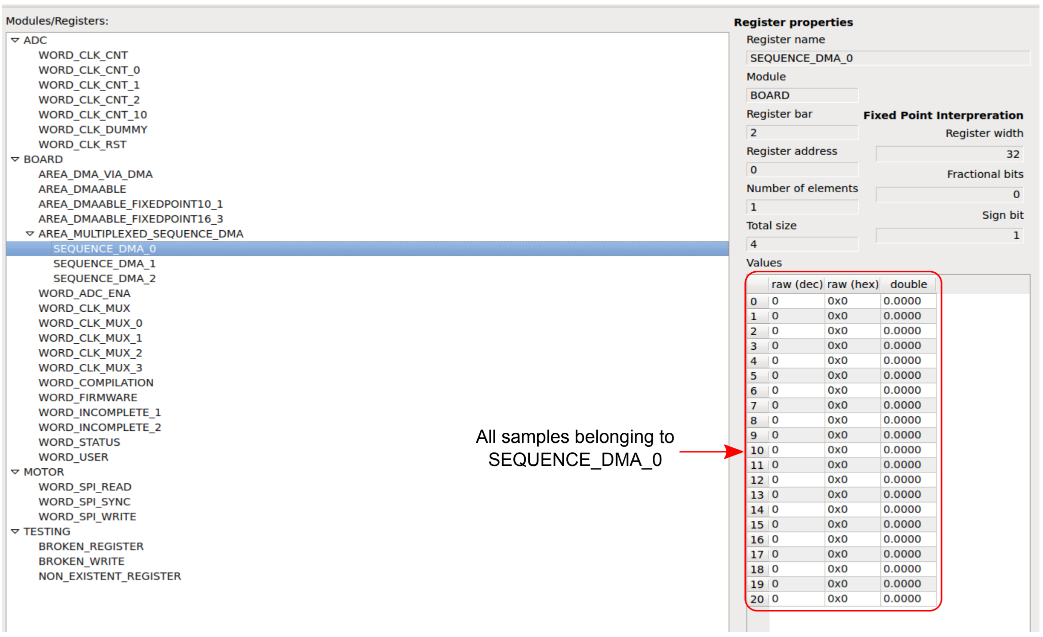
Figure A.1: The Multiplexed Data Region Representation

Such Multiplexed data regions are described in the map files as below :

```
BOARD.AREA_MULTIPLEXED_SEQUENCE_DMA 0x0 0x0 0x100  0x2
BOARD.SEQUENCE_DMA_0 0x1 0x00 0x4 0x2 32 0 1
BOARD.SEQUENCE_DMA_1 0x1 0x04 0x4 0x2 16 0 0
BOARD.SEQUENCE_DMA_2 0x1 0x06 0x4 0x2 16 0 0
```

The `BOARD` prefix denotes that, the described resource belongs to the module ‘BOARD’.The `AREA_MULTIPLEXED_SEQUENCE_DMA` keyword describes the span of the multiplexed region to be `0x100` bytes. The subsequent 3 lines describes the data on the 3 channels that have been multiplexed to this region. `SEQUENCE_DMA_0` describes that each sample of this channel takes up `0x4` bytes, and has a Fixed point representation spanning 32 bits with 0 fractional and one sign bit. `SEQUENCE_DMA_1` describes that a sampled data on the second channel takes up `0x4` bytes and has a Fixed point representation spanning 16 bits with 0 fractinal bits and no sign bit. The interpretation is similar for the 3^{rd} channel.

Figure A.2 shows `BOARD.AREA_MULTIPLEXED_SEQUENCE_DMA` in QtHardMon. Clicking on `SEQUENCE_DMA_X` demultiplexes the data from the `AREA_MULTIPLEXED_SEQUENCE_DMA` region and displays all elements belonging to the selected Sequence/(ADC channel) in the Register Values table. These values may be modified and written back to the `BOARD.AREA_MULTIPLEXED_SEQUENCE_DMA` region. QtHardmon ensures that the data would be multiplexed and put into the correct position within the area on clicking the write button.



Modules/Registers:

- ADC
  - WORD_CLK_CNT
  - WORD_CLK_CNT_0
  - WORD_CLK_CNT_1
  - WORD_CLK_CNT_2
  - WORD_CLK_CNT_10
  - WORD_CLK_DUMMY
  - WORD_CLK_RST
- BOARD
  - AREA_DMA_VIA_DMA
  - AREA_DMAABLE
  - AREA_DMAABLE_FIXEDPOINT10_1
  - AREA_DMAABLE_FIXEDPOINT16_3
  - AREA_MULTIPLEXED_SEQUENCE_DMA
    - SEQUENCE_DMA_0
    - SEQUENCE_DMA_1
    - SEQUENCE_DMA_2
  - WORD_ADC_ENA
  - WORD_CLK_MUX
  - WORD_CLK_MUX_0
  - WORD_CLK_MUX_1
  - WORD_CLK_MUX_2
  - WORD_CLK_MUX_3
  - WORD_COMPILATION
  - WORD_FIRMWARE
  - WORD_INCOMPLETE_1
  - WORD_INCOMPLETE_2
  - WORD_STATUS
  - WORD_USER
- MOTOR
  - WORD_SPI_READ
  - WORD_SPI_SYNC
  - WORD_SPI_WRITE
- TESTING
  - BROKEN_REGISTER
  - BROKEN_WRITE
  - NON_EXISTENT_REGISTER

Register properties

Register name: SEQUENCE_DMA_0
Module: BOARD
Register bar: 2
Register address: 0
Number of elements: 1
Total size: 4

Fixed Point Interpreration

Register width: 32
Fractional bits: 0
Sign bit: 1

Values

	raw (dec)	raw (hex)	double
0	0	0x0	0.0000
1	0	0x0	0.0000
2	0	0x0	0.0000
3	0	0x0	0.0000
4	0	0x0	0.0000
5	0	0x0	0.0000
6	0	0x0	0.0000
7	0	0x0	0.0000
8	0	0x0	0.0000
9	0	0x0	0.0000
10	0	0x0	0.0000
11	0	0x0	0.0000
12	0	0x0	0.0000
13	0	0x0	0.0000
14	0	0x0	0.0000
15	0	0x0	0.0000
16	0	0x0	0.0000
17	0	0x0	0.0000
18	0	0x0	0.0000
19	0	0x0	0.0000
20	0	0x0	0.0000

Figure A.2: Multiplexed Region in QtHardMon