

QtHardMon User Guide

August 20, 2014

Contents

1	Using QtHardMon	2
1.1	Specifying The Crate Through The .dmap File	2
1.2	Loading The Crate Configuration/.dmap File	3
1.3	The QtHardMon Interface	3
1.4	The Preferences Menu	6
1.4.1	Changing The Font Size	6
1.4.2	Implicit read of Register Values	6
1.4.3	Limiting The Number Of Read In Values	7
1.5	Saving and Restoring Preferences	7

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:

```
<card_alias> <card_device_file_identifier> <path_to_map_file>
```

Each such line in the .dmap file describes a card of the crate. <card_alias> is an alternative name the user specifies for the card. The GUI will list the card using this alias name. <card_device_file_identifier> is the device file identifier of the plugged in card and <path_to_map_file> is the register mapping file for the card. 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
card1      /dev/mtcadummys0 ./mtca_card_registers.map
card2      /dev/llrfdummys4 ./llrf_card_registers.map
```

The first line beginning with the # indicates a comment.

Line 2 and 3 of the *example_crates.dmap* file indicates that the user wants to configure a crate that has two cards plugged in- */dev/mtcadummys0* and */dev/llrfdummys4* (*/dev/mtcadummys0* and */dev/llrfdummys4* are device file identifiers for these cards). In our example the alias names for cards represented by */dev/mtcadummys0* and */dev/llrfdummys4* are *card1* and *card2* respectively. Once *example_crates.dmap* is loaded, the cards will be referred to as *card1* and *card2* in the GUI. *mtca_card_registers.map* and *llrf_card_registers.map* are the register mapping files for */dev/mtcadummys0* and */dev/llrfdummys4* 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)

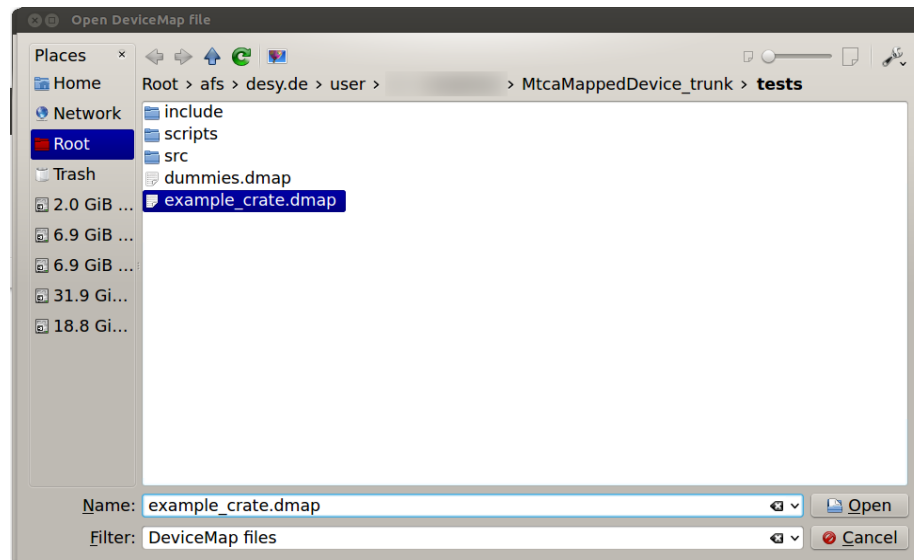


Figure 1.1: Loading the .dmap file

1.3 The QtHardMon Interface

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

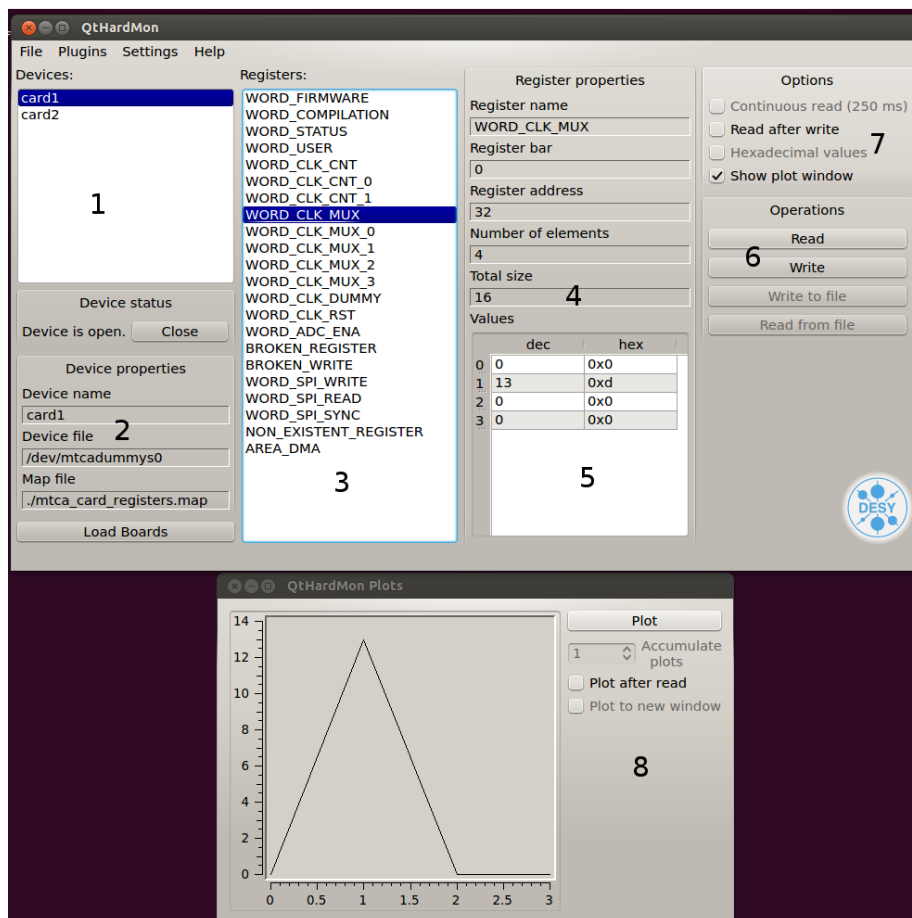


Figure 1.2: The QtHardMon Interface

the cards 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 cards, each read corresponding to each click. This behaviour is active, if the 'Read on click in register list' check-box is selected in the preferences menu (Section 1.4.2). 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).
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 representations of the value are displayed here. This field is also used to write custom values into the writeable registers of the card. The write to a register is done by setting the intended value in the decimal field and then pressing the write button in the `Register Operations` section of the GUI.
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(1.4.2). 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 1.4.3).

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 1.4.2).

1.4 The Preferences Menu

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

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 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 can be used for reading in values, when both these options are turned off

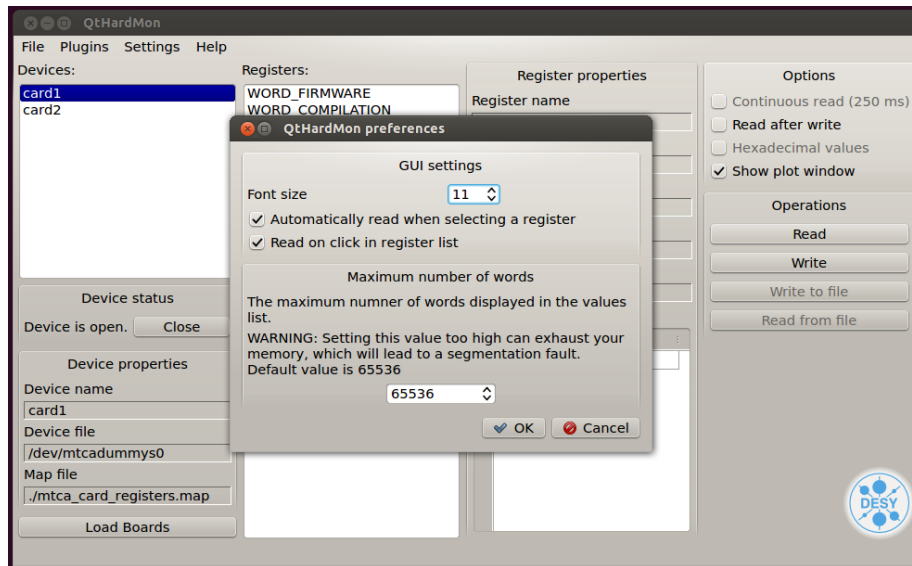


Figure 1.3: The QtHardMon Preferences

1.4.3 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 preferences menu
- The user selected settings in the GUI Options
- The user selected settings in the Plot Window