DMEC driver, EAPI, EAPI-Utility installation

Reyhaneh Yazdani, Zahari Doychev

2019-09-04T10:34:33+02:00(5e4274a)

Contents

1 Introduction

Data Modul's COM Express module comes with an embedded controller, providing embedded features, such as GPIOs, Watchdog, I2C,. . .

DMEC is a set of Linux drivers for the Embedded Controller of DMO modules.

EAPI is a library that specifies functions for user applications in order to communicate with the embedded controller. In other words, EAPI provides the possibility to avoid any software modification when changing COM Express Module.

EAPI-Utility is a GUI utility which can be used to monitor the system and to test EAPI.

This document will describe how DMEC driver, EAPI library and EAPI-Utility can be compiled and installed under Linux.

2 DMEC Driver

The dmec driver has support for the following DMO embedded controller(EC) features:

- i2c
- gpio
- watchdog
- 2 x UART
- rtm
- acpi
- 2 x PWM channel

The features have to be enabled in BIOS. The i2c, uarts and gpio need an irq number assigned in order to function properly. As an example:

- I2C: IRQ 5
- GPIO: IRQ 7
- UART1: IRQ 10
- UART2: IRQ 15

The uarts are supported by the standard linux serial driver 8250/16550.

For PWM0 and PWM1, GPIO4(COMe GPO0) and GPIO5(COMe GPO1) should be set respectively.

2.1 I2C features

The missing i2c driver feature is:

- 10-bit addresses

2.2 GPIO features

The gpio driver provides digital inputs and outputs. The inputs can be configured to generate an interrupt on different events. This can be done either using sysfs or the new gpio api character device. For more information check [1][2].

2.3 Watchdog features

The watchdog can operate in two modes standard and window. The driver should be loaded with the **win_mode** parameter set to one when window mode is used. When the watchdog driver is loaded and the watchdog is active, the watchdog is being automatically pinged. Please note that this does not work with kernel versions below 4.6. The driver does not use the locking feature of the watchdog.

NOTE: some features may not be supported. It depends on the board EC.

2.4 Driver compilation

2.4.1 Prerequisites

1. GNU/GCC compiler and related tools such as make are needed to compile the DMEC driver. For this purpose, it is required to install the following package.

```
$ sudo apt-get update && apt-get upgrade
$ sudo apt-get install build-essential
```

The installation can be verified by the following syntax:

```
$ gcc -v
$ make -v
```

NOTE: The used commands are for Debian and Ubuntu Linux distribution. For other distribution, please use the related command.

2. The linux kernel headers have to be installed in order to build the kernel module. Most of the distribution provide packages for the kernel headers.

```
$ sudo apt-get update
$ apt-cache search linux-headers-$(uname -r)
$ sudo apt-get install linux-headers-$(uname -r)
```

3. The kernel build directory can also be set using the environment variable **KERNEL_SRC**

```
$ export KERNEL_SRC=/usr/src/linux-headers-$(uname -r)
```

2.4.2 Check out

The source of DMEC driver is placed on [9]. It contains three branches for DMEC driver:

1. backport-for-v4.4 => DMEC driver for kernel 4.4
2. backport-for-v4.6 => DMEC driver for kernel 4.6
3. backport-for-v4.14 => DMEC driver for kernel 4.14
4. master => DMEC driver for kernel 4.15 and above

2.4.3 Compilation

All steps below have to be executed in the directory where the driver source code was downloaded or checked out.

```
$ make
$ make install
$ depmod -a
```

2.4.4 Loading

The driver should be auto-loaded on boot. In case the i2c device nodes are missing, *i2c-dev* and *i2c-mux* are also should be loaded.

```
$ modprobe i2c-dev
$ modprove i2c-mux
```

To check whether the modules are loaded or not, use the below command:

```
$ lsmod |grep dmec
```

The output would be list of all dmec drivers: dmec, gpio_dmec, i2c_dmec, wdt_dmec, rtm_dmec, acpi_dmec and pwm_dmec.

NOTE: some features may not be supported. It depends on the board EC.

3 EAPI

EAPI provides common API for different COM Express modules in order to unify the software control. It is developed based on the PICMG EAPI, Revision 1.0, which describes a proposal for the below areas:

- System information
  - EAPI can count the boot times and running hours of the device. Moreover, presents harware monitoring such as temperature and voltages.

- Watchdog timer
- I2C bus
- Flat Panel Brightness control
- User storage area
  - It is a part of EEPROM for storing important information by user.
- GPIO

For use EAPI during developing software, developer needs to consider function 's prototypes defiend in *PICMG*.

To control PWM, EApi Extensions is developed, which consists of functions to control PWM feature. These functions are not presented in PICMG@EAPI Specification 1.0.

3.1 EAPI compilation

3.1.1 Prerequisites

1. GNU/GCC compiler and related tools such as make are needed to compile EAPI. For this purpose, it is required to install the following package.

```
$ sudo apt-get update && apt-get upgrade
$ sudo apt-get install build-essential
```

The installation can be verified by the following syntax:

```
$ gcc -v
$ make -v
```

2. Linux-libc-dev package is needed in kernel version 4.8 and above. In Debian and Ubuntu distribution, use the below command.

```
$ sudo apt-get update && apt-get upgrade
$ sudo apt-get install linux-libc-dev
```

For other distribution, it is required to export the kernel's header files. To cover this part, please follow [13]

NOTE: please notice to use the updated version of the package

3. EAPI library will be installed by default in **/usr/local** path. It can also be changed by using the environment variable **PREFIX**.

```
$ export PREFIX=/usr/local
```

3.1.2 Check out

The source of EAPI is placed on [10]. It contains two branches for:

1. eapi-til-v4.7 => EAPI library until kernel 4.7
2. master => EAPI library from kernel 4.8 and above

3.1.3 Compilation

All steps below have to be executed in the **make** directory where API source code was downloaded or checked out.

```
$ make
$ make install
```

3.2 Log file

The API provides the possibility to log the status of the requested operations. This is intended for the user to analyze the reason of the unsuccessful operation.

In order to activate the log file, it is required to define the **LOGPATH** as following:

```
$ export LOGPATH=/logpath
```

*NOTE: If **LOGPATH** is not specified, no log file will be provided.*

4 EAPI Utility

EAPI Utility is a GUI utility which runs under Linux. It provides the possibility to use and test the related API. Functions supported in this utility are:

- Board Information
- Realtime Information
- I2C
- Watchdog
- GPIO
- Storage Area

4.1 EAPI-Utility compilation

4.1.1 Prerequisites

1. G++ compiler is needed to compile EAPI-Utility. For this purpose, it is required to install the following package.

```
$ sudo apt-get update && apt-get upgrade
$ sudo apt-get install build-essential
```

The installation can be verified by the following syntax:

```
$ g++ -v
```

2. Qmake 3.0 and Qt 5.6 or above are needed to compile EAPI-Utility. For this purpose, it is required to install Qt for Linux.

Download Qt online installer(qt5-qmake) via [11]. Then run the following commands in a path that the installer is already downloaded:

```
$ chmod u+x qt-unified-linux-x64-online.run
$ qt-unified-linux-x64-online.run
```

NOTE: the name of the installer package may differ.

After Qt installation, the below variables should be specified in .bashrc file

```
$ export QTDIR=/Qt_installed_path/5.7/gcc_64
$ export PATH=$QTDIR/bin:$PATH
$ export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
```

The installation can be verified by the following syntax:

```
$ qmake -v
```

3. Package **libgl1-mesa-dev** should be installed.

```
$ apt-get install libgl1-mesa-dev
```

4. EAPI-Utility will look for the EAPI library by default in **/usr/local/lib** path. It can also be changed by using the environment variable **PREFIX**.

```
$ export PREFIX=/usr/local
```

4.1.2 Check out

The source of EAPI-Utility is placed on [12].

4.1.3 Compilation

All steps below have to be executed in the directory, where utility source code was downloaded or checked out.

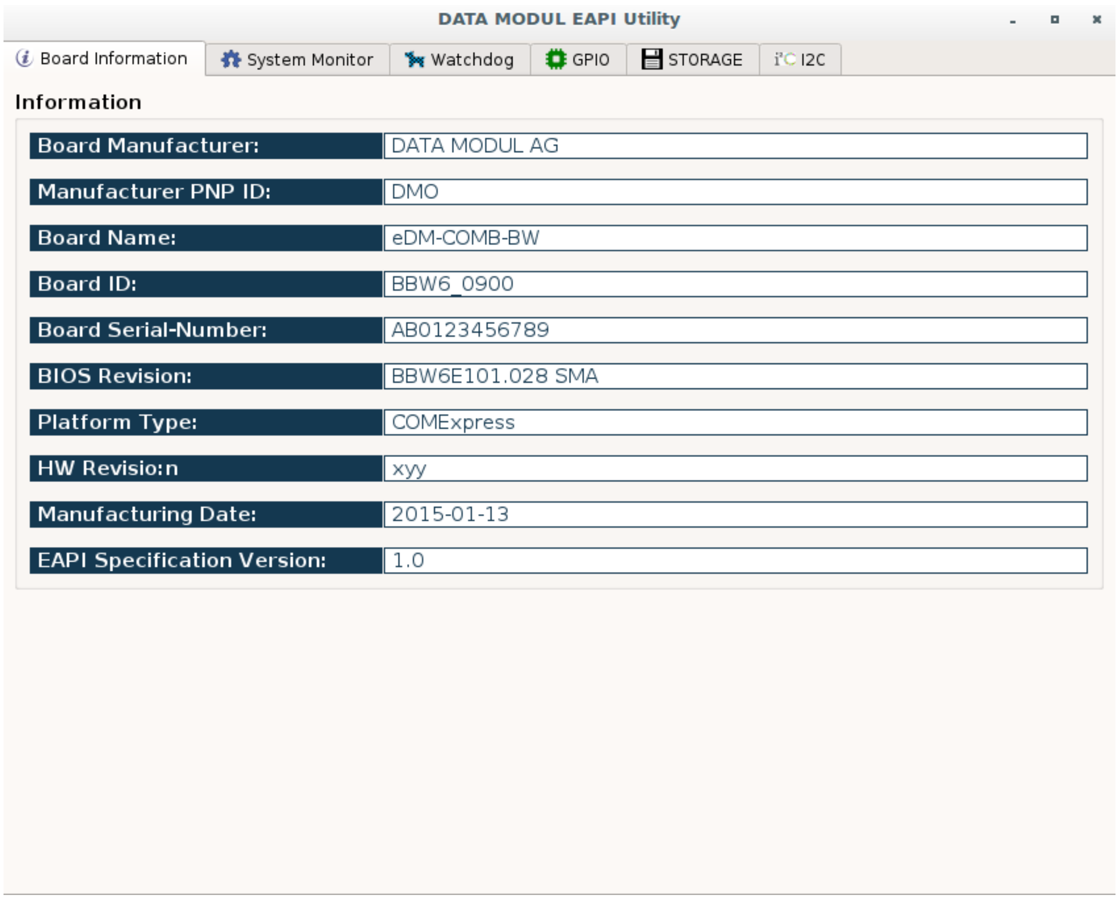
```
$ qmake gui-demo.pro
$ make
```

4.2 How use EAPI-Utility

As mentioned before, this application is developed in order to test EAPI. For each group of EAPI-functions, a specific tab is implemented, which will be explained in the following sub-sectins.

4.2.1 Board Information

In this window, all information related to the currect device are presented.



Board Manufacturer:	DATA MODUL AG
Manufacturer PNP ID:	DMO
Board Name:	eDM-COMB-BW
Board ID:	BBW6_0900
Board Serial-Number:	AB0123456789
BIOS Revision:	BBW6E101.028 SMA
Platform Type:	COMExpress
HW Revisio:n	xyy
Manufacturing Date:	2015-01-13
EAPI Specification Version:	1.0

Figure 1: Board Information Tab

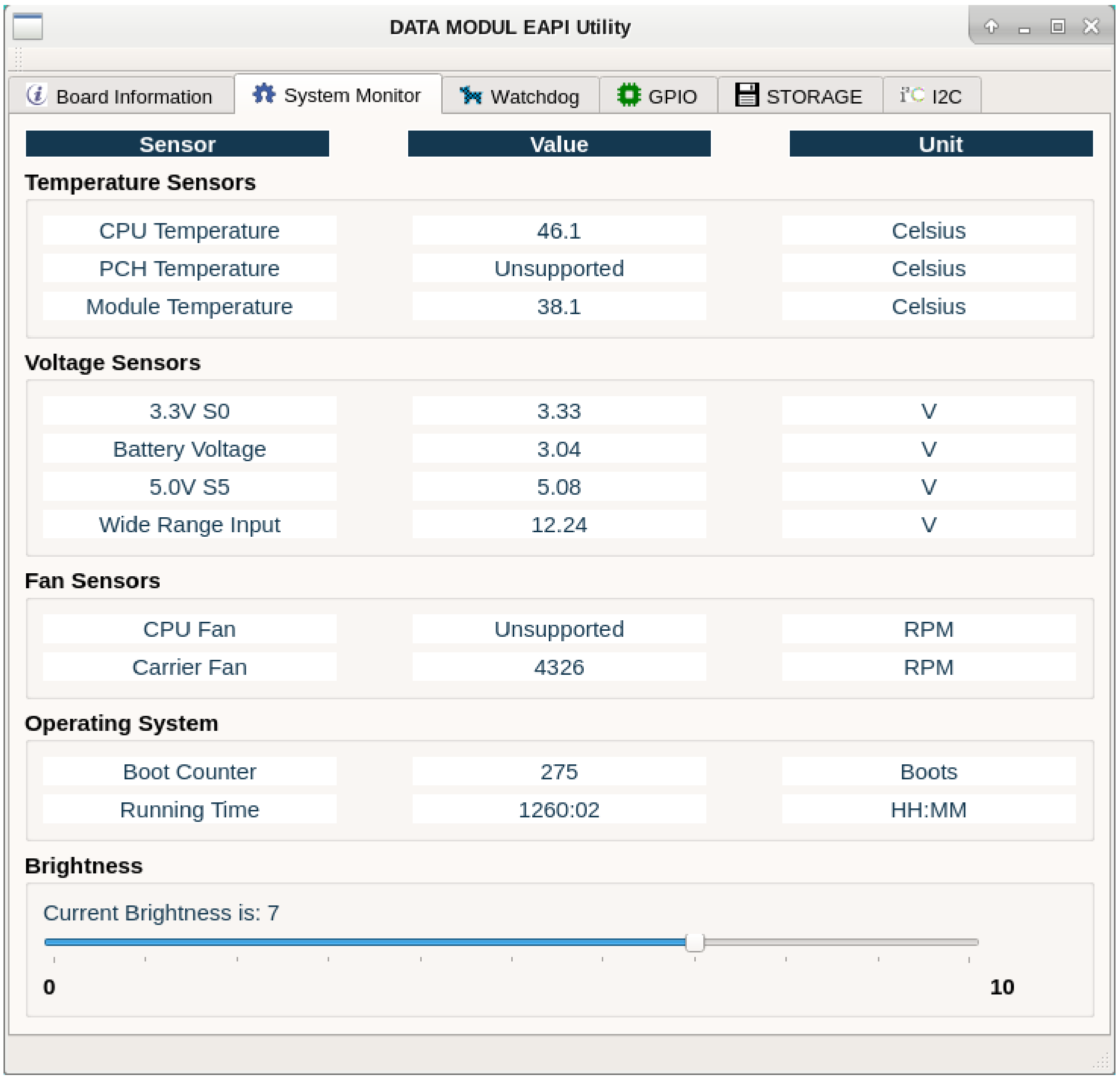
4.2.2 System Monitor

System monitor get information from hardware monitor, RTM and backlight. These information are updated continuously.

Hardware monitor contains three sensors: temperature sensor, voltage sensor and fan sensor. If hardware monitor module is not supported by the ACPI table, user will get **Unsupported** message.

RTM module provides runtime statistics as total running time in minutes and number of system boot.

The level of the brightness of the LVDS panel can be managed by the slide bar from 0 to 10. 0 value means no birghtness and 10 value means the maximum brightness.



Sensor	Value	Unit
Temperature Sensors		
CPU Temperature	46.1	Celsius
PCH Temperature	Unsupported	Celsius
Module Temperature	38.1	Celsius
Voltage Sensors		
3.3V S0	3.33	V
Battery Voltage	3.04	V
5.0V S5	5.08	V
Wide Range Input	12.24	V
Fan Sensors		
CPU Fan	Unsupported	RPM
Carrier Fan	4326	RPM
Operating System		
Boot Counter	275	Boots
Running Time	1260:02	HH:MM

Brightness

Current Brightness is: 7

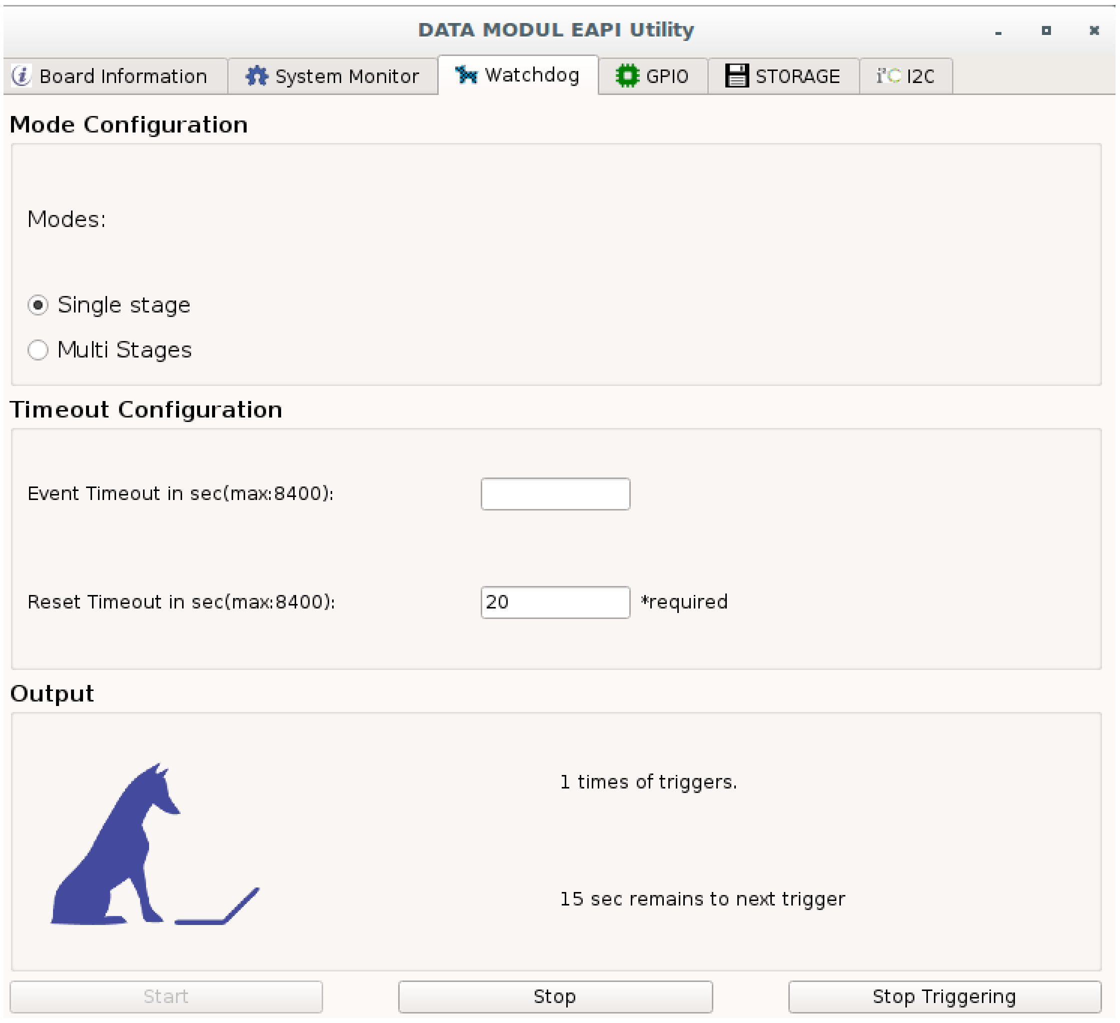
0 10

Figure 2: System Monitor Tab

4.2.3 Watchdog

The Watchdog is programmed to restart the system if after a specific time, an application crashes.

As explained in DMEC section, DMEC watchdog is implemented in two modes. Single stage and Multi stages(Window mode).



Mode Configuration

Modes:

(● Single stage) ○ Multi Stages

Timeout Configuration

Event Timeout in sec(max:8400):

Reset Timeout in sec(max:8400): 20 *required

Output

1 times of triggers.

15 sec remains to next trigger

Figure 3: Watchdog Tab

In single stage, **Reset Timeout** has to be set, however **Event Timeout(PreTimeout)** is not mandatory to be defined.

In Multi stages, both **Reset Timeout** and **Event Timeout** have to be set.

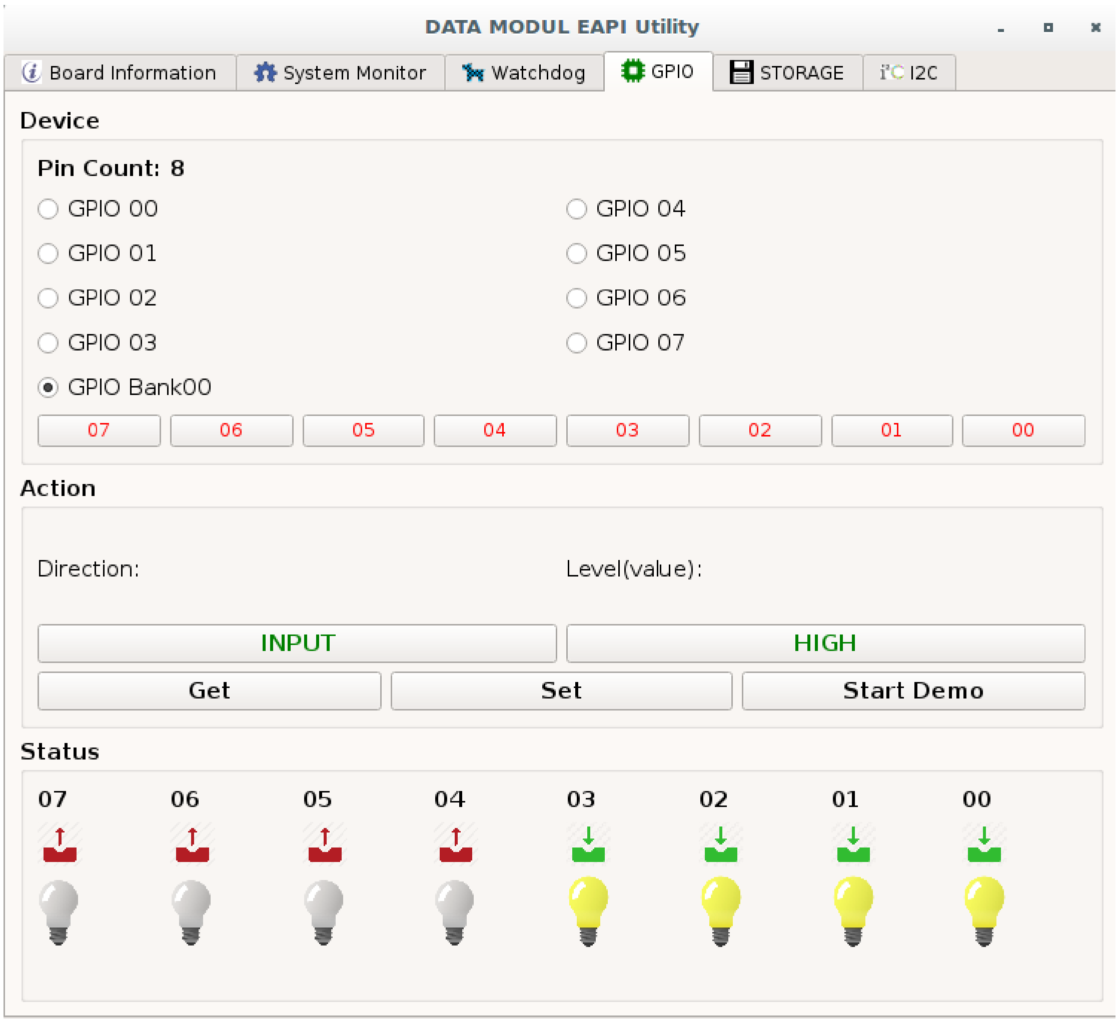
The watchdog timeout value is given in second and can be set up to 8400 seconds.

By clicking **Start** button, watchdog will repeatedly triggered withing the timeout period(In single stage, withing Reset Timeout or Event Timeout; In Multi stages within Reset timeout added by Event Timeout).

By clicking **Stop** button, the watchdog will stop.

By clicking **Stop Triggering**, triggering will stop, and therefore system will reset.

4.2.4 GPIO



Pin Count: 8

GPIO 00 | GPIO 04
GPIO 01 | GPIO 05
GPIO 02 | GPIO 06
GPIO 03 | GPIO 07
[✓ GPIO Bank00]

07 06 05 04 03 02 01 00

Direction: INPUT

Level(value): HIGH

07	06	05	04	03	02	01	00

The GPIO tab provides the possiblity to control the 8 GPIO pins.

Inside **Device** part, the specific pin through radio buttons from **GPIO 00** to **GPIO 07** can be selected. Also, the group of GPIO pins via **GPIO Bank00** can be selected too. Please notice to use **GPIO Bank00** the interested pins should be selected by clicking the corresponding number-buttons(red means selected, black means not selected).

The **Action** part provides two functionalities: **Get** direction and value of GPIO pins and **Set** direction and value of GPIO pins.

By clicking **Get** the result will be displayed in **Status** part. The on-lamp means the level of the GPIO pin is high. The off-lamp means the level of the GPIO pin is low.

The green-in-arrow means the direction of the GPIO pin is input. The red-out-arrow means the direction of the GPIO pin is output.

By clickng INPUT/OUTPUT button, user can select the GPIO direction for setting. By clicking HIGH/LOW button, user can select the GPIO level for setting.

Moreover, a demo is defined, in which by starting, 8 GPIO pins are set in way to present 0 to 255. For stop demo, just click on the button(Stop Demo) again.

4.2.5 Storage Area

It is possible that a specific area in EEPROM area is defined for user, in which user can write some important data in it and later read from the storage area.

The size of the storage area is displayed in the **Information** part.

To Read from the storage, user should set the desired offset and length, then press Read button.

To Write to the storage, user should set the desired offset and length, write the input inside **write** area, then click the write button. If the length of the input is greater than the size of the storage, write will be done only up to the size of the storage.

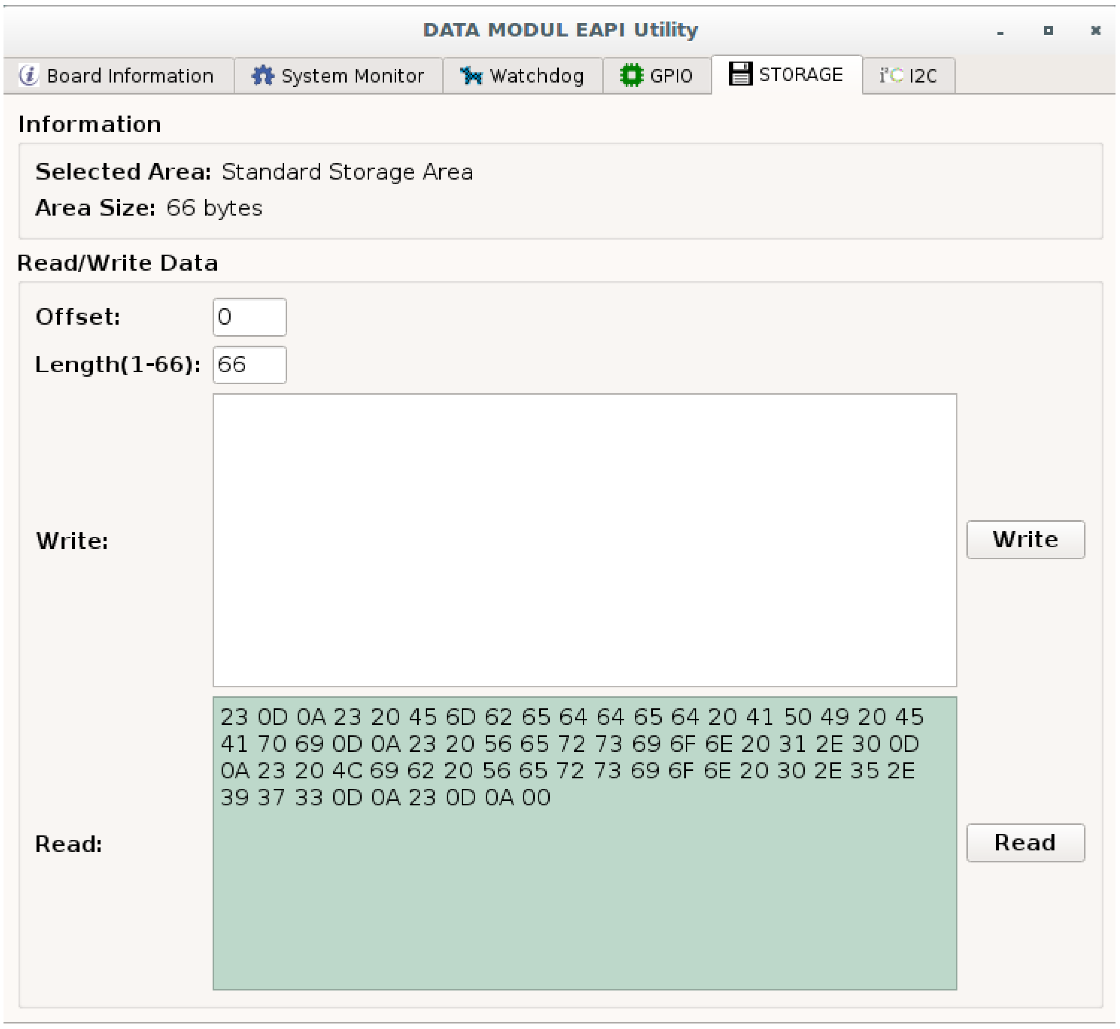
4.2.6 I2C

The Data Modul utility can access to I2C busses. These buses are displayed in **Device** part. I2C bus should be selected from the list. If **I2C select** is in red, it means no I2C bus is found. The slave address in HEX format should also be specified.

Three access protocol are defined:

- Read/Write Register: In this mode, only one byte will be read or written.
- Read/Write Continuous: In this mode, more than one byte, based on the **R-Length** and **W-Length** will be read or written.
- Write Read Raw : This mode provides universal mode for read and write. User handles read transaction by write proper offset to the I2C bus as write input.

In Control part, user should specify register offset, type of register offset(byte or word), length of read or length of write. Moreover, if it is needed to consider the write cycle timing, 0.005sec checkbox should be selected.



Information

Selected Area: Standard Storage Area
Area Size: 66 bytes

Read/Write Data

Offset: 0
Length(1-66): 66

Write:

Read:

```
23 0D 0A 23 20 45 6D 62 65 64 64 65 64 20 41 50 49 20 45
41 70 69 0D 0A 23 20 56 65 72 73 69 6F 6E 20 31 2E 30 0D
0A 23 20 4C 69 62 20 56 65 72 73 69 6F 6E 20 30 2E 35 2E
39 37 33 0D 0A 23 0D 0A 00
```

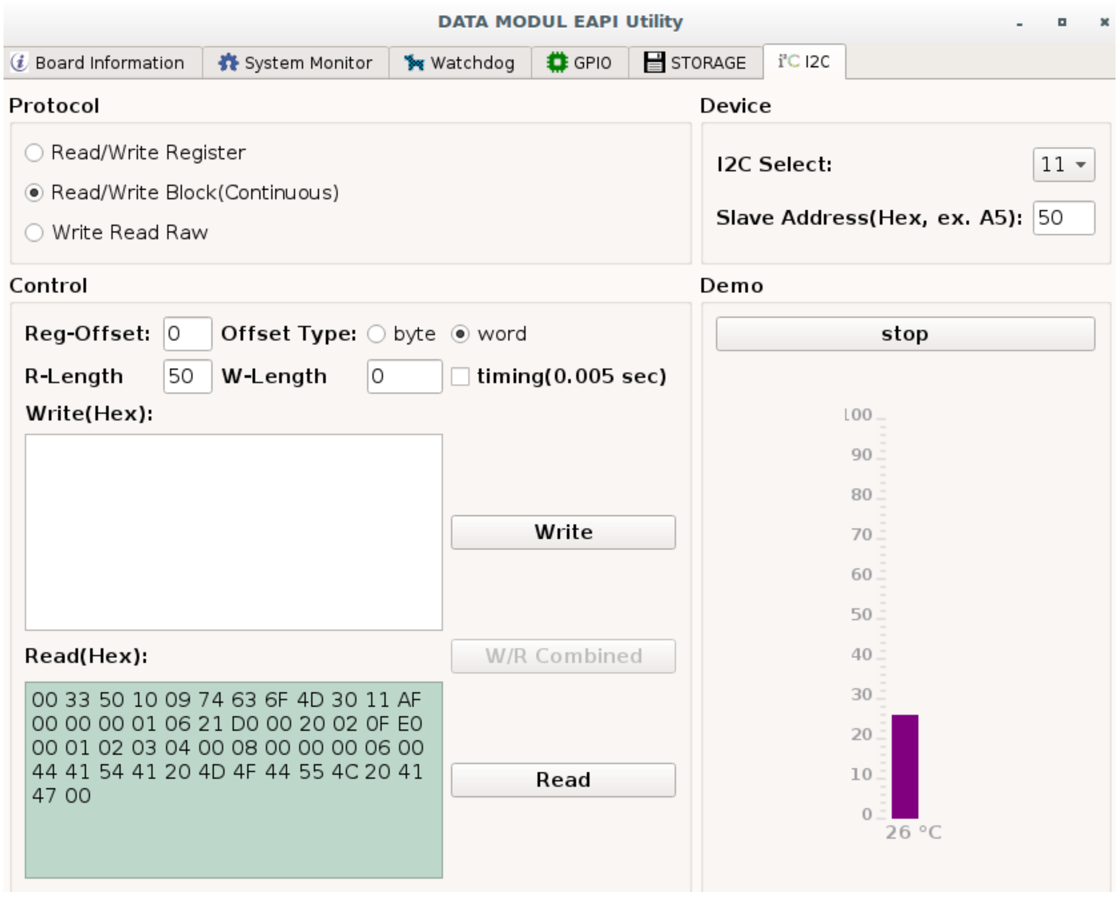
Figure 4: Storage Area Tab

Data for write has to be handed in HEX value in **write** area.

Data received by read, is displayed in **read** area as HEX value.

Demo is an example of connecting a temperature sensor to the I2C bus. The sensor has slave address: 0x48. The bar graph shows the current temperature of the environment. It will be updated by changes.

To start the demo, click the **Temperature** button in Demo part. For stopping the demo, press **stop** button.



Read/Write Register
Read/Write Block(Continuous) [✓]
Write Read Raw

I2C Select: 11
Slave Address(Hex, ex. A5): 50

Reg-Offset: 0 Offset Type: byte [✓ word]
R-Length 50 W-Length 0 timing(0.005 sec)
Write(Hex):

Read(Hex):

```
00 33 50 10 09 74 63 6F 4D 30 11 AF
00 00 00 01 06 21 D0 00 20 02 0F E0
00 01 02 03 04 00 08 00 00 00 06 00
44 41 54 41 20 4D 4F 44 55 4C 20 41
47 00
```

26 °C

Figure 5: I2C Tab