How to read or write peripheral registers

Stable: Not Stable / Revision: 08/04/2021 - 09:40

A quality version of this page, approved on *28 July 2020*, was based off this revision.

Contents

1 Article purpose

This article provides the basic information needed to start using the Devmem tool, which allows the reading and writing of peripheral registers.

2 Introduction

The following table provides a brief description of the tool, as well as its availability depending on the software packages:

✓: this tool is either present (ready to use or to be activated), or can be integrated and activated on the software package.

✗: this tool is not present and cannot be integrated, or it is present but cannot be activated on the software package.

Tool			STM32MPU Embedded Software distribution			STM32MPU Embedded Software distribution for Android™		
Name	Category	Purpose	Starter Package	Developer Package	Distribution Package	Starter Package	Developer Package	Distribution Package
Devmem	Tracing tools	Used for reading and writing in peripheral registers	✓	✓	✓	✗	✗	✗

3 Installing Devmem on your target board

You can install Devmem either with the Starter Package, the Developer Package or the Distribution Package. For the Starter Package, board internet access is required.

3.1 Using the STM32MPU Embedded Software distribution

3.1.1 Starter Package

Enter the following commands to install Devmem:

```
Board $> sudo apt-get update
Board $> sudo apt-get install devmem2
```

For more information about apt-get, see the Package_repository_for_OpenSTLinux_distribution article.

3.1.2 Developer Package

Enter the following commands:

- Download sources :

```
PC $> wget http://free-electrons.com/pub/mirror/devmem2.c
```

- Please ensure that the SDK environment for compiling Linux application setup is ready
- Compile the application :

```
PC $> make devmem2
```

- Install it on the board:

```
PC $> scp devmem2 root@<board ip address>:/usr/bin
```

3.1.3 Distribution Package

Enter the following commands:

- Build the package with bitbake:

```
PC $> bitbake devmem2
```

- Add it to the targeted image:

```
PC $> echo 'IMAGE_INSTALL_append += "devmem2"' >> meta-st/meta-st-openstlinux/recipes-st
/images/st-image-weston.bbappend
```

- Rebuild the image:

```
PC $> bitbake st-image-weston
```

You must then Flash the generated image onto your board using STM32CubeProgrammer.

4 Getting started

To use Devmem, enter the following command:

```
Board $> devmem2 [address] [type] [data]
```

Where:

- [address] corresponds to the register address you want to read
- [type] corresponds to the desired output value size (b, h, w or l, respectively byte, halfword, word or long). Assigning a type is not necessary if you only read the register - the default output type is a word.
- [data] corresponds to the data value you wish to write to the register, assigning a type is mandatory in this case . If data is not assigned, the register is only read.

5 To go further

5.1 Checking UART4 baud rate

The UART baud rate is generated from the UART clock frequency, with the USARTDIV parameter field in the USART_BRR register. By default in the OpenSTLinux Starter Package delivery, the UART4 baud rate is set to 115 000, the clock frequency CLOCK_UART4_K value is 64 000 000 Hz and USARTDIV is equal to 556. These values are known, and the USARTDIV value is retrieved by using Devmem to read the STM32MP1 registers.

On the STM32MP157 Reference Manual, we see that the base address of UART4 is 0x4001 0000 and the USARTDIV value is located at offset 0x0C.

```
Board $> devmem2 0x4001000C
/dev/mem opened.
Memory mapped at address 0xb6fc300c.
Read at address 0x4001000C (0xb6fc300c): 0x0000022C
```

This is equal to 556 decimal.

5.2 Driving LEDs with Devmem

In this example, we check the output value of GPIO port A pins 13 and 14 corresponding to LEDs 6 and 5 on DK2. According to the Reference Manual, the base address for GPIOA is 0x5000 2000 and the output value of the LEDs is located at the offset 0x14.

However, the GPIO clock must first be enabled for the results to be readable from the registers. Otherwise the resulting output value is 0:

```
Board $> devmem2 0x50002014
/dev/mem opened.
Memory mapped at address 0xb6f43014.
Read at address 0x50002014 (0xb6f43014): 0x00000000
```

The clock is named RCC_MP_AHB4ENSETR and is located at address 0x5000 0A28.

```
Board $> devmem2 0x50000A28
/dev/mem opened.
Memory mapped at address 0xb6fdca28.
Read at address 0x50000A28 (0xb6fdca28): 0x00000000
```

In this register, the first eleven bits (from LSB to MSB) allow GPIOA to GPIOK respectively to be enabled (1 enable, 0 disable). In this case only GPIOA needs to be enabled, so we write 0x1 to the register.

```
Board $> devmem2 0x50000A28 w 0x1
/dev/mem opened.
Memory mapped at address 0xb6fdca28.
Read at address 0x50000A28 (0xb6fdca28): 0x00000000
Write at address 0x50000A28 (0xb6fdca28): 0x00000001, read back 0x00000001
```

We now check the content of the register corresponding to GPIOA:

```
Board $> devmem2 0x50002014
/dev/mem opened.
Memory mapped at address 0xb6f43014.
Read at address 0x50002014(0xb6f43014): 0x00002400
```

In this case only LED 5 is enabled. The following table shows the register output value as a function of LED 5 and 6 states.

	LED 5 enabled	LED 5 disabled
LED 6 enabled	0x400	0x4400
LED 6 disabled	0x2400	0x6400

For example to enable both LEDs, set the value of the register to 0x400:

```
Board $> devmem2 0x50002014 w 0x400
/dev/mem opened.
Memory mapped at address 0xb6f43014.
Read at address 0x50002014 (0xb6f43014): 0x00002400
Write at address 0x50002014 (0xb6f43014): 0x00000400, readback 0x00000400
```

The change can be seen on your board.

6 Restricted access

Keep in mind that the Devmem tool only allows access to peripheral registers. Also all the registers in M4 and A7 cores are secured and attempting to read them causes the board to reboot.

When trying to access a register, make sure the related peripheral has not been isolated on the MCU side, otherwise access is impossible due to access rights.

Universal Asynchronous Receiver/Transmitter

Universal Synchronous/Asynchronous Receiver/Transmitter

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Reset and Clock Control

Light-emitting diode

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)