GPIO on Fujitsu mainboards

COLLABORATORS

	TITLE : GPIO on Fujitsu mainboards	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Rainer König	August 6, 2014	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.9	2014-01-07	First draft	Rainer König
1.0	2014-08-06	Updated with more driver details and new board D3313-Sx	Rainer König

Contents

1 Introduction **1**

1.1 General information 1

1.2 Required prerequisites and skills 1

2 Fujitsu OEM mainboard overview **2**

2.1 Mainboards and kernel drivers 2

2.2 Pin assignment of the GPIO connector 2

3 Using GPIO **4**

3.1 Scan the smbus 4

3.2 Initializing the GPIO pins 5

3.3 Setting values of output pins 7

4 Online resources **8**

4.1 Access GPIO from Linux user space 8

4.2 GPIO interface access conventions on Linux 8

List of Tables

2.1 Fujitsu mainboards 2
2.2 GPIO connector pinout 3

Abstract

This document explains how to use the General Purpose I/O features of current Fujitsu mainboards with various Linux distributions.

Chapter 1

Introduction

1.1 General information

Fujitsu industrial mainboards provide GPIO connector with 8 pins that can be programmed as input or output pins and that can be controlled by the PCA953x kernel module.

The kernel module should be shipped with all Linux distributions that have a kernel version >=3.1.x.

This HowTo describes the steps to setup the GPIO interface and how to use it.

1.2 Required prerequisites and skills

To do the configuration steps described in this document you need *root access* for your Linux operating system.

Important
All shell commands must be executed with root privileges.

If your distribution kernel does not provide the required device driver module then you need to compile it by yourself. Therefore you need to know how to compile kernel modules for your distribution kernel. This also requires that your system has the complete toolchain for building kernel modules installed.

You need to know how to locate and install packages for your Linux distribution.

Chapter 2

Fujitsu OEM mainboard overview

2.1 Mainboards and kernel drivers

This table shows you which mainboards you can use for GPIO and what kernel-driver is needed to access the SM-Bus interface on those mainboards.

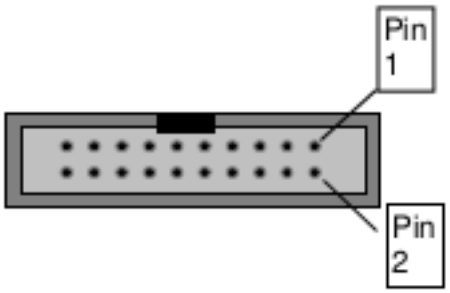
Board	SM-Bus driver	Comment
D3003-Sx	i2c-piix4	
D3076-Sx	i2c-i801	
D3236-Sx	i2c-i801	See comment below
D3313-Sx	i2c-piix4	

Table 2.1: Fujitsu mainboards

Important
On the D3236-Sx board you need to apply the kernel boot parameter `acpi_enforce_resources=lax` to avoid problems. If you forget this parameter you won't be able to scan the SM-Bus later.

2.2 Pin assignment of the GPIO connector

The following graphics shows the GPIO pinout.



Pinout of the GPIO connector

The pin assignement of the GPIO connector is defined in the following table.

Pin	Signal	Type
1	GPIO0	
2	GPIO1	
3	GPIO2	
4	GPIO3	
5	GPIO4	
6	GPIO5	
7	GPIO6	
8	GPIO7	
9	+3.3V	Power
10	GND	Power
11	+3.3V	Power
12	+5V standby	Power
13	Reserved	
14	GND	Power
15	Reserved	
16	GND	Power
17	GND	Power
18	+5V	Power
19	+12V	Power
20	+12V	Power

Table 2.2: GPIO connector pinout

Chapter 3

Using GPIO

3.1 Scan the smbus

First check if the needed kernel driver from the mainboard table is loaded.

```
# lsmod | grep i2c
```

```
1 i2c_i801               22444  0
2 i2c_algo_bit           13413  2 igb,i915
```

You should find the driver listed in the mainboard table on the output of the **lsmod** command. In the example above you see the module listed on line 1.

Important
The next important step is to load the i2c-dev driver. This driver is needed to probe the I^2C bus.

```
# modprobe i2c-dev
```

Now you need to locate the smbus.

```
# i2cdetect -l
```

i2cdetect will show a list of all available I2C buses.

```
1  i2c-0   i2c             i915 gmbus ssc                          I2C adapter
2  i2c-1   i2c             i915 gmbus vga                          I2C adapter
3  i2c-2   i2c             i915 gmbus panel                        I2C adapter
4  i2c-3   i2c             i915 gmbus dpc                          I2C adapter
5  i2c-4   i2c             i915 gmbus dpb                          I2C adapter
6  i2c-5   i2c             i915 gmbus dpd                          I2C adapter
7  i2c-6   i2c             DPDDC-B                                 I2C adapter
8  i2c-7   i2c             DPDDC-C                                 I2C adapter
9  i2c-8   i2c             DPDDC-D                                 I2C adapter
10 i2c-9   smbus           SMBus I801 adapter at f040              SMBus adapter
```

In the example above the smbus is located on the I^2C bus number 9 (shown in line 10).

Now you need to scan the smbus. Use the bus nubmer found above (in our example its 9) as a parameter to **i2cdetect**.

```
# i2cdetect 9
```

i2cdetect will ask for confirmation (answer "Y" here) and then produce some output like this:

```
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-9.
I will probe address range 0x00-0x7f.
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: 00 -- -- -- -- -- -- -- 08 -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- 31 -- 33 -- -- -- -- -- -- -- -- 3c -- 3e --
40: -- -- -- -- 44 -- -- -- -- -- -- -- -- -- -- --
50: -- 51 -- 53 -- -- -- -- -- -- -- -- -- -- -- --
60: 60 61 -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- 73 -- -- -- -- -- -- -- -- -- -- -- --
```

The GPIO controller should be located on bus address `0x3C`, we have an entry on this address so now we need to check what is there.

```
# i2cdump 9 0x3c
```

The bus number (in our example 9) and the bus address `0x3c` are provided as paramters to the **i2cdump** command. The output should look like this:

```
No size specified (using byte-data access)
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-9, address 0x3c, mode byte
Continue? [Y/n] y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: ff ff 00 ff XX XX XX XX XX XX XX XX XX XX XX XX    ....XXXXXXXXXXXX
10: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
20: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
30: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
40: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
50: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
60: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
70: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
80: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
90: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
a0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
b0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
c0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
d0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
e0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
f0: XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX    XXXXXXXXXXXXXXXX
```

The addresses `0x00` to `0x03` (the first four bytes) contain the registers and default values of the PCA9554 chip.

3.2 Initializing the GPIO pins

Now you need to load and initialize the GPIO driver for the PCA9554.

```
# modprobe gpio-pca953x
```

If you get an error when executing this command this means, that your distribution kernel doesn't provide a compiled module for the GPIO driver. In that case you have to build your own kernel and set the kernel configuration parameter "CONFIG_GPIO_PCA953x=m". This needs to be done e.g. for Fedora 20, other distributions (e.g. openSUSE 12.3) have that module already compiled into the kernel. You can also check if there is a module named "pca953x.ko" which appears to be the name for the driver used in older kernels.

Once the kernel driver is successfully loaded you need to configure the GPIO device with the following command.

```
# echo pca9554 0x3c > /sys/bus/i2c/devices/i2c-9/new_device
```

Important
In the example above we used the bus number 9 to construct the directory name `i2c-9`.

When this command was successful a new directory `/sys/class/gpio` should exist. Lets have a look what we find there:

```
1 # cd /sys/class/gpio/
2 # ls -l
3 total 0
4 --w------- 1 root root 4096 Jan  7 11:02 export
5 lrwxrwxrwx 1 root root    0 Jan  7 10:46 gpiochip248 -> ../../devices/pci0000:00/0000:00:1f ↩
      .3/i2c-9/9-003c/gpio/gpiochip248
6 --w------- 1 root root 4096 Jan  7 11:11 unexport
```

There are 2 files (`export` and `unexport`) and a directory named `gpiochip248`. Lets inspect that directory:

```
1 # ls -l gpiochip248
2 total 0
3 -r--r--r-- 1 root root 4096 Jan  7 10:46 base
4 lrwxrwxrwx 1 root root    0 Jan  7 10:46 device -> ../../../9-003c
5 -r--r--r-- 1 root root 4096 Jan  7 10:46 label
6 -r--r--r-- 1 root root 4096 Jan  7 10:46 ngpio
7 drwxr-xr-x 2 root root    0 Jan  7 10:46 power
8 lrwxrwxrwx 1 root root    0 Jan  7 10:46 subsystem -> ../../../../../../../class/gpio
9 -rw-r--r-- 1 root root 4096 Jan  7 10:45 uevent
```

The file `base` contains the base number of the first GPIO that can be used. In our example this is 248:

```
1 # cat gpiochip248/base
2 248
```

The file `ngpio` contains the number of usable GPIO pins. In our example this is 8:

```
1 # cat gpiochip248/ngpio
2 8
```

The file `label` contains the string `"pca9554"` which we provided in the **echo** above.

```
1 # cat gpiochip248/label
2 pca9554
```

At this point we know that the GPIO pins will be numbered from 248 to 255 (`base` + (`ngpio`-1). So GPIO pin 0 will have th number 248 in our driver setup and GPIO pin 7 will have the number 255.

Now we have to enable all the connector pins you want to use. So, if we want to use the pin 0 as an output pin you need to submit the following commands:

```
1 # cd /sys/class/gpio/
2 # echo 248 > export
```

The command in line 2 creates a new directory `gpio248` which contains the files we need to setup.

```
1 # ls -l
2 total 0
3 --w------- 1 root root 4096 Jan  7 11:32 export
```

```
lrwxrwxrwx 1 root root    0 Jan  7 11:32 gpio248 -> ../../devices/pci0000:00/0000:00:1f.3/ ↩
    i2c-9/9-003c/gpio/gpio248
lrwxrwxrwx 1 root root    0 Jan  7 10:46 gpiochip248 -> ../../devices/pci0000:00/0000:00:1f ↩
    .3/i2c-9/9-003c/gpio/gpiochip248
--w------- 1 root root 4096 Jan  7 11:11 unexport

 # ls -l gpio248/
total 0
-rw-r--r-- 1 root root 4096 Jan  7 11:32 active_low
lrwxrwxrwx 1 root root    0 Jan  7 11:32 device -> ../../../9-003c
-rw-r--r-- 1 root root 4096 Jan  7 11:32 direction
drwxr-xr-x 2 root root    0 Jan  7 11:32 power
lrwxrwxrwx 1 root root    0 Jan  7 11:32 subsystem -> ../../../../../../../class/gpio
-rw-r--r-- 1 root root 4096 Jan  7 11:32 uevent
-rw-r--r-- 1 root root 4096 Jan  7 11:32 value
```

Now we need to define that pin as an output pin:

```
 # echo out > gpio248/direction
```

The file `direction` defines if the pin is used as an output (`"out"`) or an input (`"in"`).

You can use the same initialization scheme for all other GPIO pins. Just echo the number (range 248..255) to the file `export` and a new directory for this GPIO will be created. If you want to erase the configuration of a pin echo its number to the file `unexport`.

3.3 Setting values of output pins

If you have defined one GPIO pin as output you can set its value to `0` or `1` by writing the value to the value file of the according directory:

```
# cd /sys/class/gpio/
# echo 0 > gpio248/active_low
# echo 1 > gpio248/value
```

The file `active_low` defines if the pin on the connector will be "high" or "low" if the file `value` contains a `1`. In the example above the pin will be "high", if you set `active_low` to `1` then the result will be "low". The default setting for `active_low` is `0`.

Chapter 4

Online resources

4.1 Access GPIO from Linux user space

http://falsinsoft.blogspot.de/2012/11/access-gpio-from-linux-user-space.html

4.2 GPIO interface access conventions on Linux

http://www.michaelhleonard.com/gpio-interface-access-conventions-on-linux/