Application Report
SPRAB39–May 2009

GPIO and PINMUX Driver for TMS320DM6446 and TMS320DM355

Frangline Jose

ABSTRACT

This application report describes the implementation of a general-purpose input/output (GPIO) and PINMUX driver that can be used from the user space. The idea is to use the low-level functions provided within the Linux® kernel in the DaVinci devices and to overwrite them with a driver that can be accessed from the user space.

Project collateral and source code discussed in this application report can be downloaded from the following URL: http://www-s.ti.com/sc/techlit/sprab39.zip.

Contents

1 Introduction

This application report shows how to build a Linux driver to access GPIO and PINMUX modules in the DaVinci devices. The driver is based on the Linux Kernel 2.6.10, which is available with the Davinci evaluation module (EVM). The document initially discusses a driver that can be used to access the peripheral registers of the device. The GPIO and PINMUX drivers are built based on this driver to access the registers. Source code for the GPIO and PINMUX drivers are provided along with this application report. A few sample applications are also included to demonstrate the usage of the driver.

2 Existing Low Level GPIO Driver

A low-level driver exists in the LSP1.20 version of the kernel that is provided with the EVM. This driver is not accessible from the user space and is used by other driver modules to configure the I/O's required by the respective peripherals. The source for the low level GPIO driver is found in the following folders:

KERNEL_FOLDER/arch/arm/mach-davinci/gpio.c file

KERNEL_FOLDER/include/asm-arm/gpio.h

The simplest option is to build a driver based on the existing low-level driver; however, there is no such low-level driver available for PINMUX. Writing to a GPIO driver alone is not of much use, as PINMUX registers play an important role in selecting the GPIO function for each pin in the device. This makes the necessity to move away from the existing low-level GPIO drivers and to use a different method to configure the respective GPIO and PINMUX registers.

3 writel and readl Kernel Functions

To directly access the device peripheral registers, the kernel provides the readb, readw, readl, writeb, writew and writel macros. These macros can be used to access and configure the respective registers in a device.

The macros are:

- unsigned readb(address) – read a byte
- unsigned readw(address) – read a word
- unsigned readl(address) – read a long word
- void writeb(unsigned value, address) – write a byte
- void writew(unsigned value, address) – write a word
- void writel(unsigned value, address) – write a long word

In DaVinci devices, memory is accessed in 32-bit long words; therefore, in this driver readl and writel functions are used.

4 Register Access Driver Using writel and readl Functions

The major number is chosen as 201 for the register access driver, 202 and 203 for the GPIO driver and 204 for the PINMUX driver. This is based on a quick survey of the existing major numbers used in the LSP 1.20. It is a better practice to verify if these numbers can be used in your system before using these major numbers in the kernel.

```
#define REGISTER_ACCESS_MAJOR 201
#define REGISTER_ACCESS_NAME  "REGISTER_ACCESS_DRIVER"
```

The following code is used to load the module in the kernel.

```
int __init init_module (void) /* Loads a module in the kernel */
{
         int status;
         printk("Initializing the register access driver \n");
         printk("This driver is for DM devices on LSP1.20 \n");

         status = register_chrdev(REGISTER_ACCESS_MAJOR,
         REGISTER_ACCESS_NAME, &register_access_file_ops);
         return status;
}
```

The following code is used to unload the module from the kernel.

```
void cleanup_module(void) /* Removes module from kernel */
{
        printk("Cleaning up register access driver \n");
        unregister_chrdev(REGISTER_ACCESS_MAJOR,
        REGISTER_ACCESS_NAME);
}
```

The following methods are available in the driver: open, read, and write.

```
struct file_operations gpio_custom_file_ops = {
     .owner      =  THIS_MODULE,
     .read       =  register_access_read,
     .write      =  register_access_write,
     .open       =  register_access_open,
};
```

The following code is used to read the registers.

```
static int register_access_read(int file_handle, unsigned int reg_num, int io_value)
{
         return davinci_readl(reg_num);
}
```

The following code is used to write to a register.

```
static int register_access_write(int file_handle, unsigned int reg_num, unsigned int reg_val)
{
        //printk("REG=0x%X, Val=0x%X\n",reg_num,reg_val);
        davinci_writel(reg_val, reg_num);
        return 0;
}
```

5 GPIO Driver

The register driver mentioned in Section 4 can be used to read and write to the GPIO drivers and also to any of the registers in the device. However, many times it may be simpler to directly address individual pins of the GPIO. Using the readl and writel macros, it is possible to read or write an entire 32-bit register. Based on the pin number provided, the GPIO driver internally uses a bit mask to read or write to the correct pins. This is the only difference between the register access driver and the GPIO driver. The GPIO driver is also divided into two different modules: one to read and write to the pins and the other to set the direction of the pins. The PINMUX registers need to be set before using the GPIO's.

The following code snippet shows the bit mask pattern generated and used to return the status of a GPIO pin.

```
if ( 0 <= reg_num <= 31  )
{
io_value = davinci_readl(DAVINCI_GPIO_BASE+0x20);
x = x << (reg_num);
io_value = io_value & x;
return (io_value >> (reg_num));
}
```

Likewise, this piece of code is used to generate the bit mask and to set or clear a pin.

```
if ( 0 <= reg_num <= 31  )
{
io_value = davinci_readl(DAVINCI_GPIO_BASE+0x14);
if (reg_val) io_value = io_value | (x << (reg_num));
else io_value = io_value & (~(x << reg_num));
return davinci_writel(io_value,DAVINCI_GPIO_BASE+0x14);
}
```

The two cases above can only be used to read or write to one pin at a time. However, if it is required to read and write to multiple pins, the entire register needs to be read and/or written. You should take care of the bitmask required in this case. To allow this 32-bit read and write, at the end of the read and write functions, the following code is added:

```
if ( (reg_num ==  DAVINCI_GPIO_BASE+0x20) ||
     (reg_num ==  DAVINCI_GPIO_BASE+0x48) ||
     (reg_num ==  DAVINCI_GPIO_BASE+0x70) ||
     (reg_num ==  DAVINCI_GPIO_BASE+0x98) )
return davinci_readl(reg_num);
```

The GPIO direction driver uses the same code except for the change in register address required; therefore, it is not discussed here.

6 Building the Driver

A simple make file is provided along with the drivers that can be used to build the modules. The path for the ARM GCC is assumed to be set in the shell. Set *KDIR* in the make file to point to the kernel folder.

Executing make in the respective folders should build the modules.

7 An Example Code That Uses the Driver's

This application report includes an example code to demonstrate the usage of the GPIO and PINMUX drivers. This sample code is tested on the TMS320DM355 device to toggle pin 16, which was verified with an oscilloscope.

The first step to test the driver is to insert the GPIO and PINMUX drivers (kernel objects) to the kernel as mentioned below.

- insmod gpio_custom.ko
- insmod gpio_custom_dir.ko
- insmod pinmux_custom.ko

Next, create a virtual file to access these drivers, assuming that the GPIO driver is assigned the Major Number 201,202 and PINMUX driver is assigned 203.

- mknod /dev/gpiocustom c 202 0
- mknod /dev/gpiocustom_dir c 203 0
- mknod /dev/pinmuxcustom c 204 0

The steps above make the GPIO and PINMUX drivers available to the user space via the virtual files /dev/gpiocustom, /dev/gpiocustom_dir and /dev/pinmuxcustom.

The following *c* code demonstrates the usage of the drivers.

Step 1:

```
Open the drivers and get the file handlers
       fdg=open("/dev/gpiocustom",O_RDWR);
       if(fdg==-1)
       {
       printf("Failed to open the driver\n");
       return 0;
       }

       fdgd=open("/dev/gpiocustomdir",O_RDWR);
       if(fdgd==-1)
       {
       printf("Failed to open the driver\n");
       return 0;
       }

       fdp=open("/dev/pinmuxcustom",O_RDWR);
       if(fdp==-1)
       {
       printf("Failed to open the driver\n");
       return 0;
```

Step 2:

Use the read function to read the PINMUX value and the write function to set or clear the desired pin. Using a bit mask can be useful as it can leave the rest of the pins untouched. To set a pin to GPIO, the corresponding bit needs to be cleared in the PINMUX register. The following code sets pin 16 and 17 as GPIO's.

```
val=read(fdp,PINMUX3,0);
val=val & 0xFFF9FFFF;
write(fdp,PINMUX3,val)}
```

Step 3:

Use the write function on the GPIO direction driver to set a particular pin as input or output. The following code sets the pin as output:

```
write(fdgd,17,0);
```

Alternately, the direction of the pin can be set by writing to the entire register.

```
val=read(fdgd,DIR01,0);
val=val & 0xFFF0FFFF;
write(fdgd,DIR01,val);
```

Use the write function on the GPIO driver to set or clear a particular pin. The following code is used to toggle the pin:

```
While(1){
write(fdg,17,0);
usleep(500000);
write(fdg,17,1);
usleep(500000);}
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265