Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería en Computadores
Introducción a los Sistemas Embebidos
Prof. Ing. Jeferson González Gómez

Tutorial: Introducción a drivers en Linux - III

Índice

1. Introducción

Este tutorial presenta la última parte del enfoque introductorio a la creación y uso de controladores de dispositivos (*drivers*) en Linux. En el desarrollo de este tutorial se creará un controlador en espacio de usuario para el módulo de entradas y salidas de propósito general (*GPIO*, en adelante) de una tarjeta Raspberrypi 2, utilizando punteros y mapeo de memoria física a memoria virtual, en espacio de usuario, así como las herramientas de construcción cruzada que provee el proyecto *Yocto* (ver tutoriales 4 y 5).

Al igual que el tutorial anterior, para el desarrollo de este tutorial será necesario contar con la hoja de datos del fabricante del Sistema en Chip (SoC) de la tarjeta Raspberrypi 2 (Broadcom BCM2835), que contiene información del funcionamiento y configuración los diferentes periféricos dentro del SoC. Dicha hoja de datos, puede descargarse de: `https://cdn-shop.adafruit.com/product-files/2885/BCM2835Datasheet.pdf`

2. "Drivers" en espacio de usuario

Como se mencionó en tutoriales anteriores, un driver para un dispositivo se implementa normalmente es espacio de kernel, de forma que se tenga un acceso más rápido y directo al hardware.

Este enfoque funciona bien para dispositivos para los cuáles no se tiene un controlador implementado o construido dentro del kernel del sistema operativo por defecto (como fue el caso de módulo GPIO, del tutorial anterior). Sin embargo, para dispositivos más comunes, como el caso del GPIO, normalmente existe una versión de driver compilada junto con el kernel, pues el sistema operativo debe estar en la capacidad de utilizar los pines de entrada y salida del chip al arrancar. En estos escenarios no es posible implementar un driver en espacio de kernel, ya que el espacio de memoria que se solicitaría (utilizando la función *request_mem_region()*) se encuentra ocupado por el controlador del sistema. Para el caso del controlador de GPIO, esto se puede verificar utilizando el comando:

```
cat /proc/iomem
```

En la salida de dicho comando se puede observar cómo las direcciones del módulo de GPIO se encuentran reservadas:

```
3f200000-3f2000b3 : /soc/gpio@7e200000
```

Afortunadamente, en Linux existen formas de escribir y leer de dispositivos que se encuentren mapeados a memoria, sin necesidad de realizar llamadas al sistema costosas en tiempo. En primer lugar, el archivo de dispositivo */dev/mem*, en sistemas Linux, contiene una imagen de la memoria del computador, por lo que puede ser utilizado para escribir y leer de dispositivos, siempre y cuando éstos se encuentren mapeados en memoria (caso usual). Esto sería suficiente en esquemas sin memoria virtual, sin embargo al contar con una unidad de manejo de memoria (MMU) las direcciones de /dev/mem no pueden ser accedidas directamente, pues serían convertidas a direcciones no correspondientes con las físicas. Para solucionar esto, la función *mmap* `http://man7.org/linux/man-pages/man2/mmap.2.html` permite mapear un rango de memoria física (a partir de una dirección base) a un puntero en memoria virtual, que es accesible al proceso en espacio de usuario. De esta forma, es posible escribir y leer a dispositivos mapeados en memoria por medio punteros, tal y como se realizaría en un esquema de programación *Bare-metal*.

3. Funcionamiento del módulo GPIO

El módulo de control de GPIO, dentro del SoC BCM2835, permite configurar el comportamiento de los pines de entrada y salida del chip. En este comportamiento se incluyen lecturas y escrituras, así como la posibilidad de establecer la dirección (entrada-salida) de un pin o la asignación de flancos de interrupción y otras funciones alternativas (SPI, UART, PWM, etc). El funcionamiento de los *GPIO* del SoC BCM2835 se encuentra descrito a partir de la página 89 de la hoja de datos del fabricante. En esta sección se presentará un resumen de las funcionalidades básicas para establecer un pin específico como salida y escribir un valor, desde espacio de usuario.

El módulo de GPIO interactúa con el procesador por medio de registros, que son accedidos por medio de una dirección base (*base address*) y un desplazamiento (*offset*). Para el caso del BCM2835 la dirección base para el *GPIO* corresponde a $0x7E200000$, esta dirección corresponde a una dirección base del SoC, sin embargo, el sistema operativo lo asigna a una dirección distinta, por medio de un *offset* a todas las direcciones. Para el caso de este tutorial, la dirección base del *GPIO* será: $0x3F200000$. Una sección de los registros de GPIO se muestran en la Fig.1

Para los objetivos de este tutorial, se abarcarán solamente la configuración de un único GPIO (GPIO2), a través del registros **GPFSEL0**, **GPSET0** y **GPFCLR0**.

Address	Field Name	Description	Size	Read/ Write
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x 7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x 7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x 7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W
0x 7E20 0014	GPFSEL5	GPIO Function Select 5	32	R/W
0x 7E20 0018	-	Reserved	-	-
0x 7E20 001C	GPSET0	GPIO Pin Output Set 0	32	W
0x 7E20 0020	GPSET1	GPIO Pin Output Set 1	32	W
0x 7E20 0024	-	Reserved	-	-
0x 7E20 0028	GPCLR0	GPIO Pin Output Clear 0	32	W
0x 7E20 002C	GPCLR1	GPIO Pin Output Clear 1	32	W

Figura 1: Segmento de los registros del módulo GPIO del BCM2835

3.1. Registro selección de función - GPFSELn

El registro GPFSEL0 permite la selección de la función que tendrá en GPIOn, según la tabla de la figura 2 (mostrada para el GPIO9). Como se nota en la hoja de datos (pag.92), para el GPIO2 se deben escribir los bits 8-6 con el valor correspondiente ($b001$), para el caso una configuración como salida).

000 = GPIO Pin 9 is an input
001 = GPIO Pin 9 is an output
100 = GPIO Pin 9 takes alternate function 0
101 = GPIO Pin 9 takes alternate function 1
110 = GPIO Pin 9 takes alternate function 2
111 = GPIO Pin 9 takes alternate function 3
011 = GPIO Pin 9 takes alternate function 4
010 = GPIO Pin 9 takes alternate function 5

Figura 2: Selección de funciones para un GPIO en registro GFPSEL0

3.2. Registro de establecimiento de salida - GPSETn

El registro GPSET0 permite escribir un 1 lógico a un GPIO específico. Para esto, debe escribirse un valor de 1 al bit correspondiente al GPIO en el registro. Por ejemplo, para el GPIO2 debe escribirse un valor de 1 en la posición 2 del registro (empezando en 0), es decir un valor de 0...00100 (0x04 en hexadecimal). El registro GPSET0 se encuentra en el *offset* 0x1C.

3.3. Registro de limpiado de salida - GPCLRn

Similar al caso anterior, el registro GPCLR0 permite escribir un 0 lógico a un GPIO específico. Para esto, debe escribirse un valor de 1 al bit correspondiente al GPIO en el registro. Por ejemplo, para el GPIO2 debe escribirse un valor de 1 en la posición 2 del registro (empezando

en 0), es decir un valor de 0...00100 (0x04 en hexadecimal). El registro GPCLR0 se encuentra en el *offset* 0x28.

4. "Driver"para módulo GPIO

En esta sección, se realizará la implementación de un *driver* sencillo para el módulo *GPIO*, específicamente para establecer un GPIO como salida y escribir un valor de 0 o 1 a dicho pin. El código del driver y su aplicación se muestra continuación (se encuentra además en el tecDgital):

```
#include <sys/mman.h> //mmap
#include <err.h> //error handling
#include <fcntl.h> //file ops
#include <unistd.h> //usleep

//Static base
static unsigned GPIO_BASE = 0x3f200000;

//Regs pointers
volatile unsigned int * gpfsel0;
volatile unsigned int * gpset0;
volatile unsigned int * gpclr0;

/*Function prototypes*/
void gpioInitPtrs();
void gpioSetMode();
void gpioWrite(unsigned char bit);


//Initialize pointers: performs memory mapping, exits if mapping fails
void gpioInitPtrs(){
  int fd = -1;
  //Loading /dev/mem into a file
  if ((fd = open("/dev/mem", O_RDWR, 0)) == -1)
            err(1, "Error opening /dev/mem");
    //Mapping GPIO base physical address
  gpfsel0 = (unsigned int*)mmap(0, getpagesize(),
             PROT_WRITE, MAP_SHARED, fd, GPIO_BASE);
  //Check for mapping errors
  if (gpfsel0 == MAP_FAILED)
    errx(1, "Error during mapping GPIO");
  //Set regs pointers
  gpset0 = gpfsel0 + 0x7; // offset 0x1C / 4 = 0x7
  gpclr0 = gpfsel0 + 0xA; // offset 0x28 / 4 = 0xA
}


//Sets GPIO2 as output
void gpioSetMode(){
```

```
  *gpfsel0 = *gpfsel0 | 0x40; //0 0100 0000
}


//Writes to GPIO2
void gpioWrite(unsigned char bit){
  if (bit) *gpset0 = 0x4; //sets bit
  else *gpclr0 = 0x4; //clears bit
}


int main(int argc, char const *argv[])
{
  gpioInitPtrs();
  gpioSetMode();

  //toggle GPIO2
  gpioWrite(1);
  usleep(1000000);
  gpioWrite(0);

  return 0;
}
```

En el código, se inicia declarando el macro de la dirección base del dispositivo (GPIO_BASE). Luego se declaran los punteros que serán usados para la configuración de los registros respectivos. La palabra reservada *volatile* le indica al compilador que el contenido del puntero no es estático y puede cambiar sin modificar explícitamente su valor en el programa (como se comporta el hardware).

La función *gpioInitPtrs()* se encarga de inicializar los punteros a los registros, por medio del mapeo de memoria física a memoria virtual del archivo de dispositivo /dev/mem (representado por la variable *fd*). Dicho mapeo se realiza explícitamente en la función *mmap* que toma como parámetros la dirección base (0 en este caso), el rango o tamaño del bloque a mapear (se utiliza la función del sistema *getpagesize()*, para obtener el tamaño de una página de memoria virtual), la bandera de protección de páginas a mapear (PROTWRITE, permite a las páginas ser escritas), el tipo de mapeo (MAPSHARED, permite que el mapeo sea compartido con otros procesos que también mapean esta región) y finalmente el *offset* que se utilizará para calcular la dirección final (en este caso se utiliza GPIO_BASE como *offset* pues se estableció una dirección base para el mapeo de 0).

La función *gpioSetMode()* se encarga de establecer el modo del GPIO como salida. Como se mencionó arriba, esto se realiza escribiendo un valor de $b001$, en los bits 8-6 del registro GPFSEL0. Para no afectar el comportamiento de los demás GPIO, se realiza una operación OR entre el valor actual del registro y un valor de $0x40$, que corresponde a la escritura mencionada en los bits 8-6.

La función *gpioWrite()* escribe un valor de 1 o 0 dependiendo del argumento *bit*. Nótese que la única diferencia está en el registro al que se debe escribir (GPSET0 para un 1 y GPCLR0 para un 0). El valor de $0x4$ corresponde al bit 2 del registro correspondiente, como mencionó anteriormente.

La función *main* se encarga solamente de llamar a las funciones descritas anteriormente, para verificar el funcionamiento del controlador.

5. Compilación

Ya que se está trabajando en espacio de usuario, no es necesario compilar el "driver" contra el kernel. La compilación, por lo tanto, se simplifica; para compilar basta con utilizar el *toolchain* para la tarjeta Raspberry pi 2, construido previamente con Yocto. Por ejemplo:

```
source /opt/poky/2.3.4/environment-setup-cortexa7hf-neon-vfpv4-poky-linux-gnueabi
$CC -o gpio gpio.c
```

6. Verificación

Para verificar el funcionamiento del "driver", se deberá o el binario generado en el paso anterior a una imagen funcional de Linux en Raspberry pi 2. Se deberá conectar un LED al GPIO2 (pin 3 físico de la tarjeta). El LED deberá encenderse por un segundo y luego deberá apagarse.

7. Evaluación

Para la evaluación de este tutorial se deberá crear un driver nuevo de GPIO, que permita: configurar el GPIO en cualquiera de todos los modos disponibles (entrada, salida o funciones especiales), leer y desplegar el modo actual, leer y desplegar el nivel lógico del GPIO, y escribir un valor al GPIO. Esto se debe realizar para al menos 8 GPIOs diferentes, tanto de manera individual (para un GPIO específico) como grupal (para los 8 GPIOs simultáneamente), en el mismo controlador . Deberá abstraer las funcionalidades del controlador en una biblioteca. Además deberá implementar un archivo de prueba que verifique el funcionamiento de la biblioteca del controlador. Debe seguir la estructura estandar (lib, src, etc), así como utilizar algún gestor de compilación (autotools, cmake, etc).

Entrega

Para la entrega, se deberá subir un archivamiento al tecDigital con nombre Nombre_Apellido_-tar.gz, que incluya todos los códigos fuentes, así como los pasos para la compilación.