Técnicas Digitales II

Entrada/Salida de Propósito General (*General Purpose Input/Output* - GPIO)

Dr.Ing.Steiner Guillermo

E/S de propósitos generales (*General-Purpose I/O o GPIO*)

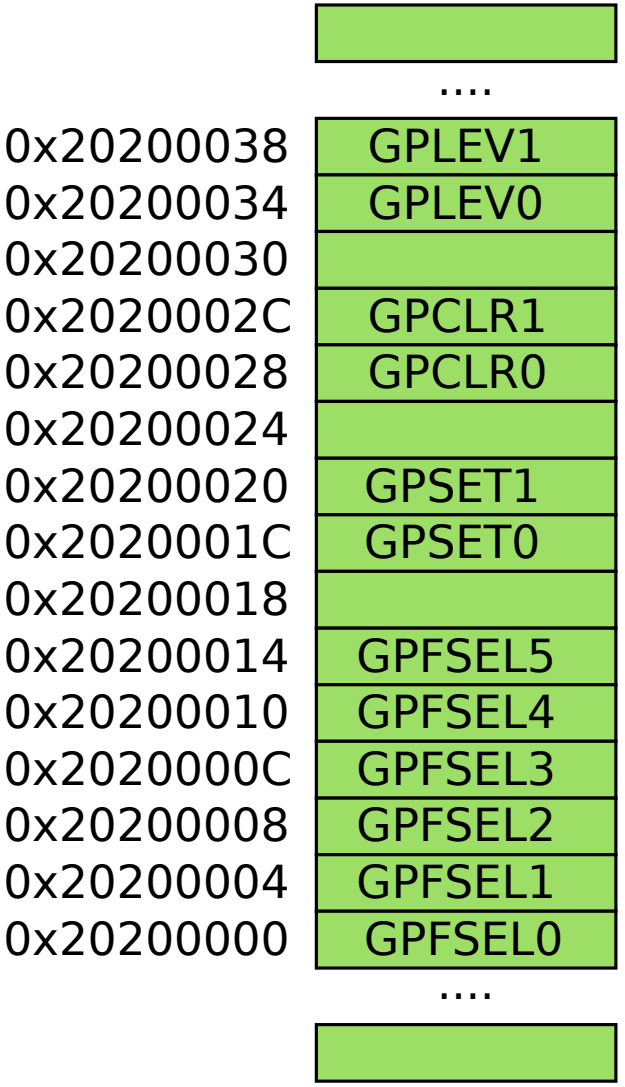
- El BCM2835, posee 54 pines de GPIO.
- El dispositivo GPIO, permite configurar a estos pines de dos formas distintas.
  - Como pines de E/S digital, permitiendo además configurar la dirección (E o S), la lectura de las entradas y escribir los valores de salida.
    Toda la configuración se realiza en la GPIO.
  - Se configura los pines como parte de un dispositivo particular.
    El resto de la configuración se realiza con el dispositivo asignado.

E/S digitales de propósitos generales (*General-Purpose Digital I/O*)

- Es la GPIO funcionando como puerto para leer y escribir señales digitales.
- Este modo, requerirá de registros para realizar las siguientes funciones:
  - Leer los valores en los pines.
  - Escribir los valores de salida en los pines.
  - Configurar el pin como entrada o salida.
  - Si el pin permite configurarse como entrada de interrupción, configurar este uso particular y otros detalles como flanco y otras condiciones de interrupción

E/S de propósitos generales (*General-Purpose I/O*)

- Los 54 pines de la GPIO del BCM2835 poseen 4 registros múltiples para su configuración.
- Se identifican con los siguientes nombres GPFSEL, GPLEV, GPSET y GPCLR.



GPFSEL registro de GPIO

- Es un registro de 6 elementos GPFSEL0 .. 5
- Es el encargado de determinar la función del pin.
- Cada pin tiene asignado 3 bit en este registro que permite determinar su función.
- El pin puede ser seleccionado como entrada o salida de E/S digital o una función específica de un periférico de E/S.

GPFSEL	Función del PIN
000	Entrada
001	Salida
010	ALT5
011	ALT4
100	ALT0
101	ALT1
110	ALT2
111	ALT3

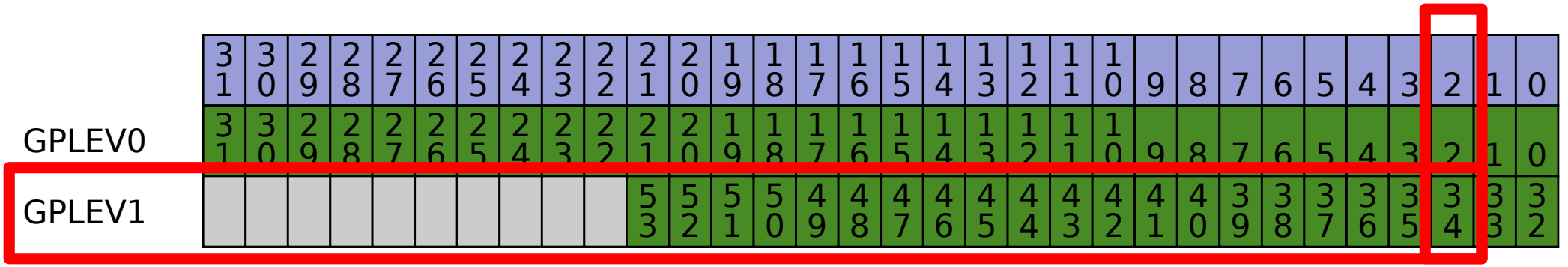
GPFSEL registro de GPIO

- Cada GPSEL controla 10 GPIO, necesitando 6 registro para los 54 GPIO que posee el BCM2835
- Por ejemplo GPIO13 es configurado por GPFSEL1[11:9]

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPFSEL0			GPIO9			GPIO8			GPIO7			GPIO6			GPIO5			GPIO4			GPIO3			GPIO2			GPIO1			GPIO0		
GPFSEL1			GPIO19			GPIO18			GPIO17			GPIO16			GPIO15			GPIO14			GPIO13			GPIO12			GPIO11			GPIO10		
GPFSEL2			GPIO29			GPIO28			GPIO27			GPIO26			GPIO25			GPIO24			GPIO23			GPIO22			GPIO21			GPIO20		
GPFSEL3			GPIO39			GPIO38			GPIO37			GPIO36			GPIO35			GPIO34			GPIO33			GPIO32			GPIO31			GPIO30		
GPFSEL4			GPIO49			GPIO48			GPIO47			GPIO46			GPIO45			GPIO44			GPIO43			GPIO42			GPIO41			GPIO40		
GPFSEL5																					GPIO53			GPIO52			GPIO51			GPIO50		

GPLEV registro de GPIO

- Son dos registros GPLEV1 y GPLEV0 y permiten leer el valor de entrada de los pines.
- Cada pin entonces tiene asignado un bit en uno de estos registros de la siguiente manera:
  - GPIO0..31 con GPLEV0[0..31]
  - GPIO32..53 con GPLEV1[0..21]
- Por ejemplo GPIO34 es leída como GPLEV1[2]



GPSET y GPCLR registros de GPIO

- El registro GPLEV no permite ser escrito.
- Para asignar un valor a la salida se debe recurrir a dos registros.
- Estos registros al igual que GPLEV, poseen 1 bit por cada pin de E/S, y la combinación de ambos permite asignar un estado determinado a cada pin.
- GPSET0..1 permite forzar un pin determinado a 1 escribiendo un 1 en el bit correspondiente de. registro.
- GPCLR0..1 permite forzar un pin determinado a 0 escribiendo un 1 en el bit correspondiente del registro.

Driver para manejo de GPIO

- En la siguiente sección se desarrollará un driver simple para acceder a las E/S digitales y en para configurar en general la GPIO.
- Este driver se llamará EasyPIO
- El driver desarrollado es similar a las librerías open-source WiringPi, compatible con todos los modelos de Raspberry PI

  http://wiringpi.com/

Mapeo de los dispositivos de E/S

- Los driver serán realizados bajo Linux
- Linux utiliza memoria virtual, esto significa que cuando se referencia a un puntero este no indica una dirección física.
- Es necesario entonces solicitar al SO que asigne las direcciones físicas de interés al espacio de direcciones del programa función **pioInit()** .
- El acceso a estas direcciones físicas se realiza a través de **/dev/mem**
- Luego **mmap()** es el encargado de poder usar a la variable gpio como un puntero a la dirección física 0x20200000.

Mapeo de los dispositivos de E/S

```
#include <sys/mman.h>
#define BCM2835_PERI_BASE 0x20000000
#define GPIO_BASE         (BCM2835_PERI_BASE + 0x200000)
volatile unsigned int *gpio; //Pointer to base of gpio
#define GPLEV0            (* (volatile unsigned int *) (gpio + 13))
#define BLOCK_SIZE        (4*1024)

void pioInit(){
   int mem_fd;
   void *reg_map;
   mem_fd = open("/dev/mem", O_RDWR|O_SYNC); // abre /dev/mem
   reg_map = mmap(
      NULL,       // dir. donde comienza el mapeo (null=no importa)
      BLOCK_SIZE,            // 4KB bloque mapeado
      PROT_READ|PROT_WRITE, // permite lect.y escr.en la mem.
      MAP_SHARED,            // acceso no exclusivo
      mem_fd,                // puntero a /dev/mem
      GPIO_BASE);            // offset a la GPIO
   gpio = (volatile unsigned *)reg_map;
   close(mem_fd);
}
```

Volatile

- La palabra clave “volatile” indica que un valor puede cambiar entre diferentes accesos, incluso si no parece ser modificado.
- Esto evita que el compilador optimice lecturas o escrituras con valores obsoletos.

```
int main(void)
{
 while(contador<100){
 }
```

```
int contador;
int main(void)
{
 100: push {lr}
 104: sub sp, sp, #28
 while(contador<100) {
 108: ldr r3, [pc, #40] ; 138
 10c: ldr r3, [r3]
 110: cmp r3, #99 ; 0x63
 114: ble 110
 }
```

```
volatile int contador;
int main(void)
{
 100: push {lr}
 104: sub sp, sp, #28
 while(contador<100) {
 108: ldr r3, [pc, #40] ; 138
 10c: ldr r2, [r3]
 110: cmp r2, #99 ; 0x63
 114: ble 10c
 }
```

Driver Easy PIO

- Luego de mampear el dispositivo, se deben escribir las funciones del driver.
- Debido a que se utilizan registro múltiples para controlar las E/S, las funciones deben calcular que registro debe utilizarse y que desplazamiento dentro de el aplicar.
- Se definen una serie de constantes utilizadas luego en las funciones

```
#define GPFSEL      ((volatile unsigned int *) (gpio + 0))
#define GPSET       ((volatile unsigned int *) (gpio + 7))
#define GPCLR       ((volatile unsigned int *) (gpio + 10))
#define GPLEV       ((volatile unsigned int *) (gpio + 13))
#define INPUT   0
#define OUTPUT 1
```

Driver Easy PIO - pinMode

- Esta función, establece el modo del pin, donde
  - **pin** es el numero del puerto.
  - **funcion** es el modo de funcionamiento.
- La función configura el registro multiple GPFSEL0..5
- Debe establecer que nro de registro usar (1)
- Y que conjunto de bits (3 por cada pin) (2)

```
void pinMode(int pin, int function) {
   int reg = pin/10; (1)
   int offset = (pin%10)*3; (2)
   GPFSEL[reg] & = ~((0b111 & ~function) << offset);
   GPFSEL[reg] | = ((0b111 & function) << offset);
}
```

Driver Easy PIO - digitalWrite

- Esta función, permite escribir en un pin de salida.
  - **pin** es el numero del puerto.
  - **val** es el valor de salida (1 o 0).
- La función utiliza los registros multiples GPSET0,1 si se debe asignar un 1 a la salida o GPCLR0,1 en caso contrario.

```
void digitalWrite(int pin, int val) {
   int reg = pin / 32;
   int offset = pin % 32;
   if (val)    GPSET[reg] = 1 << offset;
   else        GPCLR[reg] = 1 << offset;
}
```

Driver Easy PIO - digitalRead

- Esta función, permite leer un pin determinado.
  - **pin** es el numero del puerto.
  - **retorno** es el valor de leído del puerto.
- La función utiliza los registros múltiples GPLEV0,1.
- El valor de retorno será 0 o 1.

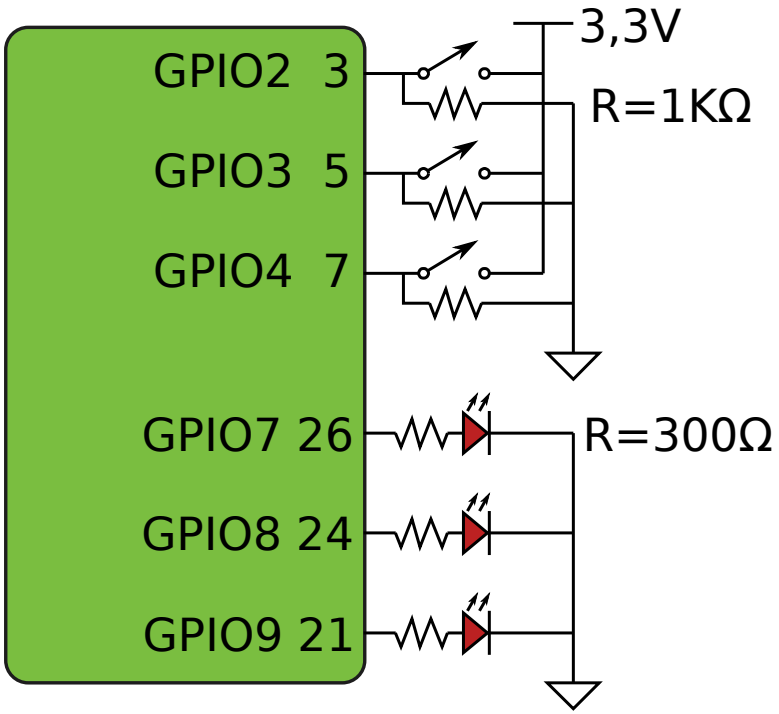
```
int digitalRead(int pin) {
   int reg = pin / 32;
   int offset = pin % 32;
   return (GPLEV[reg] >> offset) & 0x00000001;
}
```

Ejemplo del uso de GPIO

- Como ejemplo del uso del EasyPIO, se manejará un hardware con los dispositivos definidos en el recuadro.
- No hay especificaciones de niveles lógicos y corriente para el BCM2835, se realizarán las siguientes consideraciones:
  - Imax por puerto 16mA
    Imax para todos los pines de E/S 50mA
  - E/S compatible con 3,3 V no tolerante a 5V

3 llaves (entradas) llaves abiertas, la resistencia a gnd pone 0 a la entrada, llave cerrada 1.

3 leds (salidas) un 1 a la salida enciende el led a través de una resistencia limitadora.



Driver Easy PIO – ejemplo con el Hardware

- Ejemplo del uso del driver.

```
#include "EasyPIO.h"
void main(void) {
    pioInit();
    // Set GPIO 4:2 como entradas (llaves)
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    pinMode(4, INPUT);
    // Set GPIO 9:7 como salida (leds)
    pinMode(7, OUTPUT);
    pinMode(8, OUTPUT);
    pinMode(9, OUTPUT);
    while (1) { // Bucle, lee llaves y enciende led si esta cerrada
        digitalWrite(7, digitalRead(2));
        digitalWrite(8, digitalRead(3));
        digitalWrite(9, digitalRead(4));
    }
}
```

- Bibliografía

  Harris & Harris. Digital design and computer architecture: ARM edition. Elsevier, 2015. Capítulo 9.

¿ Preguntas ?