Apellidos	Nombre

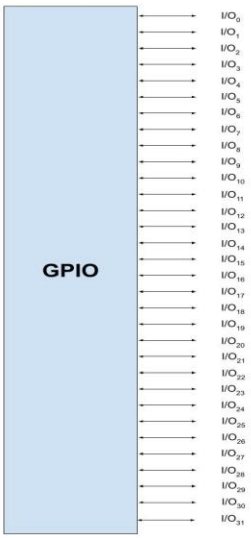
Universidad de Alcalá

Dpto Automática. 31-03-2023

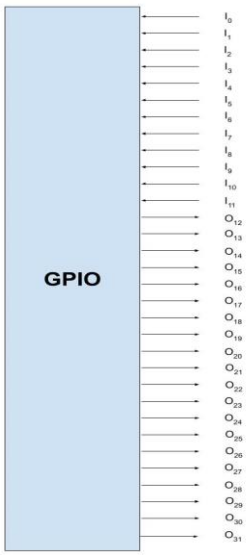
Examen Prácticas 2 y 3 - Sistemas Empotrados

Responder a las siguientes preguntas escribiendo el código necesario para implementar lo que se propone en cada caso.

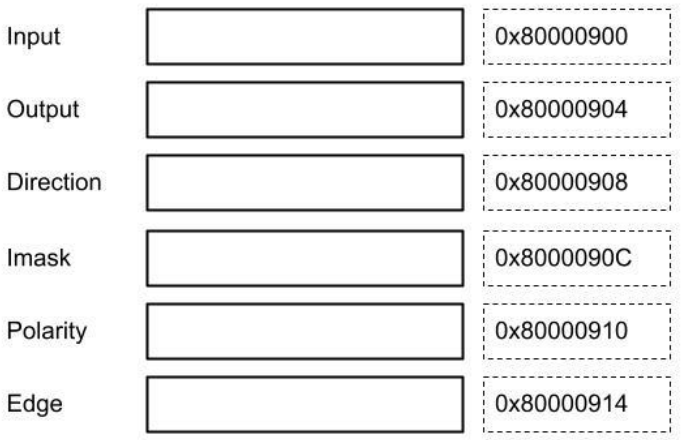
1) Un dispositivo GPIO integrado en *System on Chip* (SOC) permite controlar de forma individual un conjunto de señales del sistema. En la siguiente figura podemos ver un esquema de un GPIO capaz de control 32 señales.



Cada una de las 32 señales del dispositivo GPIO puede ser configurada como entrada o como salida, de acuerdo a las necesidades del sistema en el que se va a utilizar el SOC. En la figura de la derecha podemos ver una configuración en la que las primeras 12 señales han sido configuradas como entradas y el resto como salidas.



El controlador del GPIO integrado en el SOC basado en LEON3 cuenta con un conjunto de 6 registros, de 32 bits cada uno, ubicados en posiciones de memoria consecutivas tal como se representa en la siguiente figura.



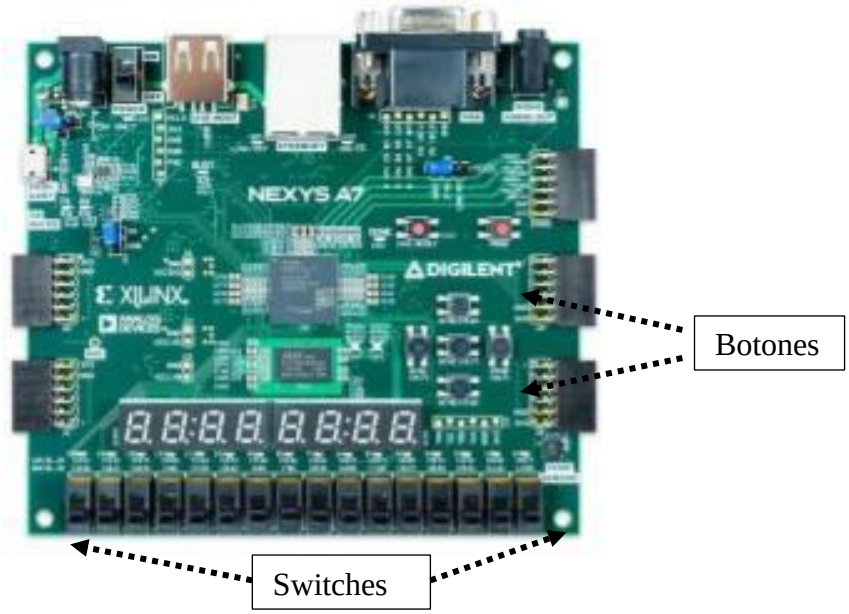
- El registro **Input** está en la dirección 0x80000900 y permite leer el valor de una señal configurada como entrada.
- El registro **Output** está en la dirección 0x80000904 y permite fijar el valor de una señal configurada como salida.
- El registro **Direction** está en la dirección 0x80000908 y permite determinar si una señal se configura como entrada o como salida.
- El registro **Imask** está en la dirección 0x8000090C y permite determinar si se enmascara o no la interrupción asociada a una señal.
- Los registros **Polarity y Edge** están, respectivamente, en la dirección 0x80000910 y 0x80000914, y permite determinar cómo se van a disparar las interrupciones asociadas a las señales, si por nivel o por flanco, y qué nivel o flanco será el seleccionado.

Declarad la estructura `struct GPIO_regs`, junto con la variable `pGPIO_REGS`, que facilita el acceso a estos registros. **(1.0 punto)**

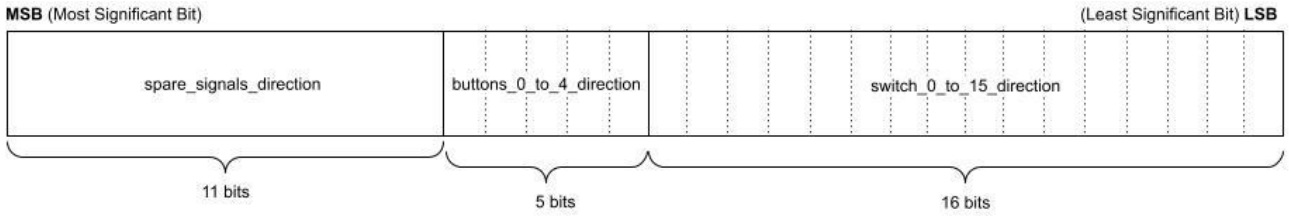
Respuesta:

```
struct GPIO_regs {    // COMPLETAD …

 volatile uint32_t Input;
 volatile uint32_t Output;
 volatile uint32_t Direction;
 volatile uint32_t Imask;
 volatile uint32_t Polarity;
 volatile uint32_t Edge;

 };
   // COMPLETAD
struct GPIO_regs * const pGPIO_REGS = (struct GPIO_regs * ) 0x80000900;
```

2) El SOC sintetizado en la FPGA de la tarjeta Nexys A7 integra este controlador GPIO para acceder a los 5 botones y 16 interruptores de los que dispone y cuya ubicación se muestra en la figura



Para controlar que queden configuradas como entradas de las señales conectadas a estos 21 elementos (5 botones y 16 interruptores) **es necesario poner a 0** el valor de los primeros 21 bits del registro **Direction** de acuerdo a la disposición que se muestra en la siguiente figura:



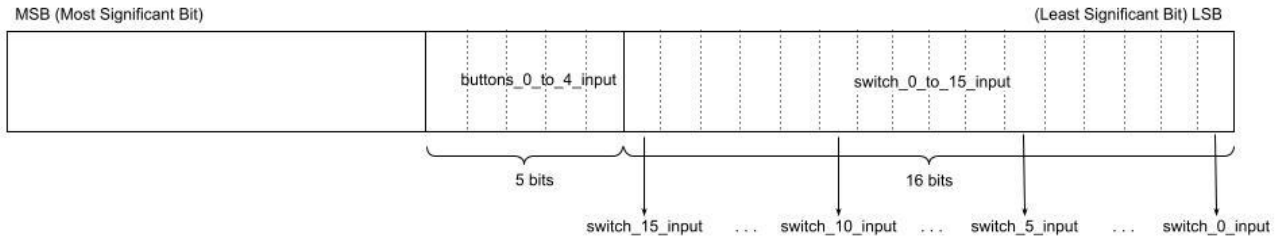
Nexys GPIO Direction Register

Teniendo en cuenta esta configuración, completad la función `nexys_set_buttons_switch_direction` para que ponga a 0 el valor de los primeros 21 bits del registro **Direction sin cambiar el resto de los bits del registro** (los etiquetados como spare_signals_direction en la figura). **(1.0 punto)**

Respuesta:

```
void nexys_set_buttons_switch_direction(){

      // ⇓  COMPLETAD  ⇓

      pGPIO_REGS->Direction &= ~0x1FFFFF;

}
```

3) Para averiguar el estado de un interruptor controlado por la GPIO es necesario leer el valor que tiene su bit asociado en el registro **Input.** De acuerdo a la siguiente figura, los 16 bits de menor peso del registro **Input** dan acceso al valor que indica el estado de los 16 interruptores.



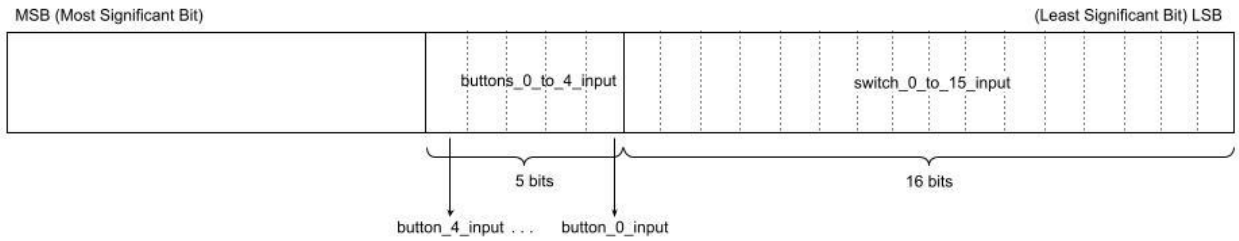
Nexys GPIO Input Register (Switch input assignation)

Teniendo en cuenta esta configuración, completad la función `nexys_get_switch_state` que recibe como parámetro el identificador del interruptor (`switch_id`, que estará entre 0 y 15) y devuelve un 0 si el bit correspondiente al Input de ese interruptor está a 0, y **un valor distinto de 0** en caso contrario. **(1.0 punto)**

Respuesta:

```
uint8_t nexys_get_switch_state(uint8_t switch_id){ //asumir que switch_id < 16

      // ⇓ COMPLETAD ⇓
      uint32_t bit_mask;
      bit_mask = (0x1 << switch_id);
      return (pGPIO_REGS->Input & bit_mask);

}
```

4) Completad, además, la función análoga a la anterior, de nombre `nexys_get_button_state`, que determina el estado de los botones. Esta función recibe como parámetro el identificador del botón (`button_id`, que estará entre 0 y 4) y devuelve un 0 si el bit correspondiente a ese botón está a 0, y devuelve un valor distinto de 0 en caso contrario. Tened en cuenta que la ubicación de los bits asociados a los 5 botones en el registro Input está de acuerdo a la siguiente figura.
(1.0 punto)

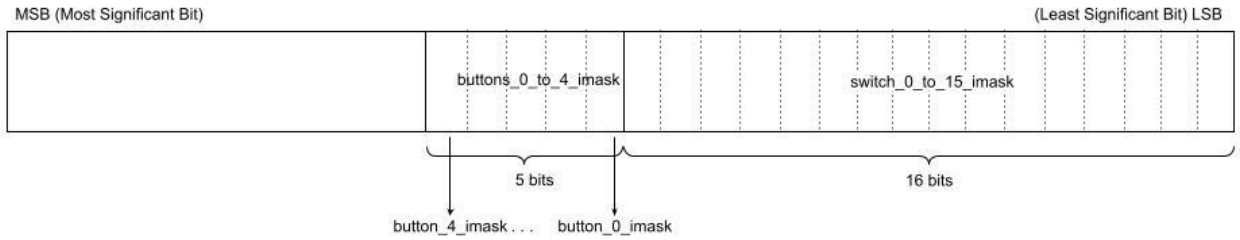


Nexys GPIO Input Register (Button input assignation)

Respuesta:

```
uint8_t nexys_get_button_state(uint8_t button_id){ //asumir que button_id < 5

      // ⇓ COMPLETAD ⇓
      uint32_t bit_mask;
      bit_mask = (0x1 << (button_id + 16));
      return (pGPIO_REGS->Input & bit_mask);


}
```

5) Para poder controlar la máscara de interrupción de cada una de las señales asociadas a los botones es necesario configurar correctamente el registro **Imask** de acuerdo a la organización que se muestra en la siguiente figura.



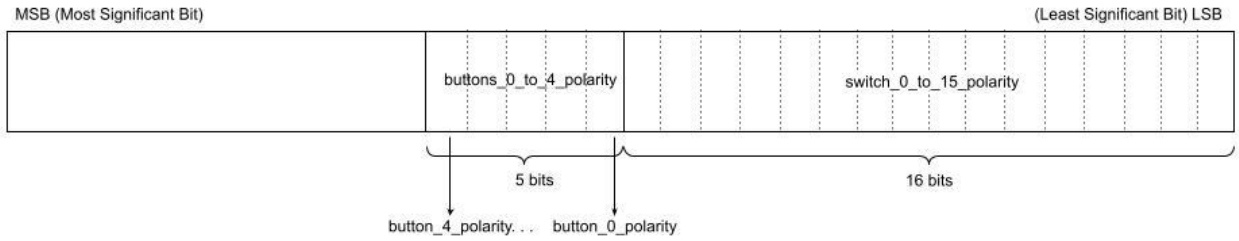
Nexys GPIO Imask Register (Button Imask assignation)

Para enmascarar la interrupción asociada a un botón es necesario que el bit imask asignado el botón en el registro **Imask** esté a 0, estando habilitada la interrupción si el bit tiene el valor 1. De acuerdo a este comportamiento, implementad las funciones `nexys_disable_button_irq` `nexys_enable_button_irq` que reciben como parámetro el identificador del botón (`button_id`) cuya interrupción se desea deshabilitar o habilitar. **(1.0 puntos)**

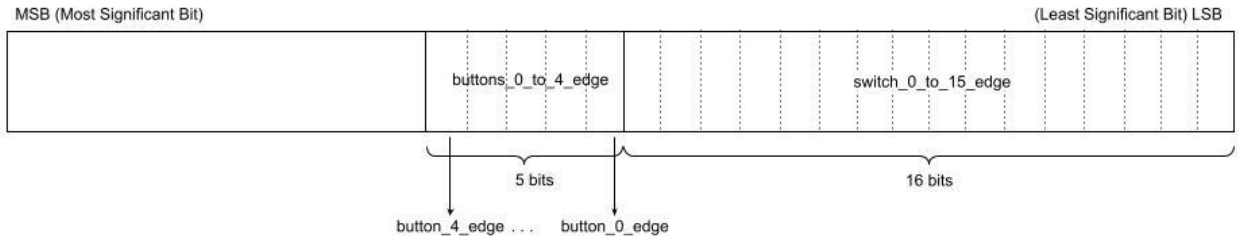
Respuesta:

```
void nexys_disable_button_irq(uint8_t button_id){ //asumir que button_id < 5

      // ⇓ COMPLETAD ⇓
      uint32_t bit_mask;
      bit_mask = (0x1 << (button_id + 16));

      pGPIO_REGS->Imask &= ~bit_mask;



}


void nexys_enable_button_irq(uint8_t button_id){ //asumir que button_id < 5

      // ⇓ COMPLETAD ⇓
      uint32_t bit_mask;
      bit_mask = (0x1 << (button_id + 16));

      pGPIO_REGS->Imask |= bit_mask;



}
```

6) Implementar, también, la función `nexys_config_button_irq` que fija el valor de los bits que controlan la polaridad y el edge de las interrupciones de los botones.



Nexys GPIO Polarity Register (Button Polarity assignation)



Nexys GPIO Edge Register (Button Edge assignation)

La función `nexys_config_button_irq` recibe como parámetro el identificador del botón (`button_id`) y el valor que deben tener el bit de los registros Edge y Polarity asociados al botón. **(1.5 puntos)**

```
     //asumir que está controlado que button_id es < 5

void nexys_config_button_irq(uint8_t button_id,
                             uint8_t polarity,
                             uint8_t edge){

     // ⇓  COMPLETAD  ⇓
     uint32_t bit_mask= 0x01 << (button_id + 16);

     if(polarity){ //Poned a 1 sólo el bit polarity correspondiente al button_id


        pGPIO_REGS->Polarity|= bit_mask;


     } else { //Poned a 0 sólo el bit polarity correspondiente al button_id


        pGPIO_REGS->Polarity&=~bit_mask;

     }
```

```
        if(edge){ //Poned a 1 sólo el bit edge correspondiente al button_id
            pGPIO_REGS->Edge|= bit_mask;

        } else { //Poned a 0 sólo el bit edge correspondiente al button_id
            pGPIO_REGS->Edge&=~bit_mask;

        }
}
```

7) Finalmente, se va a implementar un sistema que controle con dos botones la velocidad de un vehículo. Uno de los botones (el de identificador 4) aumenta en una unidad el valor de la velocidad (definido mediante una variable global `vehicle_speed`), mientras que otro botón (de identificador 0) la disminuye. El bucle del programa principal (**ya implementado**) aplicara el valor actual de esa velocidad con una llamada función `uah_vehicle_set_speed(vehicle_speed)`;

Teniendo en cuenta que la pulsación de un botón**, independientemente del botón que se pulse, tiene asociada la interrupción externa de nivel 4, y que la función denominada** `buttons_irq_handler`, **que controla el valor de** `vehicle_speed`, **y que hay que instalar como manejador de la interrupción, ya está implementada,** completad en el programa principal **toda la configuración del SOC necesaria para que el sistema trabaje correctamente.**

Al igual que se ha realizado en la práctica 3, la configuración se debe aplicar antes de alcanzar el bucle del programa principal, de forma que el sistema esté completamente configurado antes de ejecutarse la sentencia `do{ }while(1);`

La configuración deberá hacerse de acuerdo a los siguientes puntos:

- Configurar la dirección de las señales de la GPIO asociadas a los botones e interruptores, utilizando la función `nexys_set_buttons_switch_direction`
- Configurar el controlador GPIO para que las interrupciones asociadas a los botones 0 y 4 tengan el bit de Polarity = 1 y el bits de Edge = 0
- Configurar el controlador GPIO para que estén habilitadas las interrupciones de los botones 0 y 4.
- Instalar la función `buttons_irq_handler` como manejador de la interrupción externa de nivel 4.

Aplicar esta configuración con las interrupciones deshabilitadas y enmascaradas, tal como se realiza en la práctica 3, y una vez esté aplicada, desenmascarar la interrupción externa asociada a los botones, y habilitar de nuevo todas las interrupciones antes de dar paso al bucle del programa principal **(3.5 puntos)**

Respuesta:

```
uint8_t vehicle_speed;

void buttons_irq_handler(void){

	if (nexys_get_button_state(4)){
		vehicle_speed++;
	}

	if (nexys_get_button_state(0)){
		vehicle_speed--;
	}
}

int main()
{

	//Instalar como manejador del trap 0x83 la rutina
	// que habilita las interrupciones
	leon3_set_trap_handler(0x83,leon3_trap_handler_enable_irqs);

	//Instalar el manejador del trap que 0x83 la rutina
	// que deshabilita las interrupciones
	leon3_set_trap_handler(0x84,leon3_trap_handler_disable_irqs);

	//COMPLETAD CONFIGURACIÓN

	//Deshabilitar las interrupciones
	leon3_sys_call_disable_irqs();

	//Enmascarar todas las interrupciones, sólo desenmascaremos aquellas
	//que tenemos manejadas.
	leon3_mask_all_irqs();

	//Inicializar nexys
	nexys_set_buttons_switch_direction();

	//Configurar Polarity y Edge de las interrupciones de los botones 0 y 4
	nexys_config_button_irq(0, 1, 0);
	nexys_config_button_irq(4, 1, 0);

	//Habilita las interrupciones de los botones 0 y 4
	nexys_enable_button_irq(0);
	nexys_enable_button_irq(4);

	//Instalar la función button_handler como
	// manejador de usuario de la interrupción de nivel 4
	leon3_install_user_hw_irq_handler(4, buttons_irq_handler);

	//Desenmascarar la interrupción de nivel 4
	leon3_unmask_irq(4);

	//Habilitar las interrupciones
	leon3_sys_call_enable_irqs();




	//FIN COMPLETAD CONFIGURACIÓN
```

```
	velocity=60;
	//bucle principal
	do{
		uah_vehicle_set_speed(vehicle_speed);
	}while(1);
	return 0;
}
```