Apellidos	Nombre

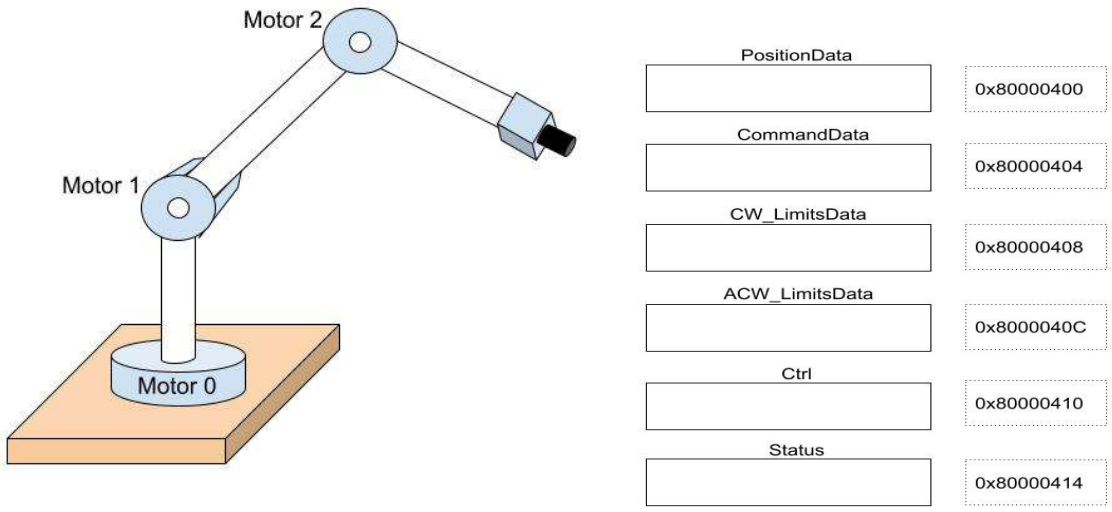
Universidad de Alcalá

Dpto Automática. 01-04-2022

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.

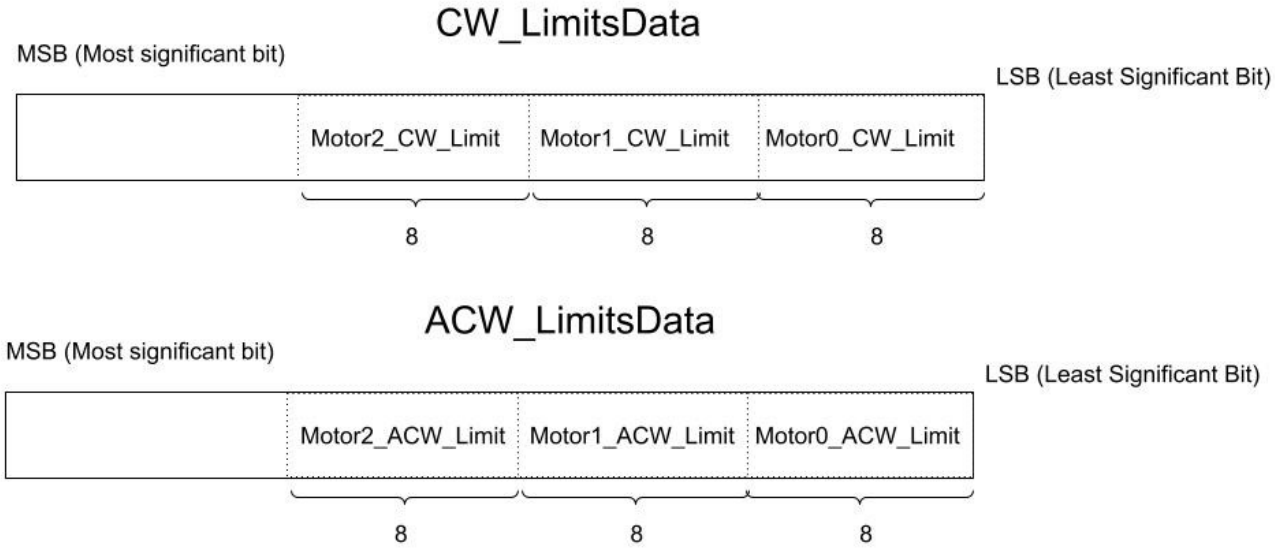
1) Se ha diseñado un brazo robot controlado por tres motores paso a paso. Para controlar esos tres motores se ha diseñado un *System on Chip* que integra, entre otros elementos, un LEON3 como CPU y **un controlador asociado a los 3 motores**. El controlador tiene ubicado el registro **PositionData** en la dirección 0x80000400, el registro **CommandData** en la dirección 0x80000404, el registro **CW_LimitsData** en la dirección 0x80000408, el registro **ACW_LimitsData** en la dirección 0x8000040C, el registro **Ctrl** en la dirección 0x80000410 y el registro **Status** en la dirección 0x80000410. Declarad la estructura `struct ROB_ARM_regs`, junto con la variable `pROB_ARM_REGS` que facilita el acceso a estos registros. **(1 punto)**



Respuesta:

```
struct ROB_ARM_regs {    // COMPLETAD …

 volatile uint32_t PositionData;
 volatile uint32_t CommandData;
 volatile uint32_t CW_LimitsData;
 volatile uint32_t ACW_LimitsData;
 volatile uint32_t Ctrl;
 volatile uint32_t Status;

 };
                                                  // COMPLETAD
struct ROB_ARM_regs * const pROB_ARM_REGS = (struct ROB_ARM_regs * ) 0x80000400
```

2)Los registros **CW_LimitsData y ACW_LimitsData** del controlador permiten establecer, respectivamente, los límites absolutos de giro de los motores en sentido del reloj (CW ClockWise) y en sentido contrario (ACW Anti ClockWise):



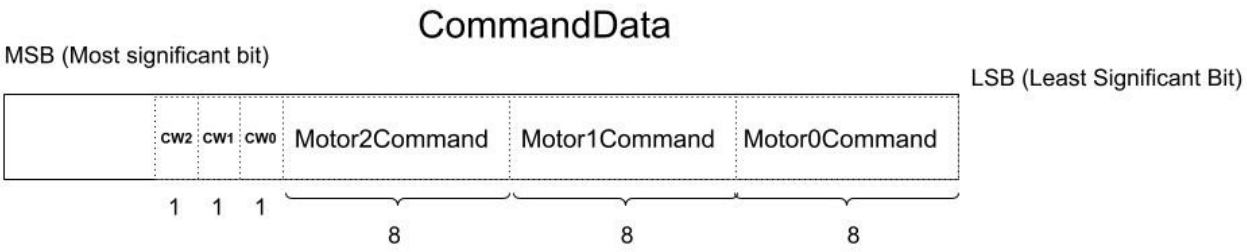
Cada uno de los registros tiene 3 campos de 8 bits destinados a determinar los límites, respecto a un 0 de referencia, de cada uno de los motores en el sentido correspondiente. Teniendo en cuenta la ubicación del campo asociado a cada motor dentro del registro, completad las funciones rob_arm_set_cw_limits y rob_arm_set_acw_limits que fijan, respectivamente, el valor de los límites para cada motor en el sentido del reloj y en el contrario. **(1.5 puntos)**

Respuesta:

```
uint8_t rob_arm_set_cw_limits(uint8_t motor0_cw_limit, uint8_t motor1_cw_limit,
                              uint8_t motor2_cw_limit){

      uint8_t error=0;
      if(motor0_cw_limit > MOTOR0_MAX_LIMIT)
         error|=0x1;
      if(motor1_cw_limit > MOTOR1_MAX_LIMIT)
         error|=0x2;
      if(motor2_cw_limit > MOTOR2_MAX_LIMIT)
         error|=0x4;

      if(!error){ //⇓ COMPLETAD ⇓

        pROB_ARM_REGS->CW_LimitsData= motor0_cw_limit;
        pROB_ARM_REGS->CW_LimitsData|=(motor1_cw_limit<<8);
        pROB_ARM_REGS->CW_LimitsData|=(motor2_cw_limit<<16);

      }
      return error;

}
```

```
uint8_t rob_arm_set_acw_limits(uint8_t motor0_acw_limit,
                               uint8_t motor1_acw_limit,
                               uint8_t motor2_acw_limit){

      uint8_t error=0;
      if(motor0_acw_limit > MOTOR0_MAX_LIMIT)
          error|=0x1;
      if(motor1_acw_limit > MOTOR1_MAX_LIMIT)
          error|=0x2;
      if(motor2_acw_limit > MOTOR2_MAX_LIMIT)
          error|=0x4;

      if(!error){ //⇓ COMPLETAD ⇓

         pROB_ARM_REGS->ACW_LimitsData= motor0_acw_limit;
         pROB_ARM_REGS->ACW_LimitsData|=(motor1_acw_limit<<8);
         pROB_ARM_REGS->ACW_LimitsData|=(motor2_acw_limit<<16);

      }

      return error;

}
```

3)El registro **CommandData** está formado por 6 campos que permiten comandar una posición absoluta a alcanzar por cada motor respecto al 0 de referencia. Los tres primeros campos, de 8 bits, denominados Motor0Command, Motor1Command y Motor2Command determinan el valor del giro que se quiere comandar, mientras que los 3 bits útiles más significativos corresponden a los campos CW2, CW1 y CW0, que determinan el sentido del giro que se comanda, siendo un 1 el valor que indica un giro en el sentido de las agujas del reloj, mientras que un 0 indica un giro en el sentido contrario.



Para manejar el contenido de este registro se ha definido una función denominada `rob_arm_set_command_data` que permite comandar la posición de **uno de los motores** (modificando los campos asociados al mismo) **SIN CAMBIAR LOS CAMPOS ASOCIADOS AL RESTO DE MOTORES**.

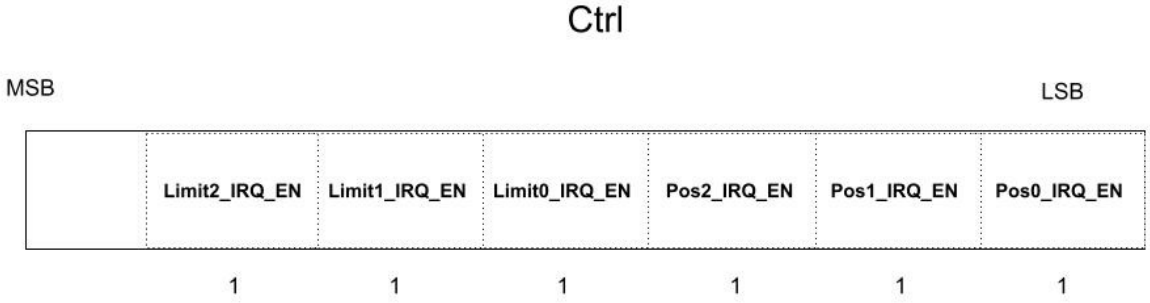
La función tiene tres parámetros:

- `motor_id`: parámetro que identifica el motor cuya posición se quiere comandar. Tomará valores 0,1 o 2 en función del motor elegido.
- `isCW`: valdrá 1 si se desea un giro en sentido de las agujas del reloj (ClockWise), y 0 si se desea un giro en sentido contrario.
- `motor_position`: parámetro que define el valor que se va a escribir en el campo MotorXCommand, siendo X el identificador `motor_id` pasado a través del parámetro

Completad el código de la siguiente función `rob_arm_set_command_data` que comanda la posición del motor cuyo identificador es `motor_id`, **SIN CAMBIAR LOS CAMPOS ASOCIADOS AL RESTO DE MOTORES** **(2.0 puntos)**

```
uint8_t rob_arm_set_command_data(uint8_t motor_id, uint8_t isCW,
                                 uint8_t motor_position){

      uint8_t error=0;

      if (motor_id>2)
        error=0x1;
      else{

           uint32_t CommandDataNextValue;

           CommandDataNextValue = pROB_ARM_REGS->CommandData; // valor actual

          // ⇓ COMPLETAD definiendo el valor que tiene que tomar CommandDataNextValue ⇓

           //Escribir 0x00 sólo en el campo MotorXCommand siendo X=motor_id
           CommandDataNextValue &= ~(0xFF<<(8*motor_id));

           //Escribir sólo el campo MotorXCommand siendo X=motor_id
           CommandDataNextValue |= motor_position<<(8*motor_id);

           //Fijar el campo CWX siendo X=motor_id

           if(isCW)
              CommandDataNextValue |= (1 << (24 + motor_id));
           else
              CommandDataNextValue &= ~(1 << (24 + motor_id));


          // FIN COMPLETAD

           pROB_ARM_REGS->CommandData=CommandDataNextValue; //Valor actualizado
      }
      return error;

}
```

4)El registro **Ctrl**, por su parte, está formado por 6 campos de un bit que permiten activar una interrupción al alcanzar cualquiera de los dos límites fijados para cada motor, y también cuando se alcanza la posición indicada en el registro CommandData. La siguiente figura y descripción aclaran la ubicación y el propósito de cada campo.



- **Limit2_IRQ_EN** Con valor 1 se dispara una interrupción 8 al alcanzar el motor 2 un límite de giro
- **Limit1_IRQ_EN** Con valor 1 se dispara una interrupción 8 al alcanzar el motor 1 un límite de giro
- **Limit0_IRQ_EN** Con valor 1 se dispara una interrupción 8 al alcanzar el motor 0 un límite de giro
- **Pos2_IRQ_EN** Con valor 1 se dispara una interrupción 9 al alcanzar el motor 2 la posición comandada
- **Pos1_IRQ_EN** Con valor 1 se dispara una interrupción 9 al alcanzar el motor 1 la posición comandada
- **Pos0_IRQ_EN** Con valor 1 se dispara una interrupción 9 al alcanzar el motor 0 la posición comandada

Los campos **Limit2_IRQ_EN**, **Limit1_IRQ_EN** y **Limit0_IRQ_EN** controlan la habilitación de la interrupción al alcanzar los límites correspondientes a los valores definidos en los registros CW_LimitData y ACW_LimitData.

Los campos **Pos2_IRQ_EN**, **Pos1_IRQ_EN** y **Pos0_IRQ_EN,** por su parte, controlan la habilitación de la interrupción al alcanzar cada motor la posición objetivo comandada en el registro CommandData.

Para poder establecer de forma independiente la habilitación y deshabilitación de estas interrupciones se han definido dos conjuntos de funciones que, en cada caso, ponen a 1 (para habilitar) o a 0 (para deshabilitar) **SOLO el bit** que corresponde al control de la interrupción asociada al motor que se pasa a través del parámetro `motor_id`. Completar el código de las cuatro funciones `rob_arm_enable_limit_irq`, `rob_arm_disable_limit_irq`, `rob_arm_enable_pos_irq` y `rob_arm_disable_pos_irq` que hacen posible el control de la habilitación o deshabilitación de las interrupciones en cada caso.

(2.0 puntos)

```
uint8_t rob_arm_enable_limit_irq(uint8_t motor_id){

      uint8_t error=0;

      if (motor_id>2)
         error=0x1;
      else { //COMPLETAD poniendo a 1 SOLAMENTE el bit LimitX_IRQ_EN, siendo X=motor_id

         pROB_ARM_REGS->Ctrl |= (1<<(motor_id+3));

      }
      return error;
}

uint8_t rob_arm_disable_limit_irq(uint8_t motor_id){

      uint8_t error=0;

      if (motor_id>2)
         error=0x1;
      else { //COMPLETAD poniendo a 0 SOLAMENTE el bit LimitX_IRQ_EN, siendo X=motor_id

         pROB_ARM_REGS->Ctrl &= ~(1<<(motor_id+3));

      }
      return error;
}

uint8_t rob_arm_enable_pos_irq(uint8_t motor_id){

      uint8_t error=0;

      if (motor_id>2)
         error=0x1;
      else { //COMPLETAD poniendo a 1 SOLAMENTE el bit PosX_IRQ_EN, siendo X=motor_id

         pROB_ARM_REGS->Ctrl |= (1<<(motor_id));

      }
      return error;
}

uint8_t rob_arm_disable_pos_irq(uint8_t motor_id){


      uint8_t error=0;

      if (motor_id>2)
         error=0x1;
      else { //COMPLETAD poniendo a 0 SOLAMENTE el bit PosX_IRQ_EN, siendo X=motor_id


         pROB_ARM_REGS->Ctrl &= ~(1<<motor_id);

      }
      return error;
}
```

5) Finalmente, y teniendo en cuenta que el controlador **genera la interrupción externa de nivel 8 cuando el motor X alcanza uno de los límites** (siempre que tenga a 1 el correspondiente bit LimitX_IRQ_EN), y **la interrupción externa de nivel 9 cuando el motor X alcanza la posición comandada** (siempre que tenga a 1 el correspondiente bit PosX_IRQ_EN), configurar el SoC de acuerdo a los siguientes puntos:

- Configurar los límites de los motores 0, 1 y 2 para el giro Clock Wise con los valores `180,120,90`.
- Configurar los límites de los motores 0, 1 y 2 para el giro Anti Clock Wise con los valores `90,120,90`.
- Configurar el sistema para que se ejecute la función `rob_arm_motor0_limit_handler` (ya implementada) cuando la posición del motor 0 alcanza cualquiera de sus límites. (Esta función no debe ejecutarse cuando el límite es alcanzado por los motores 1 y 2)
- Configurar el sistema para que se ejecute la función `rob_arm_motor2_position_handler` (ya implementada) cuándo el motor 2 alcanza la posición comandada (Esta función no debe ejecutarse si la posición comandada es alcanzada por los motores 0 y 1).

Para aplicar esta configuración completad la función `main` que se da a continuación utilizando tanto las funciones definidas en los apartados anteriores del examen como las funciones de instalación y configuración de las interrupciones que utilizaste en la práctica 3.

(3.5 puntos)

Respuesta:

```
void rob_arm_motor0_limit_handler(void){

  printf("motor 0 limit\n");
}

void rob_arm_motor2_position_handler(void){

  printf("motor 2 position reached\n");
}

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);

      //Deshabilitar las interrupciones
      leon3_sys_call_disable_irqs();

      //Enmascarar todas las interrupciones
      leon3_mask_all_irqs();
```

```
      //COMPLETAD

      //Instalar manejadores de usuario de la interrupción de nivel 8
      leon3_install_user_hw_irq_handler(8, rob_arm_motor0_limit_handler);
      //Instalar manejadores de usuario de la interrupción de nivel 9
      leon3_install_user_hw_irq_handler(9, rob_arm_motor2_position_handler);

      //Configuro límites
      rob_arm_set_cw_limits(180,120,90);
      rob_arm_set_acw_limits(90,120,90);

      //Habilito interrupción cuando se violan límites del motor 0
      rob_arm_enable_limit_irq(0);
      //Deshabilito interrupción cuando se violan límites de motor 1 y motor 2
      rob_arm_disable_limit_irq(1);
      rob_arm_disable_limit_irq(2);

      //Habilito interrupción cuando se alcanza posición en el motor 2
      rob_arm_enable_pos_irq(2);

      //Deshabilito interrupción cuando se alcanza posición en motor 0 y motor 1
      rob_arm_disable_pos_irq(0);
      rob_arm_disable_pos_irq(1);

      //Desenmascarar interrupcioines de nivel 8 y 9
      leon3_unmask_irq(8);
      leon3_unmask_irq(9);

      //FIN COMPLETAD

//Habilitar las interrupciones
leon3_sys_call_enable_irqs();

      do{


            // En este código se usaría repetidamente rob_arm_set_command_data( )
            // para determinar las posiciones a comandar durante la operación


        . . .

      }while(1);

      return 0;
}
```