PSEUDO-CÓDIGO DE LOS ALGORITMOS DESARROLLADOS

Apartado (1/3): Entrada del pulsador detectada por flanco

FICHERO "d_Boton"

Objetivo: Programar un driver sencillo que detecta una pulsación de un botón haciendo encuesta continua sobre el bit correspondiente del registro de eventos.

➔ Función "Ini_Boton": Prepara un botón pasado como parámetro para empezar a detectar las pulsaciones. No retorna. Subrutina pública (accesible desde programas externos).

Guardo registros y link register en la pila. Inicia subrutina.

si (botón = 1) entonces: #terminal = 10

si (botón = 2) entonces: #terminal = 9

prog_gpio (#TERMINAL, 0)

progResPull_gpio (#TERMINAL, 0)

habilitar detección de eventos de flanco ascendiente: bit del terminal del botón a 1 en el registro GPEDS0, manteniendo valores (peticiones) anteriores.

Restauro registros y program counter. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	Nº Botón; Nº Terminal
R1 (temp)	Parámetros
R4	Nº Terminal

NOTA: Inicialmente, no se programan las resistencias ya que por simplicidad se comenzó colocando la placa BerryClip en la misma posición que la primera práctica. Cuando se ha verificado que el código funciona, se ha desplazado a la posición indicada para esta práctica, programando las resistencias pull down como pull up, y cambiando los valores de los terminales.

➔ Función "Lee_boton": Recibe como parámetro el número del botón cuya pulsación se debe reconocer. Finaliza cuando se ha detectado la pulsación del botón. Subrutina pública (accesible desde programas externos).

Guardo registros y link register en la pila. Inicia subrutina.

si (botón = 1) entonces: #terminal = 10

si (botón = 2) entonces: #terminal = 9

boton_activado (#TERMINAL)

boton_desactivado (#TERMINAL)

Restauro registros y program counter. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	Nº Botón; Nº Terminal
R4	Nº Terminal

NOTA: ¿Qué son las subrutinas "boton_activado" y "boton_desactivado"? Son las rutinas encargadas de detectar que el botón ha sido pulsado y soltado, respectivamente. El objetivo de esta división es simplificar el diseño y estructurar mejor la función de uso, facilitando la comprensión del código. A continuación, el funcionamiento de estas subritinas:

➔ Función "boton_activado": Recibe como parámetro el número de terminal del botón cuya pulsación se debe reconocer. Finaliza cuando se ha detectado un evento de flanco descendiente. Subrutina privada.

Guardo link register en la pila. Inicia subrutina.

cargo la dirección del controlador GPIO (DIR_BASE)

creo máscara para el bit del botón

elimino detecciones anteriores: aplico máscara sobre GPEDS0

hacer (do) :

registro = cargo reg GPEDS0

resultado = (AND) máscara, registro

mientras (resultado = 0)

Restauro program counter. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0	Num Terminal; Máscara botón;
R1 (temp)	Registro GPEDS
R2 (temp)	resultado
R3	DIR_BASE

➔ Función "boton_desactivado": Recibe como parámetro el número de terminal del botón que se va a soltar. Finaliza cuando se ha comprobado que el botón ya no está pulsado. Subrutina privada.

Guardo registros y link register en la pila. Inicia subrutina.

cargo la dirección del controlador GPIO (DIR_BASE)

index_bucle = 3

creo máscara para el bit del botón

mientras (index_bucle != 0) hacer

registro = cargo reg GPLEV0

resultado = (AND) máscara, registro

si (resultado = 0) entonces

index_bucle = 3

fin si

si no, hacer

index_bucle = index_bucle - 1

fin si

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0	Num Terminal; resultado; ESPERA;
R1 (temp)	Registro GPLEV
R2 (temp)	resultado
R4	DIR_BASE
R5	Máscara botón
R6	index_bucle

ST_Delay(ESPERA)

fin mientras

elimino la detección del evento. el evento se ha atendido: aplico máscara sobre GPEDS0

Restauro registros y program counter. Fin subrutina.

FICHERO “Td_boton”

Objetivo: Probar el correcto funcionamiento del driver desarrollado.

➔ _start:

ST_Init()

Ini_Boton(1)

cargo cadena de prueba

hacer:

CHAR = (byte) CADENA_PRUEBA

si (CHAR = 0X0) entonces

break;

fin si

lee_Boton(1)

imprime CHAR

@CADENA_PRUEBA = @CADENA_PRUEBA + 1

fin bucle

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0	Parámetros
R4	CADENA_PRUEBA
R5	CHAR
R3	DIR_BASE

Apartado (2/3): Gestión de interrupciones del GPIO

FICHERO “d_CI”

Objetivo: Programar un driver sencillo para operar con el controlador de interrupciones.

➔ Función “Ini_CI”: Obtiene la dirección lógica del controlador de interrupciones y la almacena en una variable en memoria. No recibe ni devuelve parámetros. Subrutina pública (accesible desde programas externos).

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	13
R1 (temp)	DIR_FISICA
R2 (temp)	0X100
R3	DIR_BASE

Guardo link register en la pila. Inicia subrutina.

DIR_BASE = SWI 0X68 (13, DIR_FISICA, 0X100)

guardo DIR_BASE en memoria

Restauro program counter. Fin subrutina.

➔ Función “Enable_CI”: Recibe el número de línea sobre el que se va a actuar. Permite que se gestionen las peticiones de interrupción que llegan por la línea indicada. Subrutina pública (accesible desde programas externos).

Guardo link register en la pila. Inicia subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0	#LINEA
R1	DIR_BASE_CI
R2	@DIR_EnIRQ
R3	máscara línea

cargo DIR_BASE_CI

si (#LINEA < 32) hacer

    @DIR_EnIRQ = @EnIRQ1

si no, hacer

    @DIR_EnIRQ = @EnIRQ2

    #LINEA = #LINEA – 32

fin si

creo máscara para el bit correspondiente a la línea

guardo máscara en el registro @EnableIRQ correspondiente

Restauro program counter. Fin subrutina.

➔ Función “Disable_CI”: Recibe el número de línea sobre el que se va a actuar. Impide que se gestionen las peticiones de interrupción que llegan por la línea indicada. Subrutina pública (accesible desde programas externos).

Guardo link register en la pila. Inicia subrutina.

cargo DIR_BASE_CI

si (#LINEA < 32) hacer

    @DIR_DisIRQ = @DisIRQ1

si no, hacer

@DIR_DisIRQ = @DisIRQ2

#LINEA = #LINEA – 32

fin si

creo máscara para el bit correspondiente a la línea

guardo máscara en el registro @DisableIRQ correspondiente

Restauro program counter. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0	#LINEA
R1	DIR_BASE_CI
R2	@DIR_DisIRQ
R3	máscara línea

FICHERO "d_Boton"

Objetivo: Programar un driver sencillo que detecta una pulsación de un botón mediante el modelo de gestión por interrupciones.

- ➔ Función "Ini_Boton": Prepara un botón pasado como parámetro para empezar a detectar las pulsaciones. No retorna. Subrutina pública (accesible desde programas externos).

Guardo registros y link register en la pila. Inicia subrutina.

si (botón = 1) entonces: #terminal = 10

si (botón = 2) entonces: #terminal = 9

prog_gpio (#TERMINAL, 0)

progResPull_gpio (#TERMINAL, 0)

Disable_CI(#GPEDS0_IRQ)

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	Nº Botón; Nº Terminal; Parámetros
R1 (temp)	Parámetros
R2	DIR_BASE
R4	Nº Terminal

habilitar detección de eventos de flanco ascendiente: bit del terminal del botón a 1 en el registro GPFEN0, manteniendo valores (peticiones) anteriores.

creo una entrada en el vector de excepción. Quiero que mi RTI "Int_Boton" atienda las peticiones que lleguen por la línea 49. SWI 0X4B (#GPEDS0_IRQ, @Int_Boton, 0, 0, 0)

Restauro registros y program counter. Fin subrutina.

- ➔ Función de uso "Lee_boton": Recibe como parámetro el número de botón cuya pulsación se quiere detectar. No retorna nada. Finaliza cuando se ha detectado la pulsación del botón. Subrutina pública (accesible desde programas externos).

Guardo link register en la pila. Inicia subrutina.

cargo DIR_BASE

si (botón = 1) entonces: #terminal = 10

si (botón = 2) entonces: #terminal = 9

creo máscara para el bit del terminal del botón

guardo la máscara en el registro GPEDS0

EnableCI(#GPEDS0_IRQ) // Deshabilito la máscara

Sleep() // El programa detiene su ejecución aquí. Seguirá cuando la RTI haya detectado la pulsación de un botón.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	Nº Botón; Nº Terminal; Parámetros
R1	Máscara terminal
R2	DIR_BASE

\- - -

DisableCI(#GPEDS0_IRQ) // Habilito de nuevo la máscara

Restauro program counter. Fin subrutina.

➔ Rutina de atención "Int_boton":

Guardo registros y link register en la pila. Inicia subrutina.

cargo la dirección del controlador GPIO (DIR_BASE)

creo máscara para el bit del botón

resultado = (AND) máscara, registro GPEDS0

si (resultado = 0) entonces: va a FIN_RTI

Nota: se entiende que la interrupción no la ha provocado el botón solicitado. Entonces no operamos.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0	registro GPEDS; resultado;
R2 (temp)	registro GPEDS; resultado_LEV;
R4	DIR_BASE
R5	Máscara botón
R6	index_bucle

index_bucle = 3

mientras (index_bucle != 0) hacer

registro = cargo reg GPLEV0

resultado_LEV = (AND) máscara, registro

si (resultado_LEV = 0) entonces

index_bucle = 3

fin si

si no, hacer

index_bucle = index_bucle - 1

fin si

espera_microsegundos(ESPERA)

fin mientras

WakeUp()

FIN_RTI:

elimino la detección del evento. el evento se ha atendido: aplico máscara sobre GPEDS0

Restauro registros y program counter. Fin subrutina.

NOTA SOBRE LA RUTINA "espera_microsegundos": Esta rutina se ha programado con el objetivo de reemplazar a la función "ST_Delay" incluida en d_timer, ya que cambiará el modo privilegiado IRQ por el modo usuario. Entonces, la nueva rutina gastará el tiempo indicado en microsegundos ejecutando instrucciones. El programa es el siguiente:

```
espera_microsegundos:
@Recibe un valor en microsegundos y pierde el tiempo.
STMDB sp!, {R4-R5, LR}
MOV R4, R0
MOV R5, #700 @frecuencia del procesador
MUL R5, R4, R5
LSR R4, R4, #1
FOR_EM:
CMP R4, #0
BEQ FIN_FOR_EM
SUB R4, R4, #1
B FOR_EM
FIN_FOR_EM:
LDMIA sp!, {R4-R5, PC}
```

➔ Función "Fin_Boton": Recibe como parámetro el número del botón que se va a dejar de usar. No retorna resultados. Subrutina pública (accesible desde programas externos).

Guardo registros y link register en la pila. Inicia subrutina.

cargo DIR_BASE

si (botón = 1) entonces: #terminal = 10

si (botón = 2) entonces: #terminal = 9

deshabilitar detección de eventos de flanco ascendiente: bit del terminal del botón a 0 en el registro GPFEN0, manteniendo valores (peticiones) anteriores (opero con bit clear – BIC).

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	Nº Botón; Nº Terminal; Parámetros
R1	Máscara; Parámetros
R2	Parámetros
R4	DIR_BASE; Parámetros

elimino la entrada del vector de excepción. Quiero que mi RTI "Int_Boton" deje de atender las peticiones que lleguen por la línea 49. SWI 0X4C (#GPEDS0_IRQ, @Int_Boton, 0, 0, 0)

Disable_CI(#GPEDS0_IRQ)

Restauro registros y program counter. Fin subrutina.

FICHERO “Td_boton”

Objetivo: Probar el correcto funcionamiento del driver desarrollado.

El programa de prueba funciona de forma similar al del apartado 1, con ligeros cambios. Se introduce la inicialización del controlador de interrupciones y se elimina la llamada a la inicialización del timer, que ya no necesitamos. Además, se finalizará el botón 1 al final del programa.

Apartado (3/3): Medida de distancia cuando se active la interrupción

FICHERO “d_CI”

Objetivo: Programar un driver sencillo para operar con el controlador de interrupciones.

No varía con respecto del apartado anterior.

FICHERO “d_Boton”

Objetivo: Programar un driver sencillo que detecta una pulsación de un botón mediante el modelo de gestión por interrupciones.

- ➔ Función “Ini_Boton”
  No varía con respecto del apartado anterior.

- ➔ Función de uso “Lee_boton”: No recibe parámetros. Finaliza cuando se ha detectado la pulsación de uno de los dos botones y retorna su identificador. Subrutina pública (accesible desde programas externos).

Guardo link register en la pila. Inicia subrutina.

cargo DIR_BASE

creo máscara para el bit del terminal del botón 1

creo máscara para el bit del terminal del botón 2

junto las máscaras y las guardo en el registro GPEDS0

EnableCI(#GPEDS0_IRQ) // Deshabilito la máscara

Sleep() // El programa detiene su ejecución aquí. Seguirá cuando la RTI haya detectado la pulsación de un botón.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	Máscara botón 1; BOTON; #botón
R1 (temp)	Máscara botón 2
R2 (temp)	Máscara conjunta
R3 (temp)	DIR_BASE

- - -

DisableCI(#GPEDS0_IRQ) // Habilito de nuevo la máscara

cargo BOTON // variable global donde la RTI ha almacenado el número de terminal del botón pulsado

si (BOTON = 10) entonces: #botón = 1

si (BOTON = 9) entonces: #botón = 2

Restauro program counter. Fin subrutina.

NOTA: La variable global "BOTON" tiene tamaño de 1 Byte, y se leerá o modificará el dato teniendo esto en cuenta.

➔ Rutina de atención "Int_boton":

Guardo registros y link register en la pila. Inicia subrutina.

cargo la dirección del controlador GPIO (DIR_BASE)

creo máscara para el bit del botón 1

creo máscara para el bit del botón 2

creo máscara conjunta para usarla más tarde

Comparo cada máscara con el registro GPEDS (AND), y guardo la máscara y el número del terminal del botón cuya señal se ha detectado.

Almaceno en memoria (BOTON) el número del botón pulsado.

index_bucle = 3

mientras (index_bucle != 0) hacer

registro = cargo reg GPLEV0

resultado_LEV = (AND) máscara, registro

si (resultado_LEV = 0) entonces

index_bucle = 3

fin si

si no, hacer

index_bucle = index_bucle - 1

fin si

espera_microsegundos(ESPERA)

fin mientras

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	registro GPEDS; resultado;
R2 (temp)	registro GPEDS; resultado_LEV;
R3 (temp)	#botón pulsado
R4	DIR_BASE
R5	Máscara botón 1; máscara del botón pulsado
R6	Máscara botón 2
R7	Máscara conjunta

WakeUp()

FIN_RTI:

elimino la detección del evento. el evento se ha atendido: aplico la máscara conjunta sobre GPEDS0.

Restauro registros y program counter. Fin subrutina.

NOTA SOBRE LA RUTINA "espera_microsegundos": La programación se mantiene igual a la del apartado anterior.

➔ Función "Fin_Boton":

Permanece sin cambios con respecto del apartado anterior.

FICHERO "Td_boton"

Objetivo: Probar el correcto funcionamiento del driver desarrollado.

➔ _start:

// inicialización de dispositivos

Ini_CI()

Ini_dist()

ST_Init()

Ini_Boton(1)

Ini_Boton(2)

Se programa un bucle que en cada iteración detecta una pulsación de botón. Si proviene del botón 1, imprimirá un mensaje con el resultado de la medición realizada con el dispositivo HC-SR04. Si, por el contrario, la pulsación proviene del botón 2, se finalizará el programa.

Antes de finalizar el programa, se deshabilitarán los dos botones que hemos inicializado al empezar.

NOTA SOBRE EL APARTADO 3/3: En los apartados anteriores, se ha utilizado el driver del GPIO proporcionado. Debido a que por cuestiones de implementación y diseño durante la práctica anterior hay rutinas necesarias para tomar medidas en "d_gpio" que no existen en esta versión, se ha decidido crear una versión modificada del driver para el GPIO de la práctica anterior (d_gpio_extended). Se ha prescindido de todas aquellas rutinas y variables que ya contiene la versión proporcionada.

CUESTIONES A DESARROLLAR

¿Qué función de inicialización ha de habilitar la línea de interrupción de GPIO en el controlador de interrupciones, Ini_Boton o Ini_CI?

La función de inicialización "Ini_Boton" habilita la línea en el momento en el que reclamamos una entrada en el vector de excepción para que nuestra RTI atienda todas las interrupciones de la línea 49. "Ini_CI" tan solo obtiene y guarda la dirección lógica de los registros del controlador de interrupciones, sin operar sobre ellos.

¿Cómo se comportaría tu código si Lee_Boton no verifica que el botón haya dejado de estar presionado?

Caso 1: Algún factor desconocido hace que nuestras comprobaciones no den resultados positivos: No se ha definido un número de comprobaciones máximas. Esto implica que la ejecución entraría en un bucle infinito.

Caso 2: Directamente no se verifica que el botón haya dejado de estar pulsado: Atendiendo a nuestro programa de prueba, podría ocurrir que se impriman todos los caracteres simultáneamente, dependiendo del ruido en la señal del botón y, por tanto, del número de flancos descendientes que se detecten por cada pulsación real.

¿Cómo modificarías tu código si el segundo pulsador estuviese conectado al terminal 40 de GPIO?

A la hora de programar, hemos dado por hecho una serie de factores que impedirían el funcionamiento de nuestro programa en caso de que el pulsador 2 esté conectado al GPIO40.

Por un lado, encontramos el acceso a los registros del controlador del GPIO. Hasta ahora siempre hemos trabajado con terminales comprendidos entre el 2 y el 27. Estos utilizan los registros "bajos" del controlador, como son GPEDS**0**, GPLEV**0**… Sin embargo, al GPIO40 le corresponden los registros "altos" (GPEDS**1**, GPLEV**1**…). Mi programa debería integrar, allá donde utilice los registros del controlador, al menos una comprobación que determine qué registro le corresponde a cada terminal GPIO. Otra opción sería crear más constantes.

Por otro lado, debemos prestar especial atención a la gestión de las interrupciones, y es que hasta ahora hemos trabajado con la línea 49, que corresponde a los eventos detectados que se reflejan en el registro GPEDS**0**. Como los eventos del GPIO40 se registran en el GPEDS**1**, las interrupciones ahora llegan por la línea 50. Mi programa deberá determinar cuál es la línea de interrupción que corresponde a cada terminal, o bien previamente cuando inicializamos un botón guardando la línea asociada en una variable en memoria, o bien cada vez que se vaya a hacer uso de este valor.

En cualquier caso, la comprobación es muy sencilla. Tan solo se comparará el número de terminal con 31. Si es mayor, le asignamos los "registros altos". Si es menor, los "bajos".