PSEUDO-CÓDIGO DE LOS ALGORITMOS DESARROLLADOS

Apartado (1/3): Conexión y programación básica del dispositivo HC-SR04

FICHERO "d_dist1"

Objetivo: Programar un driver que detecta cambios en la señal "Echo" del dispositivo HC-SR04.

➔ Función "Ini_dist": Realiza las gestiones necesarias en el dispositivo para empezar a tomar medidas, indicando a cada terminal el modo de comportamiento. No recibe parámetros ni retorna. Subrutina pública (accesible desde programas externos).

Guardo link register en la pila. Inicia subrutina.

prog_gpio(#TERMINAL_TIG, 1)

prog_gpio(#TERMINAL_ECHO, 0)

Restauro program counter. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	Nº Terminal
R1 (temp)	Modo Func.

NOTA: ¿Qué es la subrutina "prog_gpio"? Es una subrutina pública del driver d_gpio, desarrollada en la práctica anterior para cambiar el modo de comportamiento de los terminales a In/Out.

Diferencias con respecto de la práctica anterior: d_gpio_LED ahora se llama d_gpio, con el objetivo de dejar claro que su uso no está limitado a trabajar con los terminales de los LED de la BerryClip.

➔ Mide_dist: No recibe parámetros, y retorna en R0 un booleano indicando si se ha podido tomar o no la medida. Subrutina pública (accesible desde programas externos).

Guardo link register en la pila. Inicia subrutina.

valor_terminal(8, 1)

ST_Delay(#TIME_HIGH = 15)

valor_terminal(8, 0)

bool = espera_valor(15, 1)

si (bool != 0) hacer

    bool = espera_valor(15, 0)

fin si

retornar bool

Restauro program counter. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	bool

NOTA: ¿Qué es la subrutina "valor_terminal"? Es la misma subrutina que en la práctica 1 se llamó "valor_LED". Como ahora vamos a trabajar con un rango de terminales distinto al de los LED de la BerryClip, generalizaremos el uso de esta subrutina. La dinámica de asignar un valor numérico desde el 0 hasta n (nº de terminales compatibles con esta subrutina) se mantiene por simplicidad. Entonces el código de la función es el mismo, pero varía el array de terminales con el que trabaja.

NOTA: ¿Qué es la subrutina "espera_valor"? Es una función desarrollada para detectar un valor elegido en la señal "Echo" (aunque su propósito es general). Detalle de su funcionamiento:

FICHERO "d_gpio"

➔ espera_valor: Función pública externa (d_gpio). Recibe como parámetros el valor del terminal y el valor esperado en este. Retorna 0 si no se ha detectado ese valor, y 1 si lo ha detectado. Es responsable de la tarea de sincronización entre el computador y el dispositivo HC-SR04. Subrutina pública (accesible desde programas externos).

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

num_comprob = 100000

bool = FALSE; @no se crea como variable. Solo se retornará FALSE en R0 si no se detecta el valor esperado

mientras (num_comprob > 0) hacer

accedo a la dirección base de los regs gpio con desplazamiento al GPLEV0, y guardo el contenido del registro

desplazo el contenido bit a bit hacia la derecha tantas veces como el número del terminal

valor_lev = "and" del contenido y '1' (valor de ese terminal)

si (valor_lev = Valor Esperado) entonces

bool = TRUE

break;

fin si

num_comprob = num_comprob - 1

fin mientras

Restauro registros y PC en la pila. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0	Num Terminal; bool
R1	Valor Esperado
R2 (temp)	[GPLEV0]
R3 (temp)	Valor actual
R4	num_comprob
R5	DIR_BASE

NOTA SOBRE LA RUTINA "espera_valor": Es necesario limitar el número de comprobaciones para la detección del cambio de flanco en la señal echo. En caso de flanco ascendente, para evitar entrar en un bucle infinito si no hay señal de vuelta. El caso de flanco descendiente es distinto. Según la ficha técnica del dispositivo HC-SR04, podemos deducir que siempre que el pulso de echo se activa, se va a desactivar en algún momento, ya sea por la recepción de la onda lanzada por Trig, o por la superación de un límite establecido en unos 18 mS (en este caso el flanco baja 36mS después de la subida). Sin embargo, no deja de ser una buena práctica de programación limitar el número de comprobaciones, ya que siempre pueden darse errores que en un principio no estaban contemplados. Se establece un límite de 100000 comprobaciones para los dos casos, pues se ha demostrado empíricamente que, en el caso del flanco descendente, no se espera llegar nunca a esa cantidad.

NOTA SOBRE EL APARTADO 1/3: Cuando terminé de programar el driver, el funcionamiento no era el esperado y, para poder depurar una parte que me interesaba especialmente, me vi en la necesidad de crear una subrutina privada de prueba que verificase que el valor del terminal "trig" (en GPLEV0) era el esperado en cada llamada, mostrando un mensaje en caso afirmativo. "comprueba" recibe el número de terminal a comprobar como parámetro (R0). Solo funciona para un caso particular: muestra mensaje si el valor del terminal es '1'. Para probar el otro caso (valor terminal = 0), se modificaron directamente los registros en tiempo de ejecución. Esta subrutina se eliminó cuando ya no hizo falta, y su código es el siguiente:

```
comprueba:
STMDB SP!, {R4-R8, LR}
MOV R4, R0 @No LED
MOV R7, #14
MOV R8, #1
LSL R8, R8, R7
LDR R6, =0xf8200000
SWI 0X16
LDR R8, [R6, #0x34]
SWI 0X7C
AND r8, r8, r7
CMP r8, #0
LDRNE r0, =BIEN_D
SWINE 0x2
LDMIA SP!, {R4-R8, PC}
```

Se observa que su diseño es elemental y se asume, por ejemplo, que la dirección virtual de los registros del controlador GPIO es 0XF8200000, para simplificar y agilizar la depuración.

FICHERO "Td_dist1"

Objetivo: Probar el correcto funcionamiento del driver desarrollado.

➔ _start:

ST_Init()

Ini_dist()

bool = Mide_dist()

si (bool = 0) -> muestra mensaje ERROR

si no -> muestra mensaje OK

Finaliza la ejecución del programa.

Apartado (2/3): Medida de distancias con el dispositivo HC-SR04

FICHERO “d_dist2”

Objetivo: Programar un driver que calcule la distancia entre el dispositivo HC-SR04 y un objeto, en base a los tiempos en que se detectan cambios en la señal “Echo”.

NOTA SOBRE EL APARTADO 2/3: Se actualiza “Mide_dist” para ajustarse a los nuevos requisitos, y se crea una subrutina que calculará de forma independiente la distancia en base a la diferencia entre los tiempos del cambio de señal en “Echo”. El resto de las funciones del programa se mantienen iguales al AP1.

➔ Mide_dist: No recibe parámetros, y retorna en R0 un booleano indicando si se ha podido tomar o no la medida, y en R1 la medida o "0" si no se pudo tomar.

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

valor_terminal(8, 1)

ST_Delay(#TIME_HIGH) @15

valor_terminal(8, 0)

bool = espera_valor(15, 1)

si (bool = 0) -> ir a ERROR MEDIDA

tiempo1 = ST_Read()

bool = espera_valor(15, 0)

si (bool = 0) -> ir a ERROR MEDIDA

diferencia_t = ST_Read() - tiempo1

si (diferencia_t >= 25000) -> ir a ERROR MEDIDA. nota: se tiene en cuenta que, si el dispositivo no detecta un objeto pasado un tiempo después de haber emitido la onda, el pulso del “Echo” durará unos 36 µs. Sabiendo que lo habitual es que este no supere los 18 µs, buscamos un punto medio para determinar si hay o no un objeto y si la medida es válida.

distancia = calcula_distancia(diferencia_t)

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	bool; diferencia_t; distancia;
R4	tiempo1

return TRUE & distancia;

ERROR MEDIDA: return FALSE & 0;

Restauro registros y program counter. Fin subrutina.

➔ calcula_distancia: Recibe como parámetro el tiempo que ha tardado la señal de ultrasonidos en llegar al objeto y volver en microsegundos. Retorna la distancia en milímetros. Subrutina privada.

distancia = tiempo * 11246 (0X2BEE)

distancia = distancia / 65536

Restauro program counter. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0	tiempo; distancia
R1	0X2BEE

FICHERO "Td_dist2"

Objetivo: Probar el correcto funcionamiento del driver desarrollado.

➔ _start:

ST_Init()

Ini_dist()

bool = r0 Mide_dist()

medida = r1 Mide_dist()

si (bool = 0) -> muestra mensaje ERROR

si no -> muestra mensaje OK

muestra "DISTANCIA: "

print_integer(medida)

muestra "mm"

Finaliza la ejecución del programa.

Apartado (3/3): Aumentando la precisión de las medidas

FICHERO "d_dist3"

Objetivo: Programar un driver que calcule la distancia entre el dispositivo HC-SR04 y un objeto, en base a los tiempos en que se detectan cambios en la señal "Echo". Similar al apartado 2/3, pero ahora buscamos conseguir una medición más precisa.

NOTA SOBRE EL APARTADO 3/3: La subrutina “Mide_dist” pasa a llamarse “coge_medida” y ahora es privada: solo será llamada por la nueva función dentro de este fichero “Mide_dist”.

➔ Mide_dist: No recibe parámetros, y retorna en R0 un booleano indicando si se ha podido tomar o no la medida, y en R1 la medida o "0" si no se pudo tomar.

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

@se ha definido un array en memoria de longitud para 8 medidas tamaño word, todas ellas inicializadas a -1

hacer @estructura alto nivel: do … while

media = completa_medidas(Array Medidas)

si (media = -1) -> ir a ERROR MEDIDA

guarda media en registro seguro

r_dispersion = media/16

num_medidas_validas = verifica_medidas(Array Medidas, media, r_dispersion)

mientras (num_medidas_validas != 8)

return TRUE & media; @proceso satisfactorio

ERROR MEDIDA: return FALSE & 0; @no se han podido tomar las medidas
Restauro registros y program counter. Fin subrutina.

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R2 (temp)	r_dispersion
R4	Array Medidas
R5	media

➔ completa_medidas: Recibe como parámetro una referencia a un array de medidas (r0) (L=8). Por cada posición del array en la que aparezca "-1" se toma una medida y se reemplaza. Si las medidas son válidas, retorna la media de todas las medidas del array (r0). Si 16 mediciones consecutivas son incorrectas, se retorna -1 (r0)

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

contador_incorrectas = 0 @medidas incorrectas

index = 8

mientras (index > 0) hacer

medida = array_medidas[]

si (medida = -1) hacer

medida_b = r0 coge_medida()

medida = r1 coge_medida()

si (medida_b = 1) hacer

guarda “medida” en su posición del array “array_medidas[]”

contador_incorrectas = 0 @resetea contador

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R4	contador_incorrectas
R5	index bucle
R6	Ref. array medidas
R7	suma_medidas
R8	medida

si no, hacer

contador_incorrectas = contador_incorrectas + 1

si (contador_incorrectas = 16) -> ERROR MEDIDA

fin si

fin si

suma_medidas = suma_medidas + medida

array_medidas = array_medidas + 4 @avanzo un word en el array

index = index – 1

ST_Delay(#TIME_WHILE) @100000

fin mientras

return (suma_medidas / 8); @proceso satisfactorio

ERROR MEDIDA: return -1; @se han alcanzado las 16 medidas máximas sin dato

Restauro registros y program counter. Fin subrutina.

➔ verifica_medidas: Recibe como parametros una referencia al array de medidas (r0), la media (r1) y el margen de dispersión (r2). Reemplaza todas las medidas del array por -1 cuando esta no se encuentra en nuestro margen de error. Retorna el número de medidas que verifican la condición (r0)

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

medidas_validas = 0

index = 8

mientras (index > 0) hacer

medida = array[] @medida a comprobar

aux = media – dispersion

si (medida < aux) -> ir a NO VERIFICA

aux = media + dispersion

si (media > aux) -> ir a NO VERIFICA

medidas_validas = medidas_validas + 1

TABLA DE USO DE REGISTROS	
#REG	VARIABLE
R0 (temp)	medida
R1 (temp)	aux
R4	Ref. array medidas
R5	media
R6	dispersion
R7	medidas_validas
R8	index bucle

NO VERIFICA: array[] = -1 @sustituye la medida cuando es errónea con -1, para que cuando se ejecute de nuevo completa_medidas, se rellene con una nueva medida

@array = @array + 4 //avanzo 1 word la dirección del array

index = index – 1

fin mientras

return medidas_validas;

Restauro registros y program counter. Fin subrutina.

FICHERO “Td_dist3”

Objetivo: Probar el correcto funcionamiento del driver desarrollado.

NOTA: La función principal para probar y depurar este driver es igual a la que aparece en el fichero “Td_dist2”, siguiendo las instrucciones del enunciado del apartado.

CUESTIONES A DESARROLLAR

Indica qué se debería introducir en el código para determinar el número de veces que no se detecta correctamente el pulso Echo.

En la función "completa_medidas" del Apartado 3/3 ya se ha implementado un contador de pulsos no detectados. Estos se guardan en una variable "contador_incorrectas". El problema es que, si entre varios intentos fallidos hay uno que ha dado resultado, el contador se reinicia a 0, con el objetivo de cumplir con el enunciado y tener en cuenta solo los 16 pulsos fallidos, que han de ser consecutivos.

Para determinar el número de veces que no se detecta el pulso de "Echo" podemos seguir la misma idea: cada vez que llamamos a la función coge_medida() y nos devuelva FALSE, sumamos 1 unidad a un contador que tendremos en una variable. Después ya podremos trabajar con el dato, retornarlo…

Excepción: Medidas fuera del alcance del sensor: Tal y como está diseñado el algoritmo, coge_medida() devolverá también FALSE en caso de que no se haya detectado un objeto, pero esto no implica que no se haya detectado el pulso de echo. Entonces cuando trabajamos con este caso en particular, el valor del contador ya no será estrictamente el número de pulsos no detectados.

Indica qué piensas que ocurriría en tu código si el pulso de la señal Trig es inferior a 10µs.

El dispositivo HC-SR04 requiere un pulso en la señal "Trig" de un mínimo de 10 µs para emitir una onda. Esto implica que con un pulso de duración menor no se emitirá la señal, pues puede considerarse ruido o una interferencia.

Entonces, si no se emite la onda, no detectaremos cambios de flanco en la señal "Echo", y nuestro driver, que limita esta detección a 100000 comprobaciones, asumirá que no se ha podido tomar la medida y retornará FALSE, además de un 0 para la medida.

¿Se te ocurre alguna manera de detectar si el detector de ultrasonidos no funciona?

El funcionamiento de este dispositivo se basa en la generación de una onda que rebota en un objeto (a través de Trig), y la posterior recepción de un pulso a través del terminal al que está conectado "Echo". Por esto, el algoritmo del apartado 1, que nos permite saber si el pulso de Echo se ha detectado, supone una buena herramienta para conocer si el dispositivo trabaja como se espera. El código del tercer apartado nos facilita, además, la distancia precisa entre el sensor y un objeto. Para esto podemos realizar mediciones de distancias conocidas, preestablecidas (incluso las que puedan dar lugar a un error de medición), y entonces sabremos si el pulso de "Echo" se genera en los instantes adecuados.