Zdravko Dimitrov Arnaudov

PRÁCTICA 3

1. Inicialmente, comenzaremos abordando los algoritmos de las funciones y algoritmos de prueba realizados.

• Desarrollo 1

*-función de inicialización **ini_Timer**:*

determinamos dirección base lógica del System Timer.
guardamos en variable global.

*-función de uso **timer_Act**:*

salvamos dirección base del timer.
accedemos al registro del contador.
devolvemos en registros la parte baja y la parte alta.

*-programa de prueba **EC31**:*

inicializamos el timer.
leemos parte baja del contador y guardamos en registro
implementamos bucle de retardo.
leemos contador, y guardamos diferencia.
devolvemos resultado.

• Desarrollo 2

*-función de inicialización **Ini_dist**:*

programamos señal Trig como salida
programamos señal Echo como entrada

*-función de uso **Mide_dist:***

inicializamos timer
determinamos el tiempo en microsegundos que estará activa la señal Trig : 19
activamos señal Trig
leemos contador, contador + 19
    BUCLE1: leemos contador hasta hasta que sea igual a contador + 19.

desactivamos señal Trig.
    BUCLE2: leemos Echo hasta que sea igual a 1 (inicio pulso).
        si el numero de iteraciones supera 100000, finalizar con error.

leemos contador del timer y lo salvamos en registro (T1)
    BUCLE3: leemos Echo hasta que sea que sea igual a cero (fin pulso).

leemos contador de timer (T2) y guardamos la diferencia (T = T2- T1)
despejamos usando la velocidad del sonido y obtenemos la distancia al objeto.

si hubo error
    devolver distancia 0 y en registro 1 valor 1 para indicar error.
sino
    devolver distancia aproximada en milímetros.

*-programa de prueba **EC32:***

mostramos cabecera del programa.
si usuario pulsa 1, el dispositivo mide la distancia.
si en el registro R1 hay 1.
indicar por escrito que hubo error.
si en el registro R1 hay 0.
mostrar la distancia por pantalla.
si el usuario pulsa 2, el programa finaliza.
si el usuario pulsa opción no permitida, repetir intento.

- **Desarrollo 3**

*-función de inicialización **Ini_dist**:*

programa la señal Trig como salida
programa la señal Echo como entrada
programamos Echo para que genere eventos de flanco descendente.
limpiamos la detección de eventos anteriores.

*-función de uso **Mide_dist:***

inicializamos timer
determinamos el tiempo en microsegundos que estará activa la señal Trig : 19
activamos señal Trig
leemos contador, contador + 19
BUCLE1: leemos contador hasta hasta que sea igual a contador + 19.

desactivamos señal Trig.
BUCLE2: leemos Echo hasta que sea igual a 1 (inicio pulso).
si el numero de iteraciones supera 100000, finalizar con error.

leemos contador del timer y lo salvamos en registro (T1)
BUCLE3: leemos del registro de detección de eventos hasta devolver 1.

leemos contador de timer (T2) y guardamos la diferencia (T = T2- T1)
despejamos usando la velocidad del sonido y obtenemos la distancia al objeto.

si hubo error
devolver distancia 0 y en registro 1 valor 1 para indicar error.
sino
devolver distancia aproximada en milímetros.

*-programa de prueba **EC33:***

mostramos cabecera del programa.
si usuario pulsa 1, el dispositivo mide la distancia.
si en el registro R1 hay 1.
indicar por escrito que hubo error.
si en el registro R1 hay 0.
mostrar la distancia por pantalla.
si el usuario pulsa 2, el programa finaliza.
si el usuario pulsa opción no permitida, repetir intento.

2. Hemos comenzado la práctica programando el driver del System Timer en la Raspberry Pi a partir de dos funciones: una de inicialización que registraba la dirección base lógica del mismo y otra de uso, la cual separa el registro contador en dos (parte alta y parte baja, siendo esta última los 16 bits menos significativos y que usaremos para registrar los tiempos).
En el segundo desarrollo, hemos elaborado el driver para medir distancias con el dispositivo HC-SR04 a raíz de dos funciones: una para inicializar las señales Trig y Echo como salida y entrada respectivamente y otra de uso que se encarga de devolver la distancia al objeto medido en milímetros. Esta última función presenta la mayor complejidad respecto al algoritmo puesto que envía un pulso por la señal de salida con un tiempo determinado, espera a cambios en la señal de entrada y calcula la anchura del pulso (tiempo que tarda la señal en llegar al objeto y rebotar de vuelta) con la diferencia de contadores. Finalmente, despeja la distancia usando la velocidad del sonido y la devuelve en caso de no haber error.
Por último, en el tercer desarrollo, se programa la señal Echo para que pueda producir eventos de flanco descendente y a la hora de detectar el cambio de señal a 0 en el driver, se realiza encuesta continua al registro de estado de detección de eventos hasta que se produce el cambio de flanco.

3. Para comprobar la funcionalidad de las funciones desarrolladas, hemos implementado tres programas de prueba. El primero se enmarca en el System Timer y simplemente devuelve un tiempo medido.
El segundo programa imprime una cabecera indicando que el usuario puede escoger entre medir una distancia o finalizar el programa. La distancia será imprimida por pantalla si no se ha producido ningún error , que en tal caso, se notifica por pantalla.
El tercer y último programa de prueba conforma la misma funcionalidad que el anterior pero el driver usado se enmarca en la detección de eventos.

4. A continuación, responderemos razonadamente a las cuestione propuestas.

- Para generar el pulso de activación del medidor de ultrasonidos, hemos de escoger el tiempo en microsegundos que estará activo (entre 15 y 20). Seguidamente, determinamos el tiempo de espera del bucle que será el valor del contador del timer más el tiempo previo escogido.
- Para detectar el flanco de bajada en la señal Echo, es preciso programar dicha señal para que genere eventos de flanco descendente, limpiar la detección de eventos y a la hora de leer el cambio de señal se realiza encuesta continua en el registro de estado de detección de eventos hasta que se produce el cambio de flanco.