Universidad Nacional de Córdoba

Facultad de Cs. Exactas, Físicas y Naturales

Electrónica Digital III

Trabajo Final Integrador

Docente:

- Ayarde, Martin

Alumnos:

- Amallo, Sofía
- Bonino, Francisco
- Covacich, Axel

Índice

Introducción

Para siguiente trabajo se creó un prototipo de una cinta de correr, la cual cuenta con algunas funcionalidades específicas, como serán mostradas a lo largo de los subíndices del informe.

En el siguiente repositorio de Github se puede observar el progreso de nuestro trabajo desde su inicio hasta su conclusión: [sofia-am/Electronica-Digital-III_int: Repositorio para el trabajo integrador de la asignatura Electrónica Digital III de la Universidad Nacional de Córdoba. (github.com)]

Control de un motor con PWM

Para simular el control del motor que hace funcionar a la cinta utilizamos el módulo PWM. Este módulo hereda todas las funcionalidades del módulo Timer, y nos permite establecer un ciclo de trabajo combinando dos timers, uno que determina el periodo de la señal que alimenta el motor, y otro para el duty cycle.

Para controlar el funcionamiento del mismo se utilizó un LED cuyo brillo varía con la variación del duty cycle que en la vida real podría significar una variación en las revoluciones por minuto del motor.

Realizamos el siguiente cálculo para poder traducir la velocidad en km/h a su equivalente en duty cycle (valor que se carga en el registro MR1)

- MR0 = 1000: es el periodo de trabajo.
- MR1 es el duty cycle. La velocidad máxima corresponde a un duty cycle del 100%

$$20 \text{ km/h} \text{ — } 1000$$
$$\text{vel km/h} \text{ — } X$$

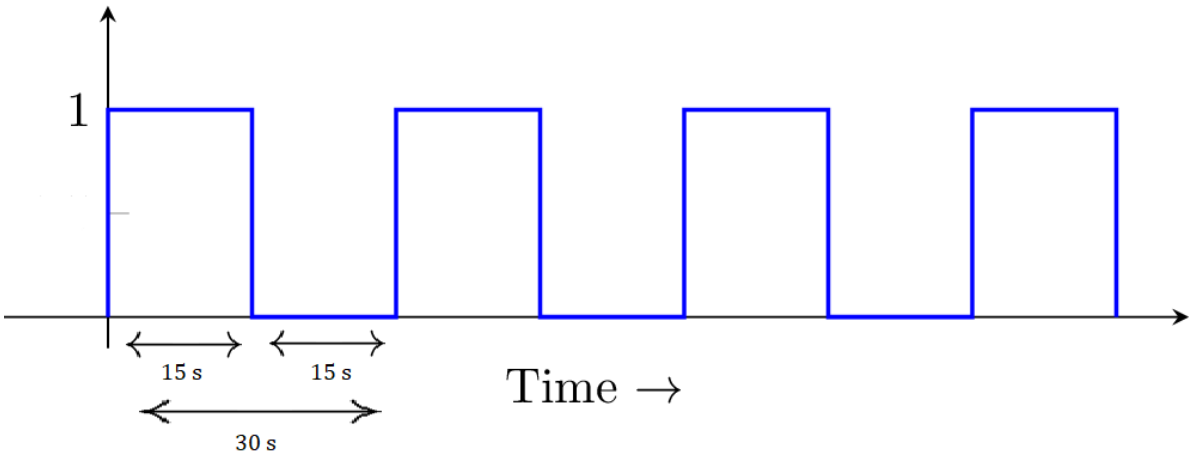
$$(\text{velocidad deseada})*1000/20 = X$$
$$(\text{velocidad deseada})*50 = X$$

donde X es el valor de carga en el módulo MR1.

Medición de temperatura del usuario con ADC

Esta funcionalidad surgió a partir del requerimiento de medir alguna magnitud analógica, en el contexto de la cinta de correr se pensó con el objetivo de medir la temperatura corporal del corredor. Para implementarlo se utilizó un sensor de temperatura LM35, que está conectado al pin AD0.0 del ADC para su conversión.

Como se trata de una magnitud que no varía con mucha frecuencia, se estableció un periodo de muestreo de 30 segundos para facilitar el testeo. Para lograr este objetivo se utilizó el Timer 1 en modo Match, con un comportamiento de toggle al hacer match. Se utilizaron drivers para establecer un match de prescaler cada 1 ms y se cargó el MR0 con un valor de 149, de esta manera se logra un match cada 15 segundos, y como el ADC inicia su conversión en un flanco de subida, el periodo de conversión queda en 30 segundos.



Muestra en pantalla con módulo UART

La función de comunicación serie por UART con una PC consta del envío de información útil cada 10 segundos mediante el Timer 0 para mostrar estadísticas en tiempo real de las pulsaciones por minuto, la velocidad a la que se corre y la temperatura corporal del corredor.

En un principio, los valores se enviaban en caracteres ASCII, por lo que si las ppm eran, por ejemplo, 60, al enviar esta información, el usuario veía reflejado en pantalla el caracter “<” que tiene un valor decimal 60 en ASCII. Lo mismo sucedía con la velocidad y la temperatura.

Para solucionar esto, se optó por descomponer el número en unidades y decenas para acotar el valor decimal al rango 0-9 y luego sumarle 48 para que en la tabla ASCII obtenga su equivalente correcto a cada número.

Para obtener las unidades, se realiza la operación módulo:

unidades = número % 10

De esta manera obtenemos el resto de dividir *número* entre 10.

Para obtener las decenas, podemos tomar el número original, restarle las unidades obtenidas como se especificó anteriormente y al resultado dividirlo por 10. Es decir:

decenas = (*número* - unidades) / 10

Finalmente, se devuelve *unidades + 48* o *decenas + 48* según lo que se necesite.

En el caso de la temperatura, al ser una variable de tipo ***float***, necesitábamos obtener el valor del primer decimal. Para esto, multiplicamos *número* por 10 para correr la coma, y así obtener las unidades de este nuevo número obtenido. Quedando así:

decimal = ((*número*10*) - (unidades + decenas)) / 100

y luego devolvemos *decimal + 48*.

Toda la información formateada se envía por UART cuando el usuario decide comenzar a monitorear sus estadísticas de rendimiento.

Control de pulsaciones con Timer en modo Capture

Se agregó la funcionalidad de medir las pulsaciones por minuto del corredor utilizando el Timer en modo Capture. Para simularlo en la placa se utilizó un pulsador que será activado de manera manual.

Se implementó además un filtro promediador móvil para que la medición se vaya haciendo cada vez más exacta. El Capture está configurado con un valor de match de prescaler cada 1 ms, es decir, la precisión con la que medimos el tiempo transcurrido entre pulsaciones es de 1 milisegundo.

Dentro del handler del Capture se obtiene el tiempo transcurrido entre 2 pulsaciones, dato que se almacena en un buffer circular para obtener un promedio entre pulsaciones mas exacto. El resultado de este promedio es dividido entre 600 para obtener las pulsaciones por minuto.

El razonamiento es el siguiente: nuestro buffer circular almacena 10 muestras en cierto tiempo T (el tiempo promedio entre las 10 pulsaciones.). Ese valor lo podemos extender a un minuto multiplicando por 60.

10 pulsaciones — T
PPM — 60 segundos
PPM = (60*10)/T medición

Almacenamiento de información usando DMA

El almacenamiento de información se realiza una vez finalizada la corrida, cuando se presiona la tecla C de Stop. Los datos a guardar serán la distancia recorrida, temperatura medida, tiempo medido desde la pulsación de la tecla D correspondida a empezar a trackear información, y pulsaciones por minuto.

Para medir el tiempo se implementó el Timer 2 en modo match cada 1 segundo para hacer el conteo de segundos. Luego se obtiene la distancia con el tiempo medido y la velocidad.

Para el cálculo de la distancia, se tuvo que pasar el tiempo medido en segundos a horas mediante el siguiente cálculo:

$$1h \text{ — } 3600s$$

$$0.00028h \text{ — } 1s$$

De esta forma, para lograr obtener la distancia recorrida en metros, hay que multiplicar además por 1000 para pasar la velocidad de Km/h a m/h

Finalmente, la ecuación para calcular la distancia recorrida en metros queda de la siguiente manera:

$$distancia = velocidad * tiempo * 0.28$$

Luego se realiza el envío de datos por DMA al presionar la tecla C, donde se activa el canal correspondiente de DMA y se realiza una transferencia memoria a memoria por el canal 0 desde un buffer de origen, el cual contiene todos los datos antes mencionados, a un buffer de destino donde quedarán almacenados los datos. No se utilizó LLI.

Control por teclado a través de GPIO

Para el control del teclado de 4 filas y 4 columnas, se optó por ocupar el puerto 2 (P2.0 - P2.7) por tener los pines contiguos y facilitar así la conexión en la protoboard.

Al trabajar en este puerto, las interrupciones generadas serán atendidas en la rutina de interrupción de EINT3.

La decodificación de las teclas presionadas se realizó mediante un algoritmo de multiplexación entre las filas y las columnas. Para esto, las filas del teclado se configuraron como pines GPIO de salida y las columnas como pines GPIO de entrada.

Cuando una tecla es pulsada y se capta una interrupción por EINT3, se envía un '1' por cada fila y se analiza el nibble superior del primer byte del puerto 2. Si la fila es la correcta, el '1' debería llegar y todas las columnas deberían estar en '1', por lo que si el nibble superior del primer byte es 0xf, estamos en la fila correcta.

Para obtener la columna, recorremos la copia almacenada de la lectura del puerto 2 hasta encontrar un '0' en alguno de los bits del nibble inferior del primer byte.

Teniendo la fila y la columna, se devuelve el valor de la coordenada en el teclado de la tecla resultante mediante la ecuación:

coordenada = (4 * fila) + columna

Una vez obtenida la coordenada (desde (0,0) hasta (3,3)), se hace un mapeo entre la coordenada y la tecla correspondiente.

Finalmente, se ejecuta la acción requerida en base a las siguientes funciones especificadas:

Coordenada	Tecla	Función
(0,3)	A	Encender la cinta
(1,3)	B	Setear velocidad ingresada
(2,3)	C	Apagar la cinta
(3,3)	D	Comenzar a trackear estadísticas
(3,0)	E	Aumentar velocidad (+1Km/h)
(3,2)	F	Disminuir velocidad (-1Km/h)
Cualquier otra	Número (0-9)	Almacenar un nuevo posible valor de velocidad a setear

Para una mejor apreciación del input del programa, se decidió mostrar por un display de 7 segmentos de cátodo común conectado al puerto 0 (P0.0 - P0.6) el valor de la tecla presionada.

Algunas observaciones

Nos gustaría usar este espacio para comentar algunas dificultades con las que nos encontramos pero pudimos solucionar:

- **Utilizar un tipo de dato muy chico**
  Por usar `uint8_t` en lugar de `uint32_t` perdíamos información y el LED que debíamos encender no brillaba tanto como esperábamos.

- **Prestar atención con los pines que usábamos**
  Tuvimos que refactorizar el código en dos oportunidades ya que al tener funcionalidades compartidas tuvimos que cambiar los módulos que usábamos para que no hubiera conflictos.

- **Tener que resetear constantemente la placa**
  No encontramos otra solución que desenchufar y volver a enchufar la placa para corregir algunos errores que obteníamos a la hora de debuggear el código, o incluso resetearla mediante el pin 4 de la misma.

- **Conectar mal los pines del módulo CP2102**
  Conectamos RX a RX y TX a TX.

- **Utilizar lógica de registros en lugar de drivers**
  Muchas veces la configuración por drivers no se comportaba como esperábamos y era más simple configurar directamente el registro.