6- Qüestionari Pràctica L6B ADC

Nom i Cognoms: Sergio Shmyhelskyy Yaskevych && Songhe Wang Grup: 31

Ejercicio 1: En primer lugar, añadiremos dos botones que sirvan para aumentar o disminuir el Duty Cycle de la señal PWM (la frecuencia del PWM será de 500Hz al igual que en el trabajo previo) y comprobaremos usando el osciloscopio de Proteus que todo funciona según lo esperado. Estos botones se leerán por detección de flanco y en cada pulsación aumentarán o disminuirán el Duty Cycle en un 5%. Los valores se saturarán para que no sean mayores que el 100% o menores que el 0%. Hay que mostrar el valor actual de Duty Cycle en el GLCD.

Realizado con PROTEUS.

Ejercicio 2: Poner el Duty Cycle al 100% y esperar hasta que el motor llegue al estado estacionario. a) ¿Cuál es el régimen máximo de giro en RPM? (el valor en RPM se puede consultar en el propio display del motor que proporciona Proteus, como se muestra en la Figura 2); b) ¿A qué frecuencia equivale en vueltas por segundo? c) Teniendo un encoder de 24 pulsos por vuelta, ¿qué frecuencia máxima tendremos como entrada? d) Cuál será el periodo máximo que podría tener nuestro bucle while en el main en caso de que quisiéramos leer el encoder detectando flancos mediante polling en una entrada normal? (como cuando leemos botones por detección de flanco).

a) Podemos observar en la simulación que el RPM máximo se alcanza cuando el duty cycle del PWM está al 100%. El valor máximo es de 365 RPM.
b) Dividiendo entre 60 obtendremos las vueltas por segundo, es decir 6.0834 RPS.
c) 24 (pulsos por revolución) · 6.0834 RPS (revolución por segundo) ≈ 145.992 Hz (pulsaciones por segundo)
d) El periodo que tendría que ser 6.8496ms (1/fmax = 1/145.992), para no perder ningún pulso.

Ejercicio 3: Intentad aligerar al máximo (sin reducir funcionalidad y sin volvernos locos) el bucle principal del programa. Usando los breakpoints de Proteus medir el tiempo que tarda en ejecutar una iteración del bucle. a) ¿cuál es el tiempo menor obtenido? b) ¿cuál es el mayor? c) Cuál sería la frecuencia máxima que podríamos muestrear usando polling? d) ¿sería suficiente para el motor que estamos simulando?

a) Después de ejecutar el programa en debug y observar el tiempo indicado en el time elapsed del proteus, hemos podido determinar que el bucle del main dura 3us. En este caso no se pulsa ningún botón.
b) El tiempo máximo que podemos obtener en el main depende del usuario, ya que dependiendo de cuando suelte el botón, tendremos un tiempo diferente. Considerando teóricamente que el usuario deja ir el botón instantáneamente, hemos obtenido un tiempo total de bucle de 20.096ms.
c) Si consideramos el caso que no se quiere usar ningún botón, tendremos una frecuencia de muestreo de 0.33MHz. En el peor caso, si se usan los botones obtendremos una frecuencia de 49.761Hz.
d) Podemos concluir que si solo queremos la funcionalidad de muestrear, entonces tendremos de sobras para hacer la muestra. Si queremos implementar otras funcionalidad como las de aumentar el duty cycle (que conlleva pintar en la GLCD) tendremos una frecuencia de muestreo de 49.761Hz, lo cual no es suficiente y implicaría perder la lectura de algunos pulsos del motor.

Ejercicio 4: Escribe una posible solución (puede ser en pseudocódigo o directamente una descripción en texto) que permitiera estimar la velocidad de giro del motor empleando polling y refrescar la interfaz de usuario (leer los botones y actualizar cada cierto tiempo la pantalla) de manera razonablemente precisa. ¿Qué frecuencia máxima se podría llegar a estimar con precisión?

Si queremos predecir la velocidad del giro del motor mediante un polling en una entrada normal debemos usar un timer. En un bucle contamos los pulsos que nos llegan del motor, por cada pulso que llega, sumaremos el valor del timer a una variable que nos guardará el tiempo total de una revolución. Saldremos del bucle cuando ya nos hayan llegado los 24 pulsos, y entonces haremos el cálculo para obtener la RPM según la frecuencia de oscilación que hemos escogido para el timer.

El procedimiento se basaría en que cada vez que se detecte alguno de los flancos (subida o bajada) proveniente de los pulsos del motor, el PIC incremente un contador mediante **polling**.

```
Bucle principal
Iniciar variables                                    // conteoPulsos, sumaTiempo
Poner Timer y conteoPulsos a 0
Leer botones (realizar las acciones relacionadas)
Inicio Bucle interno conteoPulsos < 24
        Si el timer está a punto de hacer overflow entonces Sumar valor del timer a sumaTiempo
        Si nos llega señal del motor
            Sumar valor del timer a sumaTiempo
            Actualización de variables (++conteoPulsos)
Fin del bucle interno
Calcular RPM
Refrescar la interfaz de usuario (mostrar en GLCD nueva RPM)
Fin del bucle principal.
```

//Todo y que sería mejor poner “Leer botones” dentro del bucle (para que pueda atender al usuario y no perder ningún pulso de botón) hemos decidido ponerlo fuera porque hemos visto en el apartado anterior que si realizamos acciones dentro del bucle disminuimos la frecuencia del bucle, afectando así al muestreo de los pulsos del motor (con acciones nos referimos a las funcionalidades del apartado 1).

¿Cuál sería la frecuencia máxima que podríamos estimar?

La frecuencia máxima será la frecuencia del bucle interno, dependiendo de cómo hayamos implementado el código (las actualizaciones de GLCD, botones…). Es decir 1/tiempo que tarda el bucle. Ahora sí, para que la aproximación sea más exacta, la resolución del encoder y la frecuencia de muestreo deberían ser suficientemente altas para poder capturar los cambios más “rápidos” la velocidad de giro del motor.

Ejercicio 5: Describe de manera similar a la anterior cómo se podría programar la estimación de velocidad empleando interrupciones para la lectura del encoder.

El concepto general aquí, es similar al anterior, solo que ahora el microcontrolador actualiza las variables **dentro de la rutina de interrupción**.

Config PIC: Habilitar interrupciones
Iniciar variables // sumaTiempo
Rutinas principales
Flag de interrupción del timer. Inicio del manejador
    sumamos 2^16 a sumaTiempo (en caso que el timer sea de 16 bits).
Fin del manejador
Flag de interrupción por pulso de motor. Inicio del manejador
    ++conteoPulso.
Fin del manejador

Bucle principal
Iniciar variables // conteoPulsos

Leer botones (realizar las acciones relacionadas).
Sí conteoPulso = 24 entonces :
    sumar a sumaTiempo el registro del timer (tiempo restante).
    Calcular nueva RPM
    Actualizar GLCD
    Poner conteoPulsos, sumaTiempo, timer a 0.
Fin del bucle principal.

Ejercicio 6: Describe cómo se podría hacer la estimación de velocidad empleando un módulo CCP e impleméntalo en Proteus. Para esta implementación usaremos el módulo CCP5 asociado al Timer 3. Hay que mostrar (además del Duty Cycle) el valor estimado de RPM en la pantalla del GLCD (Esto servirá para comprobar que la estimación es razonablemente precisa comparando el valor con el mostrado en el display que proporciona Proteus en el “encoder-motor”, ver Fig 2.)

Realizado con PROTEUS.

Ejercicio 7: Comenta las posibles ventajas y desventajas de los tres métodos anteriores.

Polling:

Ventajas	Desventajas
Simplicidad: La implementación mediante polling es relativamente simple y directa.	Consumo de CPU: Requiere dedicar recursos del procesador para la verificación constante del estado del encoder, lo que puede ser ineficiente.
Control preciso: Puedes tener un control preciso sobre el momento en que se realiza la lectura del encoder.	La respuesta en tiempo de polling puede ser larga, por tanto, hay probabilidad de que se ignore los pulsos de los botones si el RPM es bajo, ya que el bucle interno durará más tiempo y la lectura se hace fuera del bucle.

Interrupciones:

Ventajas	Desventajas
Al no haber bucle de polling, hay menos probabilidad de ignorar pulsos del botón. Es decir, podemos aprovechar la CPU mientras estamos haciendo el muestreo. No gastamos tiempo de CPU.	Complejidad: La implementación mediante interrupciones puede ser más compleja que el polling. Además, es muy difícil probar su funcionamiento, y todavía más encontrar posibles fallas.
Eficiencia de CPU: No es necesario realizar verificaciones constantes, ya que las interrupciones son eventos asíncronos, lo que mejora la eficiencia del procesador.	Tiempo variable: La duración de la interrupción no es fija, puede ser variable dependiendo de las condiciones del sistema (el tiempo para copiar registros, en caso de que la interrupción sea de low priority la puede interrumpir una de high…).

CCP (Capture/Compare/PWM):

Ventajas	Desventajas
Hardware dedicado: El módulo CCP está diseñado específicamente para gestionar tareas como la captura de eventos y la generación de PWM.	Limitado a hardware específico: Requiere que el microcontrolador tenga un módulo CCP disponible.
Precisión: Puede proporcionar mediciones mucho más precisas.	Configuración inicial más compleja: Configurar el módulo CCP puede requerir más esfuerzo en comparación con las soluciones de software.

Consideraciones generales:

- Tiempo de ejecución: El método de interrupciones suele ser más eficiente en términos de tiempo de ejecución, ya que solo se activa en respuesta a eventos.
- Precisión: El módulo CCP tiende a ofrecer mediciones más precisas y respuestas más rápidas que el polling.
- Uso de recursos: El polling puede consumir más recursos de CPU que las otras opciones, lo que puede ser crucial en sistemas con recursos limitados.