PRACTICA 5. INTRODUCCIÓN AL MICROCONTROLADOR V25

5.1 Introducción

En esta práctica se introduce el uso de las herramientas de programación del microcontrolador V25 que se empleará en el módulo III. Adicionalmente se describirá el uso de las placas de entrenamiento que permiten ejecutar los programas desarrollados sobre harware real.

La duración de esta práctica es de 1 sesión

Los objetivos de la práctica son:

- Describir las principales características del microcontrolador V25 haciendo especial hincapié en sus dispositivos periféricos.
- Introducir la herramienta de diseño C/C++ Paradigm que será empleada para desarrollar la programación del microcontrolador V25.
- Ser capaz de desarrollar y verificar un programa simple que interactúe con el mundo real y emplee temporizadores e interrupciones.

5.2 Trabajo previo

TRABAJO PREVIO

Recordad que se ha de presentar una copia del trabajo previo al inicio de la sesión de prácticas.

a) Leed atentamente este manual y las guías del software Paradigm y del microcontrolador V25.

b) Responded a las preguntas del cuestionario del V25 incluidas en el apartado 5.5

c) Estudiad el contenido de los ficheros de soporte ports.c y timers.c

c) Escribid una propuesta de programa pedido en el apartado 5.7.

5.3 Descripción del Microcontrolador V25

El microcontrolador V25 es un microcontrolador de 16 bits que incorpora un microprocesador similar a los de la familia Intel 8086/88 y los siguientes dispositivos principales de entrada/salida:

- Un puerto de teclado PT de 8 bits con nivel de comparación programable.
- Tres puertos de entrada/salida P0, P1 y P2 que permiten un total de 20 bits de entrada/salida y 4 bits adicionales sólo de entrada.
- Una interfaz de comunicación serie UART de dos canales.
- Un contador de base de tiempos.
- Dos temporizadores de 16 bits.

Una descripción mas detallada del microcontrolador se halla en la Adaptación de la Guia del Usuario del V25 que fue desarrollada para las prácticas del CISE IV, cuando se usaba este microcontrolador, y que está disponible en el CPET y en formato PDF dentro del servidor Weble.

Antes de seguir adelante conviene leer este documento.

Para el proyecto que desarrollamos en esta asignatura no necesitamos emplear todos los periféricos del V25. En concreto emplearemos:

- El puerto de teclado PT
- El puerto de entrada/salida P2 en modalidad de salida únicamente.
- La UART que controla el canal serie.
- El contador de la base de tiempos.
- Los dos temporizadores.

A parte de los periféricos de entrada/salida, el V25 cuenta con un controlador de interrupciones bastante completo que permite una gestión muy eficiente de los eventos de entrada/salida.

5.4 Descripción de la placa de desarrollo del V25

Dado que en el laboratorio contamos únicamente con un robot completo, resultaría muy tedioso tener que acceder a él por turnos para depurar el programa desarrollado. Es por ello que cada grupo de trabajo dispondrá de acceso a una placa de entrenamiento basada en el V25 que es idéntica a la empleada para el control del robot excepto por el hecho de que no está conectada a motores y cuentapasos reales.

La figura 1 muestra un esquema de esta placa de entrenamiento. Esta placa cuenta con 8 pulsadores asociados a las 8 líneas del puerto de teclado PT. También dispone de un pulsador de reset y de un conector DB9 para cargar el programa mediante un interfaz serie usando el canal 0 de la UART del propio microcontrolador. Es por ello que nuestro programa no podrá hacer uso de este canal si no deseamos perder la comunicación con el PC.

Las lineas de los otros puertos y el canal 1 de la UART se hallan disponibles en la tira de pines que se halla a la izquierda de la placa.

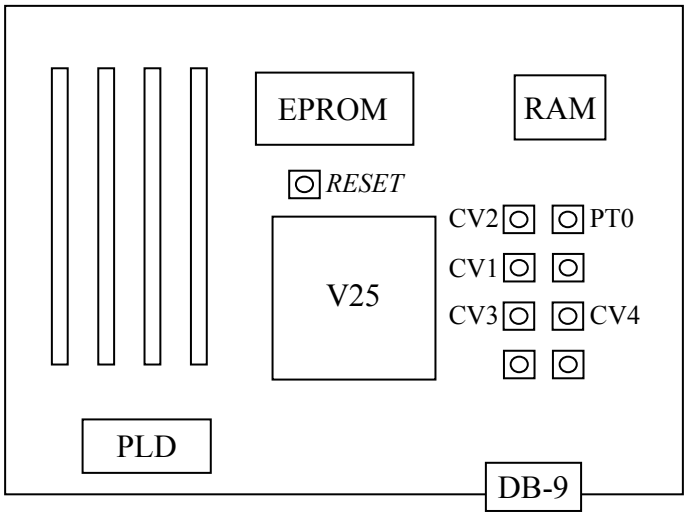

Figura 1 : Placa de entrenamiento del V25

Para poder interactuar con el mando a distancia desarrollado en el módulo II y para poder visualizar las órdenes de control de los motores asociadas al puerto P2, se ha añadido una placa adicional como la mostrada en la figura 2 sobre los pines de los puertos.

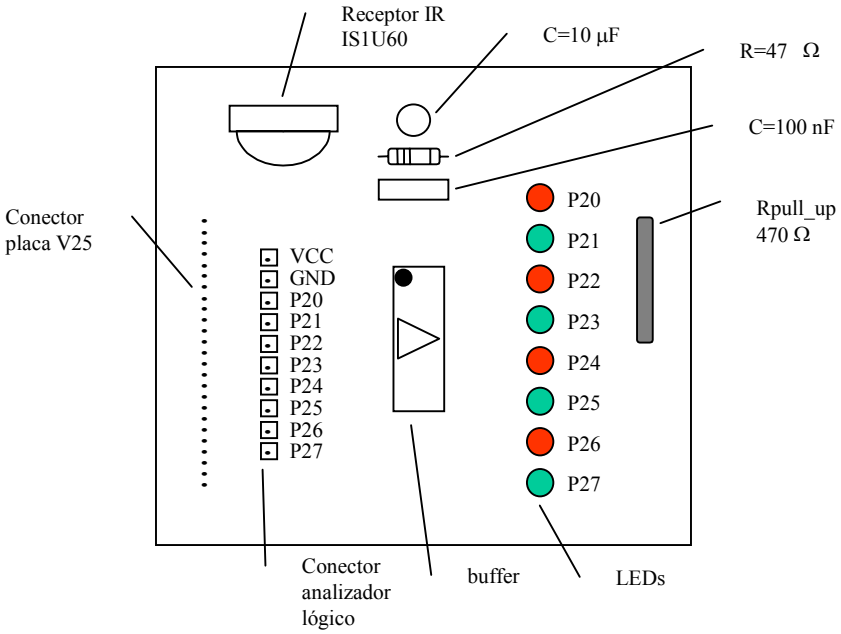

Figura 2 : Placa adicional

Esta placa adicional permite ver el estado de los 8 bits del puerto P2, cuando se hallan configurados como salida, mediante una serie de leds que se encienden cuando hay un

"1" lógico en cada uno de los bits del puerto P2. La placa, adicionalmente, cuenta con un integrado receptor de infrarrojos que realiza las mismas funciones que el circuito desarrollado en el módulo I y que se halla conectado al canal 1 de la UART del V25.

De este modo la placa de entrenamiento cuenta con los mismos interfaces de entrada salida que el robot real salvo por el hecho de que los motores se han substituido por leds y los cuentapasos por pulsadores.

5.5 Cuestionario sobre el V25

Antes de describir el trabajo a realizar en esta práctica se plantea un cuestionario para evaluar la comprensión de todos los temas descritos hasta el momento tanto en este manual como en los manuales del software Paradigm y la guía del V25.

Este cuestionario deberá incluirse como parte del estudio previo de la práctica.

Q1 ¿Cuántas líneas pueden configurarse como entrada o salida en el V25?

Q2 Indique que registros se han de modificar y que valores han de contener para configurar el puerto P2 como salida digital.

Q3 Indique que registro se ha de modificar y que valor ha de tomar para que el umbral de comparación de las líneas del puerto de teclado se establezca en 1.25V.

Q4 Indique cuantas interrupciones pueden generarse en el V25 indicando cuantas tienen registro de interrupción y cuantas son enmascarables.

Q5 Indique cual es el procedimiento para enmascarar o desenmascarar una interrupción que cuenta con registro de interrupción.

Q6 En la placa de prácticas la frecuencia f_x del oscilador de cuarzo es de 8Mhz. Indique que bits del registro PRC se han de modificar y a que valor para que la frecuencia de reloj f_{clk} sea de 4MHz y el intervalo de interrupciones de la base de tiempos sea de 2,05 ms.

Q7 Suponiendo que el registro PRC se ha programado tal y como se pedía en Q6, determine que registros se han de modificar y que valor han de tomar para que el temporizador 0 realice una interrupción cada 5ms y para garantizar que el temporizador 1 no realice ninguna interrupción.

5.6 Proyecto base

Para poder desarrollar un proyecto en el V25 usando las herramientas de desarrollo C/C++ de Paradigm es necesario crear un proyecto e incluir en él una serie de ficheros de configuración así como el código necesario para programar puertos e interrupciones.

Para simplificar el desarrollo se proporciona un conjunto de ficheros en forma de un archivo ZIP denominado "Fitxers_de_suport.zip" (disponible en Weble.upc.es). Este conjunto de ficheros constituye un proyecto base sobre el que podremos desarrollar el programa de control del robot.
En concreto contiene:

Dos ficheros timers.c y timers.h que contienen las siguientes rutinas:

void IniTimers(void) Inicializa la base de tiempos

void interrupt RSIBaseTemps(void) Rutina de servicio de interrupción de la base de tiempos

void Retard(int iTemps) Rutina que puede ser llamada por el programa principal y que detiene la ejecución del programa durante el número indicado de interrupciones de la base de tiempos.

Dos ficheros ports.c y ports.h que contienen las siguientes rutinas:

void IniPorts(void) Inicialización de los puertos PT y P2 tal y como requiere el robot.

void SetP2(BYTE byVal) Pone un determinado valor a la salida del puerto P2

BYTE GetPT(void) Lee el estado del puerto de teclado

Adicionalmente también hay ficheros de cabecera que contienen los nombres de los diferentes registros e interrupciones del V25.

Conviene que estudieis el contenido de los ficheros ports.c y timers.c para un mejor entendimiento sobre como se han de programar los puertos y las interrupciones.

Finalmente hay un fichero robot.c que únicamente contiene una función en blanco main() en el que se deberá incluir todo el programa principal desarrollado para nuestro robot.

5.7 Trabajo de laboratorio

Tal y como se ha indicado anteriormente, el objeto de esta práctica es familiarizarse con las herramientas de trabajo. Para ello desarrollaremos un programa sencillo. Empezaremos con un programa que apague y encienda los leds.

Abrid el proyecto “robot” en el entorno de desarrollo de Paradig e incluid el siguiente código dentro de la rutina main() :

```
BYTE b;              /* Variable local */

IniPorts();          /* Inicializa los puertos */
IniTimers();         /* Inicializa la base de tiempos */
enable();            /* Habilita interrupciones */
while (1)            /* Bucle infinito */
     {
     b=GetPT();      /* Leemos el Puerto de teclado */
     SetP2(b);       /* Volcamos en P2 */
     };
```

Este programa sencillo vuelca continuamente el contenido del puerto de teclado en el puerto P2. Observad que el programa principal tiene 3 partes. Comienza con la declaración de variables locales, le siguen las rutinas de inicialización, y, a continuación, se halla en cuerpo principal del programa. Dado que el sistema arranca con las interrupciones deshabilitadas, es preciso introducir una orden *enable()* después de acabar las llamadas a las rutinas de inicialización.

Después de introducir este programa deberéis compilarlo y cargarlo en la placa de desarrollo. Para poder cargar el programa el v25 debe cargar el programa monitor contenido en la EPROM por lo que se ha de pulsar el botón **reset**. Cada vez que pulsemos el botón **reset** nuestro programa se detendrá y el V25 pasará a estar controlado por el programa monitor contenido en la EPROM de la placa.

Una vez verificado el correcto funcionamiento del programa podemos empezar a desarrollar el primer programa que empleará los temporizadores.

Deseamos tener un led parpadeando moviéndose permanentemente de P20 a P27. La estructura del programa es la que se describe:

* Se declara una variable global de tipo BYTE y nombre **posicion**
* En el programa principal se inicializan los puertos y se inicia posición a 00000001b
* A partir de este momento el programa principal entra en un bucle infinito
* La RSI del temporizador 0 deberá, cada 100ms, encender el led indicado por **posición** si no hay ningún led encendido o apagar todos los leds si hay alguno encendido.
* La RSI del temporizador 1 deberá, cada segundo, actualizar la posición del 1 al bit de la izquierda de manera que se pase de 00000001b a 00000010b y así sucesivamente hasta 10000000b y a continuación otra vez a 00000001b.

Para desarrollar este programa se ha de añadir en IniTimers() el código necesario para inicializar los temporizadores 0 y 1, y se deberán de crear sus rutinas de servicio de interrupción. Recordad que para iniciar los temporizadores se han de realizar al menos 4 acciones: Actualizar el registro de control, dar valor de arranque al contador, fijar el vector de interrupción y desenmascarar la interrupción.

PRÁCTICA 6

PROGRAMA DE CONTROL DEL BRAZO DE ROBOT

1. INTRODUCCIÓN

Dentro del campo de las aplicaciones para μC se encuentran las de toma de datos analógicos y su procesamiento para gobernar periféricos. En esta última práctica se aplican las diversas técnicas y conceptos desarrollados a lo largo del curso para sintetizar un sistema completo que realiza una función de control.

La práctica que se propone consiste en la realización del programa de control de un brazo de robot con 4 grados de libertad. El sistema debe gobernar el brazo de un robot mediante un mando a distancia por infrarrojos.

El sistema completo consta de:

- Un receptor de infrarrojos que convierte la señal óptica emitida por el mando a distancia por infrarrojos en una señal digital que puede ser perfectamente leída por la UART del μC.
- Un mando a distancia gobernado por el PLD ispLSI 2032, que integra el bloque digital de control del mando a distancia, y envía el código de la tecla pulsada mediante una señal en formato serie asíncrono al emisor de infrarrojos (Módulo II).
- El μC V25 de NEC que ejecutará el algoritmo de control del sistema (Módulo III).

En esta práctica se pide al alumno el desarrollo del software del μC.

Tiempo para la realización de la práctica 5: 3 sesiones

2. TRABAJO PREVIO

a) Leer detenidamente la *Descripción del Sistema* y este manual (concretamente los puntos 3,4 y 5).
b) Realizar una descripción de las variables que se usarán en el programa y la descripción que realizará cada una de las funciones principales que se implementarán en el programa.
c) Describir los valores que deberán programarse en los registros de los temporizadores y del canal serie para que su función sea la deseada.
d) Desarrollar el código del algoritmo de control del sistema paulatinamente de manera que la mayor parte del tiempo en el laboratorio sea invertido en depurar código y NO en programar.

NOTA: El alumno será responsable directo de distribuir el trabajo previo entre las 3 sesiones de forma que en clase siempre optimice su tiempo con el entorno de programación del V25.

3. SUGERENCIAS SOBRE EL ALGORITMO DE CONTROL

En este apartado se describe un posible algoritmo de control. Es posible realizar algoritmos de control completamente distintos que realicen las mismas funciones en el robot. Caso de desear realizar un algoritmo distinto se recomienda discutirlo antes con el profesor de prácticas.

El estado por defecto del sistema debe de ser el de espera a que se pulse una tecla del mando a distancia por infrarrojos. A este estado se llega después de ejecutar la rutina de **RESET** del sistema en la que se deberán inicializar las variables del algoritmo, definir constantes, configurar todos los puertos y registros de control del V25 para cumplir con las especificaciones que aparecen en la *Descripción del Sistema* y llevar el brazo del robot a la posición de **INICIO**.

Finalizado el **RESET** del sistema, el algoritmo ejecuta un bucle cerrado esperando que se pulse una tecla del mando a distancia, y tras su detección se deberá ejecutar la rutina correspondiente.

3.1. ALGORITMO PRINCIPAL

Un posible flujograma del algoritmo de control vendría dado por los siguientes bloques:

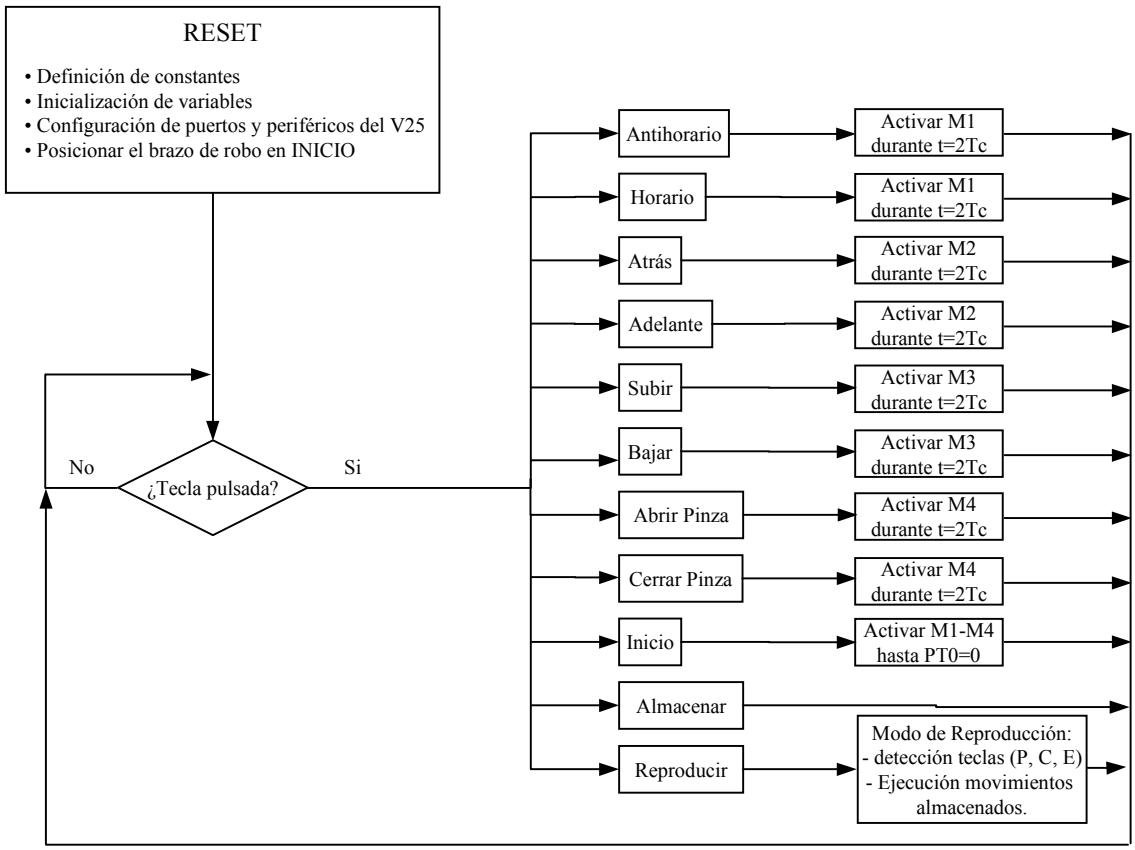

***Fig. 1**:Flujograma básico del algoritmo de control. Tc es el tiempo mínimo entre dos comandos enviados sucesivamente por el mando a distancia (20 ms).*

Es importante notar que el control de los motores debe hacerse mediante interrupciones. En concreto, el sistema no puede quedar en estado de espera mientras se mueve un motor ya que dejarían de procesarse las órdenes del teclado. Las órdenes del teclado se envían a una velocidad de 512 baudios lo cual corresponde a que una palabra de 10bits (incluyendo start y stop) se repite cada 20ms aproximadamente si se deja la tecla correspondiente apretada.

Dado que las órdenes de movimiento son discretas en el tiempo, el motor debe moverse al menos 20ms cada vez que se recibe la orden para dar tiempo a que se reciba una repetición sin que el motor se haya parado. Para evitar sucesiones de arranque-parada se propone que cada vez que se reciba una orden el motor se active durante 40ms. Transcurrido ese tiempo el motor debe pararse si no se ha recibido otra orden.

El programa no puede estar en estado de parada durante los 40ms de actuación del motor dado que no se procesarían las órdenes del canal serie durante ese tiempo. Es por ello que se propone activar el motor desde el programa principal y pararlo desde la RSI del Timer0, en modo de intervalo único. La figura 2 muestra un ejemplo de funcionamiento:

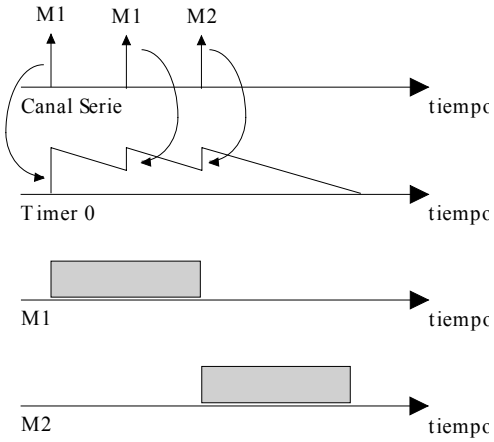

Fig. 2 – Propuesta de control de motores

En el ejemplo, la primera orden que se recibe en el canal serie es mover el motor 1, el programa principal enciende este motor y activa el temporizador 0 en modo de intervalo único. El programa principal vuelve al estado de esperar una nueva orden por el canal serie.
Al cabo de 20ms se recibe una nueva orden de M1 puesto que no se ha soltado la tecla. Dado que el temporizador se programó para 40ms, se hallará en la mitad de sus cuentas.
Con esta nueva orden se vuelve a activar M1 y se refresca otra vez el contador al valor máximo de 40ms.
Al cabo de otros 20ms se recibe otra orden, pero esta vez de activar M2, el programa principal activa M2 y refresca el timer 0 volviendo después al modo de espera de órdenes.
Finalmente, transcurridos 40ms desde la última órden, al no volverse a refrescar el temporizador, este llega a cero y se genera una interrupción. La RSI del Timer0 para entonces todos los motores.

3.2 PRINCIPALES RUTINAS DEL ALGORITMO

- **INICIO.** Se activarán los cuatro motores en modo de máxima velocidad con las direcciones *Antihorario*, *Atrás*, *Subir* y *Abrir Pinza* hasta que todos ellos alcancen sus finales de carrera (PT0=0). A continuación se detendrán los motores y se pondrán todos los registros de posición a cero, quedando el sistema en disposición de recibir órdenes por el receptor de infrarrojos.

- **PROGRAMA PRINCIPAL (MAIN).** En esta rutina se comprobará periódicamente si ha sido recibida alguna orden por el canal serie y se dirigirá el flujo del programa hacia la rutina correspondiente. Cuando se reciba una orden concreta de movimiento de un motor (antihorario, horario, arriba, abajo, etc.) se deberá poner en marcha el motor correspondiente en el sentido indicado. Así mismo se deberá activar el temporizador 0 en modo único de 40ms. Después de ello el programa deberá volver al modo de espera de órdenes por el canal serie.

- **ALMACENAR.** Esta rutina deberá almacenar la posición actual del brazo del motor. Se sugiere utilizar una variable del tipo *array* para almacenar las distintas posiciones que seleccione el usuario y que después desee reproducir. Asimismo, se deberá evitar almacenar repetidamente la misma posición de manera consecutiva debido a que se recibe varias veces la misma orden en la misma posición. Cuando el usuario pulsa la tecla de almacenar se puede recibir la orden de almacenar repetidamente, aún cuando la posición del brazo no ha variado. Esto provocaría un llenado rápido del *array* con información inútil. Por tanto, cada vez que se detecte la orden de almacenar una posición se deberá comprobar previamente que ésta no coincide con la última posición que haya sido introducida en el *array*.

- **BORRAR.** Se borrarán todos los datos de los pasos guardados en el *array*. La forma mas eficiente de hacerlo puede consistir en poner a cero la variable puntero del *array*.

- **REPRODUCIR.** Primero se deberá ejecutar la rutina **INICIO** para partir de la posición de referencia. Después se deberán rastrear las posiciones del array almacenadas de manera que se pase por todas las posiciones. Para cada posición del array se deberán activar todos los motores que se hallen fuera de la posición objetivo durante el tiempo necesario para llegar a ésta. A continuación se comprobará si existen más posiciones almacenadas en el *array* y si fuera así se repetirá el proceso anterior. Para evitar que con la inercia del motor se produzcan pasos de más no deseados, mientras se avanza y antes de las últimas 5 cuentas/pasos se pasará al modo de velocidad lenta.

  No se abandonará la rutina **REPRODUCIR** hasta que se haya alcanzado la última posición almacenada.

3.3 ASIGNACIÓN DE RECURSOS DEL V25 PARA LA APLICACIÓN

- ***Timer0 (40 ms)***

  Este temporizador se usará para controlar el movimiento del motor cuando se dan órdenes directas con las teclas. En este caso trabajará en modo de intervalo único. Cuando se genere la RSI, significará que han pasado 40ms sin ninguna orden y los motores deberán parase.

- ***Timer1 (20 ms)***

  Este temporizador será utilizado en modo de intervalo repetitivo para monitorizar la posición del brazo articulado. Con un período de 20 ms se observarán las señales de los cuenta pasos (CV1-CV4) y si se observa un cambio, se actualizarán convenientemente las variables de estado que indican la posición del robot teniendo en cuenta el sentido actual de giro del motor correspondiente.

  La RSI de este temporizador se usará también para activar los motores 20ms de cada 100ms caso de activarse la velocidad lenta por estar cerca de la siguiente posición de la tabla en modo reproducción.

- ***UART (canal 0)***

  La recepción de órdenes del mando a distancia será por interrupciones. Esta rutina únicamente deberá recoger el dato recibido por la UART y ponerlo a disposición del programa principal a través de una variable global.

  Opcionalmente, se puede considerar una estructura de *buffer* circular que permita almacenar un máximo de 10 comandos sin que haya pérdida de ninguno aunque el programa principal no recoja los datos. Este tipo de estructuras de datos es utilizado por ejemplo por la interrupción de teclado del PC.

  Existe una interrupción de la UART cuando ha habido un error en recepción. Se recomienda su captura, para que así el programa tenga conocimiento de cuándo un dato ha sido mal recibido.

Se valorará especialmente que estas rutinas sean de **tiempo de ejecución mínimo**, especialmente las dos primeras (Timer0 y Timer1). Se puede considerar, opcionalmente, incluso el realizar algunas partes en ensamblador (mediante la directiva de compilador de C: 'asm' o directamente, incluyendo un fichero en código fuente ensamblador).

4. CONVENIOS Y ESPECIFICACIONES ADICIONALES

En el modo de reproducción puede presentarse algún problema cuando alguna de las posiciones almacenadas se encuentra muy cerca de alguno de los finales de carrera de los motores. Supongamos, a modo de ejemplo, que desde la posición de inicio movemos el motor M1 (el de giro del brazo) aproximadamente 180º en sentido horario, llegando a tocar el final de carrera FC5. En esa posición almacenamos, y el robot ha detectado 200 pasos (posición A1 de almacenamiento). A continuación realizamos otro(s) movimiento(s) y volvemos a almacenar (posición A2 de almacenamiento). Si enviamos el comando 'Reproducir', el brazo volverá a la posición de inicio y activará el motor M1 en giro horario hasta que haya recorrido 200 pasos. Ahora bien, si por la imprecisión propia de los cuenta pasos, se llega al final de carrera FC5 en el paso 199 (p.e.), la protección hardware de la placa interfaz corta la alimentación en sentido horario del motor M1, independientemente de lo que indique el microcontrolador. Por otro lado, la rutina de reproducción esperará en bucle infinito a que se produzca el paso 200, que nunca llegará, lo cual generará un bloqueo total del programa de control.

Para evitar este problema existen varias alternativas. Una de ellas consiste en considerar que cuando no se ha detectado ningún cambio en la posición de uno de los motores durante 300 ms después de activarlo se considere que el motor ha llegado al fin de carrera en esa dirección. Dado que sabemos aproximadamente el número de pasos disponibles en cada motor, es posible generar una señal de error por bloqueo si se detecta que un motor no se puede mover cuando se halla lejos de sus dos finales de carrera.

Opcionalmente se propone utilizar el LCD para presentar información de la posición del brazo de robot en los 4 motores y para indicar las órdenes recibidas y dar mensajes de error.

5. GUÍA PARA LA REALIZACIÓN DEL PROGRAMA

Se aconseja y también serán valorados los siguientes puntos:

a) El programa deberá ser lo más **claro** posible, incluyendo algunos comentarios clarificadores.

b) Deberá ser modular, es decir, estructurado en rutinas de funciones relativamente simples de manera que funciones complejas se subdividan en rutinas simples, claramente identificables.

c) Se deberá minimizar el tiempo de ejecución de las rutinas de interrupción. Para ello se debe tener en cuenta que para que un código sea más rápido es posible que requiera ser más largo. Por ejemplo, si se ha pensado en hacer un bucle que se ejecuta 5 veces, puede ser mejor realizar 5 copias del código (así nos ahorramos las instrucciones de bucle).

d) Se deberá minimizar el uso de variables globales. Los intercambios de variables entre rutinas del programa principal deberá ser o por referencia o por valor, evitando el paso de información a través de variables globales. Las variables utilizadas para comunicar información entre rutinas de interrupción y el programa principal por fuerza han de ser globales y se han de deshabilitar las interrupciones mientras se modifican en el programa

principal. Las variables utilizadas dentro de las rutinas de interrupción y cuyo valor no ha de ser leído por el programa principal conviene que sean del tipo '*static*' o globales para garantizar que conserven su valor entre las distintas llamadas a la RSI.

e) El programa deberá ser paramétrico, de manera que la mayor parte de tiempos o parámetros de funcionamiento del programa sean **constantes** definidas al principio del programa. Por ejemplo, los parámetros que especifican la frecuencia de interrupción de los *timers* pueden ser constantes definidas al principio del programa principal, de manera que si en algún momento se desea modificar la frecuencia, únicamente se deba tocar ese parámetro.

Se recomienda realizar el programa comenzando con objetivos modestos para luego irlos ampliando paulatinamente. Un ejemplo podría ser el siguiente:

1) Rutina de inicio. El programa deberá activar todos sus motores en la dirección del final de carrera y esperar a que éste se active. A continuación se deberán apagar todos los motores. Ello implica la implementación de la rutina de *Reset*.

2) Conseguir que el robot obedezca a las teclas de movimiento (1-8). En esta etapa se añadiría la inicialización y uso de la RSI del canal serie 1 y las bifurcaciones del programa principal que controlan las distintas teclas. También se incluiría la programación y la RSI del Timer0 que permite apagar los motores cuando han pasado 40ms sin recibirse órdenes.

3) Implementación de los contadores de pasos. Se ha de verificar que se cuentan correctamente los pasos de los motores cada vez que un cuenta pasos cambia de estado de "1" a "0" o de "0" a "1". Ello implica implementar la funcionalidad del Timer1.

4) Añadir la capacidad de almacenamiento y reproducción (sin modo de velocidad lenta).

5) Ampliar el punto anterior para incluir el modo de velocidad lenta (añadiendo código a la rutina del Timer1).

6) Ampliación a especificación adicional punto 4. Comprobación física.

Las fases 1 a 3 pueden verificarse mediante comprobación física en la placa de evaluación disponible para cada grupo. Las fases 4 a 6 son más complejas de evaluar sin emplear el robot completo.

6. TRABAJO DE LABORATORIO

El trabajo de laboratorio consiste en construir todo el programa paso a paso. Para ello se pueden seguir las recomendaciones del punto anterior.