EJERCITO ARGENTINO
FACULTAD DE INGENIERÍA DEL EJÉRCITO
"Grl Div D Manuel Nicolás Savio"

MATERIA: Técnicas Digitales II

CURSO: V – Plan 2015 Informática

DOCENTE: Ing. Daniel Steiner.

JEFE TRABAJOS PRÁCTICOS: Ing. Ariel Dalmas Di Giovanni.

TRABAJO PRÁCTICO Nro.: 2

TÍTULO: "*Manejo de periféricos y dispositivos externos*"**.**

ALUMNOS: ALFONSO, DELCOURT, DONIS Y FUENTES

FECHA DE REALIZACIÓN: 05/06/2021

FECHA DE FINALIZACIÓN: 09/06/2021

FECHA DE ENTREGA: 11/06/2021

CALIFICACIÓN:

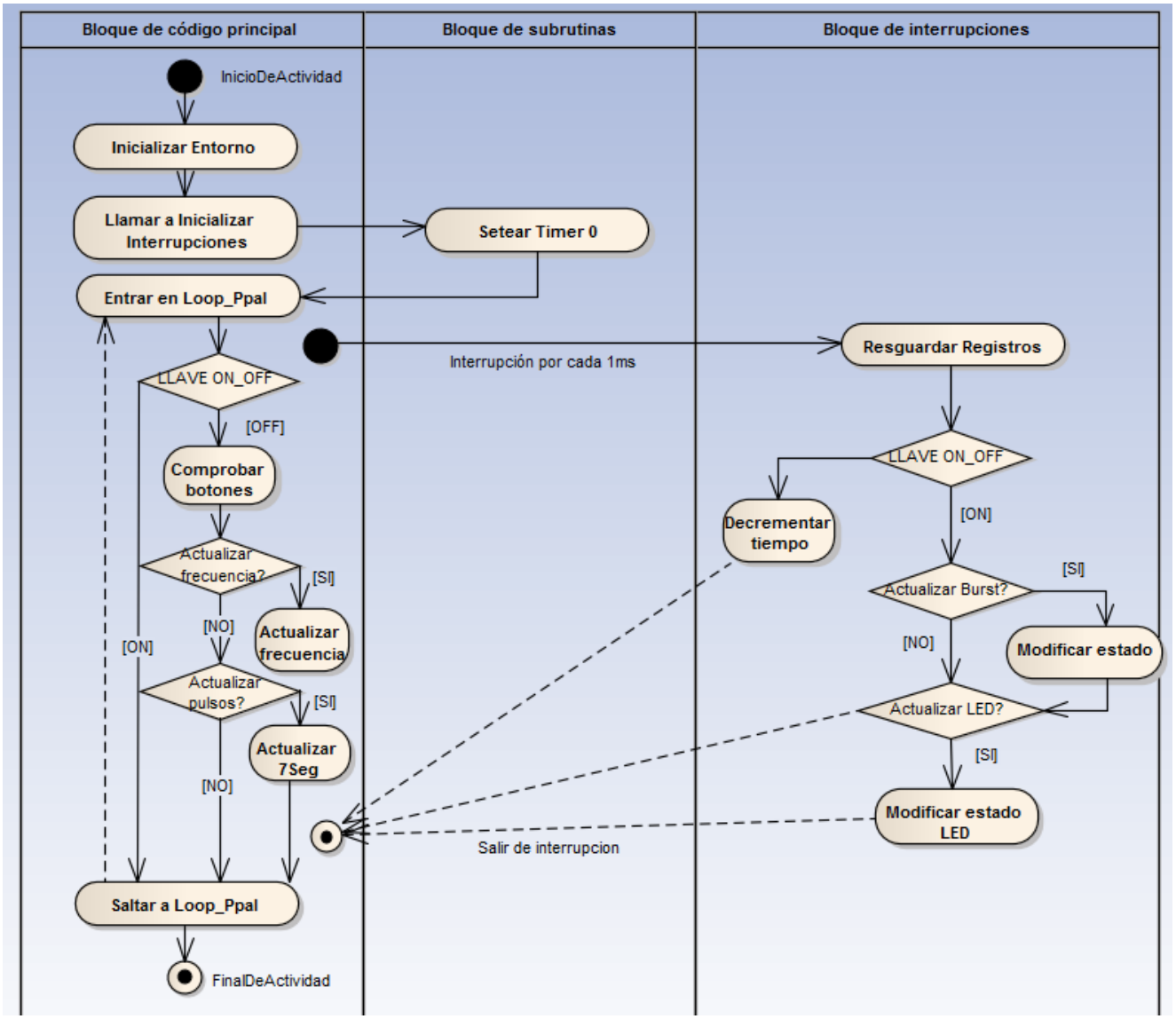
Introducción

En las distintas aplicaciones donde se utilizan microcontroladores es frecuente la necesidad de interactuar con distintos dispositivos externos como ser: tipos de sensores (digitales, analógicos, etc), actuadores (steppers, servomecanismos, etc), conversores AD y DA, etc. A su vez, es también común la necesidad de contar con una interfaz con el usuario a través de botones, llaves y presentar la información en forma visual mediante luces, displays; y también estímulos sonoros, etc.

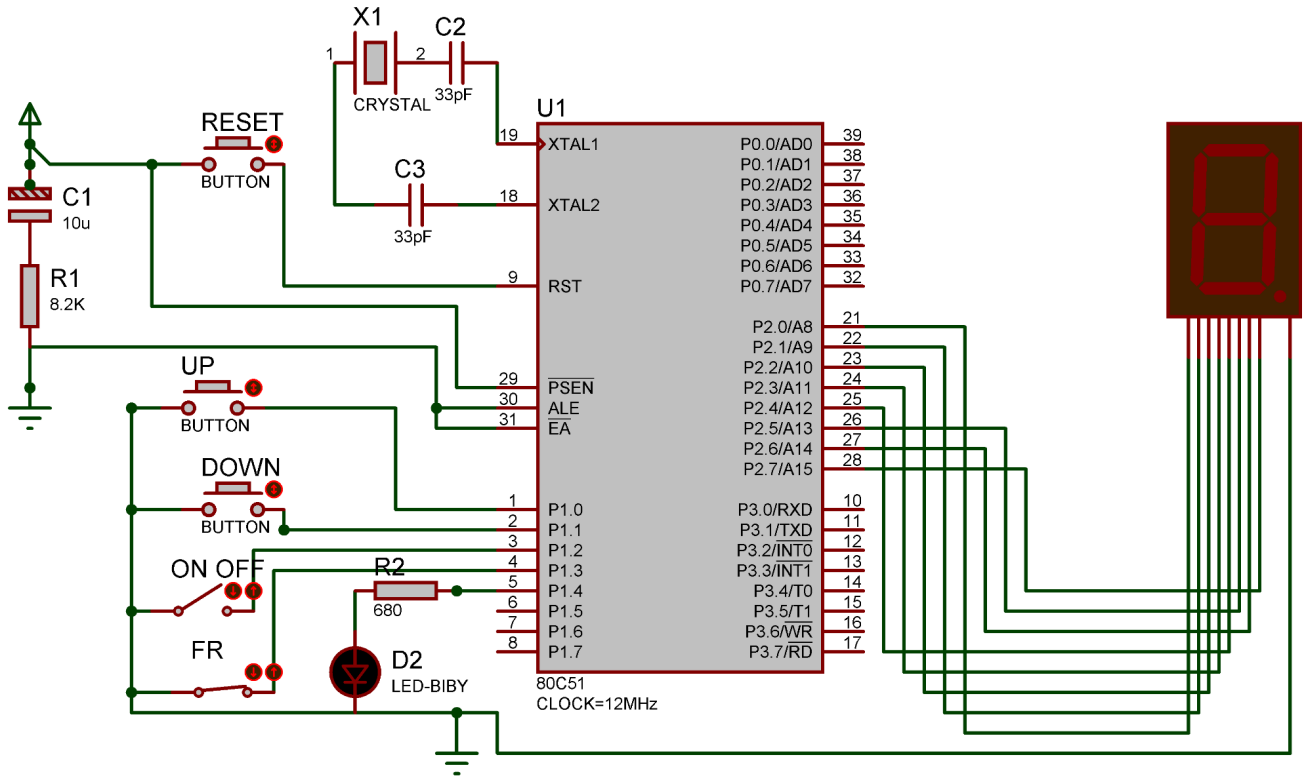
Por otro lado, los microcontroladores están integrados por diferentes periféricos que mejoran la escala de integración de un sistema e incrementa la versatilidad de aplicaciones en las que se puede utilizar un mismo microcontrolador.

Módulos de software

Diagrama de flujo:



Diseño esquemático:



Análisis del comportamiento:

Definición de constantes

Se decidió cargar los registros del timer utilizando **constantes** definidas al principio del archivo para hacer este más legible.

Ya que sólo se utiliza la interrupción del timer 0, la constante para el registro **IP** se define en 0, dándole prioridad baja a todas las interrupciones.

Luego, para la constante del registro **IE** se ponen en 1 los bits correspondientes a **EA** para la habilitación de interrupciones globales y **ET0** para las interrupciones del timer 0.

La constante del registro **TMOD** se la define en 01H, indicando que se utiliza el timer 0 en modo 1.

En la constante del registro **TCON** sólo se pone en 1 el bit **TR0**, que permite el inicio del timer 0 y lo pone a contar tiempo.

Luego de esto se definen constantes varias para el cálculo del valor de recarga para el timer.

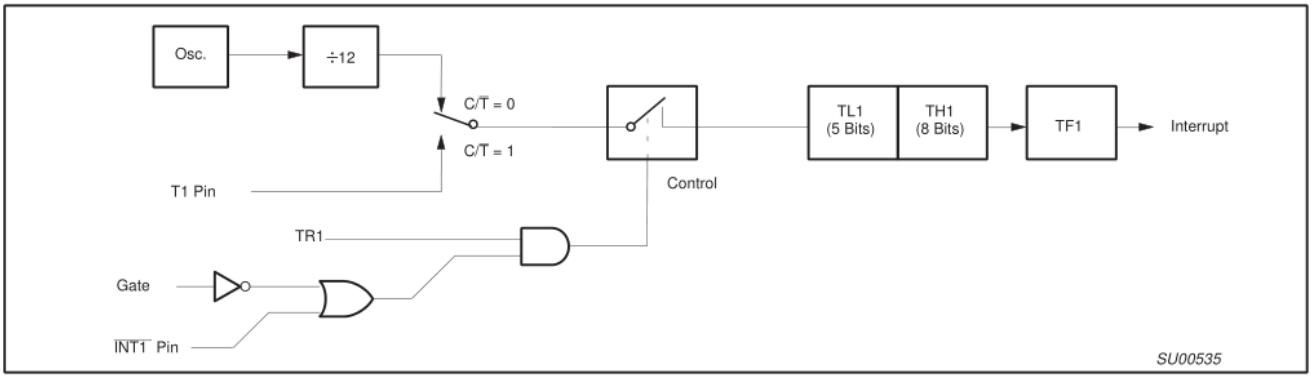


Figure 7. Timer/Counter Mode 0: 13-Bit Counter

Manual de hardware, página 7

Se utiliza un cristal de 12 MHz como es indicado en la constante **FRECUENCIA_CLOCK**.

La constante **DIV_CLOCK_INT** corresponde al valor del divisor interno de frecuencia que se puede ver en los diagramas del timer en el manual del microcontrolador.

En la constante **FREC_PERIF** se almacena el valor de la frecuencia de entrada del timer, que es 12Mhz/12 = 1Mhz. Esto quiere decir que el timer contará ciclos de 1μs.

En la constante **TICK_TMR0** se define el período según el cual se quiere realizar la interrupción en ms. Ya que el ancho de pulso para los pulsos del burst debe ser de 1ms, se le almacena el valor 1.

Con estos valores, se calcula la cantidad de ciclos que debe contar el timer y se almacena en la constante **VALOR_TIMER**. En este caso se quieren contar 1000 pulsos de 1μs para realizar la interrupción cada 1ms, por lo que se almacena en la constante el valor 1000. Es debido a este valor que se tomó la decisión de utilizar el timer en **modo 1** y no en autorecarga; ya que en éste último caso sólo se disponen de 8 bits y no es posible cargar el valor de 1000d, es decir 03E8h.

La constante **CICLOS_TMR0_DETENIDO** contiene la cantidad de ciclos que el timer se encontrará detenido en la ISR mientras se lo reconfigura. Esto podrá verse mejor más adelante, en la rutina de interrupción. En este caso, el valor almacenado es 7.

Finalmente se calcula el valor de recarga del timer. Ya que la cuenta del timer es ascendiente, se le debe cargar el valor complementario de lo que se quiere contar. El timer realizará la interrupción al desbordarse del valor FFFFh. En este cálculo se deben considerar los ciclos en los que el timer estará detenido en la ISR indicados por CICLOS_TMR0_DETENIDO. Es por esta razón que en la constante **RECARGA_TMR0** se almacena el valor FFFFh - VALOR_TIMER + CICLOS_TMR0_DETENIDO. En este caso, el valor final es de FC1Eh.

Luego de definir todas las constantes para la configuración del timer, se definen nombres para los puertos en los que se encuentran los botones, llaves, LED, salida de burst, y el 7 segmentos.

Finalmente se definen variables que se utilizarán para definir la cantidad de milisegundos contados por el timer (la cuenta en la variable MILISEGUNDO es descendiente), la cantidad de pulsos que deben hacerse, el tiempo de pulso del LED, el período de burst, y el período de verificación para las llaves y botones.

Luego se definen 2 bits para la comprobación del estado de la llave de frecuencia en el sector de memoria para guardar bits en forma individual, LLAVE_FR_ANT y XOR_P.

También se definen tres variables más que se utilizan para resguardar el contexto antes de entrar a la interrupción.

main

Estas son las primeras instrucciones en ejecutarse cuando se realiza un **RESET**. Se carga la variable **MILISEGUNDO** con el valor máximo posible de 251, que corresponde al período del LED cuando se utiliza la frecuencia de 2Hz, que es la frecuencia por defecto. Se le suma además un 1 para diferenciar esta situación como el inicio del período del LED. También se define la variable **CANT_PULSOS** en 0 para que el 7 segmentos se inicie mostrando el valor mínimo de pulsos, y la variable **TIEMPO** con su valor máximo de 100 para verificar el estado de las llaves y botones cada 100ms.

Finalmente se llama a la subrutina **InitPerif_Interrupciones** para configurar el timer y la habilitación de interrupciones antes de entrar al loop principal.

loop_ppal

Una vez iniciado el timer, el microcontrolador quedará en el loop principal infinito hasta que se realice una interrupción y deba atenderla entrando a la rutina de interrupción.

Mientras la llave de ON/OFF esté encendida y se estén realizando los bursts, no se comprueba nada y sigue en el loop infinito.

En el caso de que la llave de ON/OFF esté apagada, dentro de este loop se realiza la **verificación** de la **llave** de **frecuencia**, los **botones**, y la **actualización** del valor en el **7 segmentos**. Estos valores sólo pueden ser modificados si la llave de ON/OFF se encuentra apagada para no interferir con el burst o el período del LED una vez que se encienda la llave y se envíen los pulsos.

Se toma como frecuencia por defecto la de 2Hz, y esta será cambiada sólo ha cambiado el estado de la llave desde la última entrada al loop_ppal, cambiando los períodos del burst y el LED. La comprobación del cambio de estado de la llave se hace mediante una operación **XOR** realizada con ANL y ORL, utilizando el carry para mover el bit correspondiente del estado de la llave y estado anterior de la llave. En este caso no es posible utilizar la operación XRL ya que la comparación debe realizarse sobre 2 bits

y no 2 bytes, lo que no es soportado por esta operación, como se puede ver en el manual del microcontrolador.

Luego se realizan las comprobaciones de los botones para verificar si se cambió la cantidad de pulsos.

Ya que apretar un botón es una acción que lleva más de 1ms, se decidió agregar un **contador**, la variable **TIEMPO**, para que estas **comprobaciones** se realicen cada **100ms**. Al utilizar los pulsadores, cuando se presiona (cambian de estado) es posible que la propia fricción produzca que la señal se vuelve inestable por un pequeño tiempo en el cual se produce el cambio (jitter). De no hacer esto, presionar el botón 1 vez podría llevar a varios incrementos de la cantidad de pulsos. Si aún no pasaron 100ms desde la última comprobación, se **sale** de la rutina de **interrupción** sin hacer nada más.

En el caso de que sea momento de realizar las **comprobaciones**, se recarga el valor del contador TIEMPO, y se verifican los estados de los botones.

Dentro de las verificaciones para los **botones** con los que se define la cantidad de pulsos se programó la cuenta como **cíclica** de 0 a 9. Es decir que si el usuario tiene un 9 en CANT_PULSOS y presiona el botón de UP, la cuenta volverá a 0. Lo contrario aplica al caso en el que se encuentra un 0 en CANT_PULSOS y se presiona el botón DOWN.

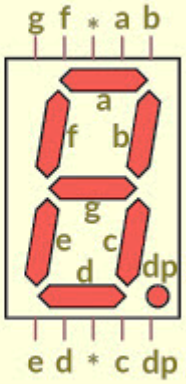
En estas también se carga el valor **REPETICION**, que indica la cantidad de veces que se debe subir/bajar la salida de burst. Se toma el doble de valor que en CANT_PULSOS ya que se están teniendo en cuenta tanto las veces que se debe fijar la salida en alta como baja (se debe subir una vez y bajar una vez para obtener un pulso completo).

La actualización del valor en el **7 segmentos** se explica con mayor detalle en la sección “Actualización del valor en el 7 segmentos”.

Luego de realizar las comprobaciones y el refresco si es necesario vuelve a saltar al inicio del loop.

Actualización del valor en el 7 segmentos

Para la actualización del 7 segmentos se comienza por crear una **tabla** en la memoria de código donde estén almacenados los **valores** del 7 segmentos. Para acceder a un valor de la tabla lo que se hace es posicionar el **DPTR** en la primera posición de la tabla y luego sumándole la cantidad de filas dando así el número de índice del dato al cual nosotros queremos acceder.



Cátodo Cómun								
Común*	Número	g	f	e	d	c	b	a
GND	0	0	1	1	1	1	1	1
GND	1	0	0	0	0	1	1	0
GND	2	1	0	1	1	0	1	1
GND	3	1	0	0	1	1	1	1
GND	4	1	1	0	0	1	1	0
GND	5	1	1	0	1	1	0	1
GND	6	1	1	1	1	1	0	1
GND	7	0	0	0	0	1	1	1
GND	8	1	1	1	1	1	1	1
GND	9	1	1	0	1	1	1	1

Valores utilizados para TABLA_7SEGMENTOS

Mediante la subrutina **ACTUALIZAR_7SEG** nosotros posicionamos el DPTR en el inicio de la tabla y buscamos en la tabla el valor del DPTR más el valor de la cantidad de pulsos actual del burst, dándonos así como resultado el valor que necesita el 7 segmentos para mostrar la cantidad de pulsos actuales. Por último enviamos ese valor a P2 para actualizar el 7 segmentos. Se pueden encontrar saltos a esta subrutina cada vez que se modifica la cantidad de pulsos.

InitPerif_Interrupciones

En esta subrutina se **inicializa** el **timer** y **habilitan** las **interrupciones** según las constantes mencionadas en la sección “Definición de constantes”.

Primero se inicializan los valores de recarga para el timer según lo calculado, cargando las partes baja y alta de la constante en **TL0** y **TH0**. Cabe aclarar que en este proceso se utilizan las instructivas del compilador **#LOW** y **#HIGH** para tomar los valores alto y bajo de la constante, pero no requieren de más ciclos de máquina para hacerlo ya que esto es resuelto por el compilador.

Luego de esto se inicializan **TMOD**, **TCON**, **IP** e **IE** según las **constantes** definidas anteriormente. Es importante notar que se inicia el timer y se habilitan las instrucciones a último momento, antes de salir de la subrutina, ya que debe estar el timer configurado antes de iniciarlo.

Isr_Tmr0

Esta es la rutina de servicio de **interrupción** para el timer 0. Las instrucciones dentro de ella serán ejecutadas cuando se realice una interrupción cada 1ms, como es indicado en el salto largo a esta etiqueta luego de definir el vector de interrupción al principio del programa.

Lo primero que se realiza aquí es **resguardar** el **contexto** antes de realizar lo pertinente a la rutina de interrupción.

Ya que se utilizarán el acumulador, R2, R3 y R4, los valores dentro de estos deben **resguardarse** antes de ser utilizados. Adicionalmente, ya que se van a realizar sumas, lo que podría modificar el bit de carry, también se resguarda la PSW. Esto se hace guardándolos en el **stack**, es decir que mientras se ejecuta la rutina de interrupción el stack contendrá la dirección de regreso al main, y además los registros que se resguardaron. Es por esto que antes de retornar de la interrupción se sacan de la pila estos valores con la instrucción POP.

Luego se realiza la recarga del timer. Antes de cargar los registros nuevamente, se **deshabilitan** las **interrupciones** y la **cuenta** del timer 0. Es a partir de este momento que se empiezan a contar los ciclos utilizados para almacenar el valor en la constante **CICLOS_TMR0_DETENIDO**. Con el timer parado, se recargan los valores de **TL0** y **TH0** sumándoles al valor de recarga lo que ya tengan dentro, ya que es posible que se hayan acumulado algunos ciclos en el tiempo desde que se entró a la rutina de interrupción. Una vez **recargados** los **registros** del **timer**, se vuelve a iniciar la cuenta y se habilitan las interrupciones. Es hasta aquí que se cuentan los 7 ciclos que está detenido el timer.

Luego de esto, comienza el código que realizará las operaciones pedidas en el enunciado del trabajo.

La primera **verificación** comprueba si la **llave** de **ON/OFF** se encuentra encendida. A partir de aquí se definen dos **flujos**:

- Si la llave está **apagada** (**0**), todavía no deben realizarse los burst ni encenderse el LED, sino que el usuario está configurando la cantidad de pulsos y la frecuencia utilizando la llave de frecuencia y los botones. Es decir que si la llave de ON/OFF se encuentra apagada deben realizarse las comprobaciones del estado de los botones.
- Si la llave está **encendida** (**1**), significa que deben iniciarse los burst y encenderse el LED, o se está en el proceso de realizar los pulsos. Es decir que si la llave de ON/OFF se encuentra encendida, se debe contar la cantidad de MILISEGUNDOs transcurridos, y REPETICIONes que se hayan hecho de pulsos, manteniendo la frecuencia de parpadeo del LED.

Se analiza el código de los dos flujos por separado:

Llave ON/OFF apagada (0)

Si la llave de ON/OFF está **apagada**, de la instrucción que verifica el estado de la llave de ON/OFF se salta a la etiqueta **COMPROBAR_LLAVES**.

Aquí se ponen en 0 la salida del burst y el LED ya que la llave está apagada, y decrementa la variable TIEMPO que se utiliza en el loop principal como período de tiempo para comprobar los botones. En caso de que hayan pasado los 100ms de TIEMPO, recarga el valor y sale de la interrupción.

Llave ON/OFF encendida (1)

Si la **llave** de ON/OFF está **encendida**, se deben realizar los pulsos de **burst** o se está en proceso de esto.

Primero se verifican los valores de REPETICION y MILISEGUNDO. Estas indican la cantidad de pulsos que faltan realizarse y la cantidad de milisegundos contados respectivamente.

Luego, se divide el flujo en cinco situaciones distintas. Las instrucciones para cada una están bajo etiquetas indicativas de la cantidad de **milisegundos contados** hasta el momento de la interrupción, tomando la frecuencia por defecto (2Hz).

Nota: dentro de estos bloques de código se utilizaron comentarios inline. Si bien se tiene en cuenta el comentario del profesor indicando que no son nativos de assembler, se los dejó incluidos en el código a fin de darle mayor claridad y permitir que los registros utilizados se puedan identificar más fácilmente al leer el código.

- ***Inicio del período del LED*** *(y burst)*

  La etiqueta que corresponde a esta situación es **MIL0**.

  Se ejecutan las instrucciones bajo esta etiqueta en el caso en que **MILISEGUNDO = 251**(2Hz)**/126**(10Hz) o cuando se salta aquí desde **MIL0**. Este momento es el inicio del período del LED, lo que también se corresponde con el período de un nuevo burst de pulsos.

  Se verifica si se eligió como cantidad de pulsos 0, y si es el caso sólo decrementa MILISEGUNDO. Si se eligió realizar más de 0 pulsos, salta a MIL0_ACTIVAR.

  Aquí se fijan en 1 la salida del LED y se comienza el burst. Luego se decrementan los dos contadores correspondientes y se sale de la rutina.

- ***Inicio del período del burst****, pero no del LED*

  La etiqueta que corresponde a esta situación es **MIL100**.

  Se ejecutan las instrucciones bajo esta etiqueta en el caso en que **MILISEGUNDO = 150**(2Hz)**/75**(10Hz). Este momento es el final del período del burst, y debe comenzar uno nuevo.

  Se verifica si se eligió como cantidad de pulsos 0, y si es el caso sólo decrementa MILISEGUNDO. Si se eligió realizar más de 0 pulsos, salta a MIL100_ACTIVAR.

  Aquí se inicia el nuevo burst, se recarga la variable REPETICION, y se decrementan los dos contadores correspondientes antes de salir de la rutina.

- ***Final del ancho de pulso del LED***

  La etiqueta que corresponde a esta situación es **MIL250**.

Se ejecutan las instrucciones bajo esta etiqueta en el caso en que **MILISEGUNDO = 0**. Este momento es el final del ancho de pulso del LED, y debe complementarse su valor.

En el caso que el LED esté apagado, comienza un nuevo período saltando a MIL250.

Si está encendido, lo apaga y recarga la variable MILISEGUNDO para que nuevamente se cuenten 250/125 ciclos. Luego decrementa el contador correspondiente y sale de la rutina.

- ***Final del burst***

  La etiqueta que corresponde a esta situación es **REP0**.

  Se ejecutan las instrucciones bajo esta etiqueta en el caso en que **REPETICION = 0**. Este momento es el final del burst luego de realizar todos los pulsos, y se debe esperar al final del período de burst para comenzar uno nuevo.

  Simplemente apaga la salida de burst y decrementa el contador correspondiente antes de salir de la rutina.

- ***Duración del burst***

  La etiqueta que corresponde a esta situación es **COMPLEMENTO**.

  Se ejecutan las instrucciones bajo esta etiqueta mientras todavía queden pulsos para realizar en el burst.

  Complementa el estado de la salida de burst para realizar la subida o bajada de un pulso y decrementa los contadores correspondientes antes de salir de la rutina.

Con esto, se cubren todos las operaciones requeridas en el enunciado del trabajo.

Documentación utilizada

- Manual de Arquitectura – Familia 80C51
- Conjunto de Instrucciones y Guía del Programador – Familia 80C51

Pruebas y ensayos.

Las mismas se realizaron utilizando la aplicación IAR Embedded Workbench y fueron demostradas en las explicaciones dadas en el punto de Módulos de Software.

Conclusiones

A partir de la realización del presente trabajo práctico se logró generar un código de assembler basado en la arquitectura 80c51 no bloqueante, es decir, que no se ve interrumpido por las subrutinas utilizadas

por distintos sectores del sistema al ser requeridas en simultáneo, apoyándonos en la utilización de interrupciones del módulo de Timer del microcontrolador estudiado.

Por otro lado, se pudo verificar la facilidad que otorga utilizar un set de instrucciones amplio que, por ejemplo, permita todas las operaciones lógicas sobre variables booleanas, ya que de no poseerlas se puede llegar a presentar la situación de comprometer mayor cantidad de ciclos de procesamiento para una misma tarea.

Asimismo, pudimos afirmar conceptos estudiados durante la cursada y comprobar el correcto funcionamiento del mismo a partir de la simulación realizada con el IDE provisto.