

Práctica 10

Uso de Temporizadores/Contadores del uC ATmega1280

Objetivo: Mediante esta práctica el alumno aprenderá la programación y uso básico del Temporizador 0 y 2 del microcontrolador ATmega1280.

Material:

- Computadora Personal (con AVR Studio)
- Tarjeta T-Juino.
- Programa Terminal.

Equipo:

- Computadora Personal con USB, AVRStudio y WinAVR

Teoría:

- Programación del Timer 0 del microcontrolador
- Programación del Timer 2 del microcontrolador
(Diagrama, Funcionamiento, Registros de configuración y operación)

Desarrollo:

1) Crear y compilar proyecto:

- Utilice el programa AVR Studio para crear un proyecto llamado **Prac10** donde los archivos del proyecto deberán ser los correspondientes al listado 2 (Prac10.c) y listado 3 (Timer0.c) **Nota:** todos los archivos (*.c y *.h) deberán estar en el mismo directorio del proyecto.
- Compile el proyecto (realizar correcciones en dado caso que existan)
- Una vez compilado el proyecto, el archivo (Prac10.hex) deberá ser cargado al T-Juino. Este archivo se encuentra en la carpeta llamada “default” generada por el compilador en el directorio del proyecto (p.e. C:\uPyuC\Prac10\default).

Listado 1. Timer0.h

```
#ifndef _TIMER0_H
#define _TIMER0_H

#include <inttypes.h>

/* Función para inicializar el Timer0 y generar */
/* la temporización de 1 Sec. */
void Timer0_Ini ( void );

/* Función para verificar bandera del segundo */
uint8_t Timer0_SecFlag ( void );

#endif /* _TIMER0_H */
```

Listado 2. Prac10.c

```

#include <avr/io.h>
#include "Timer0.h"

/* incluir lo necesario para usar UART0 */

int main() {

    /* llamar a función para inicializar puertos E/S */
    /* llamar a función para inicializar UART0 */

    Timer0_Ini();          /* Inicializar Timer0 para 1 sec.*/
    while(1){              /* == main loop == */
        if( Timer0_SecFlag() ){ /* ¿ha pasado 1 Segundo? */
            /* instrucciones para encender LED */
            UART0_puts("1 segundo\n\r");
            /* instrucciones para apagar LED */
        }
    }                      /* fin del loop principal */
    return 0;              /* <-- no se llega aqui */
}

```

Listado3. Timer0.c

```

#include <avr/interrupt.h>
#include <inttypes.h>

static volatile uint8_t SecFlag;

void Timer0_Ini ( void ){

    TCNT0=0x06;          /* Inicializar valor para el timer0 */
    TCCR0A=0x00;         /* inicializa timer0 en modo 0 (normal) */
                          /* Inicializar con fuente de osc. Int. */
    TCCR0B=0x03;         /* con Prescalador 64 */
    TIMSK0=0x01;         /* habilita interrupcion del Timer0 */
    sei();               /* habilita interrupciones (global) */
}

uint8_t Timer0_SecFlag ( void ){

    if( SecFlag ){
        SecFlag=0;
        return 1;
    }
    else{
        return 0;
    }
}

ISR (TIMER0_OVF_vect){ /* TIMER0_OVF_vect */
    static uint16_t mSecCnt;
    TCNT0+=0x06;        /* reinicializar Timer0 sin perder conteo */
    mSecCnt++;          /* Incrementa contador de milisegundos */
    if( mSecCnt==1000 ){
        mSecCnt=0;
        SecFlag=1;      /* Bandera de Segundos */
    }
}

```

- d) Una vez cargado el programa, la tarjeta T-Juino deberá estar encendiendo un LED (en algún puerto) cada segundo. Este programa utiliza como base de tiempo el temporizador **Timer0** inicializado en modo 0 (normal) para que se genere una interrupción cada un milisegundo aproximadamente. Esto ocurre cuando el Timer se desborda (pasa de valor FF a 00) y se activa **TOV0**. La rutina de servicio de interrupción (**ISR: Interrupt Service Routine**) asociada a la interrupción lleva un conteo de los milisegundos en la variable **mSecCnt**. Una vez que el conteo llega a 1000 entonces se inicializa a cero para nuevamente llevar dicho conteo, además otra variable tipo bandera llamada **SecFlag** se activa para indicar que ha transcurrido un segundo.

Modificaciones a realizar al programa:

- a) Realice los cambios necesarios para manejar el mismo esquema de tiempo base del **Timer0** pero ahora utilizando el modo **CTC** del temporizador.
- b) Cambiar la lógica de la ISR para solo implementar un contador (de 64 bits) de milisegundos. Implementar la función:
uint64_t millis(void) :
La cual retorna el conteo actual de milisegundos.
- c) Diseñe e implemente la función **void UART0_AutoBaudRate(void)**, la cual ajusta el baud rate dependiendo de la velocidad del dato recibido, tomando como base la duración del bit de inicio (Start Bit) del dato, **suponiendo que el bit menos significativo será '1'**. Esta función deberá funcionar dentro del rango de **8,000 a 200,000 Bauds**.
Nota: Hacer uso del Timer0 para contabilizar el periodo.
- d) Reutilizar las funciones de la Práctica 5:
void Clock_Ini(uint64_t millis): función para inicializar el reloj en milisegundos iniciando desde 01/01/1970 00:00:00 (UNIX Epoch).
void Clock_Date_Display(): función para el desplegado del estado del reloj y fecha en formato **“hh:mm:ss dd/mm/aaaa”**.
- e) Con estas funciones, implementar el siguiente listado:

Listado 4. Prac10.c

```
#include <avr/io.h>
#include "Timer0.h"

/* incluir lo necesario para hacer uso de UART0 */

void UART0_AutoBaudRate(void);

int main() {

    /* llamar a función para inicializar puertos E/S */
    /* llamar a función para inicializar UART0 */

    UART_AutoBaudRate(0);
    UART_clrscr(0);
    UART_gotoxy(0,5,1);
    UART_puts(0,"Autobauding done. UBRR0=");
    itoa(UBRR0,str,10);
    UART_puts(0,str);
    UART_puts(0,'\n\r');

    Timer0_Ini();          /* Inicializar contador de millis.*/
    Clock_Ini(23,59,50);

    while(1){               /* == main loop == */
        if( Timer0_SecFlag() ){ /* ¿ha pasado un Segundo? */
            gotoxy(5,2);
            Clock_Display();
            gotoxy(5,3);
        }
    }                         /* fin del loop principal */
    return 0;               /* <-- no se llega aquí */
}
```

Después de dejar correr el programa durante una hora, por favor responder la siguiente pregunta:
¿Por qué existe la diferencia de tiempo transcurrido? (Asumiendo que el temporizador fue configurado correctamente)

Comentarios y Conclusiones.**Bibliografía.**