Universidad Autónoma de Baja California

Facultad de ciencias químicas e ingeniería

Materia.

Microprocesadores y Microcontroladores.

Maestro.

Garcia Lopez Jesus Adan.

Alumno

Gonzalez Cardiel Luis Enrique

Matricula:

1217258

Grupo:

561

Trabajo:

Practica No. 10

16/11/2018

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:

a) 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.

b) Compile el proyecto (realizar correcciones en dado caso que existan)

c) 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 las 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 las 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 32 bits) de milisegundos.Implementar la función:

uint32_t millis(void):

La cual retorna el conteo actual de milisegundos desde que inició el sistema.

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.

Parte2: Uso del Temporizador/Contador 2

Descripción: Añadir al proyecto el **Listado 4 y 5**; y modifique los archivos necesarios para generar en una base de tiempo de 1 segundo usando el temporizador 2 (Timer2) con el cristal de 32.768 KHz externo como fuente de oscilación.

Para la implementación considere:

a) El Listado 4 (Timer2.h) muestra la función prototipo de inicialización del Timer2 y función de verificación de la bandera indicadora del segundo.

b) Diseñe e implemente la función Timer2_Ini(uint8_t baseT) que tiene como objetivo

c) inicializar el Timer2 para generar desbordes a una base de tiempo determinada. Es decir, la uncón recibe un parámetro con el cual determina la inicialización para el desborde dado por el parámetro. Los valores válidos para este parámetro son de 1 a 8, por tanto se puede inicializar el temporizador de 1 a 8 segundos como base de tiempo.

Ejemplo de uso: Timer2_Ini(4); /* inicializar Timer 2 para generar */
/* hacer Flag igual 1 cada 4 seg. */

d) Modifique el programa para que muestre en la PC vía puerto serie (UART0). El formato del reloj deberá ser tipo 24Hrs, (hh:mm:ss). Reutilizar las funciones de la Práctica anterior.

e) Reutilizar las funciones de la Práctica 5:
void Clock_Ini(<parámetros>): función para inicializar el reloj.
void Clock_Update( ): función para actualizar el reloj (segundos, minutos y horas).
void Clock_Display( ): función para el desplegado del estado del reloj.

f) Utilice el programa Terminal (MTTTY) en la PC con la misma configuración del puerto serie para recibir el mensaje enviado por T-Juino.

g) Sustituya el programa principal de la parte 1 al del Listado 6 para verificar el funcionamiento correcto mediante el programa Hyperterminal en la PC.

Listado 4. Timer2.h

```
#ifndef _TIMER2_H
#define _TIMER2_H
#include <inttypes.h>

/* Funcion para inicializar el Timer2 para generar N seg.   */
void Timer2_Ini( uint8_t baseT );

/* Funcion para verificar bandera del segundo        */
uint8_t Timer2_Flag ( void );

#endif  /*  TIMER2 H */
```

Listado 5. Timer2.c

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

static uint8_t  Flag;

void Timer2_Ini( uint8_t baseT ){

/* Configurar Timer2 para usar cristal externo según */
/* secuencia dada por la hoja de especificaciones    */

}

uint8_t Timer2_Flag ( void ){
   if( Flag ){
      Flag=0;
      return 1;
   }
   else{
      return 0;
   }
}

/* Rutina de Servicio de Interrupción  para atender la interrupción */
/* por Comparación del Timer2  (TCNT2 y OCR2 son iguales)           */
ISR(SIG_OUTPUT_COMPARE2A){

       Flag = 1;     /* Activa Bandera para indicar los 1 segundos */

}  /* fin de la rutina de servicio de Interrupción (RSI) */
```

Listado 6. Prac10.c

```
#include <avr/io.h>
#include "Timer0.h"
#include "Timer2.h"

/* incluir lo necesario para usar UART0 */

void UART0_AutoBaudRate(void);

int main(){

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

    UART0_AutoBaudRate();
    clrscr();
    gotoxy(5,1);
    UART0_puts("Autobauding done. UBRR0=");
    itoa(UBRR0,str,10);
    UART0_puts(str);
    UART0_puts('\n\r');

    Timer0_Ini();                /* Inicializar contador de millis.*/
    Timer2_Ini(7);               /* Inicializar Timer2 para 7 sec.*/
    Clock_Ini(23,59,50);

    while(1){                    /* == main loop ==               */
        if( Timer2_SecFlag() ){  /* ¿ha pasado un Segundo?        */
            Clock_Update();
            gotoxy(5,2);
            Clock_Display();
            gotoxy(5,3);
            UART0_puts("millis=");
            /* itoa solo convertirá los 16bits menos significativos */
            itoa(millis(),str,10);
            UART0_puts(str);
         }
    }                            /* fin del loop principal        */
    return 0;                    /* <-- no se llega aquí          */
}
```

Después de dejar correr el programa durante unos minutos, por favor responder la siguiente pregunta: *¿Por qué existe la diferencia en el conteo de milisegundos?* (Asumiendo que ambos temporizadores fueron configurados correctamente y está no es la causa raíz de la discrepancia)
Estos son los dos osciladores utilizados: XTAL 16MHz, XTAL 32KHz

Teoría:

Programación del Timer 0 del microcontrolador

Registros del Timer/Counter0

Para controlar los modos de funcionamiento y la frecuencia de trabajo, el timer0 tiene asociados varios registros los cuales se describen mas adelante. A parte de los registros, el timer0 tiene la capacidad recibir una frecuencia de reloj externa mediante el pin T0 (PB0) y de cambiar el estado de la salida en el pin OC0, que es la salida del comparador del timer:

TCCR0: (Timer/Counter Control Register) Este registro configura la frecuencia a la que trabajará el timer, el modo de trabajo y si el timer controlará la salida del pin asociado a el, en este caso el pin OC0.

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Registro TCCR0 (Timer/Counter Control Register)

- FOC0: Escribiendo un 1 en este bit, fuerza a que se realice una coincidencia de comparación y actúa sobre el pin OC0 si se usa.
- WGM01:0: Configura el modo de funcionamiento del timer0.

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

- COM01:0: Estos bits configuran el uso del pin OC0 con el modulo de comparación dependiendo del modo que se trabaje.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Salida del comparador en modo NO-PWM (Normal ó CTC)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at BOTTOM, (non-inverting mode)
1	1	Set OC0 on compare match, clear OC0 at BOTTOM, (inverting mode)

Salida del comparador en modo FAST PWM

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

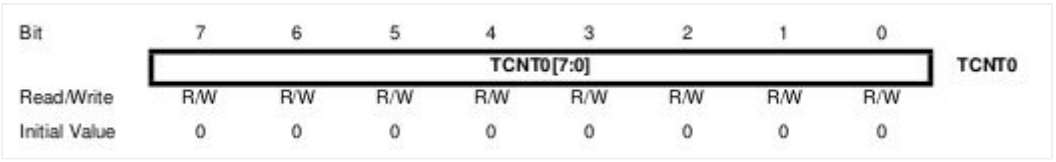
Salida del comparador en modo PHASE CORRECT PWM

- CS02:0: Con estos bits se selecciona la fuente del reloj para el timer, puede ser interna con un prescaler o externa con una señal de reloj en el pin T0.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

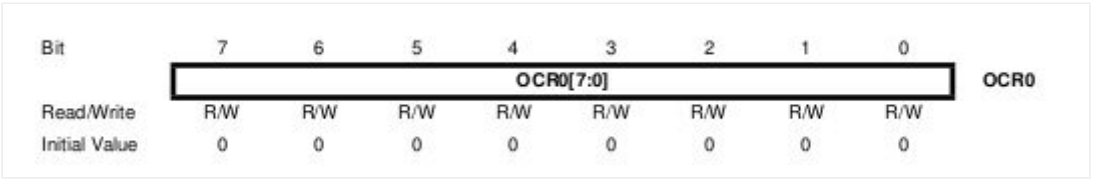
Configuración del prescaler para frecuencia del Timer/Counter0

TCNT0: (Timer/Counter Register) Este registro es el que guarda la cuenta del Timer/Counter0 (0 a 255).



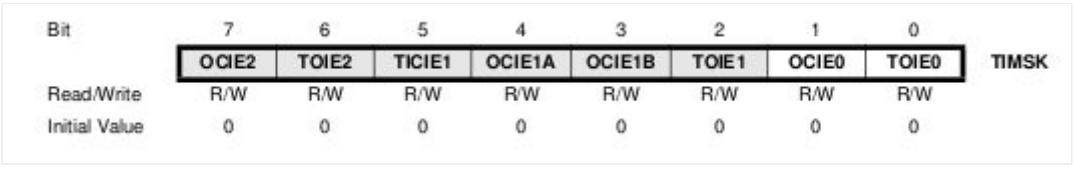
Registro TCNT0 (Timer/Counter Register)

OCR0: (Output Compare Register) Aquí se guarda el valor a comparar con el registro TCNT0. Cuando el valor del registro TCNT0 coincide con el valor guardado en este registro, se realiza el evento programado para la coincidencia en comparación, ya sea generar una interrupción o cambiar el estado del pin OC0.



Registro OCR0

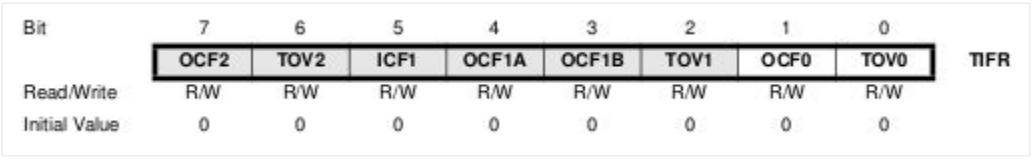
TIMSK: (Timer Interrup Mask Register) En este registro se encuentran los bits de habilitación de interrupciones para cada timer, a nosotros nos interesa el bit TOIE0(0) y OCIE0(1) que son los correspondientes al timer0.



Regitro TIMSK (Timer/Counter Interrup Mask Register)

- TOIE0: Escribir a 1 en este bit habilita la interrupción por desbordamiento del TimerCounter0 (Timer/Counter Overflow Interrupt Enable 0) si también se habilitan las interrupciones globales con el bit I en SREG.
- OCIE0:Escribir a 1 en este bit habilita la interrupción por coincidencia en la comparación del Timer/Counter0con el registro OCR0 (Output Compare Interrupt Enable 0), si también se habilitan las interrupciones globales con el bit I en SREG.

TIFR: (Timer Interrup Flag Register) En este registro se encuentran las banderas de interrupciones para cada timer, a nosotros nos interesa el bit TOV0(0) y OCF0(1) que son los correspondientes al timer0.



Registro TIFR

- TOV0: Cuando este bit se pone a 1 se genera la interrupción por desbordamiento del Timer/Counter0. Este bit se limpia cuando cuando entra a la función que atiende la interrupción o escribiendo un 1 en este bit.

- OCF0: Cuando este bit se pone a 1 se genera la interrupción por coincidencia en la comparación del Timer/Counter0 con el registro OCR0. Este bit se limpia cuando cuando entra a la función que atiende la interrupción o escribiendo un 1 en este bit.

Como se mencionó anteriormente, los timers son complicados, así que en este primer post dedicado al Timer/Counter0 se verá su funcionamiento en modo Normal con ejemplos que usen poleo e interrupciones y fuentes de reloj internas y externas. Se hace esto de explicar un modo en cada post con el fin de profundizar y utilizar todas las características del timer.

Timer/Counter0 en Modo Normal

En este modo el timer0 cuenta desde BOTTOM hasta TOP (BOTTOM = 0, TOP = 255) y al pasar la cuenta empieza de nuevo desde BOTTOM. El timer puede generar interrupciones de desbordamiento (al pasar de TOP a BOTTOM) y de coincidencia en comparación con el registro OCR0.

En estos primero ejemplos se usara el reloj del CPU como fuente de reloj para el timer, descartando el pin T0.

Para saber a que frecuencia y con que periodo trabajará el timer, es necesario saber algunas fórmulas para calcular tiempos con exactitud.

Para calcular la frecuencia del timer, se divide la frecuencia del CPU entre el prescaler deseado:

$$F_{Timer} = \frac{F_{CPU}}{Prescaler}$$

Fórmula para calcular la Frecuencia del Timer

Por lo que el periodo de la cuenta del timer sería el inverso a la frecuencia del timer:

$$T_{Timer} = \frac{1}{F_{Timer}}$$

Fórmula para calcular el Periodo de cuenta del Timer

Esto significa que el timer se incrementara cada ***Ttimer*** segundos, si queremos saber cada cuanto tiempo se desbordará el timer, se multiplica el periodo del timer por la resolución del timer, que en este caso es de 8 bits (256):

$$T_{overflow} = T_{Timer} \cdot Resolución\ Timer$$

Fórmula para calcular el tiempo de desbordamiento del Timer

Con la fórmula anterior podemos saber cada cuanto tiempo se desborda el timer, pero si queremos saber el que valor que debe de tener el timer para un determinado tiempo dentro del rango ***Toverflow*** se aplica una sencilla regla de 3:

$$Resolución\ Timer \rightarrow T_{overflow}$$

$$(Cuenta\ del\ Timer) \overset{?}{\leftarrow} T_{Deseado}$$

$$Cuenta\ del\ Timer = \frac{T_{Deseado} \cdot Resolución\ Timer}{T_{overflow}}$$

Fórmula para calcular el valor del Timer para determinado tiempo.

Por ejemplo: Supongamos que tenemos nuestro micro con una frecuencia de 8 Mhz y un prescaler para el timer de 1024, la frecuencia del timer sería:

Ft = 8000000 Hz / 1024 = 7812.5 Hz

Tt = 1 /7812.5 Hz = 0.000128 segundos = 128 microsegundos

Tovf = 128 microsegundos * 256 = 32768 microsegundos = 32.8 milisegundos

Con estos valores el timer se desbordaría cada 32.8 milisegundos, si queremos medir en intervalos de 10 milisegundos, aplicamos la regla de 3 para saber que valor tendrá el timer cuando hayan transcurrido 10 ms:

Cuenta del Timer = 10 ms * 256 / 32.8 + 1 = 79.4

Esto quiere decir que cuando el timer haya contado 79 pasos habrán transcurrido aproximadamente 10 ms.

Programación del Timer 2 del microcontrolador

Como TIMER2 es un temporizador de 8 bits (como TIMER0), la mayoría de los registros son similares a los de los registros TIMER0. Además de eso, TIMER2 ofrece una función especial que otros temporizadores no tienen: funcionamiento asíncrono.

TCCR2: El Registro de control del temporizador / contador - TCCR2 es el siguiente:

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

A diferencia de otros temporizadores, TIMER2 nos ofrece una amplia gama de prescaladores para elegir. En TIMER2, tenemos 8, 32, 64, 128, 256 y 1024

Table 20-9. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

TCNT2: En el registro del temporizador / contador - TCNT2, se almacena el valor del temporizador. Como TIMER2 es un temporizador de 8 bits, este registro tiene 8 bits de ancho.

Bit	7	6	5	4	3	2	1	0	
(0xB2)	TCNT2[7:0]								TCNT2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TIMSK: La máscara de interrupción del temporizador / contador - Registro TIMSK es la siguiente. Es un registro común para todos los temporizadores.

Bit	7	6	5	4	3	2	1	0	
(0x70)	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	TIMSK2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TOIE2 - Habilitar interrupción de desbordamiento temporizador / contador2 . Configuramos esto a '1' para habilitar las interrupciones de desbordamiento.

TIFR: El registro de banderas de interrupción del temporizador / contador - TIFR es el siguiente. Es un registro común para todos los temporizadores.

Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	–	–	–	–	–	OCF2B	OCF2A	TOV2	TIFR2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TOV2 - Indicador de desbordamiento Timer / Counter2 . Este bit se establece (uno) cada vez que el temporizador se desborda. Se borra automáticamente cada vez que se ejecuta la rutina de servicio de interrupción correspondiente (ISR). Alternativamente, podemos borrarlo escribiendo '1' en él.

Comentarios y Conclusiones.

El saber configurar de manera adecuada los timers nos ayuda mucho ya que con estos podemos realizar relojes, o bien contar cuánto lleva encendido ´x´ dispositivo y proceder a realizar cualquier actividad ya sea dormirlo, mandar un mensaje, o activar una bocina, con esto podemos observar que a partir de los timers podemos realizar una alarma. Fue interesante como se puede implementar un timer externo con la configuración del Timer2, ya que con esto se logró implementar un reloj externo de 32678hz ya que este es el que se utiliza en los relojes y así poder implementar tiempos de 1 segundo exactos

Bibliografía.

http://microcontroladores-mrelberni.com/timer0-avr/