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. 11

01/12/2018

Práctica 11

Generador de Frecuencia mediante los Temporizadores del uC ATmega1280

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

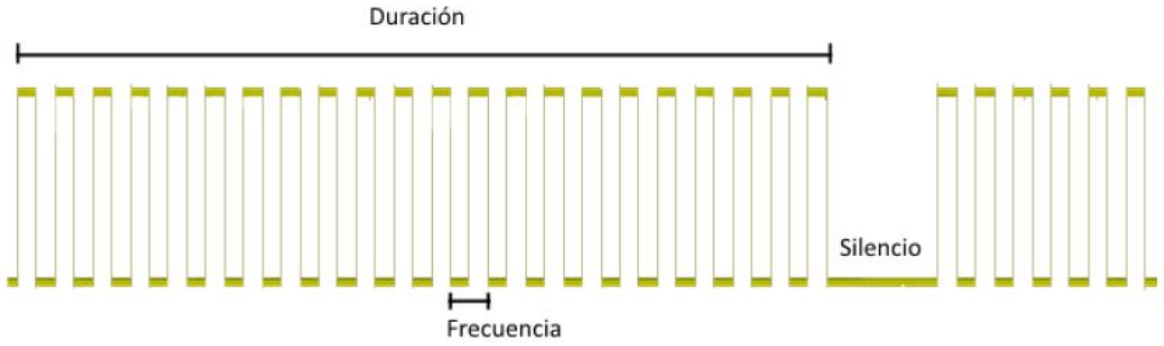
Material:
- Computadora Personal (con AVR Studio)
- Tarjeta T-Juino.
- Componentes Electrónicos.

Equipo:
- Computadora Personal con USB, AVRStudio y WinAVR

Teoría:
- Teoría Básica de Música.
- Programación del Timer2 del microcontrolador como generador de frecuencia (Diagrama, Funcionamiento, Registros de configuración y operación)
- Investigación sobre lectura y escritura de Datos en la Memoria del Programa (PROGMEM)

Desarrollo:

El propósito de la práctica es la reproducción de una canción en base a tonos, mediante el generador de frecuencia. En la siguiente figura se puede visualizar lo que llamaremos una nota:



Donde cada nota estará dada por una frecuencia y una duración, como se muestra en la siguiente estructura:

```
struct note{
      uint16_t freq;
      uint16_t delay;
};
```

Y una canción va a estar dada por un arreglo de notas. Es importante tomar en cuenta que es necesario insertar un tiempo de silencio entre cada nota con una duración de 10 ms, para poder distinguir cuándo inicia y termina una.

Para poder llevar a cabo lo anterior es necesario hacer uso de los Timer0 y Timer2 conjuntamente, y para esto se requiere realizar los ajustes necesarios para que el Timer2 se encargue de generar la frecuencia, donde la salida del generador se verá reflejado en el Pin OC2B, y el encargado de llevar el conteo del tiempo con base de 1 ms será el Timer0.

Las frecuencias de las notas usadas como ejemplo son las siguientes:

```
#define c 261
#define d 294
#define e 329
#define f 349
#define g 391
#define gS 415
#define a 440
#define aS 455
#define b 466
```

```
#define cH 523
#define cSH 554
#define dH 587
#define dSH 622
#define eH 659
#define fH 698
#define fSH 740
#define gH 784
#define gSH 830
#define aH 880
```

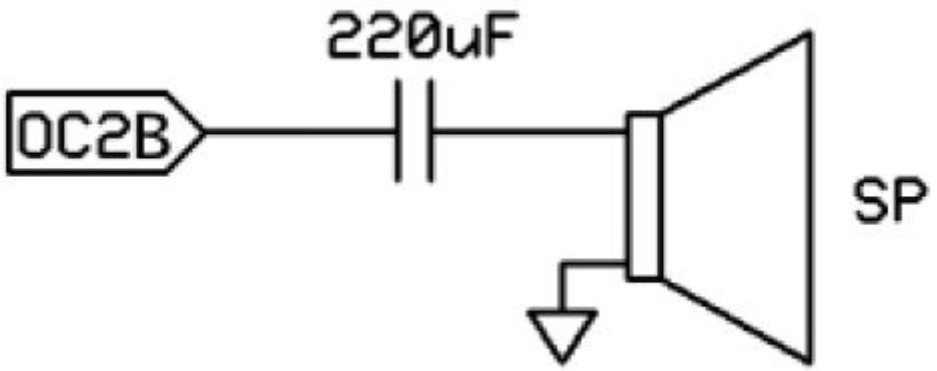
Las cuales las puedes encontrar en el siguiente enlace:
http://www.intmath.com/trigonometric-graphs/music.php

Y el listado del arreglo de notas de ejemplo lo pueden encontrar en el enlace de la práctica, en el archivo Prac11.c, junto con los listados Timer.h y Timer.c

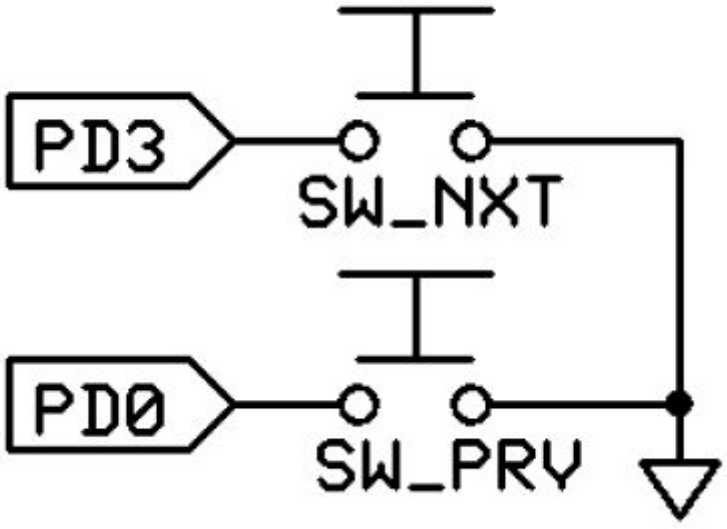
A continuación, se presenta el Listado de Timer.c, con las descripciones de las funciones y macro que se solicitan para esta práctica.

```
  /* Definir el macro que calcula los ticks en base
    a al parámetro de frecuencia (f). */
#define TICKS(f) ??

void Timer0_Ini ( void ){
  /* Permanece igual, ocasionando una interrupción
    cada 1 ms en modo CTC. */
}

ISR(_vect_){
/* Código para actualizar bandera de segundos */

/* Agregar las instrucciones necesarias para reproducir
    la siguiente nota en el arreglo dependiendo de la duración,
    e insertar los silencios entre cada nota. */
}

void Timer2_Freq_Gen(uint8_t ticks){
  /* Si "ticks" es mayor que 0 entonces, inicializa y habilita el
    Generador de Frecuencia del Timer2 con el tope dado por "ticks".
    De lo contrario se requiere deshabilitar el Generador, generando de
    esta forma el silencio (0 lógico). */
}

void Timer2_Play(const struct note song[],uint16_t len){
  /* Función que establece las condiciones necesarias para que
    el generador recorra el arreglo de notas. */
}

void Timer2_Volume(uint8_t direction){
  /* Ajusta el ciclo de trabajo para incrementar o decrementar el volumen
     de las notas que se están generando. */
}
```

Una vez codificadas las funciones anteriores, ahora es necesario alambrar el siguiente diagrama para poder conectar el T-Juino a una bocina (4Ω-8Ω) y lograr escuchar las frecuencias generadas.



Guardar las canciones (dadas en song.c) en memoria de programa, e implementar el cambio de canción utilizando dos botones como se muestra a continuación:



Teoría

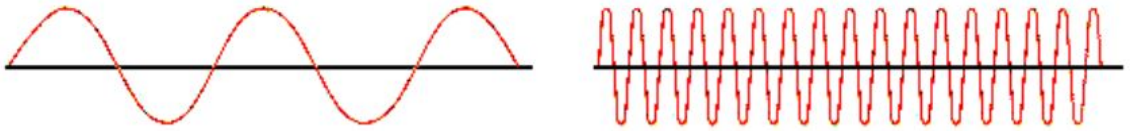
Teoría Básica de Música.

El sonido se define como la sensación producida en el oído por la puesta en vibración de cuerpos sonoros. Debemos entonces diferenciarlo del ruido, y esto se logra a través de la medición.

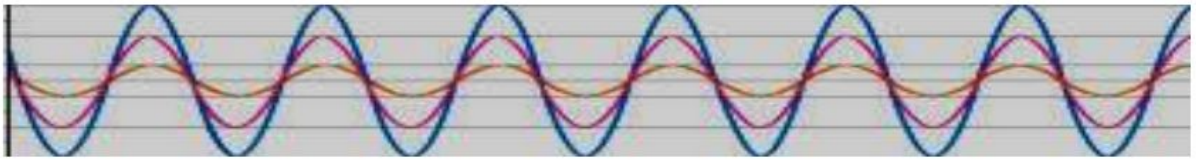
El sonido puede ser medido en sus propiedades mientras que el ruido no puede ser medido en todas

Las propiedades del sonido son:

1. altura: Nos informa la velocidad de vibración del cuerpo sonoro entre más vibre, más agudo será el sonido, entre menos vibre más grave.

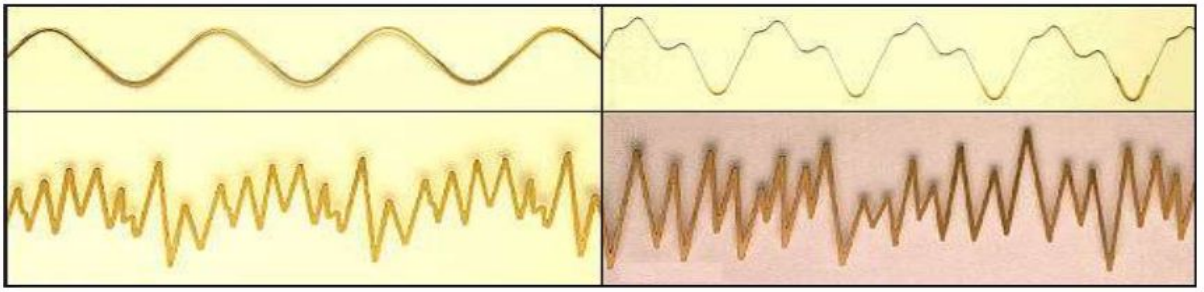


2. Intensidad: Nos habla del tamaño de las crestas o picos de la onda, es equivalente a la amplitud y volumen. “Mas grandes mas duro”

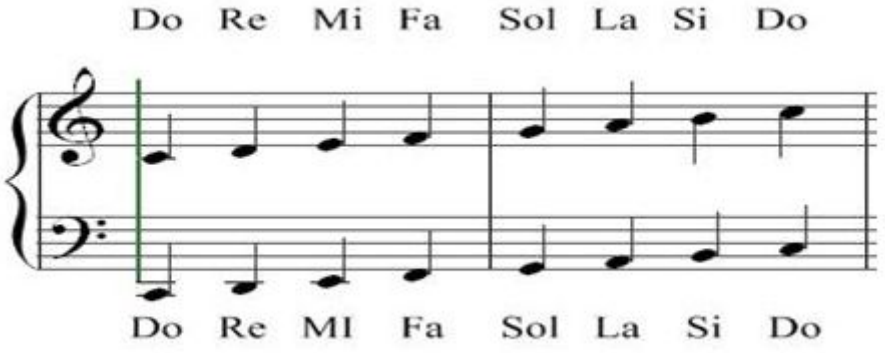


3. Duración: Nos informa del espacio temporal que ocupa desde su aparición hasta su extinción, es equivalente al tiempo.

4. Timbre: Identifica sin lugar a dudas la fuente de la cual proviene, por la forma de las ondas, asegurando que en las mismas condiciones el sonido producido será semejante al anterior.



Tonalidad



La tonalidad de las notas que forman el sistema musical occidental se representa con las palabras: Do, Re, Mi, Fa, Sol, La, Si. En los países de habla inglesa, se emplean letras del alfabeto: C, D, E, F, G, A, B. La C corresponde al Do. Existen razones históricas para que este alfabeto musical no empiece por la A (que corresponde a la nota La), sino por la C.

Para definir la tonalidad de cada nota, desde las más bajas a las más altas, el sistema de nombres se repite. Después del Si viene otro Do. La distancia o "intervalo" entre una nota y la siguiente del mismo nombre (por arriba o por abajo) se llama octava.

Dos notas separadas por una octava suenan igual pero tienen tonalidades diferentes; una está en un "registro" más alto que la otra. Este es un fenómeno natural, basado en que las frecuencias de ambas están en una proporción de 2:1.

El teclado de un piano es la referencia visual más inmediata del sistema musical occidental. Las siete notas que hemos descrito están representadas por las teclas blancas. Empezando por un Do y tocando las ocho teclas blancas que componen una octava se habrá tocado la escala diatónica de Do mayor.

Sin embargo en el espacio de esta octava hay también cinco teclas negras. Se nombra en relación con la nota blanca más próxima. Así la nota de la tecla negra situada entre el Do y el Re, se llama Do sostenido (es decir "alto") o Re bemol (es decir "bajo").

Estas notas que pueden recibir dos nombres se llaman notas enarmónicas y el contexto en el que se usan es lo que determina qué nombre es el apropiado. A menos que la armadura de clave especifique otra cosa, la regla general es llamarla sostenido cuando se sube y bemol cuando se baja.

El intervalo entre la nota de una tecla blanca y la de la tecla negra siguiente es un semitono. Dos semitonos son igual a un tono. Si miramos un teclado, se verá que entre el Si y el Do y el Mi y Fa no hay tecla negra. Esto se debe a que la octava de siete notas no está realmente dividida en intervalos iguales. De Si a Do y de Mi a Fa hay semitonos, no tonos enteros.

Programación del Timer2 del microcontrolador como generador de frecuencia

Para programar el módulo PWM timer2 AVR fase correcta en el ATmega88, se utilizan 2 registros los que son el registro TCCR2A y el registro TCCR2B.

El registro TCCR2A

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	

Los bits 7 y 6 son para elegir que la obtención de la señal PWM timer2 AVR fase correcta será por el pin OC2A y si será en forma no invertida o en forma invertida, mediante las combinaciones de estos bits según se indica en la siguiente tabla.

Table 18-4. Compare output mode, phase correct PWM Mode(1).

COM2A1	COM2A0	Description
0	0	Normal port operation, OC2A disconnected
0	1	WGM22 = 0: Normal port operation, OC2A disconnected WGM22 = 1: Toggle OC2A on compare match
1	0	Clear OC2A on compare match when up-counting Set OC2A on compare match when down-counting
1	1	Set OC2A on compare match when up-counting Clear OC2A on compare match when down-counting

Si las combinaciones de estos bits son 10 la señal PWM obtenida por el pin OC2A será no invertida, si las combinaciones de estos bits son 11 la señal PWM obtenida por el pin OC2A será invertida.

Los bits 5 y 4 son para elegir que la obtención de la señal PWM timer2 AVR será por el pin OC2B y si será en forma no invertida o en forma invertida, mediante las combinaciones de estos bits según se indica en la siguiente tabla.

Table 18-7. Compare output mode, phase correct PWM mode(1).

COM2B1	COM2B0	Description
0	0	Normal port operation, OC2B disconnected
0	1	Reserved
1	0	Clear OC2B on compare match when up-counting Set OC2B on compare match when down-counting
1	1	Set OC2B on compare match when up-counting Clear OC2B on compare match when down-counting

Si las combinaciones de estos bits son 10 la señal PWM obtenida por el pin OC2B será no invertida, si las combinaciones de estos bits son 11 la señal PWM obtenida por el pin OC2B será invertida

Los bits 3 y 2 no se utilizan por lo que se les pone a 0.

Los bits 1 y 0 junto con el bit3 del registro TCCR2B son con los cuales se elige el modo de obtener las señales PWM timer2 AVR, mediante las combinaciones de estos bits según se indica en la siguiente tabla.

Table 18-8. Waveform generation mode bit description.

Mode	WGM2	WGM1	WGM0	Timer/counter mode of operation	TOP	Update of OCRx at	TOV flag set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, phase correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, phase correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX= 0xFF
2. BOTTOM= 0x00

La combinación de estos bits que se utilizara para la generación de la señal PWM timer2 AVR fase correcta será opción 1, la que está resaltada, esto es el bit 3 del registro TCCR2B se pondrá a 0 y del registro TCCR2A su bit1 se pondrá a 0 y su bit0 se pondrá a 1.

El registro TCCR2B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/write	W	W	R	R	R	R	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Los bits 7, 6, 5, y 4 no se utilizarán en la obtención de la señal PWM timer2 AVR fase correcta por lo que se les pondrá a 0.

El bit3 trabaja junto con los bits 1 y 0 del registro TCCR2A tal como se comentó.

Los bits 2, 1 y 0 son para elegir el prescaler a utilizar para obtener la frecuencia de la señal PWM timer2 AVR fase correcta, las combinaciones de estos bits para los diversos prescaler del timer2 son los que se indican en la siguiente tabla:

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)

Investigación sobre lectura y escritura de Datos en la Memoria del Programa (PROGMEM)

Almacene los datos en la memoria flash (programa) en lugar de SRAM. Hay una descripción de los distintos tipos de memoria disponibles en una placa Arduino.

La PROGMEM palabra clave es un modificador de variable, se debe usar solo con los tipos de datos definidos en pgmspace.h. Le dice al compilador "ponga esta información en la memoria flash", en lugar de en la SRAM, a donde normalmente iría.

PROGMEM es parte de la biblioteca pgmspace.h . Se incluye automáticamente en las versiones modernas del IDE, sin embargo, si está utilizando una versión IDE por debajo de 1.0 (2011), primero deberá incluir la biblioteca en la parte superior de su bosquejo, como esto:

#include <avr/pgmspace.h>

Sintaxis

const dataType variableName [] PROGMEM = {data0, data1, data3…};

dataType- cualquier tipo de variable
variableName- el nombre de su matriz de datos

Tenga en cuenta que debido a que PROGMEM es un modificador de variable, no existe una regla estricta sobre a dónde debe ir, por lo que el compilador Arduino acepta todas las definiciones a continuación, que también lo son. Sin embargo, los experimentos han indicado que, en varias versiones de Arduino (que tienen que ver con la versión GCC), PROGMEM puede funcionar en una ubicación y no en otra. El siguiente ejemplo de "tabla de cadenas" se ha probado para que funcione con Arduino 13. Las versiones anteriores del IDE pueden funcionar mejor si PROGMEM se incluye después del nombre de la variable.

const dataType variableName[] PROGMEM = {}; // use this form
const PROGMEM dataType variableName[] = {}; // or this one
const dataType PROGMEM variableName[] = {}; // not this one
Si bien PROGMEM podría usarse en una sola variable, realmente solo vale la pena si tiene un bloque de datos más grande que deba almacenarse, lo que generalmente es más fácil en una matriz (u otra estructura de datos C más allá de nuestra discusión actual).

El uso PROGMEM es también un procedimiento de dos pasos. Después de obtener los datos en la memoria Flash, se requieren métodos especiales (funciones), también definidos en la biblioteca pgmspace.h , para volver a leer los datos de la memoria del programa en la SRAM, para que podamos hacer algo útil con ellos.

Los siguientes fragmentos de código ilustran cómo leer y escribir caracteres sin firmar (bytes) e ints (2 bytes) en PROGMEM.

```
// save some unsigned ints
const PROGMEM  uint16_t charSet[]  = { 65000, 32796, 16843, 10, 11234};

// save some chars
const char signMessage[] PROGMEM  = {"I AM PREDATOR,  UNSEEN COMBATANT. CREATED BY THE UNITED STATES DEPART"};

unsigned int displayInt;
int k;    // counter variable
char myChar;


void setup() {
  Serial.begin(9600);
  while (!Serial);  // wait for serial port to connect. Needed for native USB

  // put your setup code here, to run once:
  // read back a 2-byte int
  for (k = 0; k < 5; k++)
  {
    displayInt = pgm_read_word_near(charSet + k);
    Serial.println(displayInt);
  }
  Serial.println();

  // read back a char
  for (k = 0; k < strlen_P(signMessage); k++)
  {
    myChar =  pgm_read_byte_near(signMessage + k);
    Serial.print(myChar);
  }

  Serial.println();
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Comentarios y Conclusiones.

Para la obtención de señales PWM con el microcontrolador AVR, es muy importante saber utilizar sus temporizadores sobre los cuales ya se ha realizado en prácticas anteriores, es necesario saber configurar los registros de manera correcta, ya que a pesar de que es sencillo se pueden cometer errores a la hora de configurarlos

El ciclo de trabajo es la parte más importante a la hora de configurar el PWM ya que la frecuencia puede ser cualquiera que le configures.

Bibliografía

https://www.teoria.com/articulos/guevara-sanin/guevara_sanin-teoria_de_la_musica.pdf
https://www.monografias.com/trabajos104/teoria-musical-completa/teoria-musical-completa.shtml
http://microcontroladores-mrelberni.com/pwm-timer2-avr-modo-rapido/