

Universidad Autónoma de Baja California  
Facultad de Ciencias



Proyecto de Instrumentación  
**GENERADOR DE FUNCIONES**

Hiram K. Herrera Alcantar  
Osvaldo Rosales Pérez

Profesor:  
Dra. Eloisa del Carmen García Canseco

Fecha: 26 de Mayo del 2017

## **1. Objetivo**

Desarrollar un generador de funciones aplicando conocimientos de electrónica adquiridos en la clase de instrumentación y conocimientos de previos de programación.

## **2. Introducción**

Un generador de funciones generalmente es un equipo o software de prueba, usado para generar diferentes tipos de formas de ondas eléctricas en un rango de frecuencias establecido. Típicamente las formas de onda disponibles en un generador de funciones son la senoidal, cuadrada, triangular y diente de sierra; que pueden ser repetitivas o singulares. En general un generador de funciones se utiliza para examinar y reparar circuitos electrónicos, aunque también pueden ser utilizados para calibrar equipos, rampas de alimentación, etc [1].

El propósito de este proyecto es construir un generador de funciones sencillo, tomando como base el que se presenta en [2]; y utilizando herramientas de electrónica y programación, para esto se implementa un microcontrolador Arduino UNO en la realización del proyecto, además de diversos componentes electrónicos que se describirán más adelante. Como aportación al modelo de un generador se agrega una modulación por ancho de pulsos (PWM), que se encarga de modificar el ancho de los pulsos cuadrados.

Cabe mencionar la importancia de la instrumentación en la realización de este proyecto, pues al hacer las conexiones es necesario comprobar el valor de la corriente y voltaje que corre sobre cada componente pues de lo contrario podrían dañarse; además se debe comprobar continuidad de corriente, para esto se utiliza un multímetro. Otro punto importante de la instrumentación en este proyecto es el uso del osciloscopio para desplegar las formas de onda generadas por el dispositivo.

## **3. Materiales**

- 4 Pushbutton de 4 pines.
- 12 resistores de 10k Ohm.
- 9 resistores de 20k Ohm.
- 1 resistores de 2.2k Ohm
- 4 resistores de 1k ohm.
- 1 Potenciómetro de 50k Ohm.
- 2 Potenciómetros de 10k Ohm.
- 2 Capacitores electrolíticos de  $220\mu F$ .
- Circuito integrado LM386N.
- Batería de 9V.
- 4 diodos LED.
- Protoboard.
- Arduino UNO.
- Cable UTP.

## **4. Funcionamiento electrónico**

El generador de funciones se basa principalmente en el Arduino UNO para el cálculo y la generación de cada función. Sin embargo, sigue siendo necesario un circuito externo para manejar las señales de entrada y salida del microcontrolador. Este circuito externo consta de 4 secciones, una de entradas, otra de salidas, un convertidor digital-analógico y un amplificación de señal.

## Entradas

Las entradas que se tienen son los cuatro botones push que sirven para seleccionar el tipo de función a generar, ya sea sinusoidal, sierra, triangular o cuadrada. También están los potenciómetros, dos de éstos fijan la frecuencia de las funciones y, en caso de seleccionar la función cuadrada, el ancho de cada pulso. El tercer potenciómetro controla la ganancia o amplitud de las funciones.

## Salidas

Se tienen cuatro salidas para LEDs, cada uno corresponde a una función del generador y se encienden la momento de hacer la selección con los interruptores de entrada. También se tiene la salida digital que abarca 8 pines (D0-D7), la cual se encarga de transmitir la señal de la función seleccionada al convertidor digital-analógico.

## Convertidor digital analógico (DAC)

Éste se encarga de tomar la señal digital de salida del Arduino en la que se transmite el valor de la función y la transforma a una señal analógica.

El convertidor utilizado es uno del tipo escalera R-2R, el cual se basa en arreglos repetitivos de resistencias en una configuración de escalera, como se muestra en la figura 1.

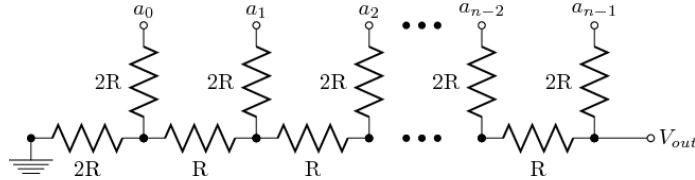


Figura 1: Convertidor digital-analógico con  $n$  entradas digitales  $a_0, \dots, a_{n-1}$  y salida analógica  $V_{out}$

Este arreglo R-2R hace que cada bit sea pesado por su contribución al voltaje de salida  $V_{out}$ . Dependiendo de cuales bits son 1 o 0, el voltaje de salida tendrá el correspondiente valor entre 0V y  $V_{ref} = 5V$ .

## Amplificador de señal

La sección del amplificador de señal sirve para que una vez obtenida una señal analógica del DAC, ésta se amplifique, pues generalmente en el proceso de conversión se pierde energía además de que las salidas del Arduino solo pueden proporcionar como máximo 5V. Para lograr la amplificación de la señal se implementó un amplificador operacional, particularmente, el circuito integrado LM386n.

El diseño general de un amplificador operacional se muestra en la figura 2 donde  $V_+$  y  $V_-$  corresponden a la señal de entrada,  $V_{S+}$  y  $V_{S-}$  son la diferencia de potencial que establece el rango de amplificación y  $V_{out}$  es la señal ya amplificada.

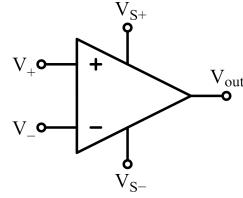


Figura 2: Amplificador operacional.

## 5. Funcionamiento general

La operación del generador de funciones es bastante simple. Como se muestra en la figura 3, es necesario alimentar el generador con 9V (terminales caimán rojo y negro), ya sea de una fuente de voltaje o una batería. Los 9V van al amplificador y al Arduino, todos los demás componentes trabajan con 5V suministrados por el mismo Arduino. La función generada, una vez amplificada, se puede medir en el pin central del potenciómetro de ganancia, conectado a la salida del amplificador. La conexión con el osciloscopio es con la sonda de prueba conectada a éste pin y la otra terminal conectada a la tierra de la fuente de alimentación.

Una vez que se conecta el circuito a la fuente de alimentación y la sonda del osciloscopio se conecta al potenciómetro de ganancia, es posible controlar la función generada y ver los cambios en la pantalla del osciloscopio.

La selección del tipo de señal es con los botones en el centro del protoboard. De los tres potenciómetros colocados en el extremo izquierdo del circuito, el primero, de izquierda a derecha, controla la amplitud de la señal, el siguiente controla la frecuencia y el último controla el ancho de los pulsos cuadrados.

La amplitud máxima para una señal es de 7V. El rango dinámico de frecuencias es de 300 – 15,000Hz para las señales sinusoidal, sierra y triangular, mientras que la señal cuadrada puede trabajar en el rango de 300 – 25,000Hz.

En la figura 4 se pueden ver los 4 tipos de señal que se pueden generar ademas de la modulación de ancho de pulso para las ondas cuadradas.

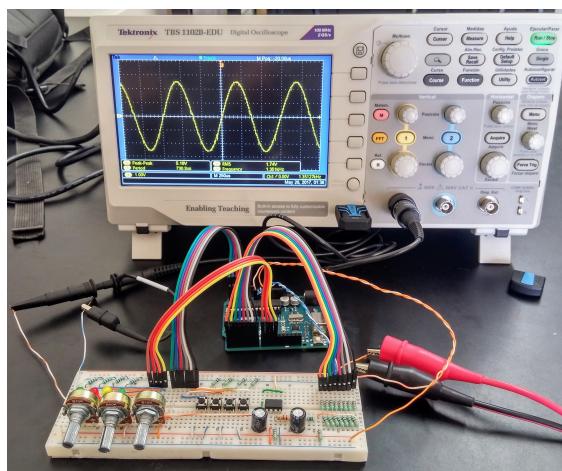


Figura 3: Generador de función en operación

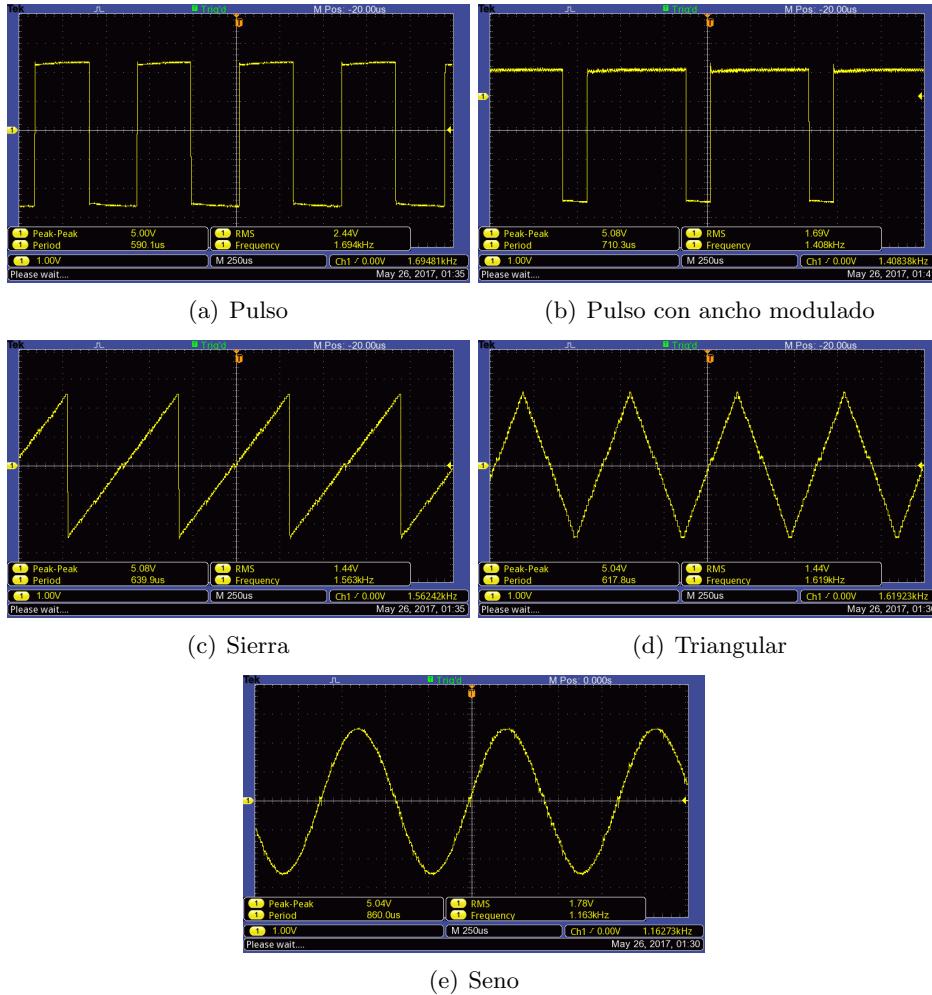


Figura 4: Resultados obtenidos.

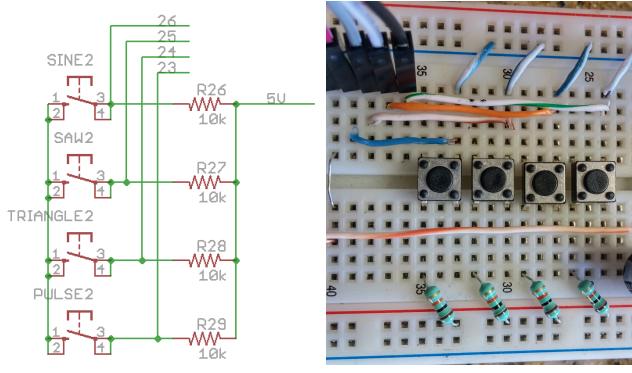
## 6. Desarrollo del proyecto.

El procedimiento para el ensamblado del dispositivo se describe a continuación

### Paso 1: Instalación de botones.

Se conectan los cuatro botones al protoboard tal como se muestra en la figura 5, éstos se conectan a tierra y a la salida de 5V del Arduino a través de un resistor de 10k Ohms cada uno. Entre cada botón y tierra se hace una conexión hacia las entradas analogicas A0-A3 del Arduino. Se conectan en el siguiente orden:

- A0 = Pulso
- A1 = Triangulo
- A2 = Sierra
- A3 = Seno



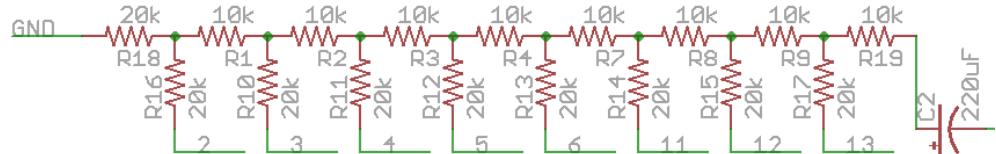
(a) Diagrama

(b) Ensamble en protoboard

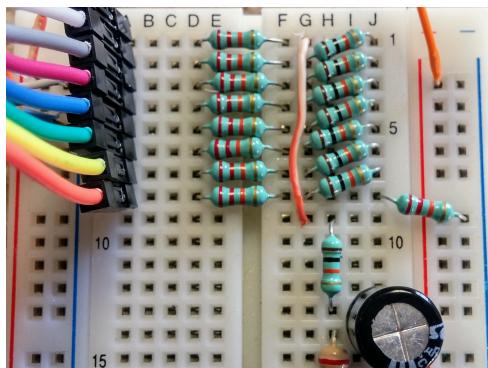
Figura 5: Conexión de botones.

**Paso 2: Conexión de convertidor digital-analógico.**

Se hacen las conexiones del convertidor digital-analógico (DAC) tal como se muestra en la figura 6.



(a) Diagrama



(b) Ensamble en protoboard

Figura 6: Conexión del convertidor digital-analógico.

Para esto se conectan primero ocho resistores de 20k Ohms al protoboard, éstos deberán ir conectados a los pines digitales D0-D7 del Arduino. Despues, se conectan 7 resistores de 10k Ohms, uno entre cada par de 20k Ohms. Un resistor de 20k Ohms se conecta a tierra y a la resistencia de 20k Ohms del pin D0. Por último, se conecta un resistor de 10k Ohms y un capacitor de  $220\mu F$  a la resistencia de 20k Ohms del pin D7.

### Paso 3: Amplificador

Primero se conecta el circuito integrado LM386N en el protoboard como se muestra en la figura 7. Se conecta un resistor de 20k Ohms a la terminal negativa del capacitor de  $220\mu F$  al final del DAC y al pin 3 del LM356N. El mismo pin también se conecta a tierra con resistor de 2.2k Ohms. Los pines 2 y 4 se conectan directamente a tierra. El pin 6 de alimentación se conecta a la fuente de 9V y el pin 5 de salida se conecta a un capacitor de  $220\mu F$ .

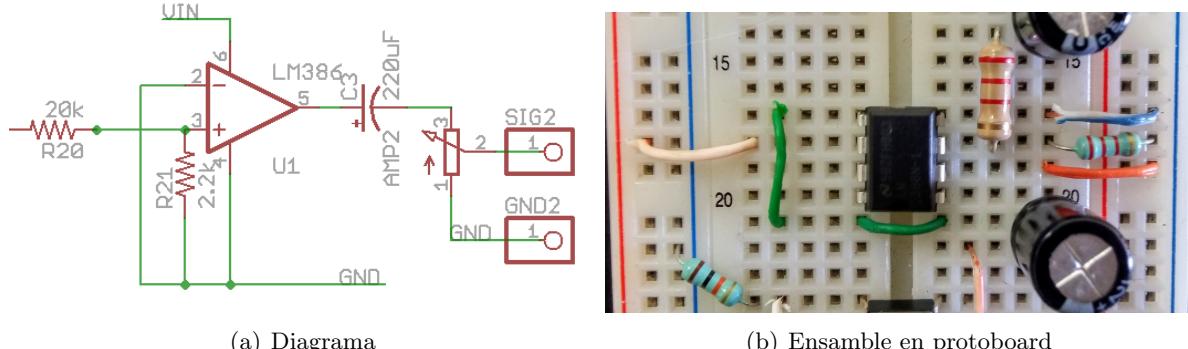
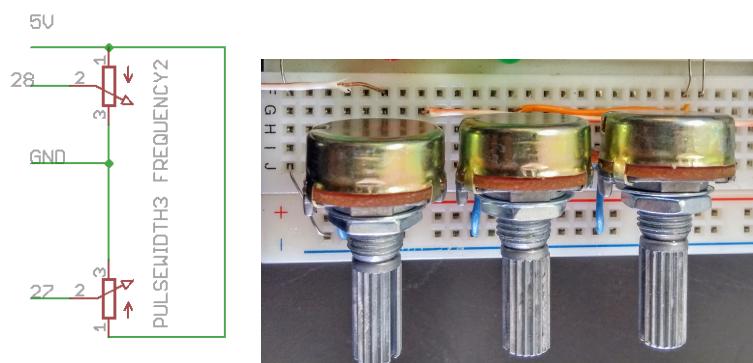


Figura 7: Conexión del amplificador de señal

### Paso 4: Potenciómetros de frecuencia y PWM y ganancia

Se conectan en el protoboard un potenciómetro de 10k Ohms para la ganancia, uno de 50k Ohms para la frecuencia y otro 10k Ohms para el PWM, como se muestra en la figura 8. El potenciómetro de ganancia se conecta al amplificador de señal como se muestra en el diagrama de la figura 7, con la terminal negativa del capacitor de  $220\mu F$  en su pin derecho, la tierra en su pin izquierdo y la salida de señal en el pin central. Los potenciómetros de frecuencia y PWM se conectan, con sus pins derechos, al pin de 5V del Arduino, a tierra con sus pins izquierdos y los pins centrales van a las entradas analógicas A4 y A5 del Arduino.

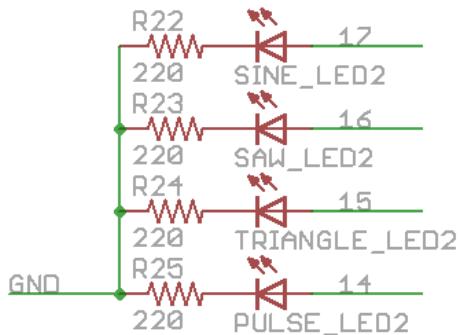


(a) Diagrama para frecuencia y PWM  
(b) Ensamblaje en protoboard de ganancia (izq), frecuencia (cen) y PWM (der).

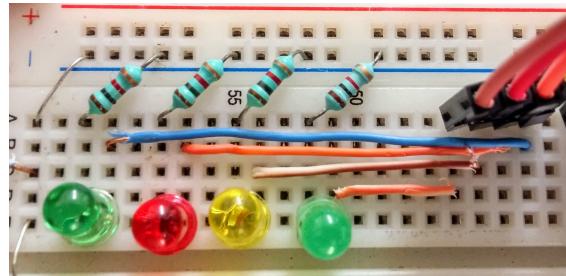
Figura 8: Conexión de potenciómetros.

**Paso 5: Conexión de LEDs** Se conectan 4 LEDs al protoboard como se muestra en la figura 9. Los cátodos se conectan a tierra con un resistor de 1k Ohm cada uno. Los ánodos se conectan a las salidas digitales D8-D11 del Arduino tomando en cuenta el siguiente orden:

- D8 = Pulso
- D9 = Triangulo
- D10 = Sierra
- D11 = Seno



(a) Diagrama de LEDs.

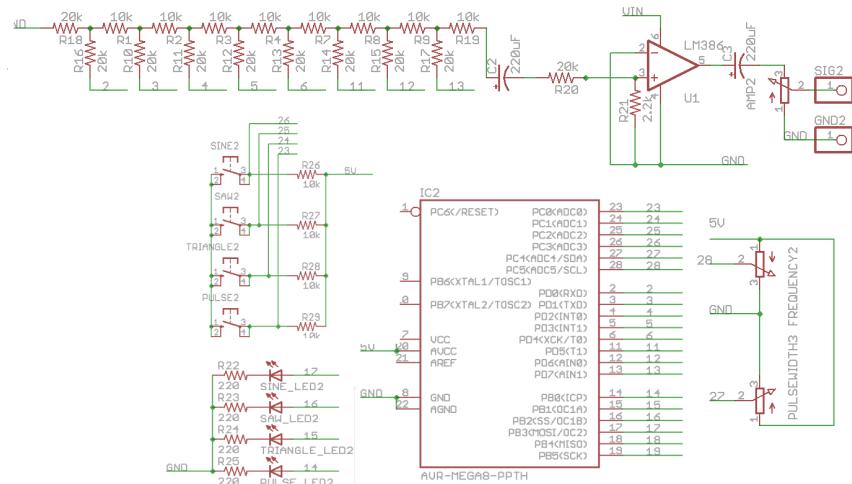


(b) Ensamble en protoboard.

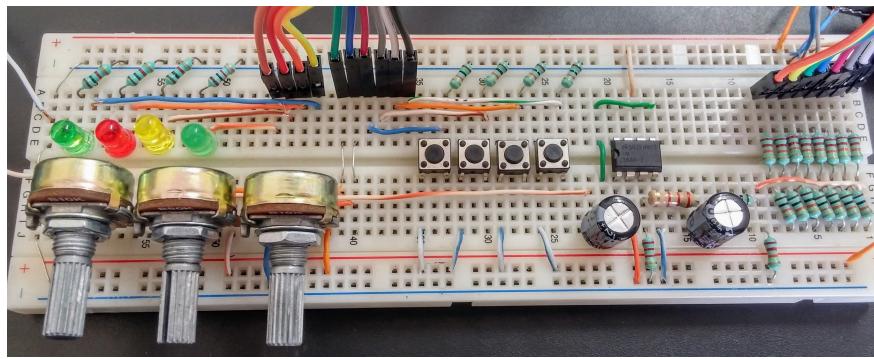
Figura 9: Conexión de LEDs

**Paso 6: Firmware** El código 1 utilizado con el Arduino se muestra en el apéndice con comentarios que explican las secciones del mismo. Este código es una versión modificada del programa desarrollado por Amanda Ghassaei en abril del 2012. El código trabaja revisando primero el estado de los botones para así determinar qué tipo de función generar. Después, lee las entradas analógicas de los potenciómetros para calcular la frecuencia que debe tener la señal y, en caso de haber seleccionado la función cuadrada, el ancho de cada pulso. Una vez leídos estos parámetros, se procede a generar la función seleccionada y a enviarla a través de el puerto digital B (pins D0-D7).

**Paso 7: Circuito completo** A continuación se muestra en la figura un diagrama completo de todo el circuito y como se ve este ya armado en el protoboard.



(a) Diagrama completo



(b) Ensamble completo

Figura 10: Generador de funciones terminado

## 7. Conclusiones

Se comprueba la importancia de la instrumentación en el proceso de realizar este proyecto pues al encontrarnos con dificultades técnicas, como que no se podía cambiar de tipo de onda al oprimir los botones, que la señal obtenida no parecía haber sido amplificada o que los potenciómetros no operaban bien al no incrementar ni disminuir los parámetros que debían de cambiar, la manera de llegar a una solución fue la de comprender bien el funcionamiento de cada componente y mediante un multímetro comprobar las conexiones y los voltajes correspondientes para cada parte del circuito.

En general, se concluye que este proyecto fue útil para reforzar y comprender conceptos de electrónica así como las habilidades manuales necesarias para la conexión de circuitos. Otro punto importante es que se aprendió como utilizar el microcontrolador Arduino, el cual resulta ser una plataforma versátil, cómoda y con gran aplicación educativa para introducir a los alumnos, como nosotros, a la electrónica y la instrumentación; Sin duda, estos conocimientos podrán ser útiles para futuros proyectos.

## Referencias

- [1] Klaas B Klaassen. *Electronic measurement and instrumentation*. Cambridge University Press, 1996.
- [2] Amanda Ghassaei. Arduino waveform generator. <http://www.instructables.com/id/Arduino-Waveform-Generator/>, 2012. [Online; accessed 26-Abril-2017].

# APÉNDICE

**NOTA:** En la linea 34 faltan valores en la definición de sine20000, para el código completo visite:  
<https://github.com/HiramHerrera/Wave-Function-Generator>

Algoritmo 1: Código implementado para el generador de funciones

```
1 //Arduino Function Generator
2 //by Amanda Ghassaei
3 //http://www.instructables.com/id/Arduino-Waveform-Generator/
4 //April 2012
5 //
6 //Modified by: Hiram Herrera & Osvaldo Rosales
7 //May 2017
8
9 /*
10 * This program is free software; you can redistribute it and/or modify
11 * it under the terms of the GNU General Public License as published by
12 * the Free Software Foundation; either version 3 of the License, or
13 * (at your option) any later version.
14 *
15 */
16
17 //in most of this code I have used the arduino portpin assignments to send data to
18 //pins, you can read more about how that works here: http://www.arduino.cc/en/
19 //Reference/PortManipulation
20
21 *****
22 some notes about pin setup:
23 4 momentary switches control waveshape (analog in 0-3)
24   -sine connects to A0
25   -triangle A1
26   -saw A2
27   -pulse A3
28 freq control pot (log taper) is connected to analog in 4
29 pulse width modulation pot (linear taper) connected to analog in 5
30 PORT D (digital pins 0-7) are 8 bit function out
31 PORT B (digital pins 8-13) are 8 bit function out
32 *****
33
34 //Storing sine wave values in size 20000 array
35 const byte sine20000 [] PROGMEM = {127, 127, ...}
36 //Defining variables
37 //wavetype storage:
38 //0 is pulse
39 //1 is triangle
40 //2 is saw
41 //3 is sine
42 byte type = 1;//initialize as square
43 byte typecurrent = 0;
44 byte typelast;
45 byte typecounter [4];
46 byte i;
47 //variables for PW pot monitoring
```

```

48 float pulseWidth;
49 int pulseWidthScaled;
50 int PWCurrent;
51 byte PWTolerance = 8; //adjust this to increase/decrease stability of PW measurement
52
53 //variables for freq pot monitoring
54 int frequency;
55 int freqCurrent;
56 byte freqTolerance = 2; //adjust this to increase/decrease stability of frequency
57     measurement
58 unsigned int freqscaled;
59
60 byte wave;
61 long t;
62 long samplerate;
63 long period;
64
65 //storage variables - I used these to cut down on the math being performed during the
66     interrupt
67 int sawByte = 0;
68 byte sawInc;
69 int triByte = 0;
70 byte triInc;
71 int sinNum = 0;
72 int sinInc;
73
74 void setup() {
75
76     //set port/pin mode
77     DDRD = 0xFF; // all outputs
78     DDRC = 0x00; // all inputs
79     DDRB = 0xFF; // all outputs
80     //TIMER INTERRUPT SETUP
81
82     cli(); //disable interrupts
83     //timer 1:
84     TCCR1A = 0; // set entire TCCR1A register to 0
85     TCCR1B = 0; // same for TCCR1B
86     //set compare match register - 100khz to start
87     OCR1A = 159; // = (16 000 000 / 100 000) - 1 = 159
88     //turn on CTC mode
89     TCCR1B |= (1 << WGM12);
90     // Set CS10 bit for 0 prescaler
91     TCCR1B |= (1 << CS10);
92     // enable timer compare interrupt
93     TIMSK1 |= (1 << OCIE1A);
94
95     samplerate = 100000;
96
97     PORTB = 0;
98     PORTB = 1<<type; //Send signal value to PORTB
99
100    //initialize variables
101    frequency = analogRead(A5); //initialize frequency
102    freqscaled = 48*frequency+1; //from 1 to ~50,000\
103    period = samplerate/freqscaled;

```

```

102 pulseWidth = analogRead(A4); //initialize pulse width
103 pulseWidthScaled = int(pulseWidth/1023*period);
104
105 //Number of steps for each wave
106 triInc = 511/period;
107 sawInc = 255/period;
108 sinInc = 20000/period;
109
110 sei(); //enable interrupts
111 }
112
113
114 void checkFreq() {
115 freqCurrent = analogRead(A5);
116 if (abs(freqCurrent-frequency)>freqTolerance){//if reading from pot exceeds
117   tolerance
118   frequency = freqCurrent;//new frequency- number between 0 and 1024
119   freqscaled = 48*frequency+1;//from 1 to ~50,000Hz
120   period = samplerate/freqscaled;
121   pulseWidthScaled = int(pulseWidth/1023*period);
122   triInc = 511/period;
123   if (triInc==0){
124     triInc = 1;
125   }
126   sawInc = 255/period;
127   if (sawInc==0){
128     sawInc = 1;
129   }
130   sinInc=20000/period;
131 }
132
133 void checkPW() {
134 PWCurrent = analogRead(A4);
135 if (abs(PWCurrent-pulseWidth)>PWTolerance){//if reading from pot exceeds tolerance
136   pulseWidth = PWCurrent;//new pulse width, val between 0 and 1023
137   pulseWidthScaled = int(pulseWidth/1023*period);
138 }
139
140
141
142 void checkShape() {//check states of buttons
143 // 4 momentary switches control waveshape
144 // -pulse connects to A0
145 // -triangle A1
146 // -saw A2
147 // -sine A3
148 typelast = typecurrent;
149 if (digitalRead(A0)==HIGH){
150   typecurrent = 1;
151 }
152 else if (digitalRead(A1)==HIGH){
153   typecurrent = 2;
154 }
155 else if (digitalRead(A2)==HIGH){
156   typecurrent = 4;

```

```

157 }
158 else if (digitalRead(A3)==HIGH){
159     typecurrent = 8;
160 }
161 //This part allows momentary switches to work properly
162 for (i=0; i<4; i++){
163     if (i==type){
164     }
165     else{
166         if ((typecurrent & (1 << i)) ^ (typelast & (1 << i))){//current diff than prev
167             and debounce
168             if ((typecurrent & (1 << i))){//currently depressed
169                 type = i;//set wave type
170             }
171             else {
172                 typecounter[i] = 12;//else set debounce counter to 12
173             }
174         else if (((typecurrent & (1 << i)) == (typelast & (1 << i)))) {//if current
175             same as prev and diff than debounce
176             if (typecounter[i] > 0 && --typecounter[i] == 0) {//decrease debounce
177                 counter and check to see if = 0
178                 if ((typecurrent & (1 << i))){//if debounce counter = 0 toggle debounced
179                     state
180                     type = i;
181                 }
182             }
183 }
184
185
186
187 ISR(TIMER1_COMPA_vect){//timer 1 interrupt
188     //increment t and reset each time it reaches period
189     t += 1;
190     if (t >= period){
191         t = 0;
192     }
193     switch (type) {
194         case 0://pulse
195             if (pulseWidthScaled <= t) {
196                 wave = 255;
197             }
198             else{
199                 wave = 0;
200             }
201             break;
202         case 1://triangle
203             if ((period-t) > t) {
204                 if (t == 0){
205                     triByte = 0;
206                 }
207                 else{
208                     triByte += triInc;

```

```

209     }
210   }
211   else{
212     triByte -= triInc ;
213   }
214   if ( triByte>255){
215     triByte = 255;
216   }
217   else if ( triByte<0){
218     triByte = 0;
219   }
220   wave = triByte ;
221   break;
222 case 2://saw
223 if (t==0){
224   sawByte=0;
225 }
226 else{
227   sawByte+=sawInc ;
228 }
229 wave = sawByte;
230 break;
231 case 3://sine
232 sinNum = t*sinInc ;
233 wave = pgm_read_byte_near(sine20000 + sinNum);
234 break;
235 }
236 PORTD = wave;
237 }
238 }
239
240 void loop() {
241   checkFreq(); //Check frequency values from POT in A5
242   checkShape(); //Check wave selection – Values from pins A0–A3
243   checkPW(); //Check Pulse Width from pin A4 in case wave shape is pulse
244   PORTB = 1<<type; //Send value to PORTB
245 }
```