Listing 3.13 Waveform Generator Source Code

```
// DACWAVE.C MPB 5-7-07
// Outputs waveforms to DAC, simulation file DAC.DSN

#include "16F877A.H"
#include "MATH.H"
#use delay(clock=20000000)
#use fast_io(D)                          // High speed output functions

int n, time=10;
float step, sinangle;
float stepangle = 0.0174533;             // 1 degree in radians
int amp[91];                             // Output instant voltage array

// ISR to read push buttons ***************************************

#int_rb
  void change()
  {
    if(time!=255)
      {if (!input(PIN_B4)) time ++;}              // Increase period
    while(!input(PIN_B4));

    if(time!=0)
      {if (!input(PIN_B5)) time--;}               // Decrease period
    while(!input(PIN_B5));

    if(!input(PIN_B6))reset_cpu();                // Restart program
    if(!input(PIN_B7))for(n=0;n<91;n++)amp[n]=0;  // Zero output
  }

void setwave()  // Arbitrary waveform values **********************
{
  amp[0] =00;amp[1] =00;amp[2] =00;amp[3] =00;amp[4] =00;
  amp[5] =00;amp[6] =00;amp[7] =00;amp[8] =00;amp[9] =00;
  amp[10]=10;amp[11]=00;amp[12]=00;amp[13]=00;amp[14]=00;
  amp[15]=00;amp[16]=00;amp[17]=00;amp[18]=00;amp[19]=00;
  amp[20]=20;amp[21]=00;amp[22]=00;amp[23]=00;amp[24]=00;
  amp[25]=00;amp[26]=00;amp[27]=00;amp[28]=00;amp[29]=00;
  amp[30]=30;amp[31]=00;amp[32]=00;amp[33]=00;amp[34]=00;
  amp[35]=00;amp[36]=00;amp[37]=00;amp[38]=00;amp[39]=00;
  amp[40]=40;amp[41]=00;amp[42]=00;amp[43]=00;amp[44]=00;
  amp[45]=00;amp[46]=00;amp[47]=00;amp[48]=00;amp[49]=00;
  amp[50]=50;amp[51]=00;amp[52]=00;amp[53]=00;amp[54]=00;
  amp[55]=00;amp[56]=00;amp[57]=00;amp[58]=00;amp[59]=00;
  amp[60]=60;amp[61]=00;amp[62]=00;amp[63]=00;amp[64]=00;
  amp[65]=00;amp[66]=00;amp[67]=00;amp[68]=00;amp[69]=00;
```

```
  amp[70]=70;amp[71]=00;amp[72]=00;amp[73]=00;amp[74]=00;
  amp[75]=00;amp[76]=00;amp[77]=00;amp[78]=00;amp[79]=00;
  amp[80]=80;amp[81]=00;amp[82]=00;amp[83]=00;amp[84]=00;
  amp[85]=00;amp[86]=00;amp[87]=00;amp[88]=00;amp[89]=00;
  amp[90]=90;
}

void main()  //************************************************
{
  enable_interrupts(int_rb);      // Port B interrupt for buttons
  enable_interrupts(global);
  ext_int_edge(H_TO_L);
  port_b_pullups(1);
  set_tris_D(0);

  // Calculate waveform values ***********************************

  step=0;
  for(n=0;n<91;n++)
  {
    if(!input(PIN_B0)) amp[n] = 100;     // Square wave offset
    if(!input(PIN_B1))                   // Calculate sine values
    { sinangle = sin(step*stepangle);
      amp[n] = floor(sinangle*100);
      step = step+1;
    }
    if(!input(PIN_B2)) amp[n] = n;       // Triangular wave
    if(!input(PIN_B3)) setwave();        // Arbitrary wave
  }

  // Output waveform vales ***************************************

  while(1)
  { for(n=0;n<91;n++) {output_D(100+amp[n]); delay_us(time);}
    for(n=89;n>0;n--) {output_D(100+amp[n]); delay_us(time);}
    for(n=0;n<91;n++) {output_D(100-amp[n]); delay_us(time);}
    for(n=89;n>0;n--) {output_D(100-amp[n]); delay_us(time);}
  }
}
```

incrementing and decrementing count. The sine output is the most interesting, as it is calculated using the sine function from the `math.h` library. These values are assigned to the `amp[n]` array for output after being calculated, since to calculate each and output it "on the fly" would be too slow.

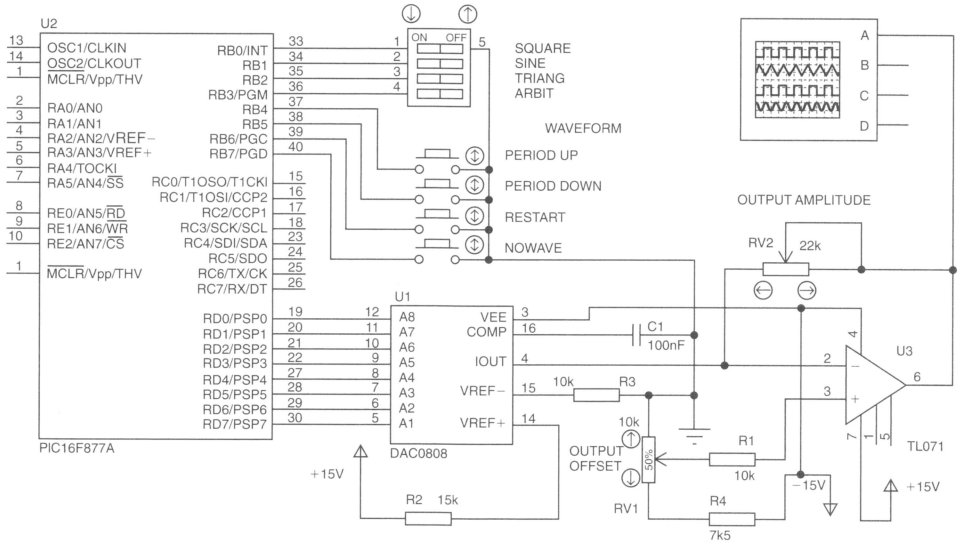

Figure 3.12: Waveform Generator

The program source code is shown in Listing 3.13. This is only a demonstration of the digital waveform generator principle, and a more sophisticated design is required to produce a waveform with a better resolution at higher frequencies. It serves only to illustrate some relevant features of C and the principle of waveform synthesis that may be used in high-performance digital signal processors, such as the dsPIC range. This is an application where critical sections of the code could be written in assembler for higher speed.

The main object of the program is to generate instantaneous voltages in sequence to produce a square, sine, triangular, and arbitrary waveform. The mid-value for the output is 100_{10}. Instant values ranging between −100 and +100 are added to this value to produce the output.

For the arbitrary pattern, most values are 0 in this example, with an increasing value at intervals of ten steps. This produces a pulse-modulated triangular waveform, which might be used to test a digital filter, but any other repetitive pattern can be entered as required. The arbitrary sequence is generated from the values entered into the array `amp[n]` in the function `setwave()` at the source code edit stage. A mechanism for entering these externally in hardware could easily be added, but that is rather tedious to demonstrate.

For the other waveforms, the values are calculated. The square wave is just a set of constant maximum (+100) and minimum (−100) values, and the triangular wave is an