```
//  Ascii to Morse code Noise maker     by Leland Boelens 09/29/2018

      const int Rly1=2;      // Digital output to drive relay
      const int Mot1=3;      // PWM output pins to drive vibrating motors
      const int Mot2=5;      // or also additional digital outputs
      const int Mot3=6;
      const int Mot4=9;
      const int Mot5=10;
      const int Mot6=11;

void setup()          // ****  INITIALIZATION    ************
{     // initialize a pro mini ATMega 328 surface mount @ 5V_16 MHZ

        // initialize pins and turn off the output driver
     pinMode(Rly1, OUTPUT), digitalWrite(Rly1, LOW);

     pinMode(Mot1, OUTPUT), digitalWrite(Mot1, LOW);
     pinMode(Mot2, OUTPUT), digitalWrite(Mot2, LOW);
     pinMode(Mot3, OUTPUT), digitalWrite(Mot3, LOW);
     pinMode(Mot4, OUTPUT), digitalWrite(Mot4, LOW);
     pinMode(Mot5, OUTPUT), digitalWrite(Mot5, LOW);
     pinMode(Mot6, OUTPUT), digitalWrite(Mot6, LOW);

        // initialize the analog inputs wired to the trim pots
     pinMode(A2, INPUT);
     pinMode(A3, INPUT);

        // program unused analog inputs as INPUT_PULLUP
     pinMode(A0, INPUT_PULLUP);
     pinMode(A1, INPUT_PULLUP);
     pinMode(A4, INPUT_PULLUP);
     pinMode(A5, INPUT_PULLUP);
     pinMode(A6, INPUT_PULLUP);
     pinMode(A7, INPUT_PULLUP);
}
       //         setup global values
       //
     // Array contains bytes of morse code and a bit count for the
     // dits and dahs pointed to by an ASCII character reference.
     // The 3 LSBits contain a value of 1, 2, 3, 5, and 6, with "7"
     // representing a 4, also with (0 and 4 representing a 6). The
     // choice of 0 and 4 representing a 6 bit Morse Code is that
     // the LSB of the 6 bit code is in the 4 bit position and the
     // 2 and 1 bit positions are both at 0.    The first test is to
     // eliminate a 6 being present, then do the conversion of 0 and 4
```

```
// if present to a bit count of 6, and then convert the 7 if
// present to a bit count of 4.
// The usuable numbers of 1,2,3,4,5,& 6 are present with a Morse
// Code dit=0, and dah=1, of 1 to 6 bits contained in Mcode packed
// with zeros to the left but only reading the # bits in the bit count

const byte Ascii_Morse20[16][1]={
                                B110, // no code
                                B11010100,    // "!"
                                B01001000,    // """
                                B110, // no code
                                B110, // no code
                                B110, // no code
                                B00010101,    // "&"
                                B01111000,    // "'"
                                B01101101,    // "("
                                B10110100,    // ")"
                                B110, // no code
                                B01010101,    // "+"
                                B11001100,    // ","
                                B10000100,    // "-"
                                B10101000,    // "."
                                B01001101,    // "/"
                                };

const byte Ascii_Morse30[16][1]={
                                B11111101,    // "0"
                                B11110101,    // "1"
                                B11100101,    // "2"
                                B11000101,    // "3"
                                B10000101,    // "4"
                                B00000101,    // "5"
                                B00001101,    // "6"
                                B00011101,    // "7"
                                B00111101,    // "8"
                                B01111101,    // "9"
                                B00011100,    // ":"
                                B110, // no code
                                B110, // no code
                                B10001101,    // "="
                                B110, // no code
                                B00110000,    // "?"
                                };
```

```
const byte Ascii_Morse40[16][1]={                    // upper case
                                B01011000,    // "@"
                                B10010,       // "A"
                                B0001111,     // "B"
                                B0101111,     // "C"
                                B001011,      // "D"
                                B0001,        // "E"
                                B0100111,     // "F"
                                B011011,      // "G"
                                B0000111,     // "H"
                                B00010,       // "I"
                                B1110111,     // "J"
                                B101011,      // "K"
                                B0010111,     // "L"
                                B11010,       // "M"
                                B01010,       // "N"
                                B111011,      // "O"
                                };

const byte Ascii_Morse50[16][1]={                    // upper case
                                B0110111,     // "P"
                                B1011111,     // "Q"
                                B010011,      // "R"
                                B000011,      // "S"
                                B1001,        // "T"
                                B100011,      // "U"
                                B1000111,     // "V"
                                B110011,      // "W"
                                B1001111,     // "X"
                                B1101111,     // "Y"
                                B0011111,     // "Z"
                                B110, // no code
                                B110, // no code
                                B110, // no code
                                B110, // no code
                                B110, // no code
                                };

const byte Ascii_Morse60[16][1]={                    // lower case
                                B110, // no code
                                B10010,       // "A"
                                B0001111,     // "B"
                                B0101111,     // "C"
                                B001011,      // "D"
                                B0001,        // "E"
```

```
                                    B0100111,     // "F"
                                    B011011,      // "G"
                                    B0000111,     // "H"
                                    B00010,       // "I"
                                    B1110111,     // "J"
                                    B101011,      // "K"
                                    B0010111,     // "L"
                                    B11010,       // "M"
                                    B01010,       // "N"
                                    B111011,      // "O"
                                    };

    const byte Ascii_Morse70[16][1]={                    // lower case
                                    B0110111,     // "P"
                                    B1011111,     // "Q"
                                    B010011,      // "R"
                                    B000011,      // "S"
                                    B1001,        // "T"
                                    B100011,      // "U"
                                    B1000111,     // "V"
                                    B110011,      // "W"
                                    B1001111,     // "X"
                                    B1101111,     // "Y"
                                    B0011111,     // "Z"
                                    B110, // no code
                                    B110, // no code
                                    B110, // no code
                                    B110, // no code
                                    B110, // no code
                                    };



//const char testString[] ={"The quick brown fox jumped over the lazy dog's back
0123456789 times     "};
//const char testString[]={"SOS   "};
//const char
testString[]={"thequickbrownfoxjumpedoverthelazydog'sback0123456789times  "};
 const char testString[]={"eeekkeeka.33??,,...,,''''"};
```

```
 const byte bitDelay=(80);
            //     ************  MAIN  LOOP   ***********************
void loop(){
  byte n=0;
  byte ascii=0;
  byte AsciiCount=0;

  // Read in an ASCII character

  AsciiCount=sizeof(testString);    // get # characters +"/0" (null) in array
  for (n=0; n< (AsciiCount-1); n++)
  {
  ascii= testString[n];             // get a character from ASCII string
  sendAscii(ascii);                 // send ASCII out to be converted to
  }                                 // Morse Code, send bits and repeat.

  }   //            ***********  end bracket for main loop  *****************


   //       ****************************************************************
   //                  Process ASCII into Morse Code and transmit code
   //       ****************************************************************

void sendAscii(byte ascii)      // the variable ascii is typed and loaded
{
  // ascii sent over when function is executed

  byte Mcode=0;
  byte count=0;
  byte ditORdah=0;

if(ascii==0x20)         // test if an Ascii space character was sent over

{
  delay( bitDelay*7 ); // found space in Ascii text, put delay between words.
  goto NoCodeExit;
}

  Mcode = Ascii2Morse(ascii);  // get the Morse code byte (code + bit count)

  count = (Mcode & 0x07);    // Mask out the bit count for the morse code

      // Test the lower 3 bits for 6 = no code
  if (count == 0x06) { goto NoCodeExit;}
      // if == B110, not a valid morse code. Exit.
```

```
 // to get here the possible 3 bit count could be 0, 1, 2, 3, 4, 5, or "7" which
 // will change to 4. The above test weeds out a 6 which represents "no morse
code"
 // and bypasses the next step to send out the dits and dahs.
 // If a 4 or 0 is present it is bit "D2" the LSB "1 or 0" of a 6 bit morse code.
 // The 0 or 4 will be changed to a 6, and extract all 6 data bits from the byte.
 // Finally, if a 7 is present, it is changed into the 4 it needs to be.

 // The conversion tests follow

  if (count == ((0x0) | (0x4))) // if 6 bit morse code, then set count to 6 &
shift
  { count=6, Mcode=Mcode >>2; }  // right by 2 to align the D2 bit with LSB D0

  else{ Mcode=Mcode>>3;} // else right shift by 3 to align the 1 to 5 bit codes

  if (count==7) { count=0x04; }
           // since 1,2,3,5,& 6 are already BCD make B111=B100 which is 4

  // output the code using 0 for dit, 1 for dah, transmitting 1 to 6 bits

  // send out morse code bit by bit separating the dits and dahs

  while (count != 0)
  {
  ditORdah = bitRead(Mcode, 0);     // sort out the dit and dah to send
  if (ditORdah==0)  { dit(); }
    else { dah(); }

  Mcode=Mcode>>1;         // shift the next bit into position to read
  count=count-1;         // advance count toward zero for the exit loop test
  }

NoCodeExit:     // proper exit point after a dit or dah would be transmitted

    delay( bitDelay *3 ); //  delay between characters
}
```

```
  //       ***************************************************************
  //                    ASCII to Morse Code conversion
  //       ***************************************************************

 byte Ascii2Morse( byte data)
 {
 byte Mcode=0;
 byte lowNibble= data & 0x0F;
 byte highNibble= data & 0xF0;
 highNibble=((highNibble >>4 ) & 0x0F);

  if ((data >= 0x20) & (data <= 0x7F))    // Ascii range test
  {
   // Ascii is in range, get the corresponding morse code
   // D7--------D3 & sometimes D2 is code..  D2---D0 # bits to send

    if (highNibble==2) {Mcode=Ascii_Morse20[lowNibble][0];}
  else if (highNibble==3) {Mcode=Ascii_Morse30[lowNibble][0];}
    else if (highNibble==4) {Mcode=Ascii_Morse40[lowNibble][0];}
      else if (highNibble==5) {Mcode=Ascii_Morse50[lowNibble][0];}
        else if (highNibble==6) {Mcode=Ascii_Morse60[lowNibble][0];}
          else if (highNibble==7) {Mcode=Ascii_Morse70[lowNibble][0];}
  }
     else { Mcode= 0x06;} // Ascii invalid, insert no morse code 0x06

     return(Mcode);
  }

  //       ***************************************************************
  //                    Morse Code dit and dah output routines
  //       ***************************************************************

void dit()
{
 int freq=analogRead(A2);
 freq=freq>>2;
 digitalWrite(Rly1, HIGH);
 tone(9, 1500);
 analogWrite(Mot1, 255);
 delay(2);         // get motor started at full speed
 analogWrite(Mot1, freq);  // Then dial back
 delay(bitDelay);
 analogWrite(Mot1, 0);  // motor off
 digitalWrite(Rly1, LOW);
 noTone(9);
```

```
  delay(bitDelay);
}

void dah()
{
  int freq=analogRead(A2);
  freq=freq>>2;
  digitalWrite(Rly1, HIGH);
  tone(11, 1370);
  analogWrite(Mot3, 255);
  delay(2);         // get motor started at full speed
  analogWrite(Mot3, freq);  // Then dial back
  delay(bitDelay*3);
  analogWrite(Mot3, 0);  // motor off
  digitalWrite(Rly1, LOW);
  noTone(11);
  delay(bitDelay);
}

   //        ****************************************************************
   //                    USER TEST FUNCTIONS
   //        ****************************************************************

void SOS()   // a test to drive the dit and dah functions, to test that code
             // this program was built from the output toward the Ascii input
{
  dit();
  dit();
  dit();
  dah();
  dah();
  dah();
  dit();
  dit();
  dit();
  delay(600);
}


  void test1()       // a test to verify that the hardware is working
  {
  int freq=0;
  int period=0;

     digitalWrite(Rly1, HIGH);
```

```
   delay(100);
   digitalWrite(Rly1, LOW);
   delay(500);


freq=analogRead(A2);
period=analogRead(A3);

 freq = freq>>2;          // divide by 4 to scale analog read for PWM

 // write the PWM value to each motor
 analogWrite(Mot1, freq);
 analogWrite(Mot2, freq);
 analogWrite(Mot3, freq);
 analogWrite(Mot4, freq);
 analogWrite(Mot5, freq);
 analogWrite(Mot6, freq);

  }
```