1300 Henley Court
Pullman, WA 99163
509.334.6306
www.store.digilent.com

Lab 7a: Digital Signal Generator

Revised April 25, 2017
This manual applies to Unit 7, Lab 7a.

1 Objectives

1. Use a programmable IIR filter to generate a single frequency sine wave.
2. Use PWM to generate an audible output tone over the Basys MX3 speaker.
3. Display the frequency of the synthesized sine wave on the LCD and 7-segment display.

2 Basic Knowledge

1. Knowledge of C or C++ programming
2. Working knowledge of MPLAB® X IDE
3. How to display text on a character LCD
4. How to display numbers on the 4-digit 7-segment display
5. Understanding of Finite Impulse Response Digital Filters

3 Equipment List

3.1 Hardware

1. Basys MX3 trainer board
2. Workstation computer running Windows 10 or higher, MAC OS, or Linux
3. Standard USB A to micro-B cables

In addition, we suggest the following instruments:

4. Analog Discovery 2

3.2 Software

The following programs must be installed on your development work station:

1. Microchip MPLAB X® v3.35 or higher
2. PLIB Peripheral Library
3. XC32 Cross Compiler
4. WaveForms 2015 (if using the Analog Discovery 2)
5. Iowa Hills Software for IIR and FIR Filters

4 Project Takeaways

1. How to implement digital filters in C using a PIC32 microprocessor.
2. How to use the PIC32 processor to make a signal generator.
3. How to create analog output using pulse-width modulation.
4. How to use change frequency of synthesized signals.

5 Fundamental Concepts

5.1 IIR Digital Filters

As introduced in Unit 7, an infinite impulse response (IIR) digital filter can be mathematically expressed by Eq. 5.1 below. Zeros of the transfer function, H(z), are the roots of the numerator polynomial. Poles of the transfer function are the roots of the denominator polynomial. We will use specific placement of the poles and zeros to cause the output to ring or oscillate indefinitely at specific frequencies.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{M-1} b_i \cdot z^{-i}}{\sum_{k=0}^{N-1} a_k \cdot z^{-k}} = \frac{b_0 + b_1 \cdot z^{-1}\ b_2 \cdot z^{-2} + \cdots b_{M-1} \cdot z^{-M-1}}{a_0 + a_1 \cdot z^{-1}\ a \cdot z^{-2} + \cdots a_{N-1} \cdot z^{-N-1}} \quad \text{Eq. 5.1}$$

From a table of Z transforms, the z transform of a sine wave is a second order IIR filter that is expressed by Eq. 5.2. It has two complex poles at $e^{\pm j2\pi Fo/Fs}$ and two zeros at the origin.

$$H(z) = \frac{sin(2\pi f_0/F_S) \cdot z^{-1}}{1 - 2 \cdot \cos\left(\frac{2\pi f_0}{F_S}\right) \cdot z^{-1} + z^{-2}} = \frac{Z_1 \cdot z^{-1}}{(1 - p_0 \cdot z^{-1}) \cdot (1 - p_1 \cdot z^{-1})} \quad \text{Eq. 5.2}$$

5.2 The Z-plane Unit Circle

The z plane is a unit circle that describes a surface of complex numbers using polar coordinates. Poles and zeros are the roots of the denominator and numerator polynomials, respectively expressed in Eq. 5.1 or Eq. 5.2. These roots can be represented in the frequency domain as a vector, as shown in Fig. 5.1 and expressed by Eq. 5.3. The variable *r* represents the magnitude of the vector and the angle Θ represents the frequency normalized by the sampling frequency, expressed in radians.

$$p_{0,1} = r_i \cdot e^{\pm j\theta_i} \quad \text{Eq. 5.3}$$

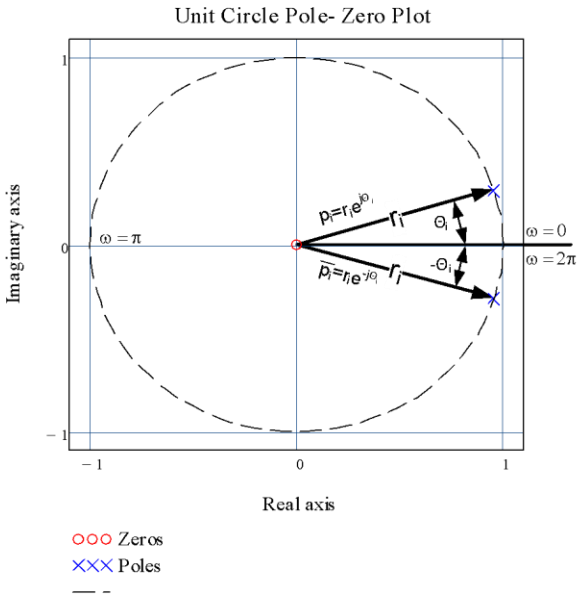

Figure 5.1. Complex poles plotted on the z-plane for the 7500 Hz sine wave oscillator.

It is beneficial to understand how the pole-zero plots on the z-plane are representations of digital filters. To explain it in a single sentence, digital filters can be written as transfer functions in the form of Eq. 5.2, the poles and zeros of which can be expressed in the frequency domain as a vector. Figure 5.1 illustrates the set of complex poles for the transfer function of a sine function at 7500 Hz when the sampling frequency, Fs, is set to 160000 Hz. Since the digital filter design process generates a unique polynomial, the placement of the poles and zeros in the z-plane are equally unique. Zeros on the unit circle will cause the numerator of the transfer function to be zero, causing the filter to output a zero amplitude signal when excited at the frequency, ω, corresponding to the angle Θ. Zeros at the origin contribute nothing to the dynamics of the transfer function.

In contrast, poles on the unit circle cause the denominator of the transfer function to equal zero, hence the output of the filter will be infinite and it will oscillate indefinitely when excited at the frequency, ω, corresponding to the angle Θ. Poles outside the unit circle indicate that the filter is unstable and the amplitude of the oscillations will increase toward infinity. Poles just inside the unit circle will cause the filter to oscillate; however, the amplitude is bounded (fixed maximum). We will take advantage of this characteristic to make the eight oscillators used in Lab 7a.

The poles in Fig. 5.1 are those represented by blue X's and the zeros by red circles. Note that the poles and zeros that do not lie on the real axis exist in complex conjugate pairs. The z-plane frequency, ω, relates to the real-world signal frequency by noting that ω = π for f = the Nyquist rate, which is half of the sampling rate in samples per second.

5.3 An IIR Digital Oscillator

Although as References 6 and 7 prove the recursive filter signal generation using the fixed point math approach is not novel, it does offer significant memory and computational efficiency over the commonly used table look-up methods and algorithms based on trigonometry functions. The Timer2 ISR for the IIR approach to signal generation requires 1.6582 µs of the 6.25 µs PWM period, resulting in 26% CPU time loading. The *gen_tones* IIR function requires only 364 bytes of program memory and 24 bytes of data storage. Although the table look-up method requires minimal program memory, to achieve the same degree of resolution will require 160 K bytes of data storage memory. Table look-up methods can use fixed point mathematics. The look-up tables themselves use

floating point sine functions when generated during an initialization, but only to that extent. The function method uses the sine trigonometric function to generate the output in real time. Since the trigonometric function method uses floating point math throughout the signal synthesis process, it is confined to low frequency applications. However, this method does offer the highest degree of resolution.

From the previous discussion, we can generate an oscillator of any frequency between zero and half the sampling frequency by using Eq. 5.2, or by placement of the poles in Eq. 5.3. In either case, the desired oscillator transfer function is computed using Eq. 5.4. Since the input to the digital filter is a unit impulse, it only exists for the computation of the y[1] output, which can be set as the digital filter initial condition without the need for an input signal at all.

$$y[n] = -a_1 \cdot y[n-1] - a_2 \cdot y[n-2], with\ a_1 = -\cos\left(\frac{2\pi F_0}{Fs}\right), a_2 = 1, y[1] = \sin\left(\frac{2\pi F_0}{Fs}\right), and\ y[0] = 0, for\ n > 1 \qquad \text{Eq. 5.4}$$

The C program functions shown in Appendix A provide code to create a sine wave at 5000 Hz. Listing A.1 defines the global constants and declares the public functions and variables. Listing A.2 initializes the audio output for the defined 5000 Hz oscillator and is called from the main function infinite loop. The filter coefficients for the default 5000 Hz IIR filter are configured here for convenience. The variable *tones_active* is set in Listing A.3 and starts the oscillator and is reset in the main function loop to stop the oscillator.

Listing A.3 provides for the capability to generate different frequencies using a common IIR filter algorithm shown in Listing A.4. Since the oscillator is started with an impulse input, the outputs are initialized to start with the pre-computation of y[n-1], as shown in Eq. 5.4. The function in Listing A.3 is called to start the oscillator output.

The function shown in Listing A.4 is called from the Timer 2 ISR shown in Listing A.5 at the PWM period rate if the *tones_active* variable is set TRUE.

6 Problem Statement

You are to create a digital sine wave generator capable of synthesizing eight constant amplitude signals at frequencies specified in Section 8.1. The frequency of the sine waves will be selected by setting one of the eight slide switches. The audible output will be enabled and the frequency displayed on the 4-digit 7-segment display for five seconds when BTND is pressed. The LCD will display the frequency of the selected sine wave.

7 Background Information

See the discussion of IIR filters in section 6.3.2 of the Unit 7 text.

8 Lab 7a

The objective for this lab is to create a programmable digital oscillator as modeled in Fig. 8.1.

8.1 Requirements

1. The peripheral bus clock is to use a divisor of 1 (set in config_bits.h).

2. The PWM period is to be set for 160000 Hz. (This PWM period was selected to take advantage of the analog low-pass filtering of the PWM output to produce a clean sine wave shown in Fig. 8.4.)
3. The eight Basys MX3 slide switches are set to generate a single frequency tone as listed in Table 8.1.
4. When BTND is not pressed, the LCD will display as shown in Fig. 8.2.
5. When BTND is pressed, the LCD will display as shown in Fig. 8.3 and output the tone selected for 5 seconds.
6. The frequency of the 8 synthesized tones are to be measure using the Analog Discovery 2, as shown in Fig. 8.4, and added to Table 8.1 as shown below. The analog signal trace shown in Fig. 8.4 was captured using the MIC analog input. No code is required because the MIC on the Basys MX3 board is picking up the speaker audio. The amplitude of the signals you capture may differ from Fig. 8.4 because of the MIC Volume and Speaker Volume potentiometer settings.
7. Record amplitudes of the sine wave for the 8 tones without modifying the MIC Volume and Speaker Volume settings.
8. Record the 8 sets of filter constants for the frequencies listed in Table 8.1.

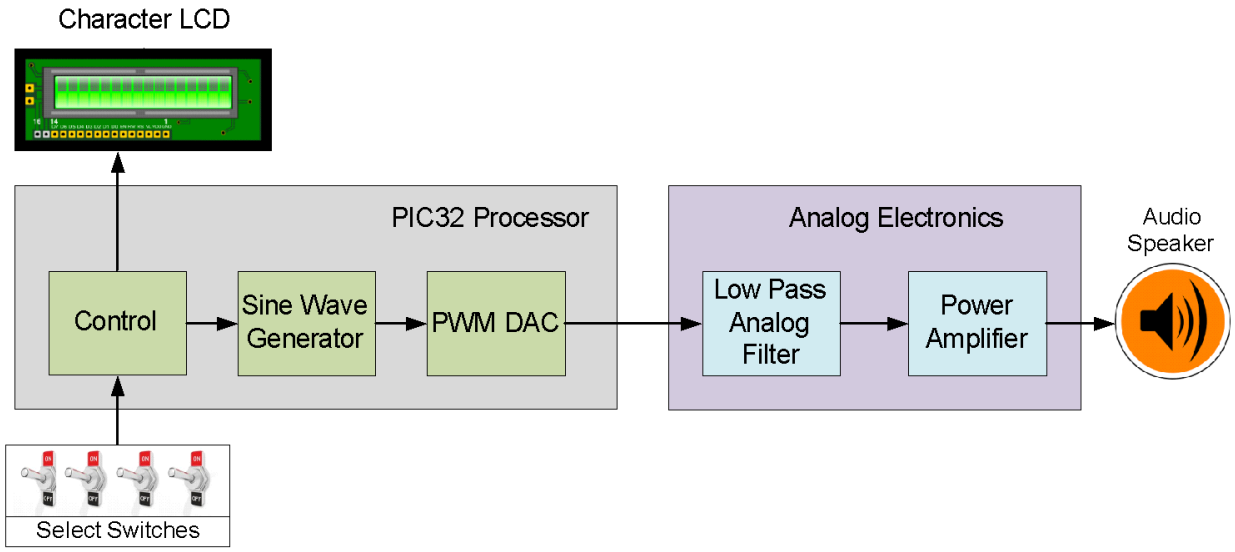

Figure 8.1. Block diagram for Lab 7a.

Table 8.1. Frequency Select.

On Switch	Osc. Frequency	IIR Filter Coefficients			Peak Amplitude	Measured Frequency	Frequency Error
		a_0	a_1	a_2			
SW0	500Hz						
SW1	1500Hz						
SW2	2500Hz						
SW3	3500Hz						
SW4	4500Hz	0x05A0	0x7E01	0x4000	± 1.0V	4496Hz	0.088
SW5	5500Hz						
SW6	6500Hz						
SW7	7500Hz						

Figure 8.2. Tone Generator display when not generating tone.

Figure 8.3. Tone Generator display when generating tone.

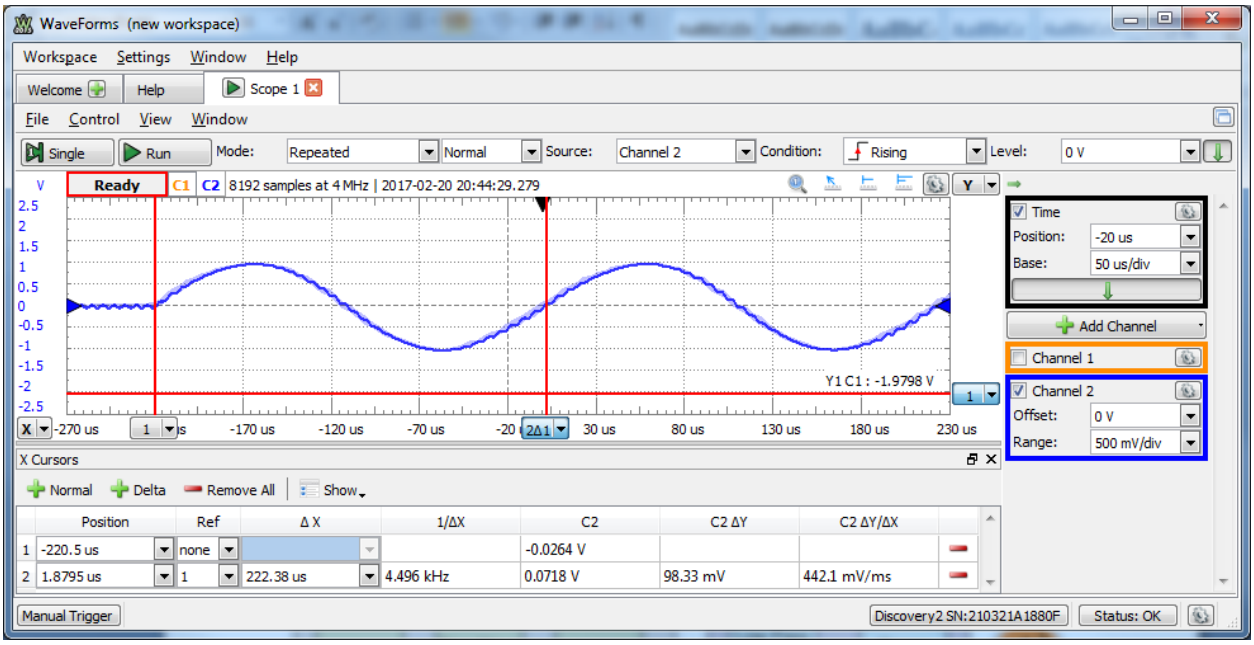

Figure 8.4. Audio signal capture for a 4500 Hz synthesized signal.

The control flow diagrams shown in Fig. 8.5 and Fig. 8.6 give an overview of an approach to this design. The code provided in Listings A.1 through A.5 can be used for the sine wave generator portion of Fig. 8.1. The Analog Electronics portion of Fig. 8.1 is provided on the Basys MX3 circuit board as shown in Fig. B.1 of Appendix B.

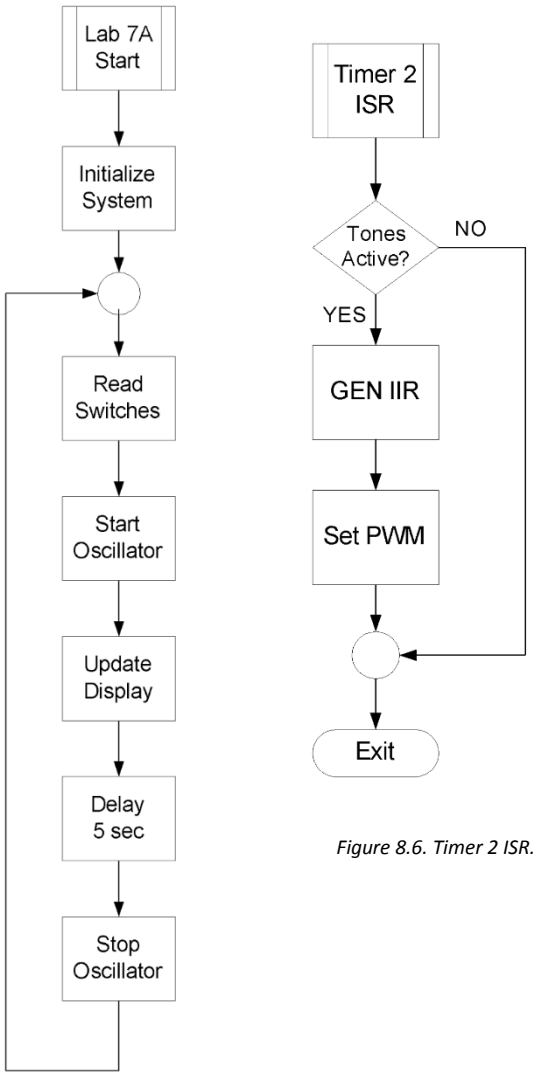

Figure 8.5. Main function control flow diagram.

Figure 8.6. Timer 2 ISR.

8.2 Design, Construction, Testing

In previous labs, steps were laid out to lead the student through effective design, construction, and testing procedures. This is now left up to the student to complete on their own, ensuring the design requirements listed in section 8.1 are met.

9 Questions

1. Since the computed poles for the digital oscillator lie exactly on the unit circle, what affect do you expect truncation and round off errors to have on the oscillator output?
2. How does the statistical integer rounding code used in the *gen_tones* function shown in Listing A.4 help prevent signal fading? (Hint: See what happens when you comment out that portion of that code and run the program.)
3. Why is it preferable to synthesize a sine function rather than a cosine function?
4. What precautions are necessary if a signal of more than one frequency is generated by adding outputs?
5. What conclusion do you draw from the data recorded in Table 8.1?

10 References

1. Basys MX3 Trainer Board Reference Manual.
2. C Programming Reference.
3. PIC32 Family Reference Manual, Timers Section 14: http://ww1.microchip.com/downloads/en/DeviceDoc/61105E.pdf.
4. Steven W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing, http://www.dspguide.com/pdfbook.htm.
5. "Design of Digital Filters", https://web.eecs.umich.edu/~fessler/course/451/l/pdf/c8.pdf.
6. "A Fixed-Point Recursive Digital Oscillator for Additive Synthesis of Audio", T. Hodes, J. Hauser, J. Wawrzynek, A. Freed, and D. Wessel, http://www.jhauser.us/publications/1999_Hodes_DigitalOscillator.pdf.
7. "An Implementation Method for Low Frequency Digital Oscillator", L. Yan and H. Yang, http://ijssst.info/Vol-17/No-25/paper10.pdf.
8. The Synthesis of Sound by Computer, Section 4.2: Additive Synthesis, http://cmc.music.columbia.edu/musicandcomputers/chapter4/04_02.php.

Appendix A: Sine Wave Generator Software

Listing A.1. Definitions and Global Variable Declarations

```
   #define PWM_CYCLE_RATE  160000                          // PWM frequency
   #define PWM_ZERO_OFFSET ((int) (PWM_CYCLE_RATE >> 1))    // 50/50 PWM output
   #define PWM_MAX         ((int) ( GetPeripheralClock() / PWM_CYCLE_RATE ))
   #define Q14             (1 << 14)             // Filter scale factor
   #define PIx2            ((float) (3.14159265359 *2))
    #define TONE            5000                            // Test frequency

   struct coefficients           // IIR denominator coefficients
   {
       int a0;
       int a1;
       int a2;
   } ;

   typedef struct coefficients filter;

// Public functions
       void audio_init(void);
       void start_tones(int tone);

// Global variables
static filter tone_filter;       // Current Filter coefficients
static filter filt_f1;           // Computed filter coefficients
int tones_active = FALSE;        // Generate output control
static int y[3] = {0};           // Filter output array
```

Listing A.2. Initialize Audio Output

```
void audio_init(void)
{
       // H(z) = z(sin(wo*T)) / (z^2 - 2*z*cos(wo*T) + 1)

float w;
   w = ((float) (TONE * pi2)) / PWM_CYCLE_RATE;
   coeffs->a0 = (int) (sin(w) * Q14/2); // Limit initial impulse to 1/2
   coeffs->a1 = (int) (2*cos(w) * Q14);
   coeffs->a2 = Q14;

   tones_active = FALSE;               // Disable Timer ISR oscillator

       // Initialize PWM output following the procedure used for Lab 6a but use RB14
       // for Output Compare Channel 1
       // Set the PWM period for PWM_CYCLE_RATE
       // Initialize PWM output for 50% duty cycle
}
```

Listing A.3. Initialize Tone Filter

```
void start_tones(int tone)
{
   tone_filter.a0 = filt_f1.a0;         // Initialize filter coefficients
   tone_filter.a1 = filt_f1.a1;
   tone_filter.a2 = filt_f1.a2;
   y[0] = tone_filter.a0;               // Initialize oscillator with impulse
   y[1] = 0;
   y[2] = 0;
   tones_active = TRUE;                 // Enable Timer ISR oscillator
```

```
}
```

Listing A.4. Tone Generating IIR Filter

```
static void gen_tones(void)
{
int pwm;

// Oscillator IIR filter
   y[2] = y[1];                  // Shift outputs and compute new output
   y[1] = y[0];

// Compute IIR result
   y[0] = (tone_filter.a1 * y[1] - tone_filter.a2 * y[2]);

// Provide rounding function to avoid truncations due to Fixed Point math because of
// division by Q14 that causes oscillations to die out.
   if(y[0] < 0)
   {
       y[0] = y[0] - Q14/2;
   }
   else
   {
       y[0] = y[0] + Q14/2;
   }

   y[0] = y[0] / Q14;              // Remove Q14 scaling for fixed point math

// Compute new pwm output
   pwm = (y[0] * PWM_MAX / Q14 ) + (PWM_MAX / 2);

// Limit output range for 0 the maximum PWM
   if(pwm < 0)
   {
       pwm = 0;
   }
   if(pwm > (PWM_MAX - 1) )
   {
       pwm = PWM_MAX - 1 ;
   }
   SetDCOC1PWM(pwm);             // Set PWM duty period
}
```

Listing A.5. Tone Generating Output Control

```
void __ISR(_TIMER_2_VECTOR, IPL2SOFT) Timer2Handler(void)
{
   if(tones_active == TRUE)    // Call IIR filter only if turned on
       gen_tones();
   INTClearFlag( INT_T2 );     // Clear the interrupt flag
} // End of Timer2Handler ISR
```

Appendix B: Audio Output Hardware

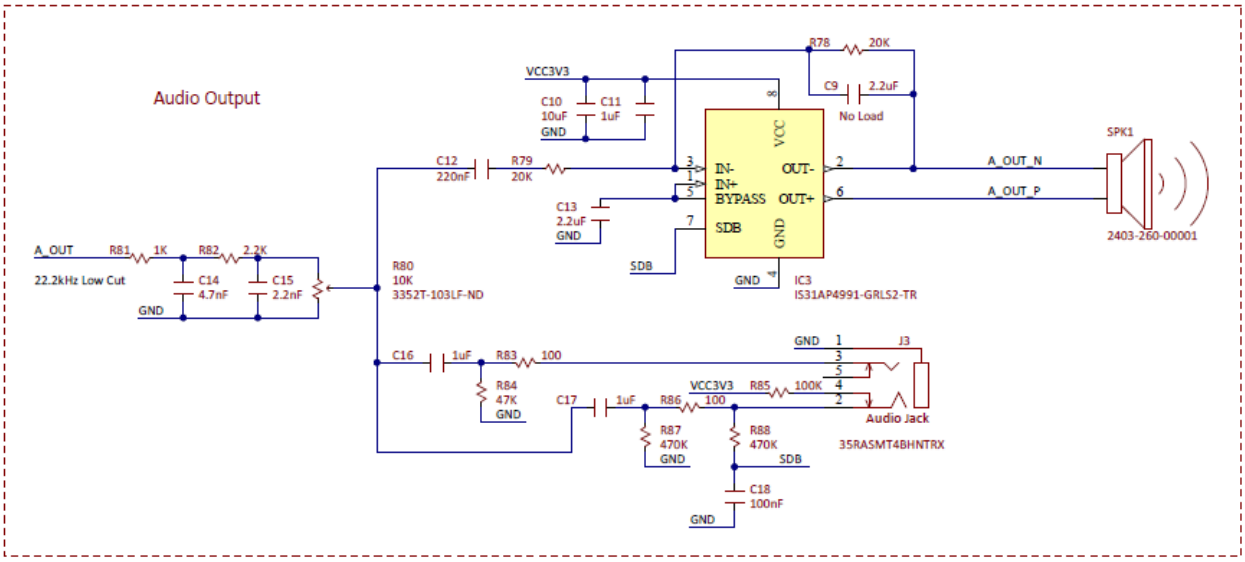

Figure B.1. Basys MX3 low-pass filter and audio amplifier.