Revision: May 7, 2014
Richard W. Wall, University of Idaho, rwall@uidaho.edu

www.digilentinc.com
1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Project 14: Discrete Fourier Transforms

Table of Contents

Purpose

The purpose of this project is to demonstrate how to use a PIC32 processor to decode the DTMF tones generated by a touch tone telephone. The method used in this project uses Discrete Fourier Transforms (DFT) to determine the energy if a signal in selected frequency bands. The DFT computation is based on the Goertzel Algorithm.

Minimum Knowledge and Programming Skills

1. Knowledge of C or C++ programming

2. Working knowledge of MPLAB® X IDE
3. LCD Interface (See Project 6)
4. Understanding of Finite Impulse Response Filters
5. Understanding of Discrete Fourier Transforms
6. The Goertzel DFT algorithm
7. **Development of the Goertzel Algorithm**
8. DTMF tones

Equipment List

1. chipKIT^{TM} Pro MX7 processor board with USB cable
2. PmodSTEP Stepper Motor Driver module (Used for performance timing)
3. Logic analyzer, or oscilloscope (Suggestion - Digilent Analog Discovery)
4. DTMF Tone Generator
5. Wave to C Code Converter

Software Resources

1. MPLAB ® X Integrated Development Environment (IDE)\
2. MPLAB ® XC32 C/C++ Compiler
3. XC32 ® C/C++ Compiler Users Guide
4. Wav to Code converter

References

1. chipKIT^{TM} Pro MX7 Board Reference Manual
2. C Programming Reference
3. PIC32 Family Reference Manual, Section 14: http://ww1.microchip.com/downloads/en/DeviceDoc/61105E.pdf
4. “A Discrete Fourier Transform Based Digital DTMF Detection Algorithm”, http://www.rootsecure.net/content/downloads/pdf/paper_dtmf.pdf
5. “Intelligent Detection of DTMF Tones using a Hybrid Signal Processing Technique with Support Vector Machines”, http://www.idsia.ch/~nagi/conferences/itsim_dtmf_detection.pdf

6. "System and method for precise DTMF signal detection US 6560331 B1"
http://www.google.com/patents/US6560331

7. "Performance of dual tone multi-frequency signal decoding algorithm using the sub-band non-uniform discrete Fourier transform on the ADSP-2192 processor."
http://www.scl.ece.ucsb.edu/rsharadh/DTMF_NDFT_2003.pdf

8. "Specification of Dual Tone Multi-Frequency (DTMF) Transmitters and Receivers"
http://www.etsi.org/deliver/etsi_es/201200_201299/20123502/01.01.01_60/es_20123502v010101p.pdf

DTMF

Touch tone dialing was introduced by AT&T for telephones in 1963 as a replacement for rotary dials[i]. The primary advantage of touch tone dialing over rotary dialing is that the dialing information can be carried in the audio band where as the dialing information for rotary dialing is DC. The rotary phones (circa 1891-present day) created pulses in the voltage supplied by the telephone exchange to power the telephone. Rotary dials are mechanical devices that use a cam to actuate a mechanical switch. Over time the mechanics of the rotary dial would break or begin to operate so slowly that the pulses could no longer be accurately decoded. Conventional touch tone operated telephones have 12 keys: 0 through 9 as well as the "*" and "#" keys. The full implementation of the touch tone key pad as shown in Figure 1 contains 16 keys. The additional four keys are designated A through D. Some phone systems use the additional keys for phone system management and control.

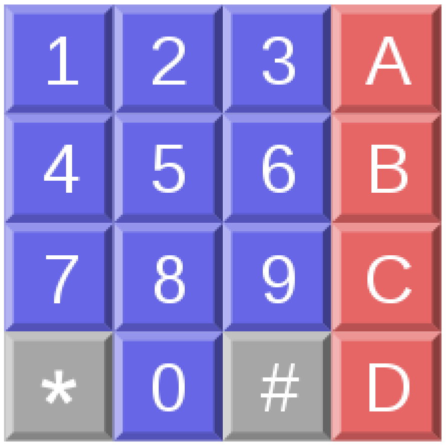


Figure 1. Key organization for 16 Key touch tone pad [ii]

Touch tone phones generate a multi tone signal containing the combination of two signals at set frequencies. As illustrated in Table I, there are four row frequencies (697 Hz, 770 Hz, 852 Hz, and 941 Hz) and four column frequencies (1209 Hz, 1336 Hz, 1477 Hz, and 1633 Hz). Whenever a key is pressed, the signal is generated that consists of the two frequencies specified by the key's row and column. DTMF standards specify that the frequencies of the eight DTMF tones be accurate to within 1.8%. This tolerance results in a tone band width of 25Hz based on the row one frequency to 59 Hz for the column four. This has ramifications when designing a digital filter to identify the tone pairs when a key is pressed.

Table I. DTMF keypad frequencies

	1209 Hz	**1336 Hz**	**1477 Hz**	**1633 Hz**
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Introduction to Goertzel Algorithm

The Goertzel Algorithm is named after its inventor and was first described by Gerald Goertzel in 1958[1]. It is an algorithm that is commonly used to compute the magnitudes of signals in a selected and restricted frequency bands based upon the computation of the Discrete Fourier Transform (DFT) expressed by Eq. 1.

$$\tilde{X}(k) = \sum_{n=0}^{N-1} \tilde{x}_n \cdot W^{n \cdot k}, \; W = e^{-i \cdot 2 \cdot \pi / N} \qquad n = 0, 1, 2, \cdots N-1 \qquad \text{Eq. 1}$$

$\tilde{x}_n$ denotes a truncated sequence of data samples and $\tilde{X}(k)$ is the complex magnitude of the k_{th} frequency component in the form $a_k + i\, b_k$ with *i* being the imaginary operator. The frequency represented by the k_{th} element is determined by the expression $\Delta\omega = 2 \cdot \pi \cdot \frac{Fs}{N}$ where *Fs* is the sample frequency rate and *N* is the truncated sequence length. It can be shown that the non recursive expression shown in Eq. 1 can converted to the recursive first order digital filter expressed by Eq. 2 and Eq. 3.

$$y(n) = W_N^{-k} \cdot y(n-1) + \tilde{x}(n), \; \textit{where } n = 1, 2, \cdots N-1 \textit{ and } y(0) = \tilde{x}(n) \qquad \text{Eq. 2}$$

[1] Goertzel, G. (January 1958), "An Algorithm for the Evaluation of Finite Trigonometric Series", *American Mathematical Monthly* **65** (1): 34–35, doi:10.2307/2310304

$y(N) = W_N^{-k} \cdot y(N-1) = \tilde{X}(k)$ Eq. 3

However Eq. 2 and Eq. 3 use the factor W_N^{-k} which is a complex variable that we would like to avoid when implementing in microprocessor code to gain the advantage of speed of computation.

Enduring additional mathematical gymnastics, the difference equation expressed by Eq. 2 can be expressed by Eq. 4.

$y(n) = \tilde{x}(n) - W_N^{-k} \cdot \tilde{x}(n-1) + 2 \cdot \cos(\alpha) \cdot y(n-1) - y(n-2),\ where\ \alpha = \frac{2\cdot\pi\cdot k}{N}$ Eq. 4

This recursive equation can be graphically represented as shown in Figure 2. The intermediate results expressed by V_k*(n)* and V_k*(n-1)* are used to compute the output. Eq. 3 tells that only the terms for V_k*(N)* and V_k*(N-1)* are needed to compute the final result for $\tilde{X}(k)$.

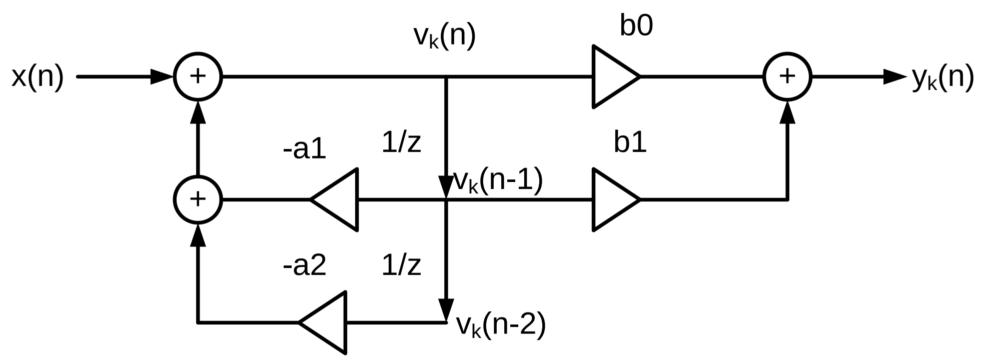


Figure 2. A Direct Form II flow diagram that describes the recursive form for computing the Goertzel Algorithm

The expression for determining the intermediate values V_k*(n)*, V_k*(n-1)* and V_k*(n-2)* is shown by Eq. 5.

$v_k(n) = \tilde{x}(n) - a1 \cdot v_k(n-1) - a2 \cdot v_k(n-2)$ Eq. 5

$where\ n = 2,3,\cdots N,\ v_k(0) = \tilde{x}(0),$
$v_k(1) = \tilde{x}(1) - a1 \cdot v_k(0)\ with\ \ a1 = \propto = -2 \cdot \cos\left(\frac{2\cdot\pi\cdot k}{N}\right)\ and\ a2 = 1$

If the samples of the signal are real-valued (normally the case) then all of the results for V_k*(n)*, V_k*(n-1)* and V_k*(n-2)* are real values because the coefficients a1 and a2 are real. The expression shown for Eq. 5 must be iterated until *n = N* before the values of V_k*(N)* and V_k*(N-1)* can be used in the output expression shown in Eq. 6.

$y(N) = b0 \cdot v_k(N) + b1 \cdot v_k(N-1) = \tilde{X}(k)\ where\ b0 = 1\ and\ b2 = W_N^{-k}$ Eq. 6

Since the coefficient *b2* is complex, the resulting computation for $\tilde{X}(k)$ is also complex. Squaring the results by multiplying $\tilde{X}(k)$ by its complex conjugate will allow the elimination of all complex values.

Again, through mathematical effort, the result can be expressed using only the values for $V_k(N)$ and $V_k(N\text{-}1)$ as shown in Eq. 7.

$$\left|\tilde{X}(k)\right|^2 = v_k(N)^2 - 2\cdot\alpha\cdot v_k(N)\cdot v_k(N-1) + v_k(N-1)^2,\ \propto\, = -2\cdot\cos\left(\frac{2\cdot\pi\cdot k}{N}\right) \qquad \text{Eq. 7}$$

Note that the term "α" used in Eq. 7 is the same as that used in Eq. 5. Hence, only one constant is required to compute the energy component of a signal at a specified frequency.

The blue lines in Figure 3 shows the results of taking the DFT of a signal that contains at all eight DTFM tones using a window that is sampled at 8KHz consisting of 205 samples. The red lines are the results of using the Goertzel algorithm specifically tuned for the eight DTMF frequencies. Even while using the same size of data set, improved tuning is achieved by not restricting the bin number used to generate the argument of the cosine function, α, to be an integer. Rather compute α using the expression shown in Eq. 8.

$$\propto\, = -2\cdot\cos\left(\frac{2\cdot\pi\cdot Fi}{Fs}\right)\ \textit{where Fi is the frequency of interest and Fs is the sample rate} \qquad \text{Eq. 8}$$

The improvement is evident by observing the graph shown in Figure 3 that shows three different approaches to computing energy-frequency spectrum of a signal containing all eight DTMF tones with equal amplitude. The DFT plot shows the results obtained by applying the conventional DFT transformation expressed by Eq. 1. The plot labeled G2 uses the Goertzel algorithm where the α term is computed as shown in Eq. 7. Lastly, the plot labeled G1 uses the Goertzel algorithm where the α term is computed as shown in Eq. 8. For all instances except on, the Goertzel algorithm that uses the α term as defined by Eq. 8 shows a greater sensitivity as indicated by the magnitude of the energy.

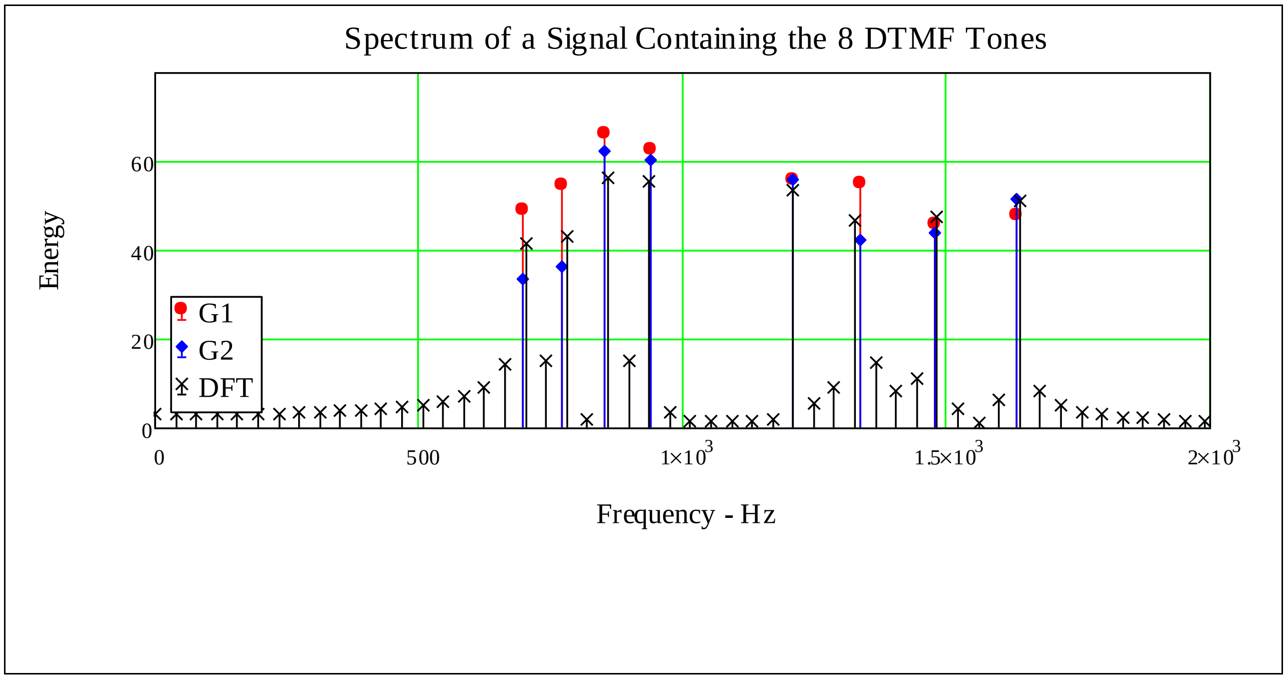


Figure 3. Comparison of DFT output and Goertzel Algorithm output of a signal containing all 8 DTMF tones

Using the recommend 205 for the block size and a sampling rate of 8 KHz results in a frequency resolution of 39 Hz. Consider the separation of frequency of the tones generated for row one and row 2. The difference of these two frequencies is 73 Hz. Using a sample block size of 205 Hz, Figure 3 clearly demonstrates that the energies of the row tones are separated by less than two DFT bins. The minimum threshold level can also be determined from Figure 3 by observing that any energy that exists due to a full amplitude adjacent tone will be below 20 as opposed to the Goertzel output of greater than 40 for tones at the frequency specified for the Goertzel algorithm.

Figure 4 show a plot of a signal containing DTMF tones that automatically generated at the specified maximum rate. The minimum periods of tone burst and inter-tone gap is 50 ms. It the DTMF tones are generated by someone pressing a touch tone key pad then the duration of the tone bursts and the period between the tone bursts will be much longer and we would not expect the intervals to be uniform. Hence the algorithm implemented in the processor must be adaptive to all possible tone burst patterns.

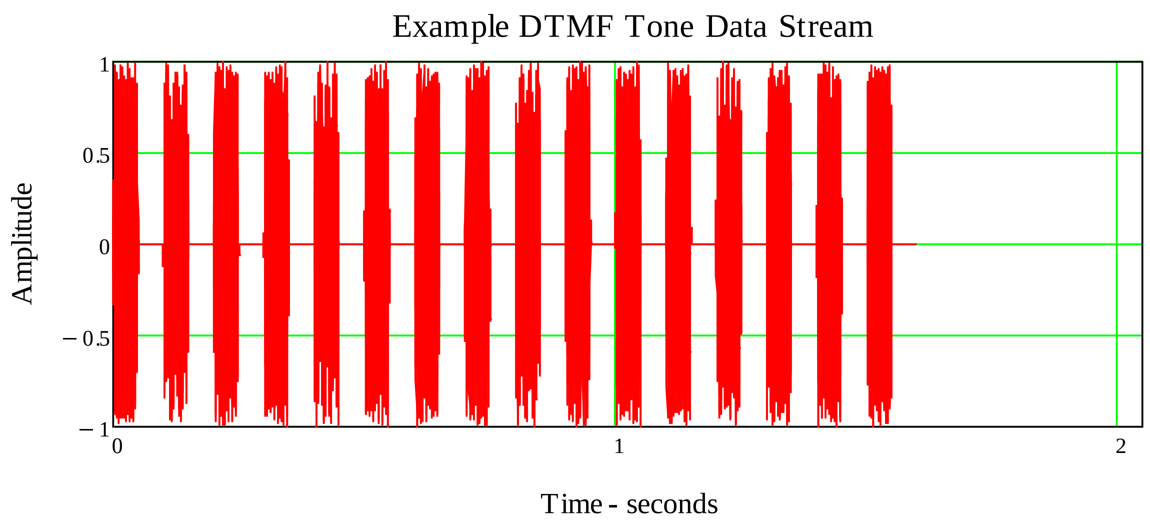


Figure 4. Plot of data for a sequence of DTMF tones spaced at 50 ms intervals.

C Code for Goertzel Algorithm:

The C code shown in Listing 1 performs the Goertzel DFT on a block of data for a single frequency. It is necessary to call this function eight times to determine the energy levels at the eight DTMF frequencies. A pointer to a signed character data array is passed as the first argument in the function call statement.

The second argument is the constant alpha that is associated with the frequency bin number as described above. The third parameter, N, is the length of the data array. An integer representing the scaled energy value associated with the frequency bin is returned by this function. This function uses only fixed point mathematics.

Listing 1. Goertzel Function

```
/* goertzel  FUNCTION DESCRIPTION *****************************************
SYNTAX:           int goertzel(signed char *blk, int alpha, int N);
KEYWORDS:         DFT, Goertzel, algorithm
DESCRIPTION:      This function computes the energy of the signal at a single
                  frequency using the Goertzel algorithm.
PARAMETER 1:      char pointer to the array of the block of sampled analog
                  data
PARAMETER 2:      The constant associated with the bin number for the
                  frequency of interest.
PARAMETER 3:      The number of elements (data points) to process
RETURN VALUE:     Integer value (32 bit signed) of the energy of the signal
NOTES:            None.
END DESCRIPTION **********************************************************/
int goertzel(signed char *blk, int alpha, int N)
{
int v0, v1, v2, y;
int i;
    v2 = blk[0];                /* Initialize filter */
    v1 = blk[1] + alpha*v2/AS;
    for(i=2; i<N; i++)          /* Digital filter algorithm */
    {
      V0 = blk[i] + (alpha*v1)/AS - v2; /* Use fixed point math */
      V2 = v1;                          /* Propagate past results */
      v1 = v0;
    }
    y = v1*v1 - (alpha*v1*v0)/AS + v0*v0; /* Square to compute energy */
    return  y/(N/2); /* Scale based upon block size */
} /* End of goertzel */
```

C Code for Project Implementation

The program code organization for this example is shown in Figure 6. The digitized samples analog signal containing the sequence of DTMF tones is stored in a memory array. The timer interrupt is programmed to read the next element out of the data file once each 0.125ms. This simulates reading an analog to digital converter that can sample an analog signal at 8 KHz. The Timer ISR sets a flag when a block of data of a preset size has been read into a buffer. The process control then passes the buffered block of data to the Goertzel Algorithm function that computes the energy level of the signal in the frequency band for the eight DTMF tones. The eight energy levels are stored in an eight element array. This array is passed to the tone decoder that compares the energy levels for the frequencies that correlate with the signals generated the row and column of the key that is pressed. This function determines which two tones (if any) have energy above a preset value. The "T*one Decoder*" function then returns an ASCII character value assigned to the key. This value is displayed on the LCD.

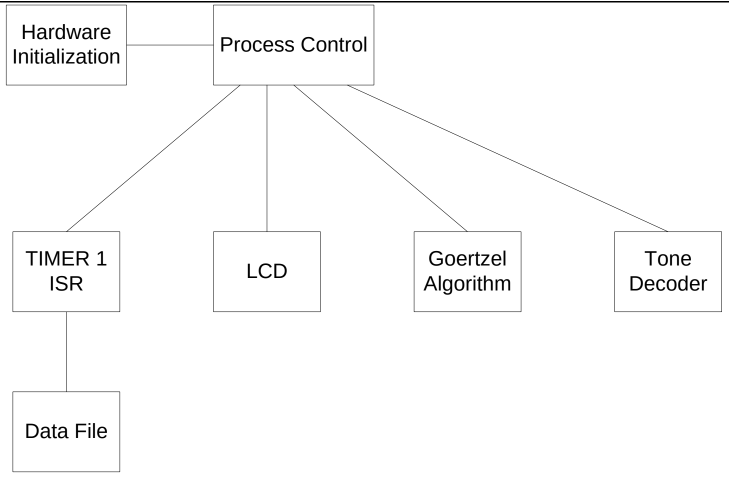


Figure 5. Software partitioning for the DTMF decoder

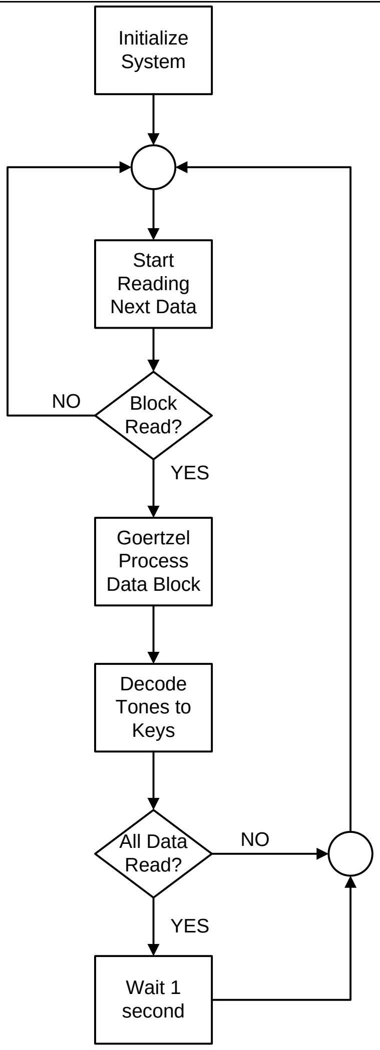


Figure 6. DTMF decoding control flow diagram

The control process implement for the demonstration program is shown in Figure 6 above. After initialization, the program waits for a flag to be set in the Timer interrupt service routing that indicates that a buffer of data is ready to be processed. Data collection is suspended until the background task resets this flag indicating that it is okay to fill the buffer with new data. The background task decodes each buffer of data to determine the key that was pressed as indicated by the two tones present or a special code that indicates that no tones were detected that were above a preset threshold value. An algorithm is used that rejects multiple decoding of the same key unless there is an intervening detection of no key pressed. The entire process is repeated after a delay of one second.

C Code for Dual Tone to Key conversion:

Listing 2 contains the C code that processes the array of eight integer values that contain the energy levels of the DTMF tones as determined by the Goertzel function. The values in the array are arranged such that the first four elements are the levels of the row tones and the second four are the column tones. There are two maximum value search program loops, one for the row frequencies and one for the column frequencies. Each search loop starts by initializing either the *row* variable or the *col* variable to zero. If the level in the element of the *tones* array is greater than the threshold value and the value currently stored in the *row* or *col* variable the current value in the *row* or *col* variable is replaced thus remembering the largest value. When a new maximum value is found, two other variables, *r* and *c* record the *tones* array element number as a 1 in the bit position associated with the element number. For example, if elements 2 and 6 of the *tones* array are the maximum of the two sets of four tones, then the value of *r* is set to 2^2 or 4 and the value of c is set to 2^6 or 64. When the two variables r and c are logically ORed together the result is 68 or using hexadecimal notation, 0x22. The key values are then decoded in a switch-case table as the key for the number 5. If no valid combination of row and column tones is identified, the default key code is 0 indicating a no-press period.

Listing 2. Key decoding function.

```
/* key_code FUNCTION DESCRIPTION *************************************
SYNTAX:         char key_code(int *tones);
KEYWORDS:       DTMF codes, key, decode
DESCRIPTION:    The magnitude of the tone signals is contained in the array of
                8 integers. The tones are grouped into two groups of 4
                elements. The number of the elements with the largest value
                for the two groups indicate the row and column keys.
RETURN VALUE:   char value containing the ASCII representation of the key. If
                no valid row and column frequency above the TT_LIMIT threshold
                a value of 0 is returned.
Notes:          None
END DESCRIPTION **********************************************************/

char key_code(int *tones)
{
int i, r = 0, c = 0;
int row = TT_LIMIT, col = TT_LIMIT; /* Minimum tone levels to be valid */
char key = 0;
    for(i=0; i<4; i++)  /* Search for element containing the largest value */
    {
        if(tones[i] > row)  /* If largest and greater than threshold */
       {                     /* save the element number - range 0 - 3 */
            row = tones[i];
            r = 1 << i;
       }
    }
    for(i=4; i<8; i++)  /* Search for element containing the largest value */
```

```
{
   if(tones[i] > col)  /* If largest and greater than threshold */
   {                   /* save the element number - range 4 - 7 */
       col = tones[i];
       c = 1<<i;
   }
}
switch (r|c)    /* Decode the row-column combination */
{
   case 0x11:
       key = '1';
       break;
   case 0x12:
       key = '4';
       break;
    case 0x14:
       key = '7';
       break;
   case 0x18:
       key = '*';
       break;
   case 0x21:
       key = '2';
       break;
    case 0x22:
       key = '5';
       break;
   case 0x24:
       key = '8';
       break;
   case 0x28:
       key = '0';
       break;
   case 0x41:
       key = '3';
       break;
   case 0x42:
       key = '6';
       break;
    case 0x44:
       key = '9';
       break;
   case 0x48:
       key = '#';
       break;
   case 0x81:
       key = 'A';
       break;
   case 0x82:
       key = 'B';
       break;
   case 0x84:
       key = 'C';
       break;
   case 0x88:
       key = 'D';
       break;
   default:
       key = 0;
}
```

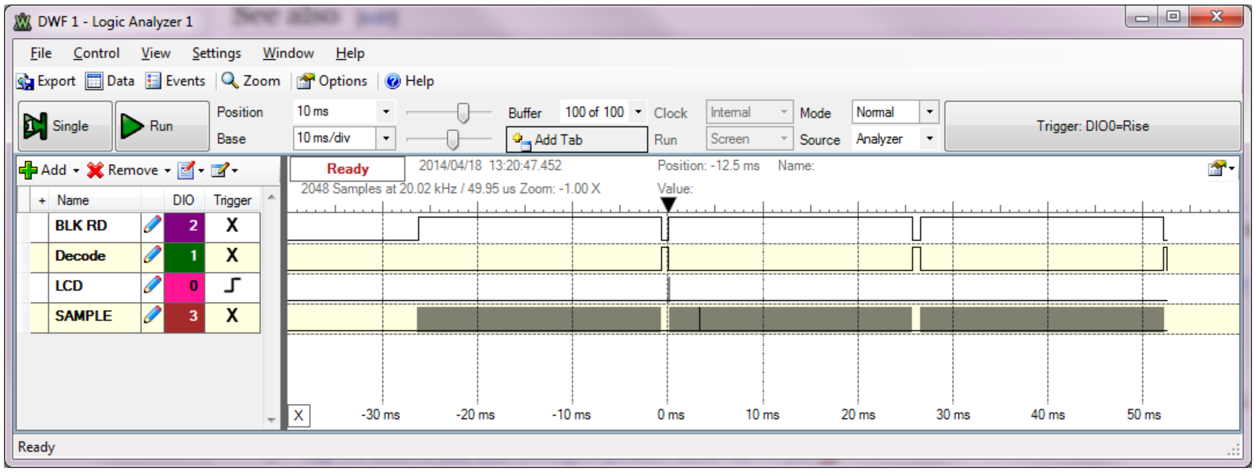
```
    return key;
} /* End of key_code */
```

Code Performance

Table II provides a performance measure for this implementation of the Goertzel algorithm. This data was determined from measurements illustrated in Figure 7 and Figure 8. 25.65 ms is required to collect a 205 sample window of the input signal. The Goertzel algorithm and the LCD display update require 0.806 ms which means that sampling is halted for approximately 6 samples. Since the minimum tone burst is 50ms, the maximum processing window size is 400 samples. Using a sample window of 205, there is always at least one window will be entirely within one tone burst or entirely within one inter-tone gap. Worst case energy level detection would occur then two buffers of data straddle either a tone burst or an inter-tone gap.

Table II. Task code execution timing

Task	Execution Time
Sample Block	25.65 ms
Decode Tones	800 µs
Update Display	6.15 µs



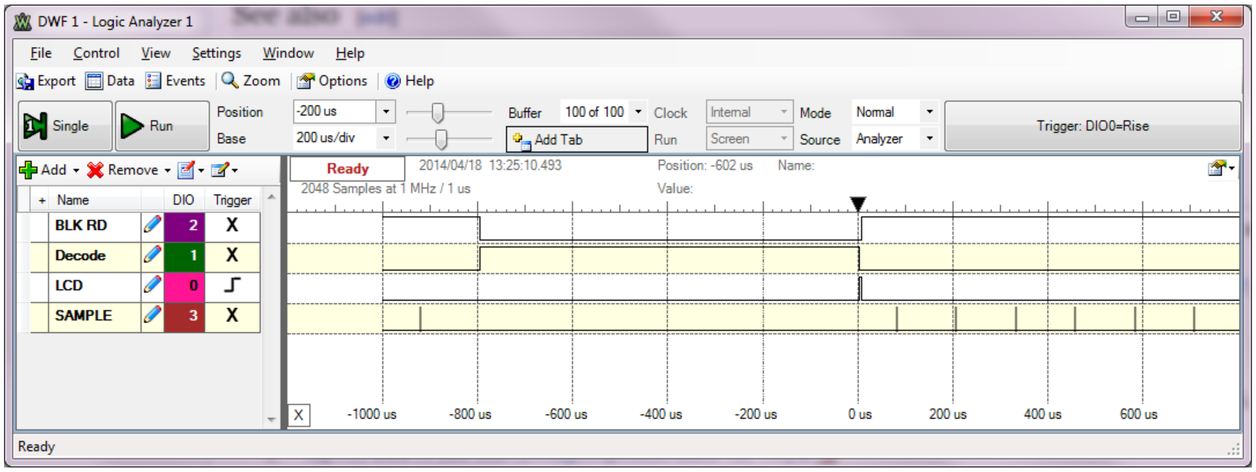
Position 10 ms
Base 10 ms/div
Buffer 100 of 100
Clock Internal
Run Screen
Mode Normal
Source Analyzer
Trigger: DIO0=Rise

Ready 2014/04/18 13:20:47.452
2048 Samples at 20.02 kHz / 49.95 us Zoom: -1.00 X
Position: -12.5 ms Name:
Value:

Name	DIO	Trigger
BLK RD	2	X
Decode	1	X
LCD	0	⎍
SAMPLE	3	X

Ready

Figure 7. Timing of the DTMF decoder program.



Position: -200 us
Base: 200 us/div
Buffer: 100 of 100
Clock: Internal
Run: Screen
Mode: Normal
Source: Analyzer
Trigger: DIO0=Rise

Ready
2014/04/18 13:25:10.493
2048 Samples at 1 MHz / 1 us
Position: -602 us
Name:
Value:

Name	DIO	Trigger
BLK RD	2	X
Decode	1	X
LCD	0	⎍
SAMPLE	3	X

Ready

Figure 8. Expanded view of the time required for decoding and LCD update.

Final Remarks

Since the sampling is interrupt driven and represents a foreground task, it is possible to modify the C code to continue sampling the data while computing the tone decoding algorithm and displaying the results in the background. Background tasks are the functions that the processor executes when there no higher priority operations waiting for execution. Operating the DTMF decoder system in this manner requires dual data buffers. One data buffers would be getting loaded with the sampled data while processing the data out of the other. As in the batch mode method first presented, it is necessary that the time required for tone decoding be less than the time required for filling a buffer with data.

[i] http://en.wikipedia.org/wiki/Rotary_dial
[ii] http://en.wikipedia.org/wiki/Dual-tone_multi-frequency_signaling#.23.2C_.2A.2C_A.2C_B.2C_C.2C_and_D