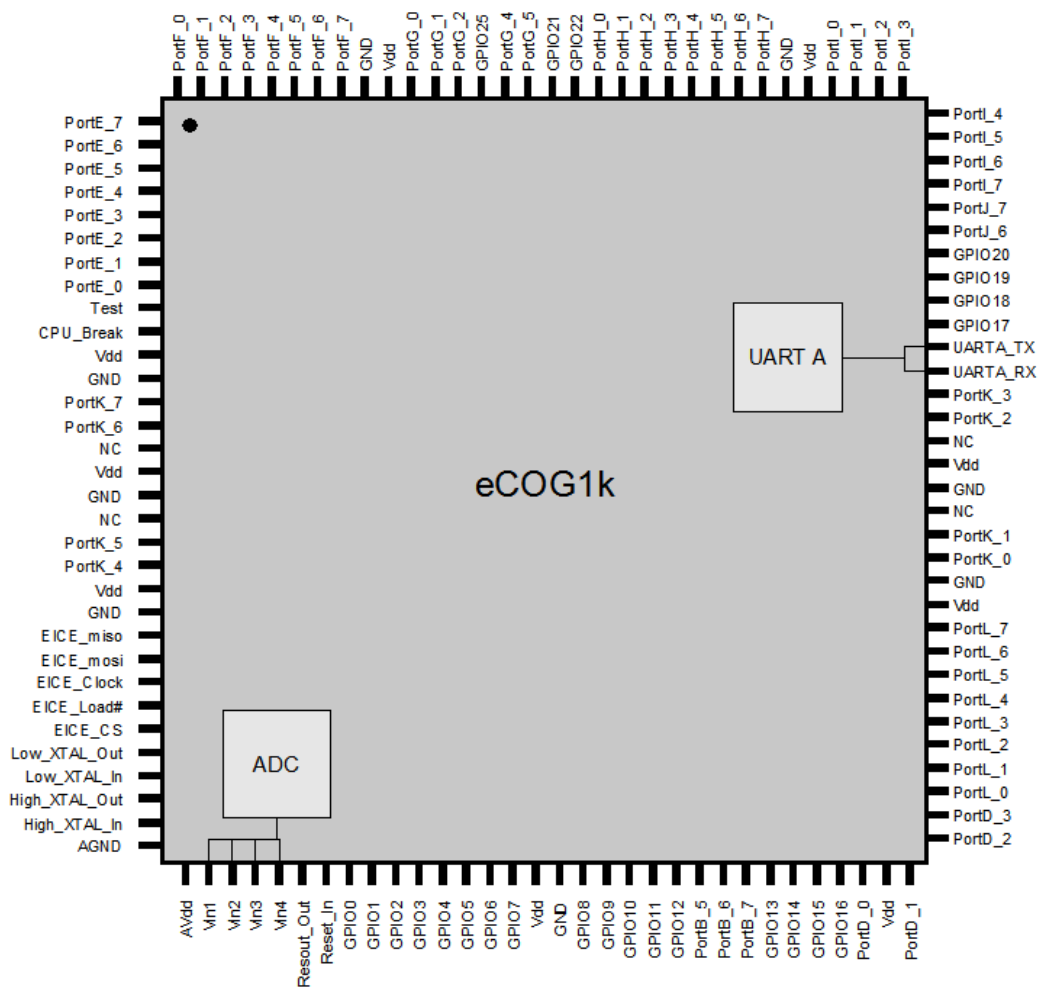


AN045 – Software DTMF Decoder

Version 1.0

This application note describes a method for decoding DTMF signals in software on the eCOG1k microcontroller.



Confidential and Proprietary Information

©Cyan Technology Ltd, 2008

This document contains confidential and proprietary information of Cyan Technology Ltd and is protected by copyright laws. Its receipt or possession does not convey any rights to reproduce, manufacture, use or sell anything based on information contained within this document.

Cyan Technology™, the Cyan Technology logo and Max-eICE™ are trademarks of Cyan Holdings Ltd. CyanIDE® and eCOG® are registered trademarks of Cyan Holdings Ltd. Cyan Technology Ltd recognises other brand and product names as trademarks or registered trademarks of their respective holders.

Any product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Cyan Technology Ltd in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Cyan Technology Ltd shall not be liable for any loss or damage arising from the use of any information in this guide, any error or omission in such information, or any incorrect use of the product.

This product is not designed or intended to be used for on-line control of aircraft, aircraft navigation or communications systems or in air traffic control applications or in the design, construction, operation or maintenance of any nuclear facility, or for any medical use related to either life support equipment or any other life-critical application. Cyan Technology Ltd specifically disclaims any express or implied warranty of fitness for any or all of such uses. Ask your sales representative for details.



Revision History

Version	Date	Notes
V1.0	17/10/2006	First release.

Contents

1	Introduction	6
2	Glossary	6
3	Overview.....	6
4	The Goertzel Algorithm.....	7
4.1	Sample Rate.....	7
4.2	Block Size.....	7
4.3	Precomputed Constants	7
4.4	Sample Processing.....	8
4.5	Block Processing	8
4.6	DTMF Constants	8
4.7	Detection Limits	8
5	Hardware	9
5.1	Circuit Diagram.....	9
5.2	eCOG1k Peripheral Requirements.....	9
5.2.1	ADC	9
5.2.2	Serial Interface	9
5.2.3	Timer	9
5.2.4	GPIO.....	10
6	Analogue to Digital Converter.....	11
6.1	Configuration	11
6.2	Input Constraints	11
6.3	ADC Calculation	11
7	Program Flow	12
7.1	Flow Chart	12
7.1.1	Main Function	12
7.1.2	Calibration Function.....	13
7.1.3	DTMF Decode Function	14
7.1.4	ADC End Of Conversion ISR.....	15
8	Calibration	17
8.1	Essential Parameters	17
8.1.1	Minimum Signal Strength (DIFF_IN).....	17
8.1.2	Idle Level (MIN)	17
8.1.3	Target Frequency Coefficient (Filters[])	17
8.1.4	Detection Limit (Detection_Limits).....	17
8.2	Example.....	18

8.3	Fine Tuning.....	20
9	Testing the DTMF Decoder	21
9.1	Run the Example Program	21
9.2	Test Results.....	21
Appendix A	Program Files.....	22
Appendix B	Global Variables and Constants.....	23
B.1	Global Variables	23
B.2	Constants	24
Appendix C	Function Details	25
C.1	main.c.....	25
	<i>main()</i> 25	
	<i>timer_init()</i>	25
	<i>Filter_Calibrate()</i>	25
	<i>find_min()</i>	25
	<i>find_max()</i>	26
	<i>find_thre()</i>	26
	<i>DTMF_decode()</i>	26
	<i>timer_handler()</i>	26
	<i>uart_putstr()</i>	27
	<i>putchar()</i>	27
C.2	adc.c.....	28
	<i>adc_start()</i>	28
	<i>adc_stop()</i>	28
	<i>adc_read()</i>	28
	<i>adc_handler()</i>	28
C.3	filter.c.....	29
	<i>Goertzel_Algorithm()</i>	29
	<i>Goertzel_Filter()</i>	29
Appendix D	Testing	30
Appendix E	Further Improvements.....	31
Appendix F	References.....	32

1 Introduction

This application note describes a method for decoding DTMF signals in software on the eCOG1k microcontroller. The example uses the Goertzel algorithm which provides an efficient method of measuring the levels of known frequencies in a set of input data samples.

2 Glossary

A table of abbreviations used in this document.

ADC	Analogue to Digital Converter
DTMF	Dual Tone Multi Frequency
eCOG1k	Cyan Technology target 16-bit microcontroller
GPIO	General Purpose I/O
ISR	Interrupt Service Routine
UART	Universal Asynchronous Receiver/Transmitter

3 Overview

Dual-tone multiple-frequency (DTMF) signalling is used in telephony applications for sending information, usually key presses, over the telephone line to the exchange or call switching centre. The information must be encoded into signals in the voice-frequency band, as higher frequencies cannot be transmitted through a traditional telephone system.

A typical telephone keypad has 12 to 16 keys, arranged in a 4x3 or 4x4 array. DTMF encoding assigns two frequencies to each key; one frequency indicates the row and the other frequency indicates the column within the keypad matrix. In this way, a 4x4 keypad requires eight frequencies to encode all 16 keys.

In DTMF encoding, the eight different frequencies include four high band frequencies for the columns and four low band frequencies for the rows in the keypad. When a key is pressed, the low band frequency for the row and the high band frequency for the column are transmitted simultaneously. The selected high and low tones are superimposed. Each tone signal must be present for at least 40ms on the communication channel, and the gaps or pauses between tones must also be at least 40ms.

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

Figure 1. DTMF telephone keypad frequencies

The DTMF decoding process must identify the high and low band frequencies present in the input signal to find the key pressed.

4 The Goertzel Algorithm

The Fast Fourier Transform (FFT) is often used to detect different frequency levels in a signal or set of data values, but it requires significant processing overhead. The Goertzel algorithm looks at specific, predetermined frequency values rather than the entire spectrum, and in such cases it is much more efficient than the FFT algorithm. It is ideally suited to DTMF decoding, where the application needs to find the signal levels at only eight specific frequencies.

Some processing is performed on every input sample. When a block of samples has been received, then a final calculation gives the signal power at the target frequency.

A number of parameters must be selected or calculated for the application.

- Select the sample rate.
- Select the block size N .
- Precompute one sine and one cosine term.
- Precompute one coefficient.

These can all be calculated once before running the main algorithm, and either stored as constants or hard-coded into the application.

4.1 Sample Rate

The sample rate may be determined by the hardware or the application. For example, a sample rate of 8kHz is often used in telecommunications applications. In this example, the eCOG1k ADC sample rate is fixed at 7.8125kHz.

4.2 Block Size

The Goertzel algorithm block size N is similar to the number of points in an equivalent FFT. It determines the frequency resolution or bin width. If the sample rate is F Hz and the block size is N samples, then the bin width is equal to F / N .

This suggests that a large block size is best, in order to get the highest possible frequency resolution. However, a large block size also means that it takes a longer time to detect each target frequency, because the final calculation can be performed only when the complete block has been captured.

The relationship between the target frequencies and the sample rate should be considered. Ideally, each target frequency should fall in the centre of its frequency bin. This means that the target frequencies should all be integer multiples of F / N . This may not be possible in a practical implementation.

Unlike the FFT, the Goertzel algorithm does not require that the block size N is a power of 2.

4.3 Precomputed Constants

It is simple to calculate the constant values required during processing, once the sample rate and block size are selected.

$$k = \left(\frac{N \times f}{F} \right)$$

$$coeff_k = 2 \cos \left(\frac{2 \times \pi \times k}{N} \right)$$

where N is the block size, F is the sample rate, and f is the target frequency.

4.4 Sample Processing

For each sample value $x[n]$, perform the following calculations.

$$y_0 = (coeff_k \times y_1[n]) - y_2[n] + x[n]$$

$$y_1 = y_0[n-1]$$

$$y_2 = y_1[n-1]$$

The values y_1 and y_2 must be initialised to zero at the start of each block of samples.

4.5 Block Processing

After N samples have been processed as above, the magnitude of the signal at the target frequency can be calculated.

$$power = magnitude^2 = y_1^2[N] + y_2^2[N] - y_1[N] \times y_2[N] \times coeff_k$$

4.6 DTMF Constants

The value of the coefficient $coeff_k$ is constant for each target frequency. The following table lists the coefficient values for the DTMF target frequencies.

DTMF target frequency f (Hz)	Sample rate F (Hz)	N	$coeff_k$	$coeff_k \ll 5$
697	7812.5	250	1.693914	54
770	7812.5	250	1.628603	52
852	7812.5	250	1.548561	49
941	7812.5	250	1.454078	46
1209	7812.5	250	1.126743	36
1336	7812.5	250	0.952388	30
1477	7812.5	250	0.747266	24
1633	7812.5	250	0.509250	16

Table 1. DTMF target frequency coefficients

The right hand column lists the coefficient values multiplied by 2^5 (shifted left by 5 bits). These values are used in the example software to allow the use of integer arithmetic instead of floating point without losing precision.

4.7 Detection Limits

For each target frequency, a detection limit must be set. For each target frequency, a simple threshold test of the calculated signal magnitude against its detection limit indicates whether or not that frequency is present in the input signal.

Details on setting the detection limits are given later in section 6, Calibration.

5 Hardware

5.1 Circuit Diagram

The diagram below shows a simplified circuit diagram for the analogue input connection to the eCOG1k microcontroller.

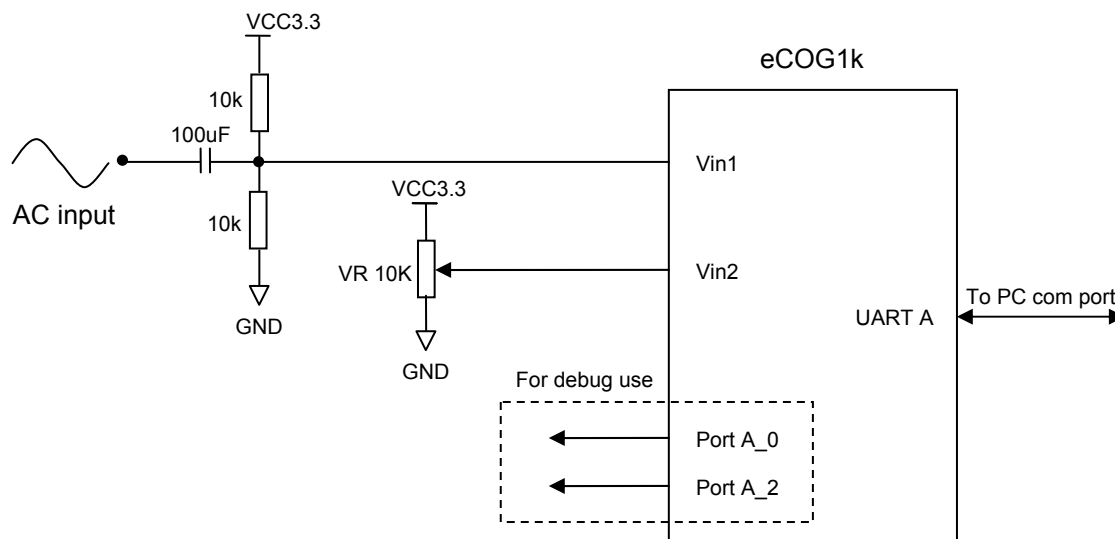


Figure 2. Analogue input circuit diagram

5.2 eCOG1k Peripheral Requirements

5.2.1 ADC

Vin1 and **Vin2** are the differential inputs of the eCOG1k ADC input. In this application program, **Vin2** is set to a voltage level of 1.65V. The AC signal input is offset to 1.65V DC. The 10uF capacitor couples the AC signals to **Vin1** and blocks any DC signals that might flow back to the AC input.

5.2.2 Serial Interface

The UART A peripheral provides a serial port, used to implement a simple user interface. It is configured for 8 data bits, 1 stop bit, no parity, no handshake and 9600baud.

5.2.3 Timer

One of the hardware timers is used to check that there is at least a 40ms gap between two successive DTMF signals.

5.2.4 GPIO

GPIO lines on portA_0 and portA_2 are used for test and debug output signals.

PortA_0 is set high when the ADC detects a valid signal level from the DTMF tone input. It stays high while the processor accumulates the required number of samples to complete a block of data.

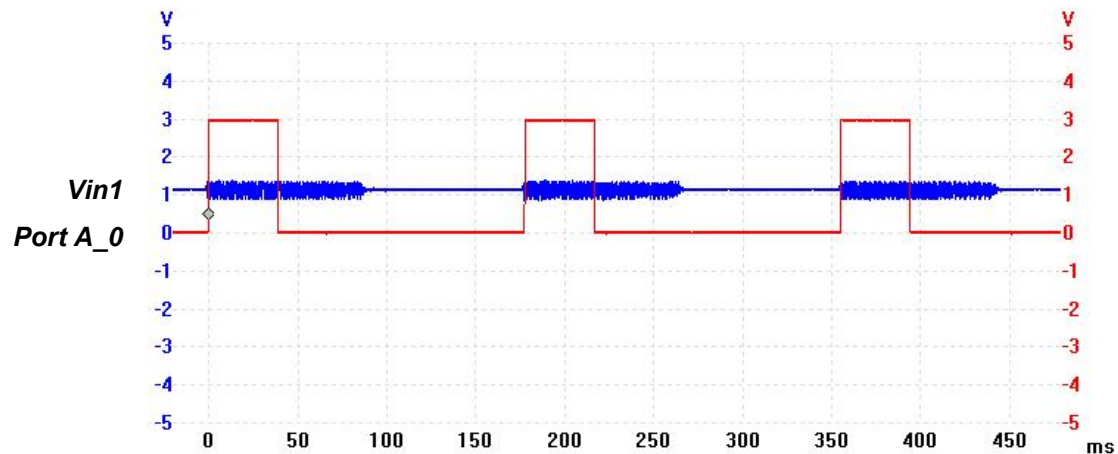


Figure 3. PortA_0 output

PortA_2 is set high when the ADC detects silence (no signal) from the DTMF tone input. This indicates the position of the gaps or pauses between two successive tones.

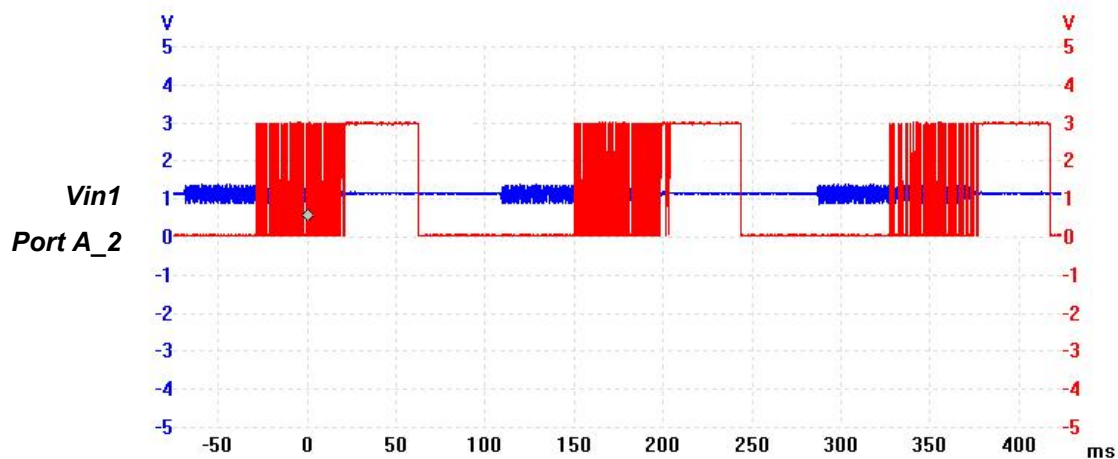


Figure 4. PortA_2 output

6 Analogue to Digital Converter

6.1 Configuration

The eCOG1k ADC is a 12-bit sigma delta design that operates at a fixed sample rate. Refer to the eCOG1k User Manual for more information. The ADC sampling frequency is controlled by the selected ADC clock source. The following table shows the available ADC sample rates.

ADC clock source	Frequency	Sample rate
Low PLL	4.9152 MHz	7.680 kHz
High reference oscillator	5.0 MHz	7.8125 kHz

Table 2. ADC clock frequencies and sample rates

In this application program, the target ADC sample rate is 7.8125 kHz.

The ADC can be operated in differential mode to measure the difference between two external analogue signals. Inputs **Vin1** and **Vin2** are used in this example application.

6.2 Input Constraints

There are some constraints for the ADC differential inputs **Vin1** and **Vin2**.

- $0 \leq V_{in1} \leq V_{cc}$
- $0 \leq V_{in2} \leq V_{cc}$
- $-1.3V \leq V_{in1} - V_{in2} \leq 1.3V$

6.3 ADC Calculation

In differential modes, the difference between the two input voltages is calculated from the ADC value according to the following equation:

$$V_{in1} - V_{in2} = \frac{ADC_OUT \times V_{REF}}{2048}$$

V_{REF} is the ADC internal reference voltage and determines the ADC scaling; typically it is 1.25V.

7 Program Flow

7.1 Flow Chart

7.1.1 Main Function

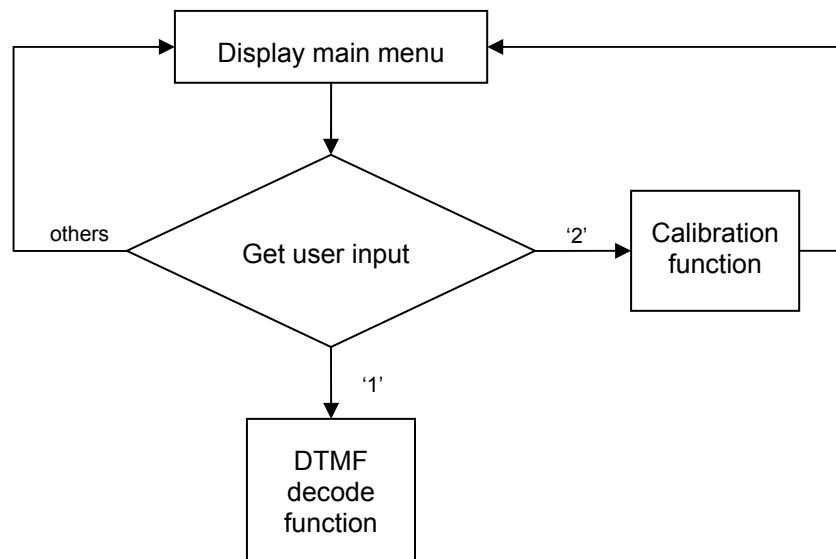


Figure 5. Flow chart: main() function

7.1.2 Calibration Function

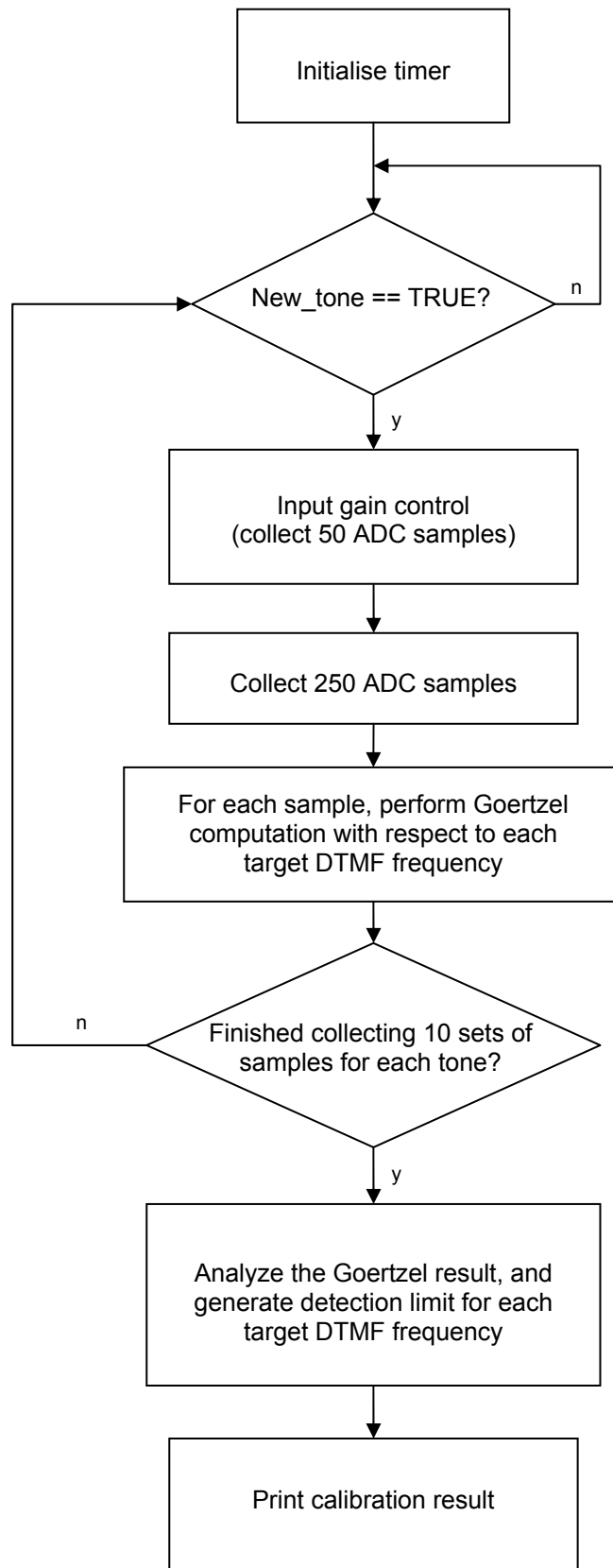


Figure 6. Flow chart: calibration function

7.1.3 DTMF Decode Function

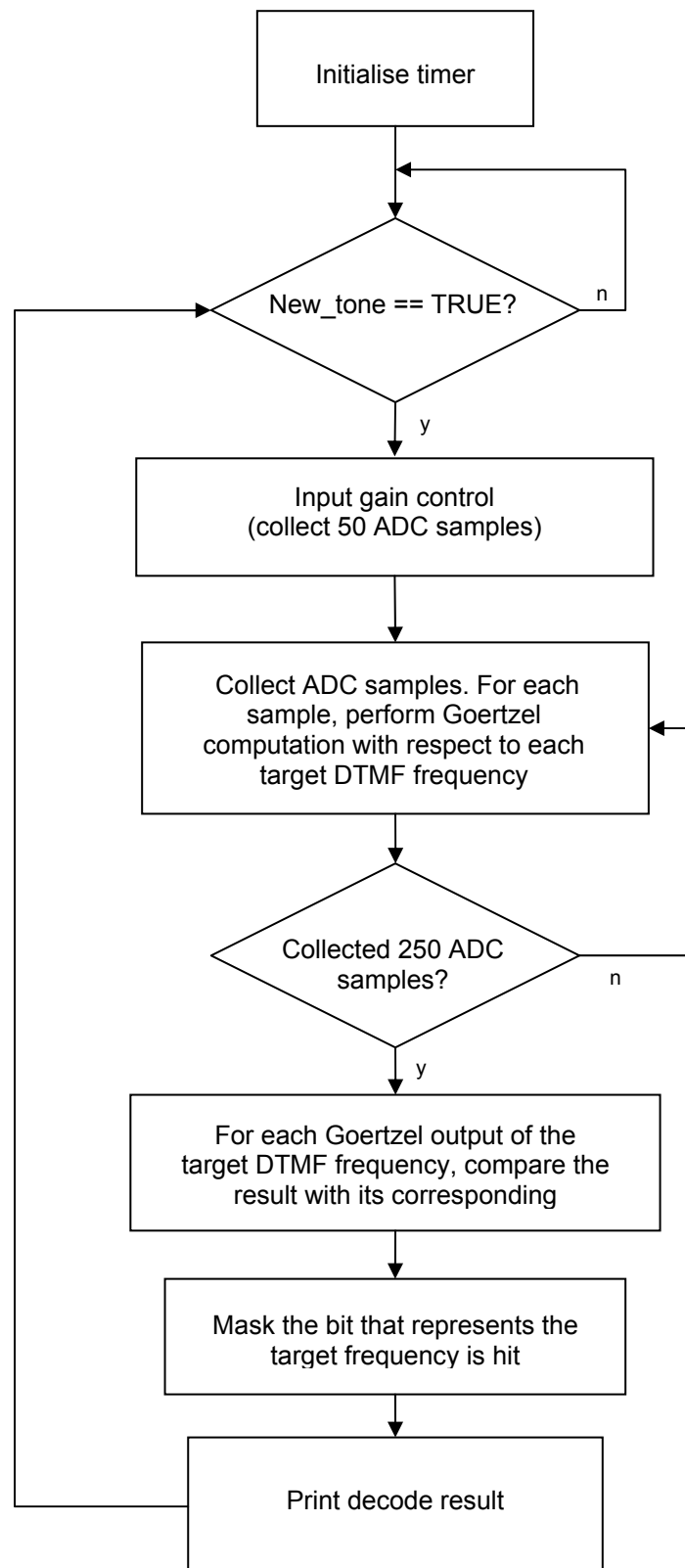
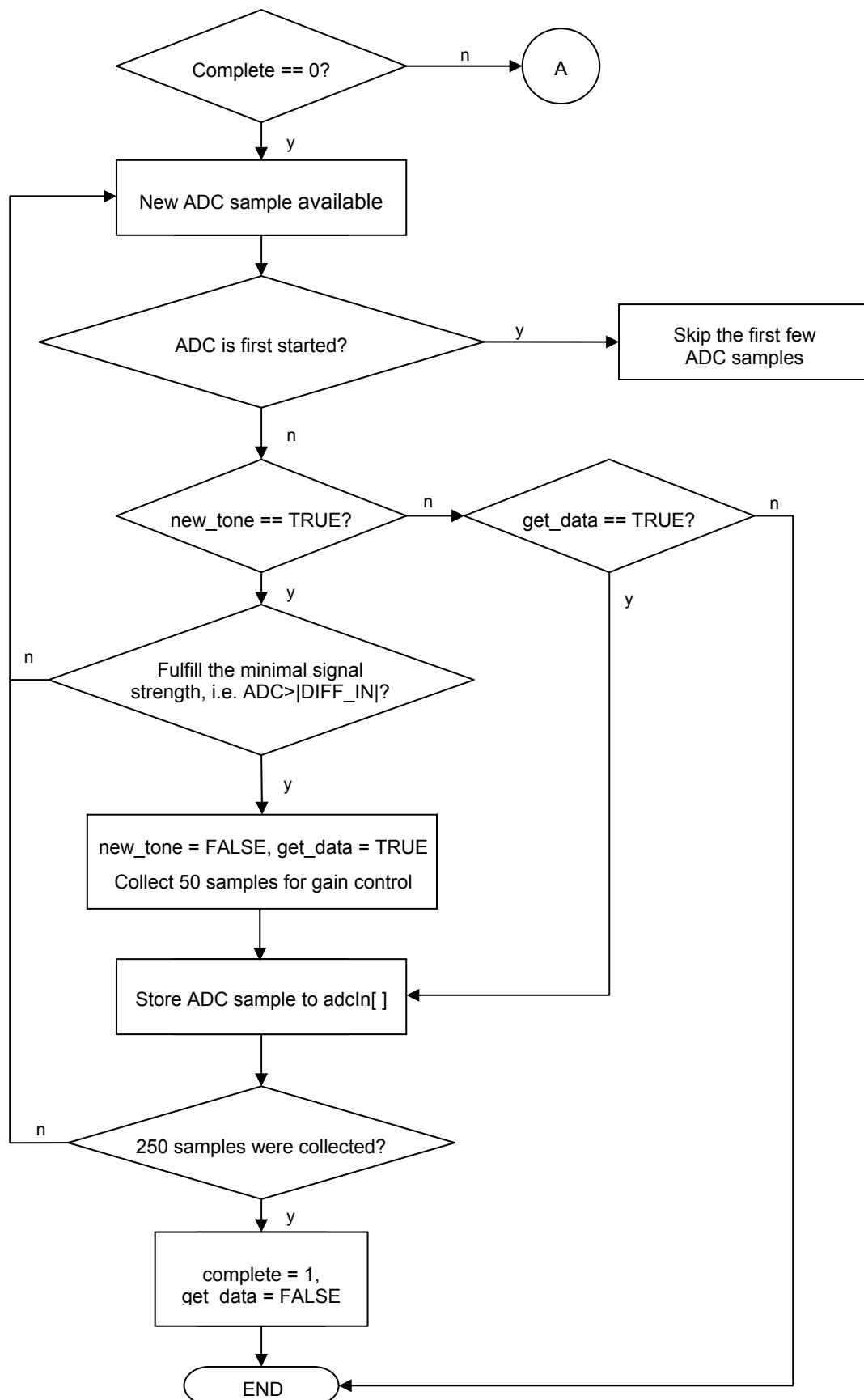


Figure 7. Flow chart: DTMF decode function

7.1.4 ADC End Of Conversion ISR

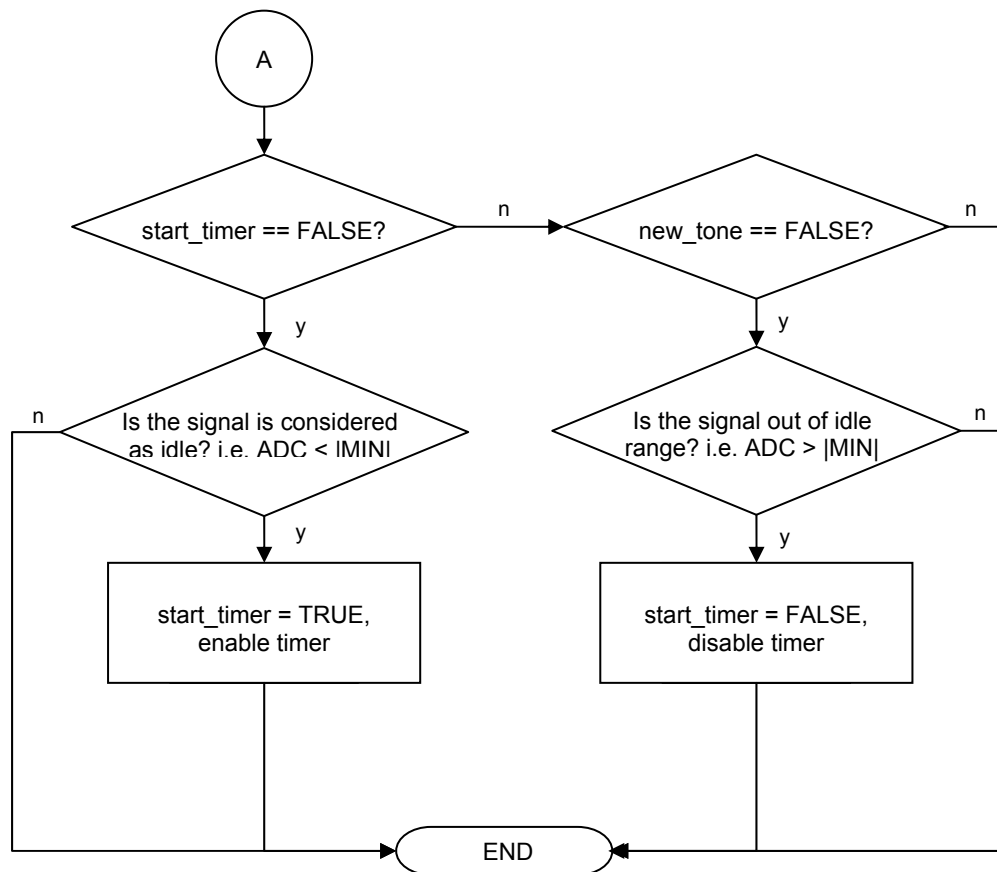


Figure 8. Flow chart: ADC end of conversion ISR

8 Calibration

8.1 Essential Parameters

8.1.1 Minimum Signal Strength (*DIFF_IN*)

An ADC sample that has a positive value higher than **DIFF_IN** or a negative value lower than **-DIFF_IN** is considered a valid tone sample. Once a valid tone sample is detected in the ADC ISR, this implies that a new tone has started. The ADC ISR continues to store the next 249 ADC samples such that a total of 250 samples are stored into the array **adcln[]** for the Goertzel calculation.

```
#define DIFF_IN          492                // about 300 mV
```

8.1.2 Idle Level (*MIN*)

An ADC sample with a value in the range of \pm **MIN** is said to be at an idle level. For DTMF tone detection, a 40ms tone gap must persist between two successive tones. Every ADC sample within the 40ms tone gap must fall in the idle level.

```
#define MIN              377                // about 230 mV
```

8.1.3 Target Frequency Coefficient (*Filters[]*)

In section 4.3, the coefficient for each target frequency is calculated and recorded in Table 1. When a valid tone sample is available, these coefficients are used in the Goertzel algorithm to determine the signal power for each of the DTMF frequencies.

```
const int Filters[NUM_FILTERS] =
{
    54, // F = 697 Hz
    52, // F = 770 Hz
    49, // F = 852 Hz
    46, // F = 941 Hz
    36, // F = 1209 Hz
    30, // F = 1336 Hz
    24, // F = 1477 Hz
    16  // F = 1633 Hz
};
```

8.1.4 Detection Limit (*Detection_Limits*)

The detection limit for each target DTMF frequency is a threshold value for that frequency. If the calculated signal power of a target frequency is above the threshold value, then that frequency component is present in the tone signal.

```
const long Detection_Limits[NUM_FILTERS] =
{
    50000, // F = 697 Hz
    10000, // F = 770 Hz
    10000, // F = 852 Hz
    10000, // F = 941 Hz
    10000, // F = 1209 Hz
    10000, // F = 1336 Hz
    3000,  // F = 1477 Hz
    200000 // F = 1633 Hz
};
```

8.2 Example

When the example program starts, a main menu is displayed to the user through the serial port. For the serial port settings, please refer to section 5.2.2.

```
DTMF Decoder Menu
1.      Start DTMF decoder
2.      Calibrate the Goertzel filters
```

Enter '2' to activate the calibration process.

```
Calibration start...
Please input 10 consecutive signals for tone 1, press <CR> when ready
```

At this point the calibration process requests the user to feed in 10 successive tone signals for tone '1'. The file *wav/tone1.wav* contains a sample of the tone signal for tone '1'. Press <enter> when the tone signals are ready. Play the file *tone1.wav* ten times to let the calibration process collect enough samples. To ensure that the process has detected the tone samples, observe the debug signal on **Port A_0**. This output is set high when the ADC detects a valid signal level on the DTMF tone input. If **Port A_0** does not go high, check the input signal strength or adjust the value of the constant **DIFF_IN**.

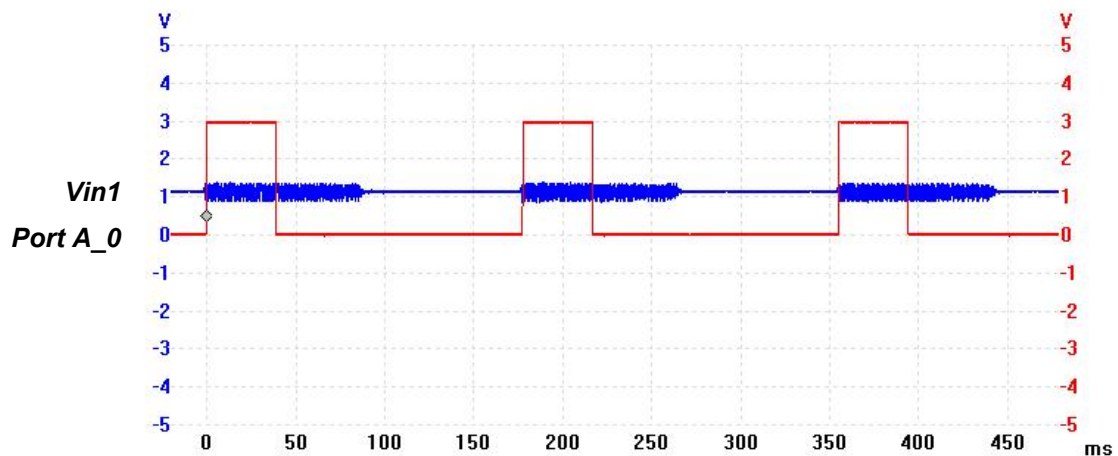


Figure 9. Port A_0 goes high showing a valid signal level is detected

After the calibration process has finished gathering tone samples for tone '1', it requests tone samples for tone '2'. Again, press <enter> and play the file *tone2.wav* ten times.

```
Please input 10 consecutive signals for tone 2, press <CR> when ready
```

The calibration process requests the tone signals in the sequence '1', '2', '3', 'A', '4', '5', '6', 'B', '7', '8', '9', 'C', '*', '0', '#', 'D'. Ten sets of signals are collected for each tone. The input signal for each tone must be played 10 times.

The set of sample tone signals (found in the **wav** sub-directory) was recorded from a popular telephone. The 'A', 'B', 'C' and 'D' keys are rarely used on telephone networks and are not available on most telephones. During the calibration process for tone signals 'A', 'B', 'C' and 'D' any other tones may be used.

Once all the samples are collected, the calibration process outputs a results table. It is noted that due to the lack of tones for the 'A', 'B', 'C' and 'D' inputs, the detection limit for the target frequency 1633Hz has not been calibrated properly. The last column of the result table, which represents the calibration result for 1633Hz, and rows 'A', 'B', 'C' and 'D' may be ignored.

The following table shows the calibration results achieved from a test on the eCOG1k development kit:

Calibration result								
Tone	F1(697)	F2(770)	F3(852)	F4(941)	F5(1209)	F6(1336)	F7(1477)	F8(1633)
1	25008	6831	1656	117	10283	377	108	89
2	8854	7219	1987	225	248	20712	38	70
3	17121	3973	1997	355	3	358	3569	74
A	21821	4327	1949	352	2	378	7068	21
4	4326	20602	2976	162	23368	384	465	1083
5	3143	28692	2928	564	45	17012	20	49
6	2662	14722	4312	305	12	264	2990	103
B	2646	7019	4345	284	8	261	6386	59
7	3740	1309	29979	232	30856	530	358	214
8	3477	935	28140	286	60	24775	7	57
9	3010	813	22557	214	60	90	5659	114
C	3193	845	12954	226	61	86	11131	55
*	3901	563	634	35592	38850	356	154	158
0	5648	908	1822	46457	174	25727	94	67
#	4597	953	2290	68258	24	51	5918	86
D	4656	662	1771	81164	16	59	7945	52
Limit:	8854	7019	12954	35592	10283	17012	2990	21

Table 3. Calibration results

To analyse the result table, refer back to the keypad frequencies as shown in Figure 1. Each column entry represents the signal power of each tone with respect to the target DTMF frequency.

For column **F1(697)**, tones '1', '2' and '3' should contain the frequency component 697Hz. The signal power in this frequency for these keys should be higher than for the other keys.

For column **F2(770)**, tones '4', '5' and '6' should contain the frequency component 770Hz. The signal power in this frequency for these keys should be higher than for the other keys.

For column **F3(852)**, tones '7', '8' and '9' should contain the frequency component 852Hz. The signal power in this frequency for these keys should be higher than for the other keys.

For column **F4(941)**, tone '*', '0' and '#' should contain the frequency component 941Hz. The signal power in this frequency for these keys should be higher than for the other keys.

For column **F5(1209)**, tones '1', '4', '7' and '*' should contain the frequency component 1209Hz. The signal power in this frequency for these keys should be higher than for the other keys.

For column **F6(1336)**, tones '2', '5', '8' and '0' should contain the frequency component 1336Hz. The signal power in this frequency for these keys should be higher than for the other keys.

For column **F7(1477)**, tones '3', '6', '9' and '#' should contain the frequency component 1477Hz. The signal power in this frequency for these keys should be higher than for the other keys.

The last row shows the preliminary detection limits for each DTMF frequency. They are the measured minimum signal power thresholds for the target frequency components respectively.

Edit the values of the array **Detection_Limits[]** in the application program according to this results table. Once these values are entered, the DTMF decoder is ready.

8.3 Fine Tuning

The detection limits generated from the calibration process are unlikely to be ideal, the calculation of the limits is very simple. To achieve a higher accuracy and more robust DTMF decode, it will be necessary to fine tune the detection limits.

Try to run DTMF decode process as shown in Section 9.1. Open the sound file *wav/all.wav*. The sequence of tones in this file is '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '*' and '#'. Play the file several times and examine the decode result. The DTMF decode result gives some clues on adjusting the detection limits. For example:

```
Detected 1
Detected 2
No signal detected, mask = 40, total count = 578, error = 3
Detected 4
Detected 5
```

In the above example, the DTMF decoder failed to recognize tone '3'. The recorded mask value for this tone detection is 0x40. The mask value contains one bit for each target frequency. A valid key detection should have two bits set in the mask value, corresponding to the two tones for that key. The upper 4 bits represent the high band frequencies, and the lower 4 bits represent the low band frequencies.

Target frequency (Hz)	1633	1477	1336	1209	941	852	770	697
Bit	7	6	5	4	3	2	1	0

Table 4. Target frequency mask bits

Refer to **Code_Masks[]** in the application program; tone '3' should have the mask value of 0x41.

```
const int Code_Masks[NUM_CODES][2] =
{
    { 0x11, '1' },
    { 0x21, '2' },
    { 0x41, '3' },
    { 0x81, 'A' },
    { 0x12, '4' },
    { 0x22, '5' },
    { 0x42, '6' },
    { 0x82, 'B' },
    { 0x14, '7' },
    { 0x24, '8' },
    { 0x44, '9' },
    { 0x84, 'C' },
    { 0x18, '*' },
    { 0x28, '0' },
    { 0x48, '#' },
    { 0x88, 'D' },
};
```

A recorded value of 0x40 indicates that the high band frequency 1477Hz was detected successfully, but the low band frequency 697Hz was not detected. This may indicate that the detection limit for 697Hz is too high. Reduce the value of this detection limit and repeat the decode process again to see if there is an improvement.

Note that there is a minimum usable value for each detection limit, determined by the maximum signal power in all the other (invalid) tones when the target (valid) tone is detected. For example, when adjusting the detection limit for the target frequency 697Hz, the valid tones are '1', '2', '3' and the invalid tones are '4', '5', '6', '7', '8', '9', '0', '*' and '#'. From Table 3, the minimum detection limit for 697Hz is 5648, as this is the largest detected power value for any of the invalid tone input signals. Setting a detection limit for this target frequency that is smaller than 5648 may result in false tone detection when the '0' tone is received.

One possible strategy is to set the detection limits half way between the lowest detection value for the valid tones and the highest detection value for the invalid tones. For the results shown this value is 7251 for the 697Hz target frequency.

9 Testing the DTMF Decoder

9.1 Run the Example Program

On program start, a main menu is displayed to the user through the serial port.

```
DTMF Decoder Menu
1.      Start DTMF decoder
2.      Calibrate the Goertzel filters
```

Enter '1' to activate the DTMF decode process.

```
Start decoding DTMF tone...
```

The DTMF decoder is ready to decode DTMF tones. Open the file *wav/all.wav*, and play it into the analogue input. The expected output from the decoder is as follows:

```
Detected 1
Detected 2
Detected 3
Detected 4
Detected 5
Detected 6
Detected 7
Detected 8
Detected 9
Detected 0
Detected *
Detected #
```

Sometimes the decoder function may fail to decode the incoming tone, and generates an error output message:

```
Detected 5
No signal detected, mask = 40, total count = 1259, error = 4
Detected 7
```

The value in **mask** gives a clue on how to adjust the detection limit, as described in section 8.3. The value **total count** is the number of tone samples processed. The value **error** is a count of how many times no valid signal was detected. These values are useful in analysing the performance of the DTMF decoder.

9.2 Test Results

In the normal telephone network, the signal input sometimes varies. It is recommended to have a dynamic gain control to normalize the input signals, so that the full dynamic range of the ADC is used ($-1.3V \leq V_{in1} - V_{in2} \leq 1.3V$). At the start of every tone detection, the gain is calculated. The dynamic gain control is implemented by finding the maximum and minimum values of the first 50 samples, then scaling the rest of the samples such that the differential peak value is $\pm 1.3V$. Floating point calculation should be avoided to speed up the gain scaling process. In the application software, input samples are scaled so as to allow the DTMF signal level in the range of $\pm 460mV$ to $\pm 1100mV$ to be decoded, and the abided noise level is $\pm 230mV$.

The software implementation of the Goertzel algorithm was tested using the sound file *wav/all.wav*. For 30000 consecutive input tones with an amplitude range of $\pm 460mV$ to $\pm 1100mV$, centred at 1.65V dc, it was found that all tones were decoded successfully and no false hits were detected. The percentage error is close to 0%.

Appendix A Program Files

Filename	Description
<i>Assembly source files</i>	
cstartup.asm	Start up program of eCOG1k
irq.asm	Contains the interrupt vectors and the initial branch to the main code.
<i>C source files</i>	
adc.c	Contains ADC initialization and ADC End-of-Conversion ISR
filter.c	Performs the Goertzel algorithm
main.c	Main entry of the DTMF decoder
<i>Header files</i>	
adc.h	Defines constants for ADC
<i>Sound files</i>	
all.wav	Wav signal of keys '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '*', '#'
tone1.wav	Wav signal of key '1'
tone2.wav	Wav signal of key '2'
tone3.wav	Wav signal of key '3'
tone4.wav	Wav signal of key '4'
tone5.wav	Wav signal of key '5'
tone6.wav	Wav signal of key '6'
tone7.wav	Wav signal of key '7'
tone8.wav	Wav signal of key '8'
tone9.wav	Wav signal of key '9'
tone0.wav	Wav signal of key '0'
toneStar.wav	Wav signal of key '*'
toneHash.wav	Wav signal of key '#'
<i>Calibration results</i>	
Calibration_Result.txt	This a sample dump file of the DTMF calibration results

Table 5. Program files

Appendix B Global Variables and Constants

This section describes the functions of global variables and constants used in the application program.

B.1 Global Variables

Global variable name	Description
adcln[]	ADC input buffer with size of BLOCK_SIZE.
adc_skip	Counts the number of skipped ADC samples when ADC is started.
adc_count	Counts the number of samples collected and store into adcln[].
adc_errs	Counts the number of ADC error samples.
complete	A flag indicates adcln[] is filled with enough ADC samples.
get_data	A flag indicates a valid signal level is detected for a tone signal.
limit	An array stores the detection limit of each DTMF frequency after the calibration process.
new_tone	A flag ensures a valid tone gap between two successive tones.
start_timer	A flag indicates the status of a timer.
threshold	An array stores the Goertzel computation result.
y0	Intermediate result of Goertzel computation.
y1	Intermediate result of Goertzel computation.
y2	Intermediate result of Goertzel computation.
cal_gain	Flag to indicate that ADC gain control calculation should be performed
high	Maximum ADC signal value.
low	Minimum ADC signal value.
gain_cnt	Counts the number of ADC samples for the gain calculation.
gain	ADC input gain.

Table 6. Global variables

B.2 Constants

Constant name	Description
FACTOR_OFFSET	The bit shift applied to the coefficients to allow the use of integer arithmetic in the Goertzel computation.
NUM_FILTERS	Defines the number of coefficients for the Goertzel calculation.
NUM_CODES	The number of DTMF tones.
Filters[]	The precalculated coefficients for each of the DTMF frequencies.
Detection_Limits[]	Define the detection limits for each of the DTMF frequencies. The detection limit is compared with the Goertzel result to find the major frequency components of an input tone.
Code_Masks[]	Defines the DTMF tones and the corresponding mask bits.
FALSE	Equivalent to a value of 0.
TRUE	Equivalent to a value of 1.
DIFF_IN	The minimum signal strength for a real DTMF tone.
MIN	Defines the maximum signal level for a gap between tones.
BLOCK_SIZE	Defines the number of ADC samples collected for Goertzel algorithm.
NUM_DISCARD	Defines the number of ADC samples to be discarded when the ADC is first started.
DEBUG	Enable debug output signals on portA_0 and portA_2.

Table 7. Constants

Appendix C Function Details

C.1 main.c

main()

NAME: main program

SYNOPSIS: int main(int argc, char* argv[])

DESCRIPTION: This is the main entry of the application program. It shows a user menu.

RETURNS: 0

timer_init()

NAME: timer initialise function

SYNOPSIS: void timer_init(void)

DESCRIPTION: Initialise timer for the 40ms pause between two consecutive DTMF tones.

RETURN: None

Filter_Calibrate()

NAME: DTMF decoder calibration function

SYNOPSIS: void Filter_Calibrate(void)

DESCRIPTION: Ask the user to input DTMF tones to calibrate the detection limits. For each tone, 10 sets of samples are collected for Goertzel calculation. After all the DTMF tone samples are collected, this function generates a result table and the suggested detection limits for the Goertzel filters.

RETURN: None

find_min()

NAME: Find minimum function

SYNOPSIS: long find_min(long arr[10][8], int mask)

PARAMETERS: arr – An array stores the Goertzel results for each frequency.
mask – A bit mask representing the major frequency components of each tone.

DESCRIPTION: Find the minimum value among the 10 calibration inputs.

RETURN: The minimum value

find_max()

NAME: Find maximum function

SYNOPSIS: long find_max(long arr[10][8], int mask)

PARAMETERS: arr – An array stores the Goertzel results for each frequency.
mask – A bit mask representing the major frequency components of each tone.

DESCRIPTION: Find the maximum value among the 10 calibration inputs

RETURN: The maximum value

find_thre()

NAME: Find minimum threshold function

SYNOPSIS: long find_thre(long thre1, long thre2, long thre3, long thre4)

PARAMETERS: thre1 - The threshold of tone 1
thre2 - The threshold of tone 2
thre3 - The threshold of tone 3
thre4 - The threshold of tone 4

DESCRIPTION: Find the minimum value among the 4 thresholds

RETURN: The minimum threshold value

DTMF_decode()

NAME: DTMF decode function

SYNOPSIS: void DTMF_decode (void)

DESCRIPTION: Detect any incoming signals. Decode the valid DTMF tone with the Goertzel algorithm and show the output to the user.

RETURN: None

timer_handler()

NAME: Timer ISR

SYNOPSIS: void __irq_entry timer_handler (void)

DESCRIPTION: Disable timer interrupt and set new_tone flag to indicate 40ms idle time is over. The DTMF decoder is ready to decode the next tone.

RETURN: None

uart_putstr()

NAME: Serial port string output function
SYNOPSIS: void uart_putstr(const char *out)
PARAMETERS: out - output character string
DESCRIPTION: Sends the character to the DUART Channel A
RETURN: None

putchar()

NAME: Serial port character output function
SYNOPSIS: int putchar(int c)
DESCRIPTION: Sends the character to the DUART Channel A
RETURN: Transmitted character

C.2 adc.c

adc_start()

NAME: ADC start function
SYNOPSIS: void adc_start(unsigned int config)
PARAMETERS: config - ADC configuration content
DESCRIPTION: Initialise and start ADC according to the configuration content
RETURN: None

adc_stop()

NAME: ADC stop function
SYNOPSIS: void adc_stop(void)
DESCRIPTION: Disable ADC interrupt
RETURN: None

adc_read()

NAME: ADC function
SYNOPSIS: void adc_read(unsigned int config, const char *name)
PARAMETERS: config - ADC configuration content
 name - description name for ADC input
DESCRIPTION: Initialise and start ADC. Collect ADC samples until complete == 1
RETURN: None

adc_handler()

NAME: ADC ISR
SYNOPSIS: void __irq_entry adc_handler (void)
DESCRIPTION: This is the ADC end-of-conversion interrupt service routine. All the valid ADC samples are stored in adcIn[], and an index variable adc_count is incremented accordingly. When an ADC sample is available, first the signal level is checked. If the signal level is large enough (input > |DIFF_IN|), it is considered as a valid signal sample, and the new_tone flag is cleared so that the following samples are considered as the same tone. Once enough samples have been taken, then a complete flag is set. It then checks that a 40ms pause is present between 2 tones. The input signal for the tone gap must be small enough (input < |min|) for 40 ms. Once the 40ms pause is complete, the new_tone flag is set, and the system is ready to detect the next tone.
RETURN: None

C.3 filter.c

Goertzel_Algorithm()

NAME: Goertzel algorithm

SYNOPSIS: long Goertzel_Algorithm(int factor, int size)

PARAMETERS: factor - filter coefficient
size - size of ADC samples

DESCRIPTION: Implement the Goertzel algorithm.

RETURN: The power level of the incoming signal with respect to the target frequency

Goertzel_Filter()

NAME: Goertzel filter

SYNOPSIS: void Goertzel_Filter(int factor, int i, int pos)

PARAMETERS: factor - filter coefficient
i - target frequency in Filters[]
pos - the current position of ADC sample to be proceeded

DESCRIPTION: Implement the Goertzel algorithm on sample-by-sample basis.

RETURN: None

Appendix D Testing

This example DTMF decoder was tested with input signals from a normal telephone. The following diagram shows the circuit used to connect the telephone to the eCOG1k development board for the test.

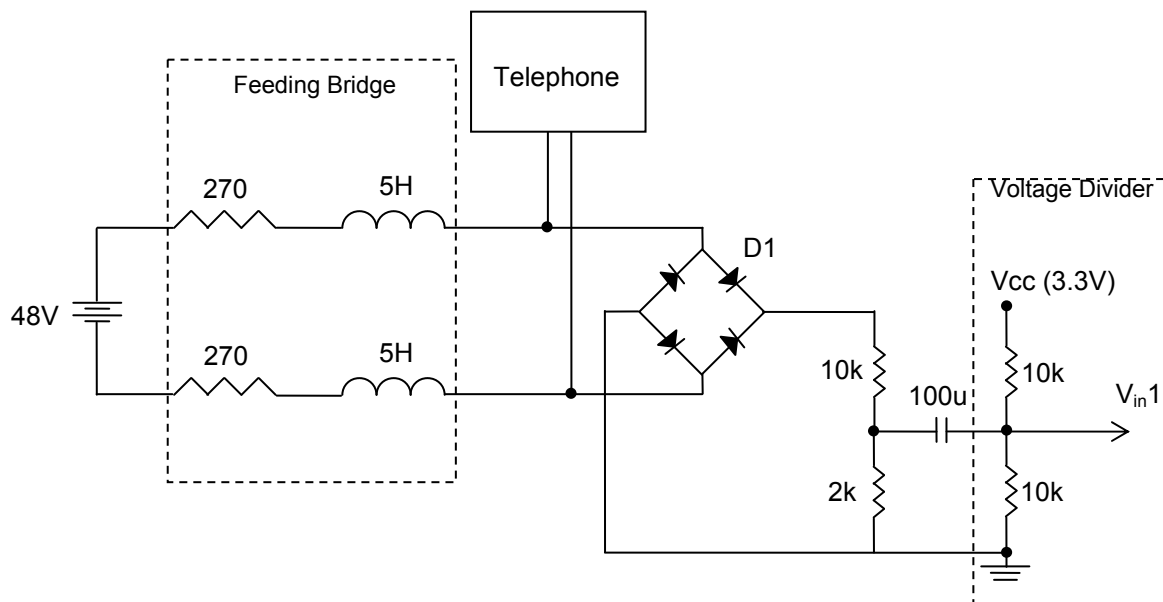


Figure 10. Test circuit with telephone

This circuit provides power to the telephone and allows analogue tone signals to be captured from it. The telephone is a.c. coupled to a voltage divider at the analogue input. The divider offsets the input signal to a centre voltage of approximately 1.65V, in the middle of the ADC input signal range. It also attenuates the telephone signal.

Tests were performed with more than 1200 consecutive input tones at amplitude of $\pm 200\text{mV}$. The results showed that all tones were successfully decoded, and no false hits were detected. The execution time for the DTMF decode algorithm was 39.7ms, for capturing and processing a block size of 300 samples (including 50 samples for gain control and 250 samples for DTMF decode), and performing the Goertzel filter computation for all eight target frequencies.

Appendix E Further Improvements

This application note presents a basic implementation of a DTMF decoder, as an example or proof of principle to demonstrate (a) the Goertzel filter technique and (b) that this can be executed in real-time on the eCOG1k device. Further improvements are possible, if desired.

- In the example, the essential parameters shown in section 8.1 are defined in the header files. For increased flexibility, these parameters may be stored in an external EEPROM or in the information block of the eCOG1k internal flash and may be changed at run time without recompiling the application. The information block is used to hold user data such as application configuration data, product serial numbers and version numbers. For further details about the flash information block, please refer to the eCOG1k User Manual and to application note AN032.
- To improve the performance of the DTMF decoder, one suggestion is that each input frequency should be detected twice by the Goertzel algorithm before reporting it as a valid detected frequency. To implement this, the input signal buffer can be divided into two halves, with each half of the buffer storing 125 samples. Keeping the same total number of samples ($125 \times 2 = 250$) means that the double detection process takes no longer than the single detection process. At the beginning of an input signal, the first buffer is filled with 125 ADC samples. While the first buffer is being processed, the next 125 samples are stored in the second buffer. When the first Goertzel calculation is complete and the second buffer is full, the second Goertzel calculation is executed. This reduces the non-detection rate and false tone rate, and improves the robustness and accuracy of the DTMF decoder.

Appendix F References

- [1] The Goertzel Algorithm
Kevin Banks, Embedded Systems Design, 28 August 2002
<http://www.embedded.com/showArticle.jhtml?articleID=9900722>
- [2] Digital Decoding Simplified Sequential Exact-Frequency Goertzel Algorithm
Eric Kiser, Circuit Cellar Issue 182, September 2005
- [3] Goertzel Algorithm
Wikipedia
http://en.wikipedia.org/wiki/Goertzel_algorithm