Goertzel Implementation

Zi-hao Liu
Electrical and Electronic Engineering
University of Bristol
Bristol, United Kingdom
fx19583@bristol.ac.uk

Kuan-Lin Lee
Electrical and Electronic Engineering
University of Bristol
Bristol, United Kingdom
vm19584@bristol.ac.uk

Zhi-ying Lin
Electrical and Electronic Engineering
University of Bristol
Bristol, United Kingdom
pt19585@bristol.ac.uk

***Abstract*—This paper is concerned with the implementation of the Goertzel algorithm via Code Composer Studio 5 (CCS5) in order to detect tones for the Dual Tone Multi-Frequency application (DTMF) signaling system.**

Keywords—Goertzel, algorithm, CCS5, DTMF, signal, system.

I. INTRODUCTION

The application of the DTMF signaling system is mainly used for push-button digital telephone sets. Nowadays, its utility has been extended to electronic mail and even telephone banking systems which involve sending DTMF signals after user selects option.

The DTMF signaling system consists of two frequency band - lower and upper, each containing four fixed frequency signals. A combination of two frequency (one from each band) is used to represent a specific digit (0 to 9), character (A, B, C or D) or symbol (* or #), as shown in Table I.

TABLE I. DTMF SIGNAL TABLE

Dual Tone Multi-Frequency Table		Upper band frequency			
		1209Hz	1336Hz	1477Hz	1633Hz
Lower band frequency	697Hz	1	2	3	A
	770Hz	4	5	6	B
	852Hz	7	8	9	C
	941Hz	*	0	#	D

In the system, there are two types of signal processing, coding and decoding. They can be interpreted as the generation and detection of the DTMF signals.

For coding, or generation of signal, two sinusoidal sequences (representing the combination of two frequency) with finite length will be added together to form a single sample representing a digital, character or symbol. E.g., pressing button 7 results in the tone for 852Hz and 1209Hz being generated simultaneously.

On the other hand, the Goertzel algorithm will be used as a filter for the detection of the signal at the receiver end. The algorithm is derived from the Discrete Fourier Transform (DFT), it utilizes the periodicity of the phase factor $e^{\frac{-2\pi jk}{N}}$ in order to reduce the computational complexity. The Goertzel algorithm is more numerically and computationally efficient for situations where a small number of selected frequency components is required to compute (only 16 samples of the DFT are required for the 16 combination of tones). This makes the algorithm well suited for embedded applications such as the DTMF signaling system.

II. UNDERSTANDING THE GOERTZEL ALGORITHM

The major principle of implementation of Goertzel algorithm in the application of DTMF system is dividing the incoming signal into *N* number of samples, and each sample will be used, together with the delayed value, to calculate a $Q_n$value.

$$Q_n = x(n) + 2\cos\left(\frac{2\pi k}{N}\right) Q_{n-1} - Q_{n-2} \quad (1)$$

As shown in (1), $x(n)$ is the n^{th} sample, Q_{n-1} and Q_{n-2} are the delayed value up to two cycles before, and $2\cos\left(\frac{2\pi k}{N}\right)$ is a coefficient calculated using *N* and constant *k*, calculation for *k* is described below in (2).

$$k = N \times \frac{f_{tone}}{f_s} \quad (2)$$

The value of constant *k* determines the tone we are trying to detect, where f_{tone} is the frequency of the tone, f_s is the sampling frequency and *N* is set to a value of 205 in order to maintain the accuracy of the result. By using (1) and (2) we're then able to calculate the value of constant *k* and the $2\cos\left(\frac{2\pi k}{N}\right)$ coefficient for each frequency. Relevant data is recorded into a table shown in Table II.

TABLE II. FREQUENCY-CONSTANT K-COEFFICIENT TABLE

Frequency	k	Coefficient (decimal)	Coefficient (Q15)
697	18	1.703275	0x6D02
770	20	1.635585	0x68AD
852	22	1.562297	0x63FC
941	24	1.482867	0x5EE7
1209	31	1.163138	0x4A70
1336	34	1.008835	0x4090
1477	38	0.790074	0x3290
1633	42	0.559454	0x23CE

Therefore, if we want to detect whether the incoming signal is 770Hz frequency, we use k=20 and coefficient=0x68B1 as the values in the equation.

Fixed point arithmetic is used when performing all calculations, therefore, due to the fact that arithmetic in fixed point can only represent numbers from 0 to less than 1, the decimal values of the coefficient needs to be divided by two to be represented in Q15 format (e.g. 0x6D02 in Q15 represent a value of $\frac{1.703275}{2}$ in decimal.), on the other hand, the calculation for $2\cos\left(\frac{2\pi k}{N}\right) Q_{n-1}$ will therefore be performed twice in order to make up for halving the coefficient.

After the calculation for Q_{205} is completed, we can then use Q_{204} and (3) to calculated the square of magnitude of the Goertzel value. If the incoming signal matches the frequency we are looking for, the magnitude of the output would be a larger number, if not, the output would be very small or a value close to zero.

$$|y_k(N)|^2 = Q^2(N) + Q^2(N-1) - 2\cos\left(\frac{2\pi k}{N}\right) Q(N)Q(N-1) \quad (3)$$

Fig. 1 is a block diagram that visualizes the entire calculation process of the Goertzel algorithm. The yellow box indicates the feedback loop that will be repeated for 205 times, after the iteration completes the algorithm enters the blue box to obtain the output value.

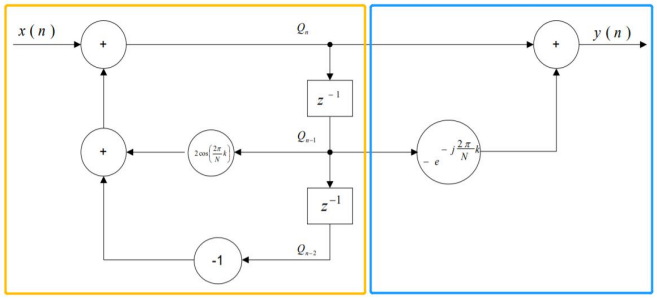

Fig. 1. *Block diagram of the Goertzel algorithm.*

By examining Fig. 1 we can see that in theory there is actually a phase factor, $-e^{-j\frac{2\pi}{N}k}$, present in the equation. Due to the fact that we are only interested in detecting the presence of a tone rather than its phase, therefore the phase can be neglected, and we can directly calculate the square of the output to obtain its magnitude.

III. Implementation of the Goertzel Algorithm

In this section we'll explore how to achieve the functionality of the Goertzel algorithm by using CCS5, as well as making an attempt to achieve two tasks, the first task is about detecting one single frequency and output its Goertzel value calculated, where the second tasks involves detecting all eight frequencies, and determine what is the digit, character or symbol that the user inputs in the console.

A. Task 1 - Detecting one single frequency

The aim of task 1 is to obtain a constant and stable output of Goertzel value when detecting one single frequency, if any slight variation of Goertzel value are observed at the output, it suggests overflow has occurred and means failure. Therefore the output should be the same every time it appears.

To support the iteration requirement for the feedback loop, clock is used in CCS5 to achieve the target of repeatedly going back and fourth between functions, as shown in Fig. 2.

```
33 void main(void)
34 {
35
36
37     System_printf("\n>>>I am in main.");
38     System_flush();
39     /* Create a Clock Instance */
40     Clock_Params clkParams;
41
42     /* Initialise Clock Instance with time period and timeout (system ticks) */
43     Clock_Params_init(&clkParams);
44     clkParams.period = 1;
45     clkParams.startFlag = TRUE;
46
47     /* Instantiate ISR for tone generation  */
48     Clock_create(clk_SWI_Generate_DTMF, TIMEOUT, &clkParams, NULL);
49
50     /* Instantiate 8 parallel ISRs for each of the eight Goertzel coefficients */
51     Clock_create(clk_SWI_GTZ_0697Hz, TIMEOUT, &clkParams, NULL);
52
53     mag1 = 32768.0; mag2 = 32768.0; freq1 = 697; // I am setting freq1 = 697Hz to test my GTZ algorithm with one frequency.
54
55     /* Start SYS_BIOS */
56     BIOS_start();
57 }
```

Fig. 2. *C code for main function in task 1.*

Line 48 and 51 signifies that the two functions, *clk_SWI_Generate_DTMF* and *clk_SWI_GTZ_0697HZ*, will be iterated, and in line 53, the variable *freq1* indicates the frequency of the incoming signal, in this case it is set to 697. Fig. 3 shows the generation of samples for the DTMF signal.

```
64 void clk_SWI_Generate_DTMF(UArg arg0)
65 {
66     static int tick;
67
68
69     tick = Clock_getTicks();
70
71 //  sample = (int) 32768.0*sin(2.0*PI*freq1*TICK_PERIOD*tick) + 32768.0*sin(2.0*PI*freq2*TICK_PERIOD*tick);
72     sample = (int) 32768.0*sin(2.0*PI*freq1*TICK_PERIOD*tick) + 32768.0*sin(2.0*PI*0*TICK_PERIOD*tick);
73     sample = sample >> 12;
74 }
```

Fig. 3. *C code for the generation of DTMF signal in task 1.*

The value of the sample is calculated based on the equation in line 72. The sine wave after the plus sign is set to 0 since in task 1 we are only detecting one frequency, therefore the second sine wave can be ignored. After sample is obtained, its value needs to be scaled down to prevent overflow, in this case it is shifted towards the right by 12, suggest the new sample is the original sample divided by 2^{12}.

Fig. 4 shows the implementation of (1) and (3) in the form of C code.

```
81 void clk_SWI_GTZ_0697Hz(UArg arg0)
82 {
83     static short delay;
84     static short delay_1 = 0;
85     static short delay_2 = 0;
86     static int N = 0;
87     static int Goertzel_Value = 0;
88     int prod1, prod2, prod3;
89     short input;
90     short coef_1 = 0x6D02;
91
92     input = (short) sample;
93
94     prod1 = (delay_1 * coef_1) >> 14;
95     delay = input + (short)prod1 - delay_2;
96     delay_2 = delay_1;
97     delay_1 = delay;
98     N++;
99
100    if (N == 206)
101    {
102        prod1 = (delay_1 * delay_1);
103        prod2 = (delay_2 * delay_2);
104        prod3 = (delay_1 * coef_1) >> 14;
105        prod3 = prod3 * delay_2;
106        Goertzel_Value = (prod1 + prod2 - prod3) >> 15;
107        Goertzel_Value <<= 8;
108        N = 0;
109        delay_1 = delay_2 = 0;
110    }
111
112        gtz_out[0] = Goertzel_Value;
113 }
```

Fig. 4. *C code of the Goertzel algorithm.*

In Fig. 4, *delay*, *delay_1* and *delay_2* corresponds to Q_n, Q_{n-1} and Q_{n-2} respectively, *coef_1* relates to the coefficient $2\cos\left(\frac{2\pi k}{N}\right)$, in this case, we want to detect frequency of 697Hz, therefore it is set to the coefficient of 697Hz referring to Table II. The value of N is incremented in line 98 every time after the three delay values has been calculated. The loop repeats for 205 times. When N increases to 206, the loop halts and enters the *if* statement to calculate the magnitude of the Goertzel value.

The output to the console panel is controlled by the *task1_dtmfDetect* function as shown in Fig. 5.

```
46 void task1_dtmfDetect(void)
47 {
48     int i, a1=0, a2=0, f1=0, f2=0, dig;
49     static int num = 0;
50
51     System_printf("\n--------------- Goertzel Algorithm (one freq) ---------------");
52     System_printf("\n>>> I am in Task 1 for the first time, please wait...");
53     System_flush();
54
55     while (1) {
56
57         num++;
58         Task_sleep(2000);  /* 0.25 second for 125us tick */
59
60         System_printf("\n>>> -%d- Goertzel value [%dHz]: %d.\n", num, freq1, gtz_out[0]);
61
62         System_printf(">>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...");
63         System_flush();
64
65         a1=0, a2=0, f1=0, f2=0;
66             }
67
68 }
```

Fig. 5. *C code for outputting the Goertzel value.*

As shown in Fig. 5, there is an infinite while loop inside the *task_dtmfDetect* function, and a *Task_sleep(2000)* command at line 58. The function waits for a fixed period of

time to allow the calculation to take place, and retrieves the data from the *gtz_out[0]* array and output it onto the console panel, and continuously carries out this iteration.

Fig. 6 below shows the result on the console screen in the case that sample is shifted right by 12.

```
[TMS320C66x_0]
>>>I am in main.
>>>I am in Task 0.
--------------- Goertzel Algorithm (one freq) ---------------
>>> I am in Task 1 for the first time, please wait...
>>> -1- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -2- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -3- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -4- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -5- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -6- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -7- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -8- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -9- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -10- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -11- Goertzel value [697Hz]: 4864.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -12- Goertzel value [697Hz]: 4864.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -13- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -14- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -15- Goertzel value [697Hz]: 4608.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
```

Fig. 6. *Result output at the console screen in the case of sample being shifter right by 12.*

In Fig. 6, the Goertzel value appears to be stable at 4608, however, at the 11th and 12th attempt, the value changes to 4864. In order to increase the stability of the result, line 73 in Fig. 3 will be changed to *sample=sample>>13*, the effect of changing 12 to 13 scales down the value of sample even further, and the corrected result is shown in Fig. 7.

```
[TMS320C66x_0]
>>>I am in main.
>>>I am in Task 0.
--------------- Goertzel Algorithm (one freq) ---------------
>>> I am in Task 1 for the first time, please wait...
>>> -1- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -2- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -3- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -4- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -5- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -6- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -7- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -8- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -9- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -10- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -11- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -12- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -13- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -14- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -15- Goertzel value [697Hz]: 1024.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
```

Fig. 7. *Result output at the console screen in the case of sample being shifter right by 13.*

In addition, if *freq1* and *coef_1* does not match, it would produce a Goertzel value of 0 (e.g. *freq1*=770 and *coef_1=0x6D02*), illustrated in Fig. 8.

```
[TMS320C66x_0]
>>>I am in main.
>>>I am in Task 0.
--------------- Goertzel Algorithm (one freq) ---------------
>>> I am in Task 1 for the first time, please wait...
>>> -1- Goertzel value [770Hz]: 0.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -2- Goertzel value [770Hz]: 0.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
>>> -3- Goertzel value [770Hz]: 0.
>>> I am leaving Task 1, please wait for a minute or so to get the next Goertzel value...
```

Fig. 8. *Result output at the console screen when freq1=770 and coef_1=0x6D02.*

B. Task 2 - Detecting all 16 digits, characters and symbols

In task 2, first the user inputs the digit, character or symbol within the list (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, *, #), the system generates the signal that contains the two frequency corresponding to the user input, and the algorithm should calculate the Goertzel value in order to determine which digit, character or symbol the user typed in.

Since we are dealing with all eight frequencies rather than one single frequency, more function will be needed for each frequency, therefore some adjustments is made to the main function as shown in Fig. 9.

```
37 void main(void)
38 {
39
40
41       System_printf("\n>>> I am in main.\n");
42       System_flush();
43     /* Create a Clock Instance */
44     Clock_Params clkParams;
45
46     /* Initialise Clock Instance with time period and timeout (system ticks) */
47     Clock_Params_init(&clkParams);
48     clkParams.period = 1;
49     clkParams.startFlag = TRUE;
50
51     /* Instantiate ISR for tone generation  */
52     Clock_create(clk_SWI_Generate_DTMF, TIMEOUT, &clkParams, NULL);
53
54     /* Instantiate 8 parallel ISRs for each of the eight Goertzel coefficients */
55     Clock_create(clk_SWI_GTZ_0697Hz, TIMEOUT, &clkParams, NULL);
56     Clock_create(clk_SWI_GTZ_0770Hz, TIMEOUT, &clkParams, NULL);
57     Clock_create(clk_SWI_GTZ_0852Hz, TIMEOUT, &clkParams, NULL);
58     Clock_create(clk_SWI_GTZ_0941Hz, TIMEOUT, &clkParams, NULL);
59     Clock_create(clk_SWI_GTZ_1209Hz, TIMEOUT, &clkParams, NULL);
60     Clock_create(clk_SWI_GTZ_1336Hz, TIMEOUT, &clkParams, NULL);
61     Clock_create(clk_SWI_GTZ_1477Hz, TIMEOUT, &clkParams, NULL);
62     Clock_create(clk_SWI_GTZ_1633Hz, TIMEOUT, &clkParams, NULL);
63
64
65     /* Start SYS_BIOS */
66     BIOS_start();
67 }
```

Fig. 9. *C code for main function in task 2.*

Moreover, unlike task 1, *freq2* into account when generating the samples. Now the sample is constructed from two sine waves created by *freq1* and *freq2* as shown in Fig. 10, where *freq1* and *freq2* are determined by referring to Table I.

```
74 void clk_SWI_Generate_DTMF(UArg arg0)
75 {
76     static int tick;
77
78
79     tick = Clock_getTicks();
80
81     sample = (int) 32768.0*sin(2.0*PI*freq1*TICK_PERIOD*tick) + 32768.0*sin(2.0*PI*freq2*TICK_PERIOD*tick);
82     sample = sample >> 8;
83 }
```

Fig. 10. *C code for the generation of DTMF signal in task 2.*

After generating the samples, the value of the sample will be inserted into the eight functions to perform Goertzel value calculations against all eight coefficients in parallel. Fig. 11 shows an example of all eight Goertzel value caluclated after inputting '7' into the system. The frequency corresponding to the digit 7 are 852Hz and 1209Hz by referring to Table I. Therefore magnitude of Goertzel value of these two frequency must be larger compared to all other frequencies. In Fig. 11 we can see that the Goertzel value for frequencies apart from 852Hz and 1209Hz are still relatively large, a few of them (770Hz and 941Hz) aren't even close to zero, this is

because when generating the value of samples, we've only shifted right by 8, suggesting that we've only scaled down the magnitude of sample by 2^8, therefore the Goertzel value at the output would be relatively larger. If the sample was shifted towards the right by 13, as we did in task 1, the Goertzel value for non-related frequencies would be smaller indeed.

```
Goertzel value of 697Hz is: 1536
Goertzel value of 770Hz is: 9472
Goertzel value of 852Hz is: 1242368
Goertzel value of 941Hz is: 8704
Goertzel value of 1209Hz is: 1330688
Goertzel value of 1336Hz is: 512
Goertzel value of 1477Hz is: 256
Goertzel value of 1633Hz is: 256
```

Fig. 11. *Goertzel value for all eight frequencies.*

After all eight Goertzel value is obtained, the values will be stored into the array *gtz_out* in the order of the magnitude of the frequency (e.g. 1536 stored in position 0; 9472 stored in position 1; 1242368 stored in position 2...). Then, two *for* loops are implemented to obtain the Goertzel value with the largest magnitude in both bands, shown in Fig. 12.

```
204         System_printf("\n>>> Calculating largest Goertzel value in lower band...");
205         System_flush();
206         for (index=0;index<4;index++){
207             if (gtz_out[index] > lower_max_reg){
208                 lower_max_index=index;
209                 lower_max_reg=gtz_out[index];
210             }
211         }
212
213         System_printf("\n>>> Calculating largest Goertzel value in upper band...");
214         System_flush();
215         for (index=4;index<8;index++){
216             if (gtz_out[index] > upper_max_reg){
217                 upper_max_index=index;
218                 upper_max_reg=gtz_out[index];
219             }
220         }
```

Fig. 12. *C code for filtering the largest Goertzel value in lower and upper band.*

Line 209 and 218 indicates that the largest Goertzel value from the lower and upper band are stored in *lower_max_reg* and *upper_max_reg* respectively, as well as their index. (the index indicates the frequency, e.g., *lower_max_index*=0 indicates 697Hz, 1 indicates 770Hz, 2 indicates 852Hz, etc, and *upper_max_index*=4 indicates 1209Hz, 5 indicates 1336Hz etc.).

A two dimensional array *valid_char* is created which stores all the digits, characters and symbols, shown in Fig. 13.

```
char valid_char[4][4]={{'1','2','3','A'},{'4','5','6','B'},{'7','8','9','C'},{'*','0','#','D'}};
```

Fig. 13. *2-d array storing all digits, characters and symbols.*

Then, the frequency and magnitude of Goertzel value related to the detected digit will be assigned by the code in Fig. 14.

```
226             a1=freq[lower_max_index];
227             a2=freq[upper_max_index];
228             mag1=gtz_out[lower_max_index];
229             mag2=gtz_out[upper_max_index];
230             detected_char=valid_char[lower_max_index][upper_max_index-4];
```

Fig. 14. *Assigning values to variables for output.*

In Fig. 14, *a1* and *a2* represents the lower and upper band frequency of the detected digit, character or symbol, respectively. Whereas *mag1* and *mag2* represents the Goertzel value of *a1* and *a2* respectively. Finally, *detected_char* is obtained by using the value of *lower_max_index*, *upper_max_index* and the array *valid_char*.

Eventually, these information will be output to the console using the following code in Fig. 15.

```
233             System_printf("\n-------------------- Result --------------------");
234             System_printf("\n>>> Detected character is -> %c", detected_char);
235             System_printf("\n>>> Information:");
236             System_printf("\n   - Lower band frequency -> %dHz", a1);
237             System_printf("\n   - Upper band frequency -> %dHz", a2);
238             System_printf("\n   - Lower band Goertzel value -> %d", mag1);
239             System_printf("\n   - Upper band Goertzel value -> %d\n\n\n", mag2);
```

Fig. 15. *C code for printing results.*

It's worth to mention that, if the user inputted something that is not valid (i.e. anything that is not included in the *valid_char* array, including smaller cased letter a, b, c and d), the Goertzel value of all frequencies will be zero, and the system outputs an error message, shown in Fig. 16.

```
191         lower_sum = gtz_out[0]+gtz_out[1]+gtz_out[2]+gtz_out[3];
192         upper_sum = gtz_out[4]+gtz_out[5]+gtz_out[6]+gtz_out[7];
193
194         if (lower_sum==0 || upper_sum==0){
195             a1=0, a2=0, f1=0, f2=0;
196             System_printf("\n\n*** Error: Invalid character detected, please try again.");
197             System_printf("\n>>> Hint: This may be because you:");
198             System_printf("\n   - entered a character that is not included within the list.");
199             System_printf("\n   - entered a lower case 'a', 'b', 'c' or 'd', they should be in upper case form.\n\n\n");
200             System_flush();
201         }
```

Fig. 16. *C code for printing error message.*

Fig.17 and Fig. 18 shows the printed result on the console screen when inputting '5' and 'A' respectively.

```
[TMS320C66x_0]
>>> I am in main.
-------------------- Goertzel Algorithm (all freq) --------------------
>>> List of valid characters -> [0,1,2,3,4,5,6,7,8,9,A,B,C,D,*,#]
>>> Please type in a character and press Enter: 5

>>> Analyzing signal...
>>> Calculating Goertzel value for all frequencies...
>>> Calculating largest Goertzel value in lower band...
>>> Calculating largest Goertzel value in upper band...
>>> Generating result...
>>> Process complete.
-------------------- Result --------------------
>>> Detected character is -> 5
>>> Information:
   - Lower band frequency -> 770Hz
   - Upper band frequency -> 1336Hz
   - Lower band Goertzel value -> 1091328
   - Upper band Goertzel value -> 1177088
```

Fig. 17. *Result on the console when user inputs 5.*

```
-------------------- Goertzel Algorithm (all freq) --------------------
>>> List of valid characters -> [0,1,2,3,4,5,6,7,8,9,A,B,C,D,*,#]
>>> Please type in a character and press Enter: A

>>> Analyzing signal...
>>> Calculating Goertzel value for all frequencies...
>>> Calculating largest Goertzel value in lower band...
>>> Calculating largest Goertzel value in upper band...
>>> Generating result...
>>> Process complete.
-------------------- Result --------------------
>>> Detected character is -> A
>>> Information:
   - Lower band frequency -> 697Hz
   - Upper band frequency -> 1633Hz
   - Lower band Goertzel value -> 1270272
   - Upper band Goertzel value -> 1239296
```

Fig. 17. *Result on the console when user inputs A.*

Fig. 19 shows error message pops up when smaller cased 'a' is inputted.

```
-------------------- Goertzel Algorithm (all freq) --------------------
>>> List of valid characters -> [0,1,2,3,4,5,6,7,8,9,A,B,C,D,*,#]
>>> Please type in a character and press Enter: a

>>> Analyzing signal...
>>> Calculating Goertzel value for all frequencies...

*** Error: Invalid character detected, please try again.
>>> Hint: This may be because you:
   - entered a character that is not included within the list.
   - entered a lower case 'a', 'b', 'c' or 'd', they should be in upper case form.
```

Fig. 18. *Result on the console when user inputs a.*

In addition, the system will pop up error message if more than one digit, character or symbol is inputted. This is

achieved by the code shown in Fig. 20, and the printed result on the console panel is shown in Fig. 21.

```
77      do{
78          System_printf("------------------ Goertzel Algorithm (all freq) ------------------");
79          System_printf("\n>>> List of valid characters -> [0,1,2,3,4,5,6,7,8,9,A,B,C,D,*,#]");
80          System_printf("\n>>> Please type in a character and press Enter: ");
81          System_flush();
82          scanf("%s", user_input);
83
84          len=strlen(user_input);
85
86          if (len==1){
87              System_printf("\n>>> Analyzing signal...");
88              System_printf("\n>>> Calculating Goertzel value for all frequencies...");
89              System_flush();
90              break;
91          }
92          else{
93              System_printf("\n*** Error: Multiple character input detected, please enter only ONE character at a time.");
94              System_printf("\n>>> Hint: In order to detect a series of character, input them with spaces separated in between.")
95              System_printf("\n    e.g. -> 1 0 2 4 A 9 C B.\n\n\n");
96              System_flush();
97          }
98      }while(1);
```

Fig. 20. *C code for error handling multiple inputs.*

```
------------------ Goertzel Algorithm (all freq) ------------------
>>> List of valid characters -> [0,1,2,3,4,5,6,7,8,9,A,B,C,D,*,#]
>>> Please type in a character and press Enter: hello

*** Error: Multiple character input detected, please enter only ONE character at a time.
>>> Hint: In order to detect a series of character, input them with spaces separated in between.
    e.g. -> 1 0 2 4 A 9 C B.
```

Fig. 21. *Result on the console after multiple input is detected.*

Last but not least, the system is able to process a series of digit, characters or symbols if they are separated by spaces during the input. (e.g. inputting '8 F', will result in two consecutive result being outputted). The system will then manage to automatically retrieve the digit, character or symbol after the space and continue its operation until it reaches the last element.