Implementing the Goertzel Algorithm for the TMS320C66x

Christopher White
Faculty of Engineering
University of Bristol
Bristol, United Kingdom
gd19031@bristol.ac.uk

Amy Williams
Faculty of Engineering
University of Bristol
Bristol, United Kingdom
ll19356@bristol.ac.uk

***Abstract*—This report will outline our implementation of the Goertzel algorithm in order to detect multiple frequencies in a dual tone multi-frequency (DTMF) signal, using a TMS320C66x microprocessor and the C programming language. This report will then outline the results from our implementation and explain some possible applications.**

***Index Terms*—Goertzel, algorithm, DTMF, signal processing, microprocessor**

I. Introduction

Dual tone multi-frequency, or DTMF, is a method of signalling which uses pairs of frequencies to represent each digit on a keypad, as shown in Figure 1. These signals are easy and economical to generate, however it can be challenging to decode them once they are received. This requires the frequency for each tone to be found within a short period of time whilst also ignoring the affects of noise. [1]

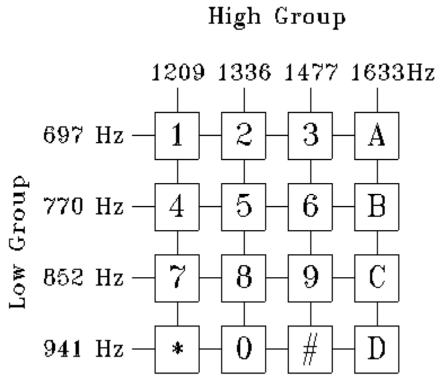


Fig. 1. Image showing the typical number-pad use of DTMF signals. Each pair of frequencies, one from the low and one from the high groups, describes a unique key on the pad.

II. The Goertzel Algorithm

Typical methods for carrying out a frequency analysis of a signal include the Fast Fourier Transform (FFT) or the Discrete Fourier Transform (DFT). However, these transforms are used to find the whole range of frequencies present in a signal which is unnecessary for our application. We are only interested in detecting whether certain frequencies are present.

The Goertzel algorithm is derived from the Discrete Fourier Transform and it is a more efficient method for evaluating individual terms of the DFT, despite being more computationally complex for calculating the whole frequency spectrum. Furthermore, the Goertzel algorithm is ideal for implementation on microprocessors and in embedded systems due to its simple structure.

The algorithm can be represented through the equations

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

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

Equations 1 and 2 can be represented by the block diagram shown in Figure 2.

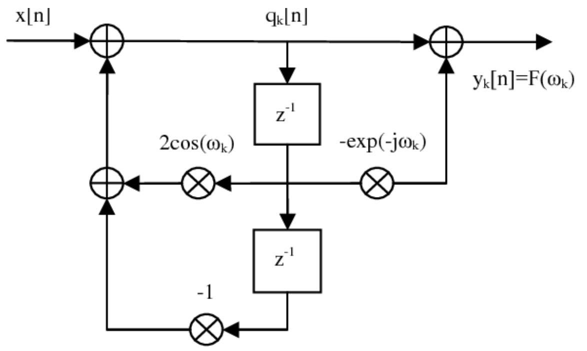


Fig. 2. Block diagram showing the functionality of the Goertzel Alogorithm. The input signal is denoted $x[n]$ and the output is denoted $y_k[n]$.

III. Single Frequency Detection

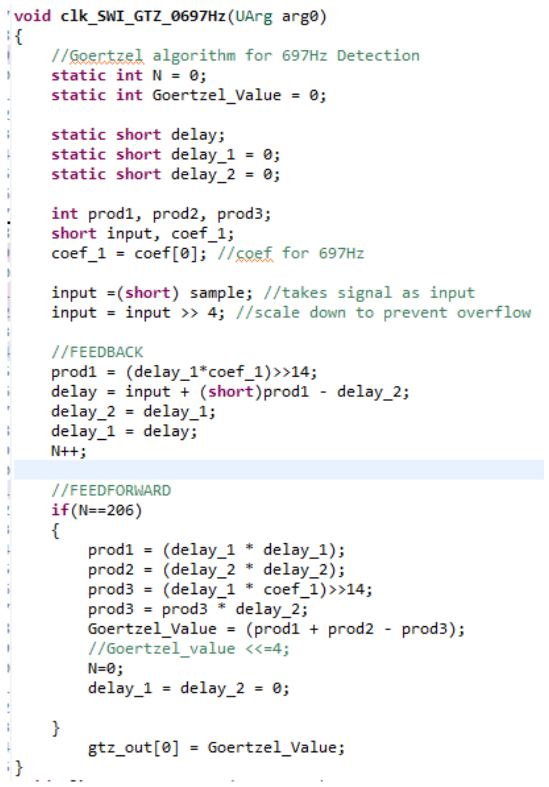
The Goertzel algorithm works by determining the weight of a given frequency which is contained within a signal. The code shown in Figure 3 focuses on a frequency of 697 Hz, so if the input signal contains this frequency a high Goertzel value will be produced. On the other hand, if this frequency is not present the output will be close to 0.

The array ‘coef’ contains coefficients for each of the frequencies shown in Figure 1 which are calculated using

$$2cos\left(\frac{2\pi k}{N}\right), \tag{3}$$

where k is the frequency and N is set to equal 205. Some of the coefficients produced are greater than 1, so the values are halved which allows them to be stored in a Q15 format.

To analyse the signal, samples are taken at fixed intervals which are individually fed into the algorithm. For each sample, $x(n)$, the feedback loop is performed, which uses Equation 1. Then, for every 206^{th} sample the feedforward loop is performed using Equation 2 and the delays generated by the feedback loop, which produces the output $|y_k(N)|$. To determine if the frequency is present within the signal the output in compared with a set value to see if the weighting is large enough.



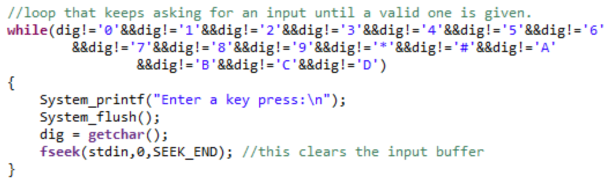
```c
void clk_SWI_GTZ_0697Hz(UArg arg0)
{
    //Goertzel algorithm for 697Hz Detection
    static int N = 0;
    static int Goertzel_Value = 0;

    static short delay;
    static short delay_1 = 0;
    static short delay_2 = 0;

    int prod1, prod2, prod3;
    short input, coef_1;
    coef_1 = coef[0]; //coef for 697Hz

    input =(short) sample; //takes signal as input
    input = input >> 4; //scale down to prevent overflow

    //FEEDBACK
    prod1 = (delay_1*coef_1)>>14;
    delay = input + (short)prod1 - delay_2;
    delay_2 = delay_1;
    delay_1 = delay;
    N++;

    //FEEDFORWARD
    if(N==206)
    {
        prod1 = (delay_1 * delay_1);
        prod2 = (delay_2 * delay_2);
        prod3 = (delay_1 * coef_1)>>14;
        prod3 = prod3 * delay_2;
        Goertzel_Value = (prod1 + prod2 - prod3);
        //Goertzel_value <<=4;
        N=0;
        delay_1 = delay_2 = 0;

    }
        gtz_out[0] = Goertzel_Value;
}
```

Fig. 3. This figure shows the implementation of the Goertzel Algorithm in the C programming language. This clock function is for calculating the Goertzel value for the 697Hz frequency.

IV. Multi-Frequency Detection

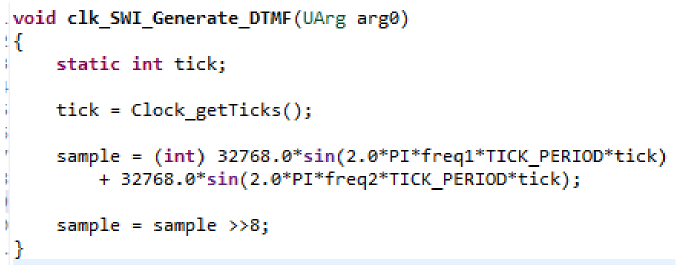
The code for multi-frequency detection builds on that of single frequency detection. Firstly, the user's input is required to select a button on the keypad. This is handled by the code shown in Figure 4. The while loop shown will continuously request inputs until a valid one is given.

Once the input has been chosen, the code in Figure 3 is applied to each frequency, through a separate clock function, and each output is stored in the array 'gtz_out'. The microprocessor loops from the sample generation function (shown in Figure 5), to each of the specific frequency clock functions 206 times before returning to task 1 and displaying the output.



```c
//loop that keeps asking for an input until a valid one is given.
while(dig!='0'&&dig!='1'&&dig!='2'&&dig!='3'&&dig!='4'&&dig!='5'&&dig!='6'
        &&dig!='7'&&dig!='8'&&dig!='9'&&dig!='*'&&dig!='#'&&dig!='A'
                &&dig!='B'&&dig!='C'&&dig!='D')
{
    System_printf("Enter a key press:\n");
    System_flush();
    dig = getchar();
    fseek(stdin,0,SEEK_END); //this clears the input buffer
}
```

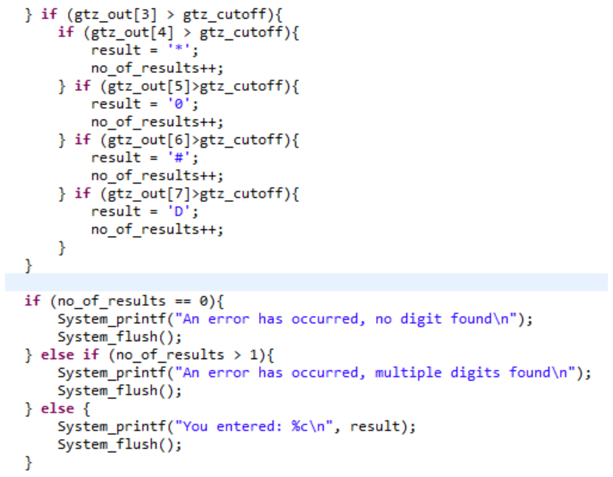
Fig. 4. The while loop that continuously asks the user for their chosen button press until they enter a valid input.



```c
void clk_SWI_Generate_DTMF(UArg arg0)
{
    static int tick;

    tick = Clock_getTicks();

    sample = (int) 32768.0*sin(2.0*PI*freq1*TICK_PERIOD*tick)
        + 32768.0*sin(2.0*PI*freq2*TICK_PERIOD*tick);

    sample = sample >>8;
}
```

Fig. 5. The clock function that generates the signal sample as the microprocessor's internal clock ticks.

As stated before, the 'gtz_out' values can be compared to set value to determine which frequencies are present and hence which digit is being represented. For this a single frequency from each group, the low group and high group, must be present. If there is not a frequency from one of the groups then no digits are represented. Alternatively, if more than one frequency from each group is present then there are multiple potential digits which could be represented. Either of these results will cause an error to display, otherwise the code simply prints the corresponding digit. The code responsible for this functionality can be seen in Figure 6.

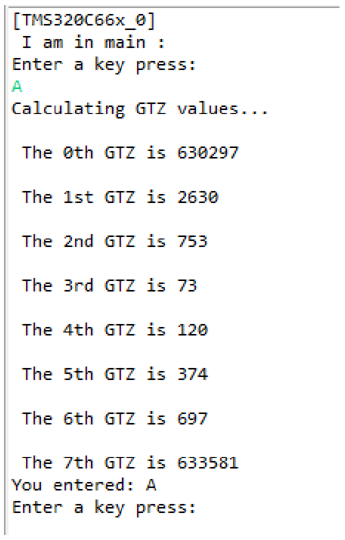


```c
} if (gtz_out[3] > gtz_cutoff){
    if (gtz_out[4] > gtz_cutoff){
        result = '*';
        no_of_results++;
    } if (gtz_out[5]>gtz_cutoff){
        result = '0';
        no_of_results++;
    } if (gtz_out[6]>gtz_cutoff){
        result = '#';
        no_of_results++;
    } if (gtz_out[7]>gtz_cutoff){
        result = 'D';
        no_of_results++;
    }
}

if (no_of_results == 0){
    System_printf("An error has occurred, no digit found\n");
    System_flush();
} else if (no_of_results > 1){
    System_printf("An error has occurred, multiple digits found\n");
    System_flush();
} else {
    System_printf("You entered: %c\n", result);
    System_flush();
}
```

Fig. 6. Part of the code that handles reconstructing the original button pressed from the Goertzel values obtained by applying the algorithm.

V. Results

Upon entering a valid button press, the microprocessor will calculate the Goertzel values and display an output similar to the one shown in Figure 7.



```
[TMS320C66x_0]
 I am in main :
Enter a key press:
A
Calculating GTZ values...

 The 0th GTZ is 630297

 The 1st GTZ is 2630

 The 2nd GTZ is 753

 The 3rd GTZ is 73

 The 4th GTZ is 120

 The 5th GTZ is 374

 The 6th GTZ is 697

 The 7th GTZ is 633581
You entered: A
Enter a key press:
```

Fig. 7. This figure shows the output displayed in the console when a valid input is entered. The program will first print out the Goertzel values before returning the original button pressed.

The Goertzel values shown in Figure 7 can be plotted as shown in Figure 8. From this plot, it is obvious which of the two frequencies are most prevalent in the signal. Therefore it is trivial to work out the original button pressed. Figure 9 shows the output when the ‘*’ button is pressed and, similarly, the two most prevalent frequencies in the signal are obvious. Therefore, we can assume that the algorithm has been implemented successfully.

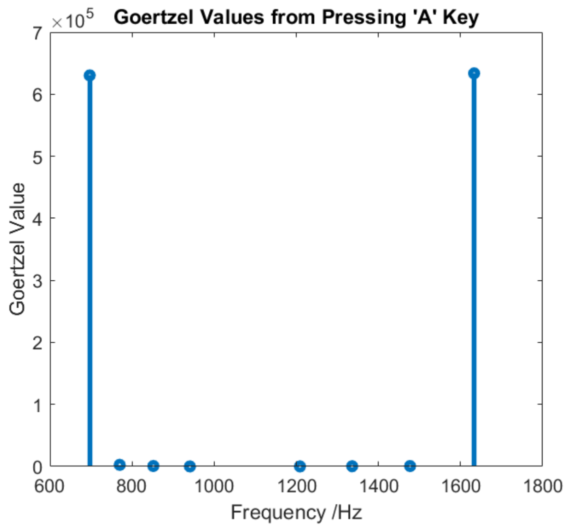


Fig. 8. This plot shows the Goertzel values for each of the eight frequencies after the ‘A’ button is pressed. IT can be seen that the Goertzel values for 697Hz and 1633Hz are significantly higher. This corresponds to the frequencies generated by pressing that button.

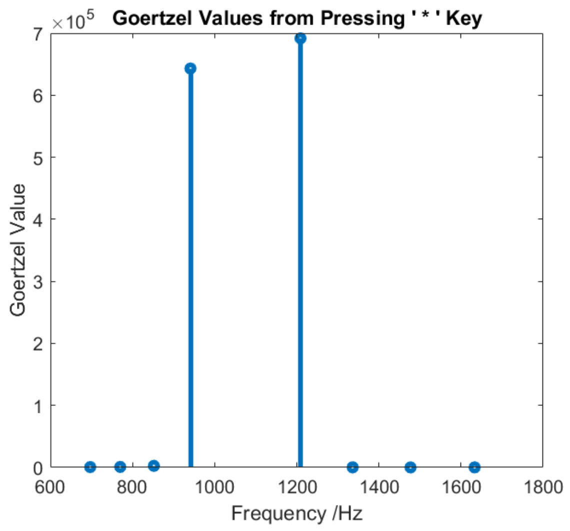


Fig. 9. This plot shows the Goertzel values for each of the eight frequencies after the ‘*’ button is pressed. It can be seen that the Goertzel values for 941Hz and 1209Hz are significantly higher. This corresponds to the frequencies generated by pressing that button.

VI. Applications

The most obvious application of DTMF is for telecommunications, such as identifying which number a customer has dialled in a telephone switching centre. More recently, companies have also started using DTMF masking where a customer uses a phone’s keypad to type in information which is then sent directly to the appropriate destination. This is particularly important for transmitting sensitive information, such as credit card numbers, as verbal communication is no longer required. This means the contact centre and its employees are removed from the line of data transmission making the transaction more secure. [2]

DTMF can also be applied in DTMF based remote control systems. These have a wide range uses, from phone-based home and office automation, [6] to control of robotics and machinery in industrial settings. [5] In each of these cases, the DTMF signal sent from a phone or remote will need to be decoded once it is received, hence the importance of the Goertzel algorithm and other methods of multi-frequency detection.

References

[1] M. J. Callahan. “Integrate DTMF Receiver,” *IEEE Journal Of Solid-State Circuits.*, vol. SC-14, No. 1, pp. 85-90, Feb. 1979.

[2] T. Critchley. “Why DTMF masking is critical to payment security,” *Computer Fraud Security.*, vol. 2015, No. 11, pp. 8-10, Nov. 2015.

[3] N. Dahnoun, “Digital Signal Processing Implementation: Using the TMS320C6000 DSP Platform,” Prentice Hall, 2000.

[4] N. Dahnoun, “Multicore DSP: From Algorithms to Real-time Implementation on the TMS320C66x SoC,” 1 edn, Wiley, 2018.

[5] T. L. V. V. Hemanth. P. Sasi Kiran. A. Suresh. “Industrial Application Of DTMF Communication In Robotics,” *International Journal of Innovative Research Development.*, vol. 2, No. 4, pp. 928-938, Apr. 2013.

[6] E. M. C. Wong. “A phone-based remote controller for home and office automation,” *IEEE Transactions on Consumer Electronics.*, vol. 40, No. 1, pp. 28-34, Feb. 1994.