Implementation, optimization and analytics of the Goertzel Algorithm in C using the TMS320 C6000 compiler

Otis Lee   
*Electrical and Electronic Engineering*  
*University of Bristol*st20714@bristol.ac.uk

Mohamed Eldebawy   
*Electrical and Electronic Engineering*  
*University of Bristol*bs19616@bristol.ac.uk

Samuel Cheung   
*Electrical and Electronic Engineering*  
*University of Bristol*jm20101@bristol.ac.uk

Abstract—This project implements the Goertzel Algorithm for interpreting DTMF signals using Code Composer Studio (CCS) in the C programming language. The algorithm is implemented both using SYS/BIOS, Texas Instrument’s own developed operating system, and without using SYS/BIOS. The implementation with SYS/BIOS is then further optimized using several techniques for the TMS320 C6000 compiler, including C intrinsics, loop unrolling and compiler switches.

Keywords— Goertzel Algorithm, Intrinsics, SYS/BIOS, compiler switches,

# Introduction: brief overview of the goertzel algorithm

A Dual Tone Multi-Frequency (DTMF) signal system comprises of digits 0 up to 9 and characters A, B, C, D, \*, #. They each are made up of 2 frequencies, one high and one low. A DTMF signal is formed by superimposing two different frequency tones from a selection of 8 available frequencies as seen in Table I.1.

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

Table I.1 Combination of eight frequencies corresponding to a character

The Fast Fourier Transform (FFT) algorithm is inefficient when applied to a DTMF system since it checks for the presence of all frequencies. The Goertzel algorithm is tailored to only detect the 8 different frequencies (16 different tone combinations), which makes it more computationally efficient They are based on equations below, and its process summarized in Figure 1:

For each desired frequency **k**, the feedback loop yields a product Q(n) at step **n.** The feedforward loop yields a Goertzel Coefficient yk(n) at step **n=206** and is stored in the array gtz\_out[8], or magnitude[8] depending on the usage. From each array, 2 of the largest coefficients are extracted, which represents the 2 most dominant frequency components in the signal. This information can further determine the key pressed or the data being extracted, depending on the user’s usage.

This report aims to cover the following parts:

* Detection of one frequency component using the Goertzel Algorithm implemented in C with TI’s SYS/BIOS operating system, sourced from user inputs.
* Detection of all 8 frequency components using the Goertzel Algorithm implemented in C with TI’s SYS/BIOS operating system, sourced from data file named ***data.bin***
* Detection of all 8 frequency components using the Goertzel Algorithm implemented in pure C language, sourced from user inputs.
* Optimization of task 2 using C intrinsics, compiler switches and other techniques

# Brief overview of optimization in C and equivalent code in assembly

## Debawy please fill in here

# Task 1: Detect 1 frequency using the Goertzel Algorithm

## Samuel Please fill in here

# Task 2: Detecting all frequencies using the goertzel Algorithm

This section aims to cover the design choice, process and goals of the code main\_gtz.c and util.c under gtz\_all\_freq. The Goertzel algorithm consists of 3 main events:

* Tone Generation
* Goertzel Computation
* Frequency Detection

A diagram of a process

Description automatically generated

Figure 0: Top-Level Flow Diagram for general Goertzel Algorithm Implementation

In the case of this task, the tone generation and computation events are undertaken by main\_gtz.c and the detection by util.c. SYS/BIOS allows these events to run in parallel and in a specific order. The roles of SYS/BIOS in the code are as follows:

* Creates a clock instance such that internal actions can be synchronized to clock ticks
  + For example: the function Clock\_getTicks() returns the number of elapsed clock cycles since the start of the program
* Instantiating interrupt service routines (ISR) for tone generation
* Instantiating ISR for computation : such that computation ISRs are called each time a sample is loaded, where N is incremented

In util.c, our frequency detection algorithm utilizes a for loop to loop through the n=8 samples that exist in the data.bin file. For each iteration, a while loop is conditioned by a flag; if detection of all frequencies is finished, the flag variable is set to 0, which exits the while loop and increments n by 1, moving onto computing the next sample. This loop structure is illustrated below:

A diagram of a algorithm

Description automatically generated

Figure I: Flow Diagram for util.c

By integrating both util.c and main\_gtz.c, we obatin the flow diagram below:Ω

A diagram of a process flow

Description automatically generated

Figure I: Flow Diagram for util.c + main.gtz.c

The result of the implementation is as follows:

|  |  |
| --- | --- |
| Sourced using data.bin file | Test case implemented using 852Hz and 1477Hz which corresponds to key 9 |
| A white text with black text  Description automatically generated | A white text with black text  Description automatically generated |
| Clock Cycles used: | |

Figure I: Console output when detection is finished

# task 3: implementing task 2 without sys/bios, input sourced by user inputs instead of binary data file

# Task 4: Optimizing task 2 using compiler switches, C intrinsics and other techniques

# Conclusion

##### References

1. N. Dahnoun, “Advanced DSP and FPGA Implementation” Lecture Slides, 2024.
2. N. Dahnoun, “Formal report: Goertzel Implementation”, 2024.
3. Texas Instruments, “module ti.sysbios.knlClock”, May 2018, <https://software-dl.ti.com/>
4. Texas Instruments, “TI-RTOS Kernel (SYS/BIOS), User’s Guide”, June 2020.
5. Texas Instruments, “module ti.sysbios.knl.Task”, May 2018, <https://software-dl.ti.com/>
6. Texas Instruments, “TMS320C6000 Compiler Optimization User’s Guide”, June 2020, <https://software-dl.ti.com/>