DSP $TMS320C6000^{TM}$ series based efficient DTMF detection using the Goertzel algorithm with optimized

1st Qingyu Zhang
Faculty of Engineering
University of Bristol
Bristol, GB
vn22984@bristol.ac.uk

2nd Shuran Yang
Faculty of Engineering
University of Bristol
Bristol, GB
rw22242@bristol.ac.uk

3rd Ruilong Liu
Faculty of Engineering
University of Bristol
Bristol, GB
hx22195@bristol.ac.uk

***Abstract*—Digital signal processors (DSPs) have a significant role to play in many applications involving signal or data processing, such as speech recognition, encoding, and decoding. One of the exceptional applications is the decoding of dual-tone multi-frequency (DTMF) signals using the Goertzel algorithm. Moreover, the optimization of software and hardware combinations is a potential research area that can enhance the efficiency and stability of algorithms. In this paper, we propose combination optimization strategies that can enhance the performance of the Goertzel algorithm. Our approach focuses on using intrinsic, compiler switches, and some compiler setting codes to improve the algorithm. By testing and combining different optimization strategies, we achieved a 16-fold improvement in execution efficiency compared to unoptimized code.**

***Index Terms*—Digital signal processors (DSPs), dual-tone multi-frequency (DTMF), the Goertzel algorithm, audio recognition, software and hardware combination optimization**

I. Introduction

The booming development of Internet of Things (IoT) devices has brought many conveniences to people's lives, such as people can communicate through the telephone system or the Internet anytime and anywhere. How to quickly and accurately process the signals transmitted by these sensor devices has become a decisive factor for the popularization of communication devices [1]. Nowadays, people mainly use a digital signal system called DTMF for communication [2]. The DTFM system as an international signal standard can be used not only in the computer field (such as voice mail), but also in the industrial field (telephone switching center) and so on. Fast and accurate signal processing of DTFM systems is a challenge [1].

Embedded microprocessors are ubiquitous when performing signal processing tasks [3]. DSPs are a special form of processor that can effectively play the role of digital signal processing in many fields with complex algorithms such as medical, industrial, and security [4].

The aim of this experiment is to decode DTMF. The commonly used methods are discrete Fourier transform (DFT) [2], fast Fourier transform (FFT) [2], digital filters [1] and the Goertzel algorithm [1] [2]. However, the first three methods have obvious defects. Both DFT and FFT algorithms have high computational complexity, which means that their operations are limited by processor performance. As for the digital filter method, its decoding accuracy is easily affected by distorted signals and noise. As early as 1958, the Goertzel algorithm was published [5]. Unlike the DFT method, which calculates based on the entire bandwidth of the signal, the Goertzel algorithm has the characteristic of mainly calculating a single item [2]. Therefore, when using the Goertzel algorithm to detect the pitch of DTFM, the computation time can be significantly saved [2].

In this article, based on the TMS320C6000™ DSP platform, we implemented the DTMF decoding using the Goertzel algorithm. In the chapter that follows, we also focuses on how to use the three optimization methods of Intrinsics instruction set, code and linear assembly components to optimize the Goertzel algorithm respectively. The purpose is to improve the decoding accuracy of DTFM and accelerate the running speed of DSP32.

II. System Architecture

A. Experiment Environment

The processor we used in this experiment is $TMS320C6000^{TM}$ DSPs produced by TEXAS INSTRUMENTS (TI). In comparison to the traditional C3x and C54x DSPs, the C6000 DSP has many advantages in architecture, the most typical performance is to simplify the control of the pipeline and increase the depth of the pipeline [6]. As shown in Fig. 1, we completed and optimised the Goertzel algorithm based on the $TMS320C6000^{TM}$ series of DSPs in Code Composer Studio 5.5.0 application by programming in C which for decoding DTFM signals in this project.

B. Design Principal

1) The Principle of the Goertzel algorithm: It can be seen from the introduction that the most effective method for decoding a small number of frequencies is to use the Goertzel algorithm [1]. The following part of this article continues to

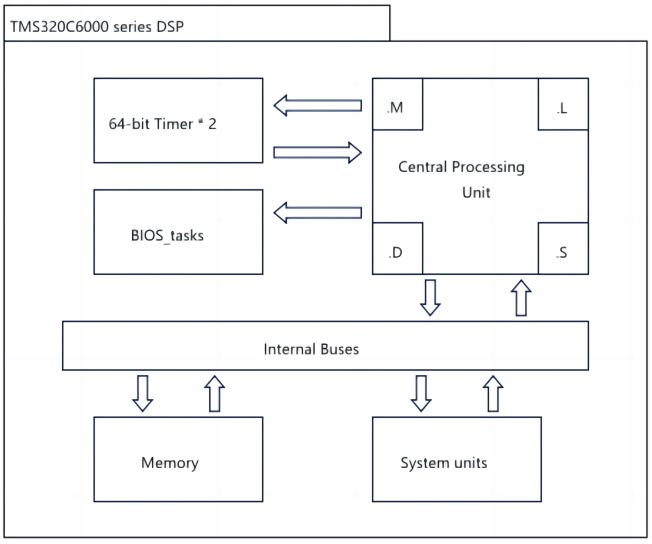

Fig. 1. DSP hardware architecture.

describe the principle of the traditional Goertzel algorithm and the improved Goertzel algorithm used in this project in more detail. As shown in Fig. 2, is a schematic diagram of the Goertzel algorithm network introduced in a traditional DSP textbook [1] [5]. This algorithm uses the Resource Sharing Approach , reducing the amount of hardware and significantly reducing the required data memory. For a length of N, the series for the traditional Goertzel algorithm is (1).

$$H_k(z) = \frac{1 - W_N^k z^{-1}}{1 - 2\cos(\frac{2\pi k}{N})z^{-1} + z^{-2}} \tag{1}$$

In Fig. 2, the traditional Goertzel algorithm outputs samples in 2 stages. In the first stage, N number of samples are first taken as input, then the data samples are shifted down through two delay elements, then zero-valued samples are taken as network input and the first stage calculation is again executed. Finally the computation of stage 2 is performed to obtain the required X(m) samples.

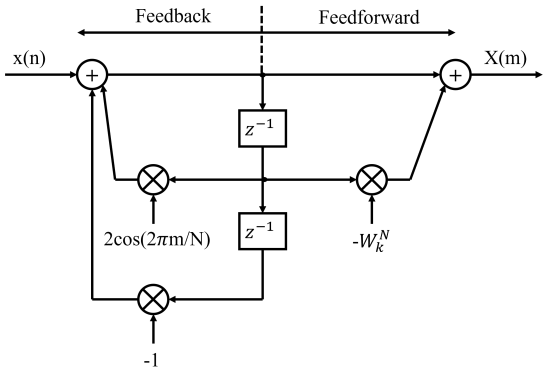

Fig. 2. Traditional DSP Textbooks Goertzel Network [5].

As shown in Fig. 3, unlike the traditional Goertzel algorithm, the improved Goertzel algorithm does not require zero-valued samples to be input in stage 1 for calculation, which makes the signal processing steps simpler. At this point, the number of stages of the optimised Goertzel algorithm for length N is (2).

$$H_k(z) = \frac{W_N^k - z^{-1}}{1 - 2\cos(\frac{2\pi k}{N})z^{-1} + z^{-2}} \tag{2}$$

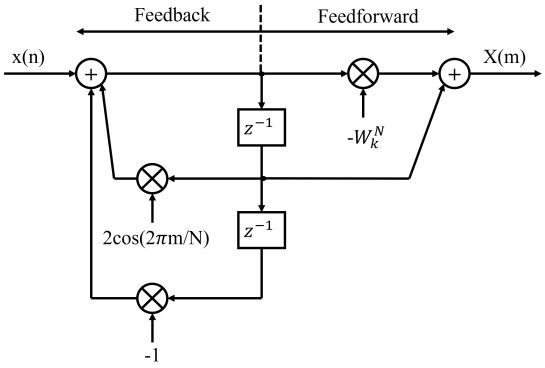

Fig. 3. A simpler Goertzel network after optimization [7].

2) Comparing DFT, FFT and the Goertzel Algorithm: In the section that follows, we will compare the efficiency and accuracy of DTFM tone detection with each of the three algorithms.

For the DFT, the (3) shows that the time complexity of the algorithm grows exponentially when the DFT method is used to decode the DTFM signal.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-\frac{2\pi n k}{N}} \tag{3}$$

For the FFT, the (4) yields a logarithmic increase in computational complexity, which is improved by decomposing samples into odd-even components.

$$X(k) = \sum_{i=0}^{N/2-1} x(2i)\, e^{-\frac{2\pi 2ik}{N}} + \sum_{j=0}^{N/2-1} x(2j+1)\, e^{-\frac{2\pi(2j+1)jk}{N}} \tag{4}$$

Despite the improved computational speed of the FFT method compared to the DFT algorithm, the Goertzel algorithm is able to reduce the number of real-valued multiplications by almost a factor of two. Not only that, but as the number of samples increases, the Goertzel algorithm always has the highest computational efficiency and accuracy of the results [2].

C. Algorithm

1) Validation:

a) Matlab Validation: In matlab, we want to use one of the coefficients 0x6D02 (corresponding to normalized angular frequency 0.85162353515625) to verify the feasibility of the Goertzel algorithm. The main logic is to use two nested loops, the outer loop traverses 2000 sinusoidal signals of different frequencies, the inner loop generates continuous sinusoidal signals of each frequency and inputs them into a combination of filtering and discrete Fourier transform (DFT) to calculate the magnitude of power of the output signal. The results are stored in the power array, and finally the powers are output through the table.

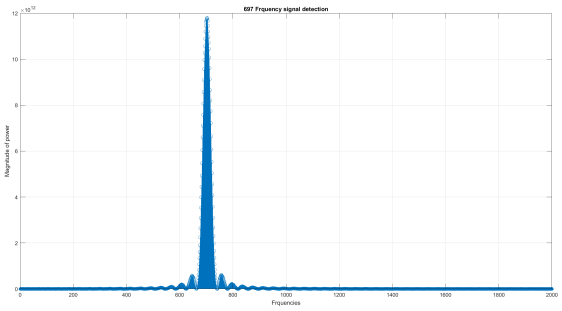

Fig. 4. Matlab code running validation.

Based on our findings shown in Fig. 4, we conclude that the Goertzel algorithm is sensitive to the normalised frequencies and their corresponding frequencies. Specifically, the power is particularly prominent when compared to other target frequencies.

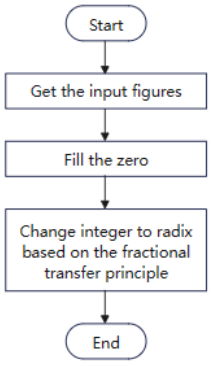

Fig. 5. A flowchart of python validation.

b) Python Validation: As shown in Fig. 5, to begin with, we obtain the input value and check if the length of the figure is a multiple of 4. If not, we use the “zfill” function to add zeros on the left side of the string until it reaches the nearest multiple of 4. This is necessary to ensure that the operands have the same number of bits and enable us to determine the correct fractional figure and symbol position. Afterwards, based on the principle of converting integers to fractional figures, we can obtain the relevant fractional figure.

The purpose of writing this python code is to analyze what all eight of the Goertzel coefficients are so that we can get a handle on the code. For the sake of brevity, let’s just take the first coefficient, 0x6D02. By analyzing the running results, we find that the normalized real value of coefficient ”0x6D02” is ”0.85162353515625” in decimal notation. Therefore, it is determined that the variable “num” represents a normalized intermediate result of discrete time series signals, which represents the cumulative value of discrete frequency response (DFR) coefficients of a specified frequency, namely the inner product operation result of the Goertzel algorithm. And it is also used to calculate the amplitude and phase information of signals at a specified frequency.

2) Implementation:

a) One Freqency Implementation: The discrete frequency response (DFR) of specified frequency is calculated by Goertzel algorithm. Specifically, it inputs a 16_bit PCM audio sampling value, performs a series of calculations using a number of coefficients, and finally gets the DFR coefficient of the input signal at the specified frequency.

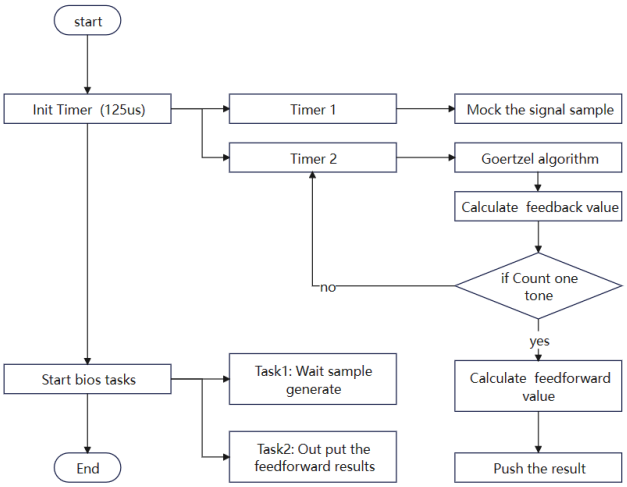

Fig. 6. Flowchart of the one frequency implementation.

As shown in Fig. 6, we first initialize the clock.Time 1 is used to mock the signal sample.At the same time ,Time 2 runs the Goeritzed algorithm.Firstly,we need to recurse several intermediates:The input parameter is a 16-bit PCM audio sampling value sample. We first converts sample to a variable input of type short and performs a multiplication of delay-1 using ”coef_1” to get an intermediate result prod1. Among them, ”coef_1” is one of the coefficients in the Goertzel algorithm, which is used to calculate the DFR coefficient of the input signal at the specified frequency. The code then adds and subtracts the input, prod1, and ”delay 2” and stores the results in a variable delay of type short. Then, ”delay 1” and delay are assigned to ”delay 2” and ”delay 1” for the next calculation.

Finally, the N variable is incremented by 1 to record the number of input sampling points that have been calculated.

This code actually implements a recursive calculation process in the Goertzel algorithm, which performs some operations between the current input sampling value and the previous sampling value, and uses the current DFR coefficient to calculate the intermediate result of the next sampling point, finally obtaining the DFR coefficient at the specified frequency.

Then we judge the result: when the number of samples of the input signal reaches ”N_const”, the code first calculates the three intermediate results prod1, prod2, and prod3 to calculate the Goertzel value. The three intermediate results represent the energy of the input signal at the specified frequency, the DFR coefficient at the specified frequency, and their product respectively. By adding these three results, the Goertzel value of the input signal at the specified frequency can be obtained, also known as the discrete frequency response coefficient.

After the current work is done, we start bios task,including waiting sample generate and out put the feed_forward results.

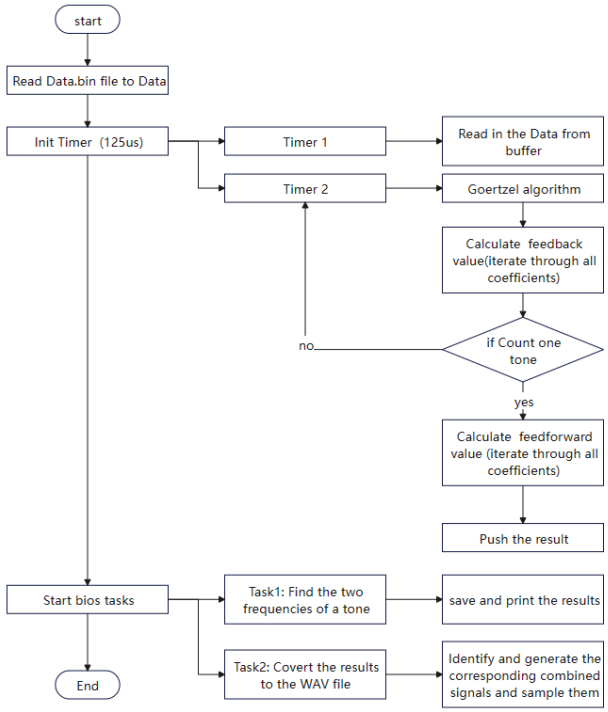

Fig. 7. Flowchart of the all frequency implementation.

b) All Frequency Implementation: As shown in Fig. 7 ,this code completes the following three sections:read Data.bin file to Data,init Timer(125us),start bios tasks.It should be noted in particular that we set the sampling rate of sample to 8000 times per second during the algorithm operation. The purpose of this is that the frequency of 8000 times per second is exactly in line with the clock of DSP operation (1/8000), so that the time of each tip can be satisfied to complete a sampling. So we don’t have to do anything else in the timer, instead , we just wait for the timer to come and we can finish sampling.

The purpose of TODO1 is to implement the feed-forward loop of the Goertzel algorithm, which is used to calculate the DFR coefficient of the input signal at the specified frequency. Specifically, this code first iterates through all of the DTMF signals to be evaluated, calculating their Goertzel values separately. For each DTMF signal, the intermediate results delay1[i] and delay2[i] of its DFR coefficient at the specified frequency need to be calculated.

The purpose of TODO2 is to implement the feed-forward loop of the Goertzel algorithm, which is used to calculate the DFR coefficient of the input signal at the specified frequency. Specifically, this code first iterates through all of the DTMF signals to be evaluated, calculating their Goertzel values separately. For each DTMF signal, the intermediate results delay 1[i] and delay 2[i] of its DFR coefficient at the specified frequency need to be calculated.

The purpose of TODO3 is to detect the frequency of the number key to determine which number key is being entered. Specifically, for each numeric key, firstly, the amplitude of its corresponding frequency (namely line frequency and column frequency) is calculated by the Goertzel algorithm, and DTMF NUM amplitude is obtained. Then, for each number key, the two values with the largest amplitudes corresponding to line frequency and column frequency are found in DTMF_NUM amplitudes respectively, and the subscripts exist-row and exist col of their corresponding amplitudes are recorded.Finally, the corresponding numeric key is found through the two subscripts and stored in the result array.

The purpose of writing TODO4 is to generate the corresponding two_tone multi_frequency signal based on the key number entered. Specifically, first, according to the input key number, determine the corresponding two frequencies, respectively assigned to the freq1 and freq2 variables; Then, for each number, the loop generates a signal of a certain length (samples-per-tone sampling points), and the value of each sampling point is obtained using the superposition of the sine waves of the two frequencies. The temp_volume variable stores the value of the current sampling point. Finally, the values of each sampling point are stored in a buffer array for subsequent output.

D. Optimization

1) Intrinsics function: Intrinsics are a special type of inline function that are commonly used in DSP programs for performing basic operations such as multiplication, addition, displacement, loading, storage, and more. Using Intrinsics in this manner enables the realization of more efficient code, as it converts function calls in high-level languages to processor native instruction sequences. This makes use of the hardware features and instruction set of the processor to directly complete operations during code execution, avoiding the costs of function calls and unnecessary instructions. As a result, the efficiency of code execution and running speed can be improved [8]. Intrinsics libraries are typically provided by compilers, and their underlying code is implemented in inline assembly language, allowing it to map directly to the underlying hardware instructions for more efficient computation.

2) Linear Assembly: Linear Assembly has two advantages, the first is instruction level parallelism optimization [9]: DSP processor usually supports some special instructions, such as SIMD instructions, vector instructions, etc., which can process multiple data at the same time, thus improving processing efficiency. Linear Assembly makes full use of these special instructions, converting some common computing tasks (including floating-point units, multipliers, accumulators, etc.) into a form, which is suitable for using these instructions, thus improving the execution efficiency of the program. The second is data locality optimization [9]: the memory access speed of DSP processor is relatively slow and the memory bandwidth is limited. To make full use of memory bandwidth and reduce access latency, we need to use local data access modes, such as sequential access and circular access. Linear Assembly optimizes memory access mode, reduces latency and bandwidth consumption during data access, and improves handling capacity and response speed of programs.

Generally, "Linear Assembly" optimization enhances program execution efficiency and response speed by optimizing DSP hardware architecture and instruction sets. Optimization results depend on factors such as program structure, data characteristics, and compiler optimization, so developers should consider these factors when optimizing.

3) Using tool: There are tools available to optimize code, including using specific compiler options, using code analysis tools, using performance analysis tools, and so on.

a) Using Instruction "#Pragma MUST ITERATE": Loop unwrapping is an optimization method that combines multiple iterations in a loop body into a single iteration. We can use the instruction "pragma MUST ITERATE" directive to tell the compiler the number of iterations to iterate over the loop to improve the efficiency of the program.By unrolling the loop, we have a better load balancing and usage of units. General steps to optimize DSP code using loop unwrapping include analyzing the performance bottleneck of the program, determining the number of times the loop is unrolled, writing loop expansion code and performing performance testing and tuning.

b) Using "restrict" Keyword: The "restrict" keyword could help optimize the code, especially for pointer access and array operations. The "restrict" keyword tells the compiler that a pointer can only access the area of memory it points to and that it will not overlap with other Pointers or areas of memory. This can help the compiler use more efficient code generation techniques, such as pointer registers, loop unwrapping, and so on, to improve the execution efficiency of the program.

c) Using "nassert" Keyword: The "nassert" keyword could be used to alignie the memory space, thereby speeding up the execution of the code. In DSP programming, using the "nassert" keyword can help optimize the code, especially in situations where fast execution is required.

III. Results

A. Preliminary Results

First, we run the detect bin file code and identify the eight keys in the bin file to represent the following characters,which shown in Fig. 8:

```
[TMS320C66x_0]
System Start
result[0]: B
result[1]: 1
result[2]: 3
result[3]: 1
result[4]: 8
result[5]: 4
result[6]: *
result[7]: 9

Detection finished
```

Fig. 8. Corresponding character.

According to DTMF frequency table dtmf_freqs and key character correspondence relationship to obtain the frequency value. Then, using these two frequency values, the DTMF signal is generated by the sine function. The length of each signal is 0.5 seconds, the sampling rate is 8000 Hz. Finally, all the signals are joined together to produce an audio file containing the DTMF signals corresponding to the eight key characters.

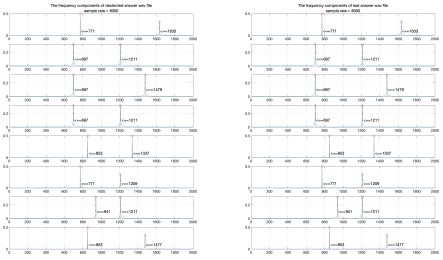

Fig. 9. Comparison of the identification results with the original results.

Finally, compare frequencies and tones sequence in the generated wav file with the audio file provided by the instructor using MATLAB as shown in Fig. 9. The conclusion drawn is that the identification was successful!

B. Optimization Results

Table I shows the results of the optimisation, in this project we used 5 strategies for the optimisation, as follows:

- Linear assembly.
- Using tools to balance two data paths.
- Using tools to alignment address.
- Intrinsics function.
- C code.

TABLE I
TABLE OF OPTIMIZATION RESULTS.

Optimisation methods	Every feedback cycles	Final feedforward cycles
Linear assembly off	422	707
Linear assembly 0	333	506
Linear assembly 1	205	252
Linear assembly 2	52	51
Linear assembly 3	56	56
Using tools (Balance)	52	51
Using tools (Alignment)	52	51
Intrinsics function	52	53
Cancel the reduction	52	46

IV. COMPARATIVE ANALYSIS

Table I collects all results, and the optimization effect can be judged by comparing the data size of every feedback cycles tdiff and final feedforward cycles tdiff_final in Table I. From Table I, it can be seen that the best optimisation method is the C code optimisation, which not only ensures that the data does not overflow by eliminating the reduction of the final power value, but also increases the speed of the code, making the optimised result as low as 46. The next best optimization can be achieved by using tools that balance two data paths, resulting in an optimized result of 51. Although the final result of the alignment address which using same method is also 51, but it is a useless optimisation because the memory itself is aligned. The high-level Linear assembly optimisation approach is not as effective because the assembly optimiser automatically generates highly parallel assembly code making it easy for the assembly optimiser to achieve handwritten assembly performance.

It is worth mentioning that not all optimization methods are positive optimization, and low-level linear assembly (off and 1) and Intrinsics function have negative optimization results.

In summary, five different methods have been used to optimise the Goertzel algorithm running in the $TMS320C6000^{TM}$ DSP processor. These optimisation methods increase the magnitude of power and reduce the cycle time of the system. We mainly recommend the C code optimisation method, which can result in cycles as low as 46. While the benefits of optimisation can be realised when the system is using advanced linear assembly methods, the benefits of further improvements are minimal.

V. CONCLUSION

This paper presents an optimized Goertzel algorithm that improves the efficiency of DTMF decoding. Unlike other research on optimization algorithms, our approach focuses on combining software and hardware to improve the algorithm's stability and efficiency. As a result, we achieved outstanding performance in both hardware and software systems, which can be applied to various applications.

We have demonstrated that implementing algorithms on the DSP system is an excellent choice. Although DSP has excellent data and signal processing performance, optimization is still a challenging task to reduce processing time. We optimized the algorithms by fully utilizing the DSP's hardware resources and using various optimization tools and codes to improve assembly execution speed, such as intrinsic and compiler switches. Ultimately, we achieved a 16-fold improvement in code execution speed compared to the original code.

In the future, we will continue to optimize our codes to make them more robust, generalized, and portable, to meet different hardware platform needs. We will also continue to focus on improving the assembly codes to enhance performance even further.

ACKNOWLEDGMENT

The authors are grateful to Professor Naim Dahnoun and the teaching assistants at the University of Bristol for arranging the necessary tools and equipment to guide us through the above research.

REFERENCES

[1] S. Selman and R. Paramesran, "Comparative analysis of methods used in the design of dtmf tone detectors," in 2007 IEEE International Conference on Telecommunications and Malaysia International Conference on Communications. IEEE, 2007, pp. 335–339.

[2] T. Joseph, K. Tyagi, and R. Kumbhare, "Quantitative analysis of dtmf tone detection using dft, fft and goertzel algorithm," in 2019 Global Conference for Advancement in Technology (GCAT). IEEE, 2019, pp. 1–4.

[3] M. Wolf, "Chapter 1 - embedded computing," in Computers as Components (Third Edition), third edition ed., ser. The Morgan Kaufmann Series in Computer Architecture and Design, M. Wolf, Ed. Boston: Morgan Kaufmann, 2012, pp. 1–50. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123884367000015

[4] O. Diouri, A. Gaga, H. Ouanan, S. Senhaji, S. Faquir, and M. O. Jamil, "Comparison study of hardware architectures performance between fpga and dsp processors for implementing digital signal processing algorithms: Application of fir digital filter," Results in Engineering, vol. 16, p. 100639, 2022.

[5] S. N. Bhavanam, P. Siddaiah, and P. R. Reddy, "Fpga based efficient dtmf detection using split goertzel algorithm with optimized resource sharing approach," in 2014 Eleventh International Conference on Wireless and Optical Communications Networks (WOCN). IEEE, 2014, pp. 1–8.

[6] P. Yin, "Introduction to tms320c6000 dsp optimization," Performance Improvement, vol. 67, p. C67x, 2011.

[7] "A simpler goertzel algorithm - rick lyons," https://www.dsprelated.com/showarticle/1386.php, (Accessed on 03/11/2023).

[8] D. Batten, S. Jinturkar, J. Glossner, M. Schulte, R. Peri, and P. D'Arcy, "Interaction between optimizations and a new type of dsp intrinsic function," in Proceeding of the International Conference on Signal Processing Applications and Technology (ICSPAT'99), 1999.

[9] A. Pegatoquet, E. Gresset, M. Auguin, and L. Bianco, "Rapid development of optimized dsp code from a high level description through software estimations," in Proceedings of the 36th annual ACM/IEEE Design Automation Conference, 1999, pp. 823–826.