修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院 情報理工学研究科 情報・ネットワーク工学 専攻　博士前期課程		
氏　　名	Wu Xiaoting	学籍番号	2031071
論文題目	FFTアクセラレーションを活用したRISC-Vベースの音声認識に関する研究		

要　旨

近年コンピュータ技術の進展や膨大な音声データの集積などに伴い、音声認識の実用化に向けた研究が加速している。音声認識では、認識をするための特徴量としてケプストラム領域の特徴量であるメル周波数ケプストラム係数（Mel-Frequency Cepstral Coefficients：MFCC）を用いるのが一般的である。しかし、現在組み込みシステムの音声MFCCを抽出するプロセスにおいて、大半の時間が高速フーリエ変換(Fast Fourier Transform：FFT)に使用され、システムのリアルタイム性と消費電力に影響が大きいと考えられる。

本研究では、RISC-Vベースのマイクロコントローラを使用し、Field Programmable Gate Array（以下FPGA）に実装されるFFTアクセラレータを提案しする。アクセラレータの原理として、通常のFFTアルゴリズムで行う浮動小数点演算を整数に置き換え、ハードウェア回路に実装することで処理速度の向上が期待できる。RISC-V Rocket のマイクロコントローラを評価ボードFPGAに実装し、長さが異なる3種類の音声サンプルデータに対してアナログ・デジタル変換(ADC)、フィルタリング、窓関数処理、FFT、およびメル特徴抽出の流れで処理を行った。そして、FFTの処理において、浮動小数点演算FFT処理と整数演算FFT処理を実装したハードウェアFFTで実験を行い、実行時間とハードウェアリソース使用率を比較した。

結果として、整数演算FFT　アクセラレータの実装により、処理速度が160倍以上に向上した。MFCC全体的な抽出パフォーマンスが20.7％向上した。ハードウェアリソース使用率として、マイクロプロセッサはSystem on Chip (SoC)の19％を占め、DDR3コントローラーとFFTアクセラレータは、それぞれ59％と16％のリソースを占めた。FFT1024、FFT512、FFT256はRISC-Vプロセッサのハードウェアリソース使用率よりそれぞれ16%、57%、77%と少なった。

浮動小数点演算を整数演算に置き換え、そしてハードウェア化によって効率が大幅に向上することができた。今後の展望として、メル特徴抽出処理の整数演算の置き換え及びハードウェア化の課題は将来の最適化方向になると期待できる。

令和３年度　修士論文

FFTアクセラレーションを活用した RISC-Vベースの音声認識に関する研究

学籍番号	2031071
氏名	Wu　Xiaoting
電気通信大学	情報理工学研究科
情報・ネットワーク工学専攻	情報通信工学プログラム
主任指導教員	範　公可　教授
指導教員	石橋　孝一郎　教授
提出日	令和 4年1月27日

目 次

第1章　序論

1.1　研究背景・目的

近年、音声認識技術の発展に伴い、様々な環境下やシチュエーションにおいて使用される機会が増加している [1, 2]。例えばカーナビゲーションの操作や会議音声の議事録化など様々な分野に応用されている [3]。また、音声認識では、認識をするための特徴量としてケプストラム領域の特徴量である MFCC(Mel-Frequency Cepstral Coefficients) を用いることが一般的である。ケプストラム領域は、対数スペクトルをフーリエ変換した領域であるため、スペクトル領域においてある箇所のみに重畳していた雑音であっても、ケプストラム領域ではその雑音が広がってしまい、ケプストラムの全ての項に対して雑音の影響を与えてしまうという欠点がある。このため、MFCC という特徴量を用いることができれば、雑音の分離がしやすく有利である。

しかし、現在組み込みシステムの音声 MFCC を抽出するプロセスにおいて、大半の時間が高速フーリエ変換 FFT(Fast Forier Transform) に使用され、システムのリアルタイム性能と消費電力に影響が大きいと考えられる [4]。

1.2　研究方法

本研究では、RISC-V ベースのマイクロコントローラを使用し、FPGA（Field Programmable Gate Array）に実装された FFT アクセラレータを提案した。

アクセラレータの原理として、通常の FFT アルゴリズムで行う浮動小数点演算を整数で実行し、これをハードウェア回路に実装することでハードウェア上の処理速度の向上が求められると考えた。これを踏まえ、まず RISC-V Rocket マイクロコントローラ というマイクロコントローラを評価ボード FPGA に実装し、長さが異なる 3 種類の音声サンプルデータに対してアナログ・デジタル変換 (ADC)、フィルタリング、窓関数処理、FFT、およびメル係数特徴抽出の流れで処理を行った。そして、中の FFT のステップにおいて、通常の double 型に基づいた FFT と整数化した FFT や HDL を用いたハードウェア FFT で実験を行い、計測できた実行時間とハードウェアリソース使用率を比較した。

1.3　論文の構成

本論文の構成として、全 6 章で構成されている。以下に、本論文の構成を記す。

第 1 章 序論

研究を行った背景として、音声認識システムに於ける背景を述べる。その上で、MFCC という特徴量を抽出する時の問題点を述べる。そして本研究でその問題点を解決するための提案について述べる。

第 2 章 音声認識の基礎

自動音声認識 ASR(Automatic Speech Recognition) の原理について紹介する。

第 3 章　MFCC について

MFCC の基礎と抽出手法について紹介する。

第 4 章　提案手法

第 3 章で提起した問題点を解決するための，本研究で提案する MFCC における FFT 加速の手法について述べる。

第 5 章　実装結果

第 4 章の提案で行った実験手順、実験結果、及び考察を行う。

第 6 章　結論

本研究の成果・今後の展望をまとめる。

第2章　音声認識の基礎

本章では、音響・音声信号の処理から音声特徴抽出手法まで、研究に関連する音声処理の基礎知識について説明する。

2.1　ASR

音声認識、いわゆる自動音声認識と呼ばれるは、通常 ASR と略される。主に人間の音声の字句コンテンツをコンピュータで読み取り可能な入力に変換すること（図 2.1）。一般的に理解可能なテキストコンテンツに変換し、バイナリエンコーディングまたは文字シーケンスの場合もある。

図 2.1: ASR

音声認識は、数学と統計、音響と言語学、コンピューターと人工知能などの基本的な分野と最先端の分野をカバーする、学際的な知識を統合する最先端のテクノロジーであり、自然な人間とコンピューターの相互作用テクノロジーの重要なリンクである。

しかし、音声認識の誕生以来、半世紀以上もの間、実際では一般的に応用されていなかった。これは、音声認識の技術的な欠陥に関連しており、その認識精度と速度は実際のアプリケーションの要件を満たすことができない一方で、音声認識に対する業界の高い期待もに関連している。

2.2　音響・音声信号の処理

音の本質は空気の振動からなる信号、すなわち波である [7]。波の波形を構成する要素は、大きく分けて振幅（音の大きさ）と周波数（音の高さ）の 2 つに分けられる。

例として、図 2.2 に，サイン波の波形と周波数特性を示す [8]。

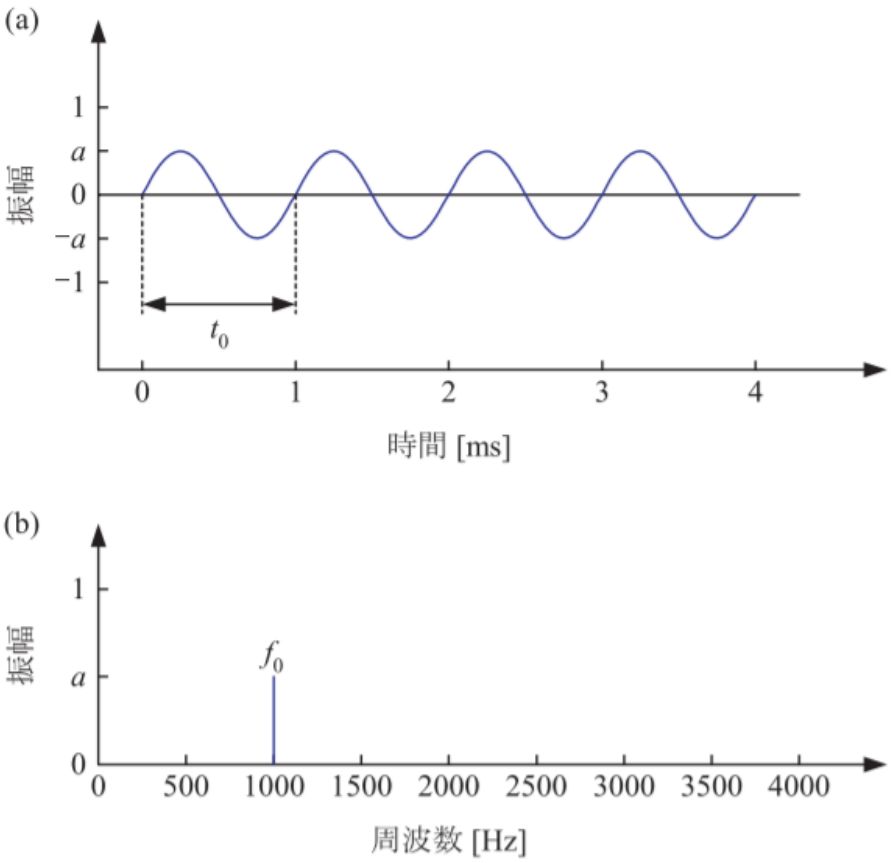

図 2.2: サイン波：(a) 波形，(b) 周波数特性

ここで，周波数 f_0 と周期 t_0 は逆数の関係である。標本化周波数を f_s とすると，振幅 a，周波数 f_0，長さ L のサイン波は，次のように定義できる。

$$s(n) = a \sin\left(\frac{2\pi f_0 n}{f_s}\right) \quad (0 \leq n < L) \tag{2.2.1}$$

音声信号処理では、音声波形（アナログ信号）そのものを操作することは少なく、主に周波数領域（音を周波数別に分解した状態、デジタル信号）に変換してから操作する。

その理由として、音声波形は、振幅と位相が時間的にゆるやかに変化する正弦波の和で構成されていると近似可能である。さらに、人間の聴覚による音声の知覚において重要な

特徴は、主として振幅情報に含まれており [9]、位相情報は通常重要な役割を果たしていない。

つまり、音声認識は音声周波数特性（スペクトル特性）のみを抽出して分析すればよいと考えられる。

2.3　音声スペクトル特性と抽出手法

ある信号波形をあらゆる周波数の正弦波に分解した時、どの周波数成分が強いかについて調べるために、「フーリエ変換」する。特に、分割したフレーム単位で考えるため、「短時間フーリエ変換」と呼んでいる。

音声は、短時間区間ごとの電力スペクトル密度（周波数領域におけるパワー特性）で測ることが多い。

短時間スペクトルを求める二つの手法はノンパラメトリック分析とパラメトリック分析である。

- ノンパラメトリック分析法：

分析対象の信号に関して、特にモデルを仮定せずに周波数分析を行う手法。万能であるが抽出すべきパラメータは多くなる。

例えば、DFT（離散フーリエ変換）分析、ケプストラム分析。

- パラメトリック分析法：

分析対象信号に対して特定のモデル化を行い、そのモデルを表現する特徴パラメータを抽出する。音声をよく表現するモデルを用意できるならば、能率的な分析が可能。

例えば：線形予測分析。

本研究ではケプストラム分析法を基づく、その中最も使われているケプストラム領域の特徴量である MFCC 特徴について研究した。

第3章　MFCCについて

本章では、MFCC の原理や流れについて説明する。

3.1　MFCC とは

人間の聴覚には、周波数の低い音に対して敏感で、周波数の高い音に対して鈍感である[1]。人間の聴覚性質、すなわち音の聞こえ方に基づいた特徴量 MFCC 特徴（メル係数特徴）は音声認識で数多く使用されている。

MFCC では人の聴覚特性に合わせて音素を変換し，スペクトルの包絡を多くの特徴量を用いて表現することで普通の分析では捉えきれなかった音素の特徴を捉えることができるようなメリットがある。

1000 [Hz] の音を 1000[mel] の音高と定められ、以下のような関係式がある。ここで、mel はメル係数特徴を表し、f は周波数を意味する。

$$\text{mel} = 2595.0 \log_{10} \left(1.0 + \frac{f}{700.0}\right) \tag{3.1.1}$$

ある音声ファイルを例として、メル係数特徴とメル周波数の関係をプロットしてみた（図 3.1）。

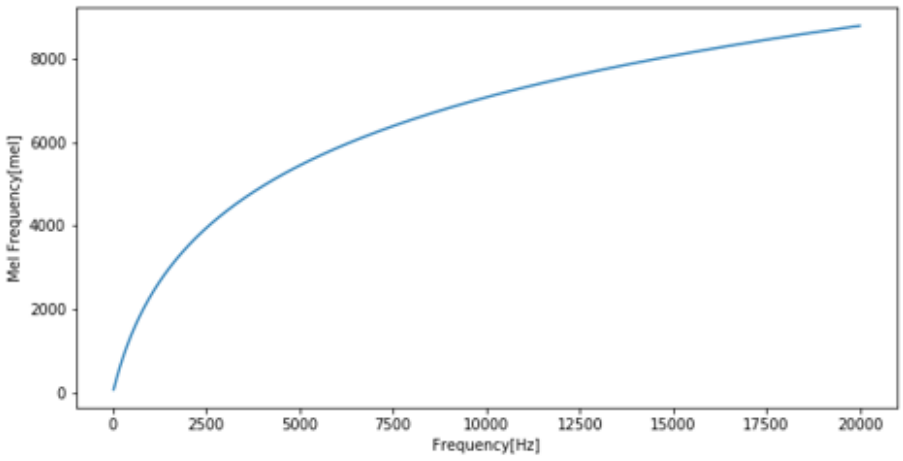

図 3.1: メル係数特徴とメル周波数の関係

メル係数特徴を用いて、周波数を選ぶことで等分割することができる。
ここでは、上記の例に分割数を 64 に設定してプロットしてみた（図 3.2）。

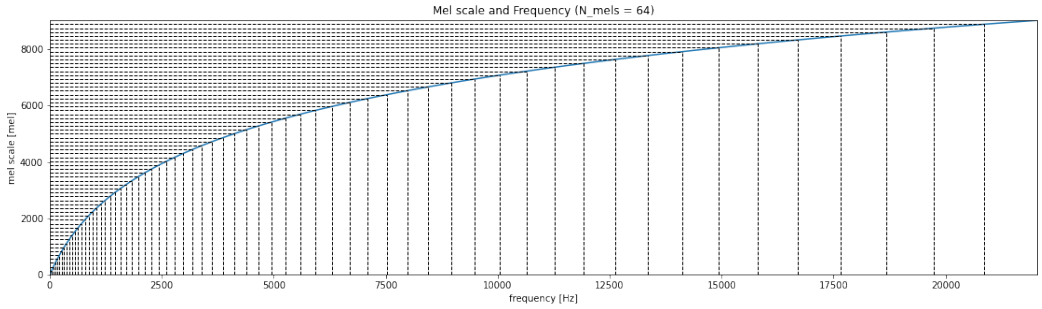

図 3.2: 周波数=64 の分割結果

その分割数を大きく設定するとより細かい目盛りのスケールを作ることができて、その逆も然りである。

3.2 MFCC の抽出手法

以下に MFCC の抽出手法を示す。

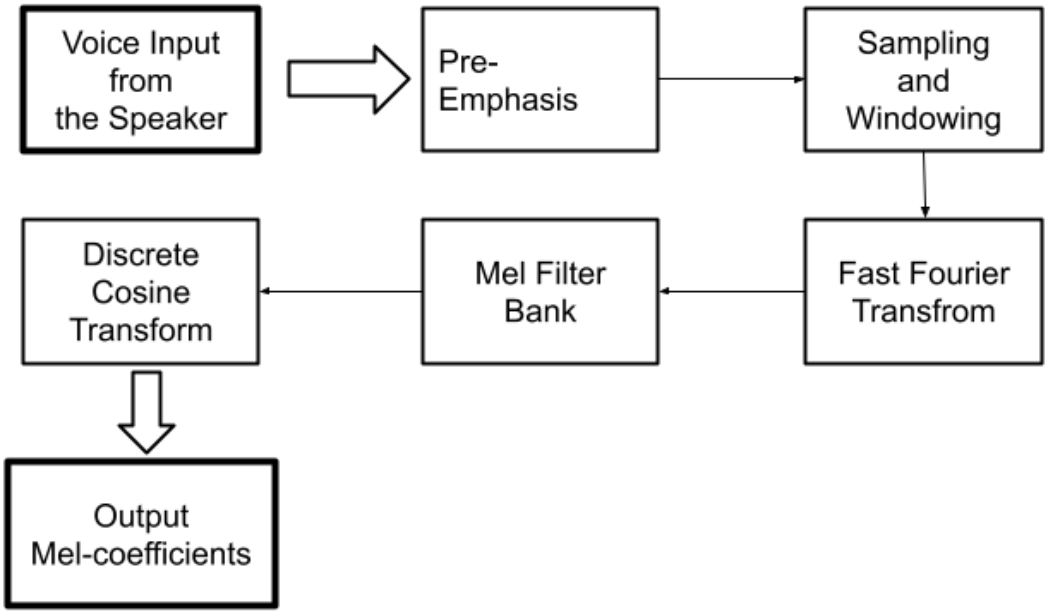

図 3.3: MFCC の抽出手順

図 3.3 の抽出手法の詳細説明を以下に記す。

3.2.1 高域強調 (Pre-Emphasis)

音声信号は、周波数が高くなるほど、振幅が小さくなる。よって、周波数成分の抽出がしにくくなる。周波数成分が高い部分を抽出しやすくするためにプリエンファシスフィルタ（pre-emphasis filter）をかける。プリエンファシスフィルタの定義は

$$y(n) = x(n) - px(n-1) \tag{3.2.1}$$

ここで、x(n) は音声波形データ、p はプリエンファシス係数。0.97 を使うことが多い。

実際にプリエンファシスフィルタをかけた波形とそのスペクトルを表示してみた。（図 3.4）。

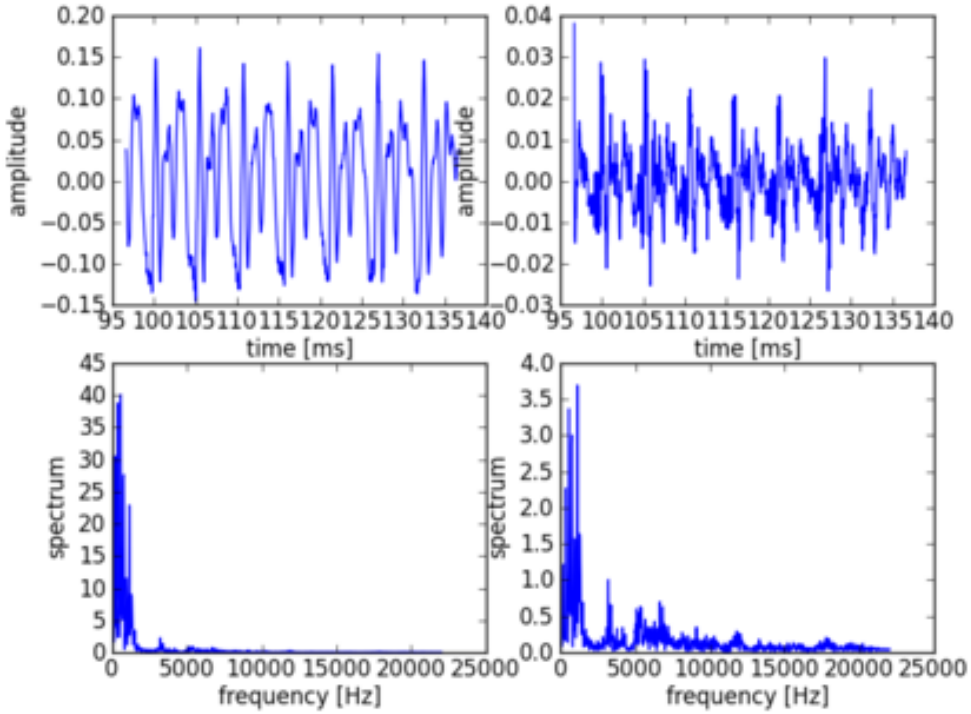

図 3.4: プリエンファシスフィルタの結果波形比較

左上が元の波形、右上がフィルタをかけた波形である。左下が元の波形のスペクトル、右下がフィルタをかけた波形のスペクトルである。プリエンファシスフィルタをかけることで高域成分が強調されているのがよくわかった。

3.2.2 窓関数 (Windowing)

不連続なデータに対してその波形の両端を減衰させるように窓関数をかける。図 3.5 にはよく使われる窓関数を示す。

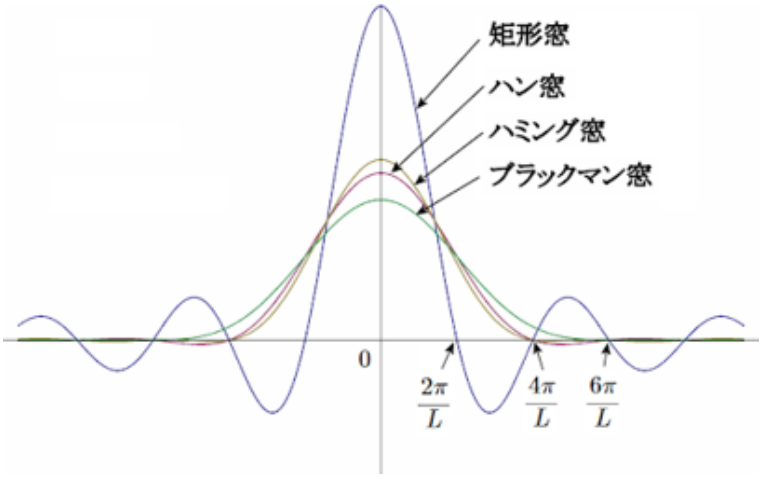

図 3.5: 窓関数比較

本研究ではハミング窓関数を使用している。

3.2.3 高速フーリエ変換 (FFT)

デジタル信号処理（DSP）では、FFT は最も基本的で有用なシステム構築ブロックの 1 つである。ここでは音声波形のパワースペクトル（振幅スペクトル）を求めるため FFT を行う。

FFT は元々DFT（Discrete Fourier Transform）を計算するための高速アルゴリズムである。DFT は 線形演算なので、行列とベクトルの積で表せる。FFT は、この行列を疎行列の積に直した形になっている。

一次元データの場合、DFT の時間計算量は $O(N^2)$ である。ここで、N はデータ長である。疎行列をベクトルにかける演算は $O(N)$ の演算量で、$logN$ 個の疎行列の積をかけるので、$O(NlogN)$ くらいの計算になる。

もっと複雑な合成数の場合、計算量の見積もりはもっと複雑になる。ここでは省略する。

行列因数分解を使用して FFT を説明する。8 点 DFT は、行列積として記述できる。ここで、$W = W_8 = e^{-i\pi/4} = (1-i)/\sqrt{2}$ とする：

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{pmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^1 & W^4 & W^7 & W^2 & W^5 \\ W^0 & W^4 & W^0 & W^4 & W^0 & W^4 & W^0 & W^4 \\ W^0 & W^5 & W^2 & W^7 & W^4 & W^1 & W^6 & W^3 \\ W^0 & W^6 & W^4 & W^2 & W^0 & W^6 & W^4 & W^2 \\ W^0 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

入力配列 a がビット反転されるように再配置し、8×8 行列を因数分解する。ここの“ . ”は 0 を意味する。

$$
\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{pmatrix}
=
\begin{pmatrix}
W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\
W^0 & W^4 & W^2 & W^6 & W^1 & W^5 & W^3 & W^7 \\
W^0 & W^0 & W^4 & W^4 & W^2 & W^2 & W^6 & W^6 \\
W^0 & W^4 & W^6 & W^2 & W^3 & W^7 & W^1 & W^5 \\
W^0 & W^0 & W^0 & W^0 & W^4 & W^4 & W^4 & W^4 \\
W^0 & W^4 & W^2 & W^6 & W^5 & W^1 & W^7 & W^3 \\
W^0 & W^0 & W^4 & W^4 & W^6 & W^6 & W^2 & W^2 \\
W^0 & W^4 & W^6 & W^2 & W^7 & W^3 & W^5 & W^1
\end{pmatrix}
\begin{pmatrix} a_0 \\ a_4 \\ a_2 \\ a_6 \\ a_1 \\ a_5 \\ a_3 \\ a_7 \end{pmatrix}
$$

$$
=
\begin{pmatrix}
1 & \cdot & \cdot & \cdot & W^0 & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & W^1 & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & W^2 & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & W^3 \\
1 & \cdot & \cdot & \cdot & W^4 & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & W^5 & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & W^6 & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & W^7
\end{pmatrix}
\begin{pmatrix}
1 & \cdot & W^0 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & W^2 & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & W^4 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & W^6 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & W^0 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & W^2 \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & W^4 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & W^6
\end{pmatrix}
\begin{pmatrix}
1 & W^0 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & W^4 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & W^0 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & W^4 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & W^0 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & W^4 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & W^0 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & W^4
\end{pmatrix}
\begin{pmatrix} a_0 \\ a_4 \\ a_2 \\ a_6 \\ a_1 \\ a_5 \\ a_3 \\ a_7 \end{pmatrix}
$$

上部の密な（dense：ゼロなし）行列を乗算する方が、下部の 3 つの疎行列（sparse：ゼロが多い）を乗算するよりコストがかかる。よって、FFT が DFT より高速な計算となる。

3.2.4 メルフィルタバンク (Mel Filter Bank)

メルフィルタバンクを説明する。フィルタバンクというのはバンドパスフィルタ [12] を複数並べたものである。ここでは、三角形のバンドパスフィルタをオーバーラップさせながら並ぶ。三角形のバンドパスフィルタの数をチャネル数と呼ぶ。

メルフィルタバンクは、バンドパスフィルタの三角窓がメル尺度上で等間隔になるように配置される。メル尺度上で等間隔にならべたフィルタを Hz 尺度に戻すと高周波になるほど幅が広い三角形になる。下の図 3.6 のようなイメージである。

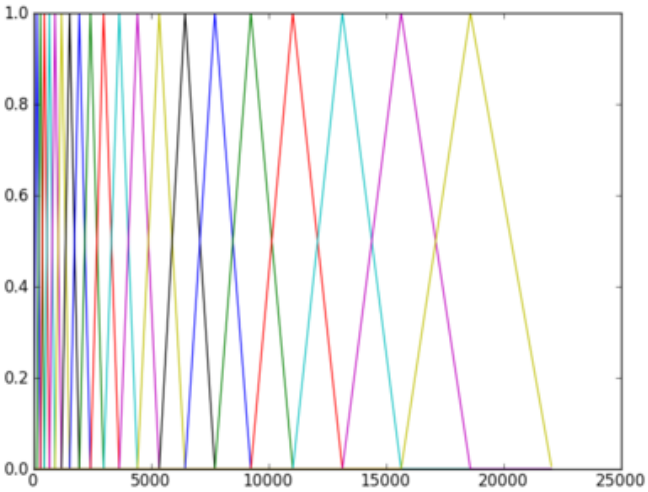

図 3.6: メルフィルタバンク

均等な幅になるようにメルフィルタバンクの個数に応じたフィルタを作成する。それを用いて、FFT 変換して求めた音声波形のパワースペクトルをメルフィルタバンクの個数のグループに分けて、メルフィルタバンクをかける。フィルタ後の振幅を足し合わせて対数をとる。

信号 $x(n)$ のフェリエ変換をとすると、ケプストラム $c(m)$ は

$$c(m) = \frac{1}{2\pi}\int_{\pi}^{\pi} \log\left|X\left(e^{j\omega}\right)\right| e^{j\omega n} d\omega \tag{3.2.2}$$

3.2.5 逆離散コサイン変換 (Discrete Cosine Transform, DCT)

最後に、逆離散コサイン変換を適用してケプストラムを導出する。ケプストラム Ci は

$$Ci = \sqrt{\frac{2}{L}} \sum_{l=1}^{L} \log m(l) \cdot \cos\left\{\left(l - \frac{1}{2}\right)\frac{i\pi}{L}\right\} \tag{3.2.3}$$

ケプストラムの低次成分 (スペクトルの包絡を表す係数) を声道のスペクトルとして音声認識の波形分析に利用できる。

3.3　まとめ

本章ではMFCCの基礎及び抽出手法を紹介した。組み込みシステムにのMFCC導出過程は、大半の時間がFFTに使用されたことが分かる。システムの性能と消費電力には大きな負担だと考えられる。

本研究の目的はFPGA上に音声のMFCC特徴抽出の効率低下に対して、FFTにかかる時間を抑えるために、整数化したFFTをハードウェアへ実装、ハードウェアFFTアクセラレータを提案する。

第4章　提案手法

本章では、提案手法及び機器構成について述べる。

4.1　提案手法

本研究の提案手法について説明する。今回提案する FFT アクセラレータのアーキテクチャは図 4.1 の通りである。

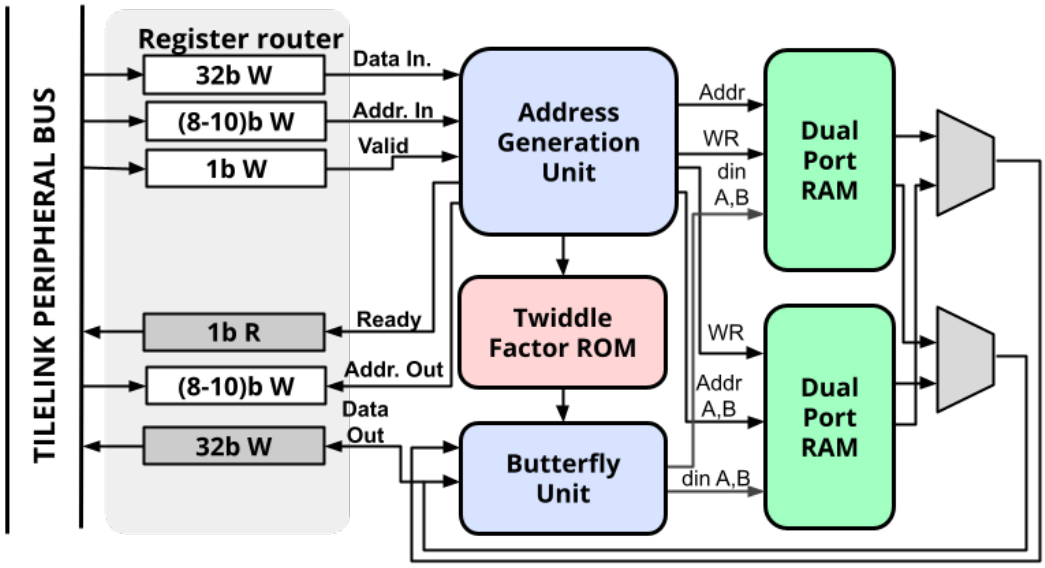

図 4.1: 提案 FFT アクセラレータのブロック図

この FFT は、2 つのサンプルに対して 1 つのバタフライユニット (Butterfly Unit, BFU) を使用して、合計および乗算演算を実行する。

デュアルポート RAM には、信号の実数成分と虚数成分、およびその計算されたスペクトルが含まれる。

データは、アドレス発生器 (Address Generation Unit, AGU)[14] からのスクランブルされたアドレスに従って送信される。このユニットは、現在のイテレーションとサンプルに基づく計算を実行することによってサンプルの順序を維持するのロジックを提供する。同様のロジックが Twiddle Factor ROM にも使用され、FFT アルゴリズムに従ってバタフライユニットの回転係数を提供する。

最後に、メモリルーティングロジックは、上書きを回避するために、両方のメモリ間で読み取りと書き込みをインターリーブする。

4.2　機器構成

本提案手法を実装するために用いた機器等を本節で紹介する。

4.2.1　ハードウェア関連

FPGA

ハードウェア化のために ARROW[15] と呼ばれる開発キット (Development kit) を用いた (図 4.2)。

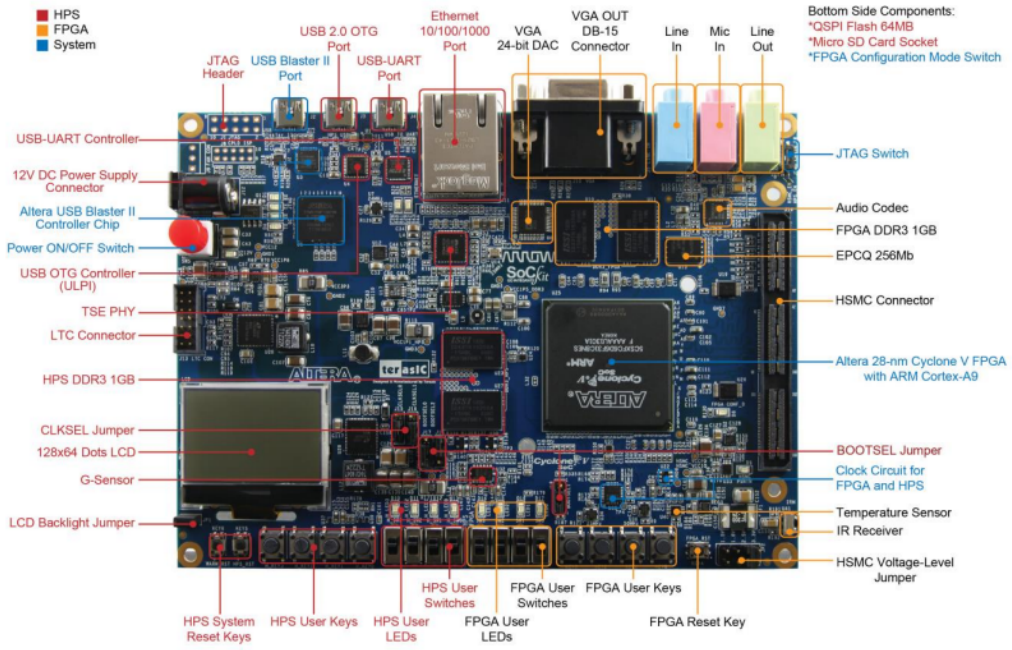

図 4.2: 評価ボード

本研究で用いた部分の構成は以下の通りである。

表 4.1: SoCKit の構成

ボード	SoCKit(ARROW 社)
FPGA	Cyclone V SoC 5CSXFC6D6F31(ALTERA 社)
Audio CODEC	SSM 2603

統合開発環境

統合開発環境には”Quartus Prime Ver.18.1”(Intel 社) を用いた。

ハードウェア記述言語 (Hardware Describe Language, HDL) による記述，論理合成，シミュレーション，実装などをサポートしているので、本研究ではVerilogファイルをQuartusによって論理合成、配置配線し、FPGAのRAMに書き込めるファイル生成などを行える。

RISC-V 環境構築

RISC-Vプロセッサ：Rocket Coreの実装はRocket Chip Generator[16]を用いた。Rocketはほぼすべての実装部分が、Chiselという Scalaをベースにした DSL（Design Specific Language）で記述されていた。

4.2.2 ソフトウェア関連

Python、C 言語、C++

本研究のソフトウェアシミュレーションではPythonを、ソフトウェア実装ではC言語とC++を用いていた。

Pythonは非常にシンプルさと分かりやすさに重きを置く言語であり、音声信号処理に関するアルゴリズムのライブラリが豊富である。開発時間を短縮できるので、アルゴリズムの効果確認には最適だと考える。

ボードで実行できるプログラミング言語が限られているため、本研究ではPythonのライブラリでシミュレーションしたアルゴリズムをC言語とC++で再プログラミングした。

第5章　実装結果

本章では実装を行った際の手順について述べる。

5.1　FFTの実装

5.1.1　２種類FFTのソフトウェア実装

実行速度を比較する為、通常のdouble型に基づいたFFTと提案する整数化FFTをC言語で実装した。整数化FFTは16-bit short integer型を用いて、-1.0から+1.0までの浮動小数点を-32768から+32767までの整数で表示させた。

第4.2.1節の開発キットへに実装を行った。実行した結果、読み込んだ1秒の音声ファイルに対し、図5.1の結果が出った。

図 5.1: ソフトウェアFFT実行結果

実行時間は表5.1の通りである。

表 5.1: ソフトウェアFFT実行時間 (ms)

従来のFFT	整数化FFT
629	11

表 5.1 から、提案手法の整数化 FFT が従来手法の浮動小数点に基づく FFT より、効率が約 60 倍改善されていることが分かる。これをハードウェアに実装すると、さらに改善することが可能だと考えられる。

5.1.2 整数化 FFT のハードウェア実装

第 4.2.1 節の開発キットへ第 4.1 節提案手法の実装を行った。

FPGA 音声サンプリング

開発キット付属の Audio CODEC から音声信号を取得した。CODEC（SSM, 2603）は、アイ・スクウェア・シー (Inter Integrated Circuit, I2C) を使用してサンプリング周波数を構成し、8、22、48、および 96kHz でデータをサンプリングできる。Audio コントローラのインターフェースは図 5.2 の通りである。

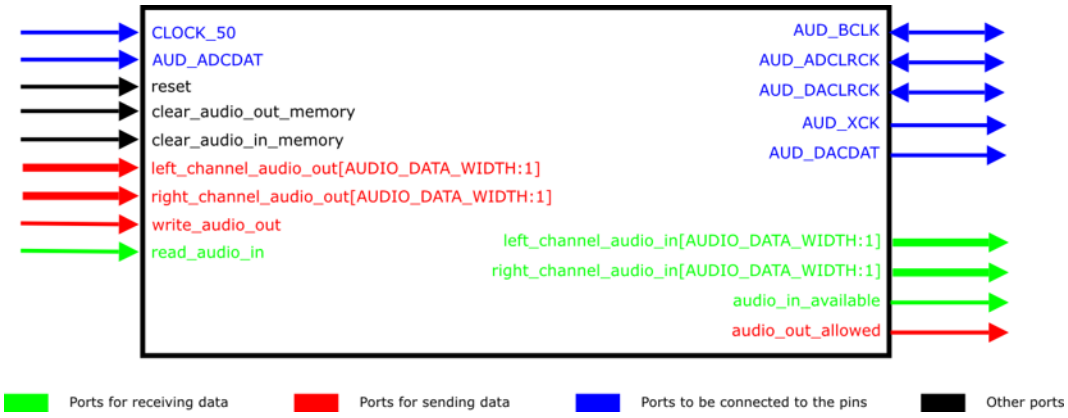

図 5.2: Audio コントローラインターフェース

第 4.2.1 節の Quartus を用いてシミュレーションの結果は図 5.3 の感じだった。

audio_in_avaliable が 1 且つ read_audio_in が 1 の場合、左チャンネル left_channel_audio_in と右チャンネル right_channel_audio_in のデュアルチャンネル音声デジタル信号を取得できる。

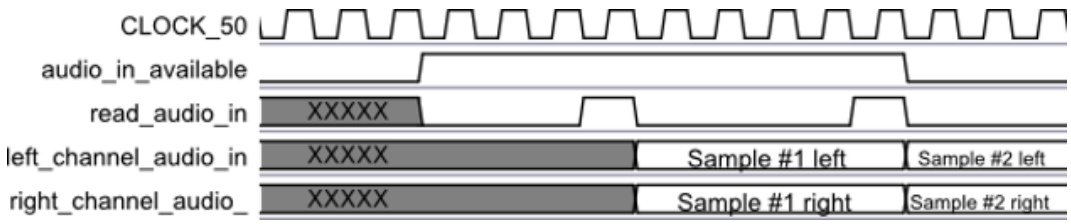

図 5.3: 音声信号取得シミュレーション結果

FFT アクセラレータ実装

ハードウェア記述言語で第 4.1 節提案手法を評価ボードに実装した。取得した音声データを異なる長さにサンプリング、比較対象と提案ハードウェアアクセラレータによる FFT を行い、結果は以下のようになった。

表 5.2: FFT 実行時間 (ms)

サンプル数	従来の FFT	整数化 FFT	HW 化 FFT
1024	300514	24306	103
512	136840	10226	47
256	56682	3779	23

表 5.2 から、従来手法だけでなく整数化 FFT ソフトウェアよりもさらに改善されていることが分かる。

5.2 FFT を MFCC への実装

5.2.1 MFCC 抽出のフロー

今回提案する FFT アクセラレータを用いた MFCC 抽出フローは以下の通りである。

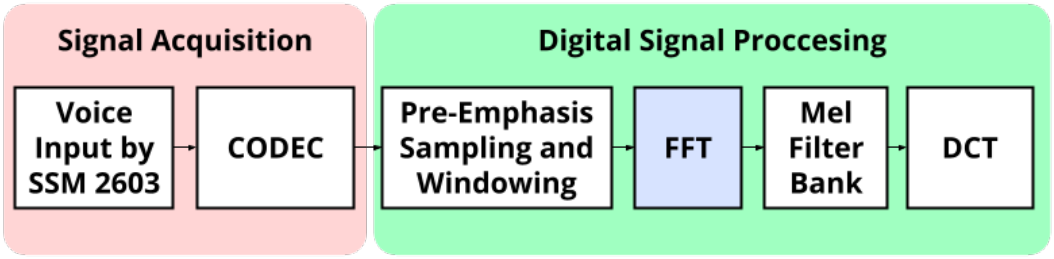

図 5.4: 提案 MFCC フロー

図 4.2 に、赤でハイライトされている *Signal Acquisition* は、設定したサンプル周波数で音声データをサンプリングする。また、緑でハイライトされている *Digital Signal Processing* がサンプルを処理して、メル係数を取得する。

まず、*Pre-Emphasis* ステージがデータを高域強調を行い、窓関数でフレーム化し、スペクトル漏れを防ぐ。*FFT* は、データを周波数領域に変換する。そして、*Mel Filter Bank* が人間の耳の動作に一致するように、低周波数帯域で十分なエネルギー情報を抽出する。最後に、*DCT* はメル係数を取得し、それらを時領域に変換し直す。

5.2.2 SoC アーキテクチャ設計

第 5.2.1 で説明されている MFCC フローは、RISC-V 環境を使用して実装された。図 5.5 は、実装された SoC のブロック図を示している。

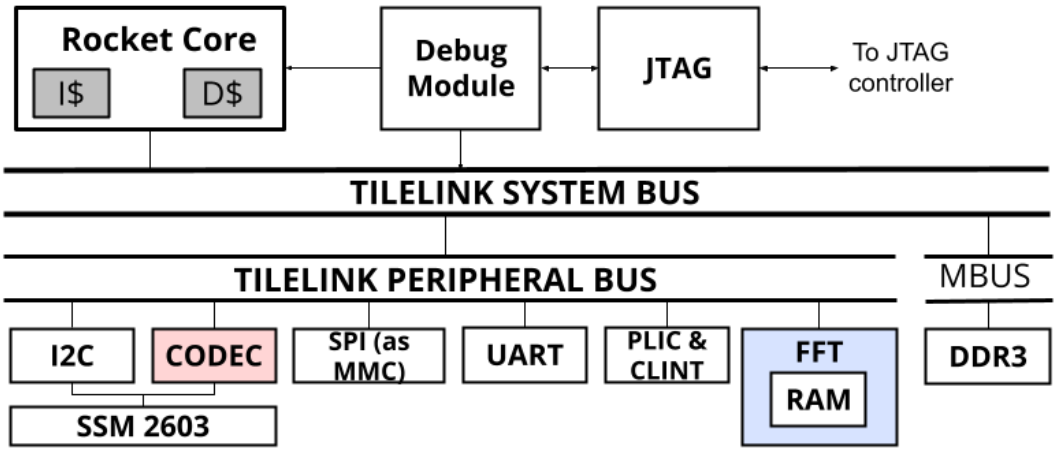

図 5.5: SoC アーキテクチャ

SoC は、16KB の命令とデータキャッシュを備えたロケットコア、デバッグモジュール、外部メモリ用の SPI、UART、システム割り込み、256MB のダイナミックランダムアクセス（DDR）メモリ、コーダー/デコーダー（CODEC）および FFT アクセラレーターで構成 されている。

そして、システムバスおよびペリフェラルバスに TileLink を使用した [17]。

CODEC 内のアナログ-デジタルコンバーターは、16 ビット符号なしでサンプリングされたデータをデジタル信号処理ステージに送信する。

測定環境は、図 5.6 のようになっている。

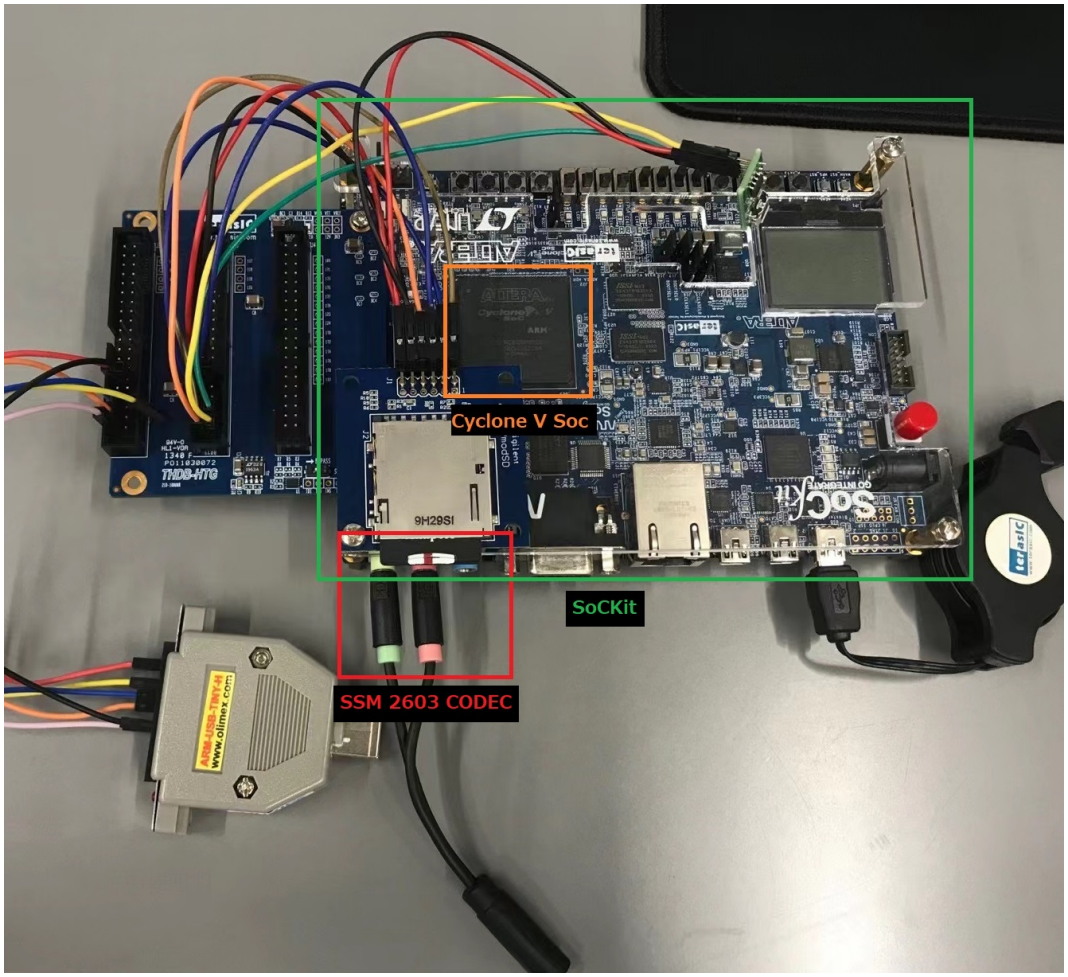

図 5.6: 評価ボード

5.2.3 結果

表 5.3 は、FPGA での SoC 実装のリソースを示している。

マイクロコントローラは、第 4.2.1 節で紹介した ALTERA Cyclone V 5CSXFC6D6F31FPGA に実装されている。

表 5.3: 各モジュールの FPGA ハードウェア使用率

	ALUTs	**FFs**	**MEM BITS**
Rocket Core	7775	5034	68608
Debug Module	829	843	0
I2C	189	135	0
CODEC	151	193	16348
SPI	318	222	128
UART	155	140	128
ROM	120	180	0
DDR3	7838	7967	237084
TilelInk Bus	2643	1188	384
FFT 1024	2182	244	65536
FFT 512	2012	244	32768
FFT 256	1987	244	16384

マイクロプロセッサは、すべての SoC の 19 % を占めた。

DDR3 コントローラーと FFT アクセラレーターは、それぞれ 59 % と 16 % のリソースの大部分を占めた。FFT は論理合成時に 1024、512、および 256 ビットの 3 つの異なる長さに構成できる。

RISC-V プロセッサと比較した FFT アクセラレータのオーバーヘッドは、それぞれ 16 %、57 %、および 77 % と少なった。TileLink バスは、SoC の 1 % 未満を占めた。

表 5.4 は、実装された MFCC の実行時間を示している。

表 5.4: クロックサイクルでの MFCC 実行パフォーマンス

	Execution Time @50 MHz @48 KSPS				
MCycles [ms]	**Filter**	**Hann Wind.**	**FFT 256**		**Mel coefficient Extraction**
			HW	**SW**	
32768 Samples	675.4 13.5	3127.2 62.5	326.1 6.5	54056.8 1081.1	212567.1 4251.3
65536 Samples	1363.4 25.2	6871.9 137.4	654.2 13.1	105750.6 2115.0	468336.4 9766.8
131072 Samples	2701.5 54.0	12267.4 245.3	1306.6 26.1	240768.4 4815.3	934005.8 18680.1

本実験では、比較のため異なる長さのサンプリングデータを使用した。

このシステムには、フィルタリング、窓関数処理、FFT、およびメル係数抽出の処理が含まれている。 メル係数抽出以外は、固定小数点数のみを使用してソフトウェアまたはハードウェアに実装された。256 サンプルの FFT アクセラレータのパフォーマンスはソフトウェアの整数化 FFT の 160 倍になった。 FFT アクセラレータは、全体的な MFCC

の実装を 20.7 % パフォーマンスを向上させた。

実装されたプロセッサは整数演算のみをサポートする一方で、メル係数抽出には浮動小数点演算が必要である。よって、提案手法の FFT アクセラレータ実装後、主要なオーバーヘッドはメル係数抽出に表示されている。コストパフォーマンスの面ではあまり良くないため、今後の改善の余地がある。

第6章　結論

6.1　結論

本研究では、RISC-V ベースのマイクロコントローラを使用し、以下 FPGA に実装される FFT アクセラレータを提案しする。アクセラレータの原理として、通常の FFT アルゴリズムで行う浮動小数点演算を整数に置き換え、ハードウェア回路に実装することで処理速度の向上が期待できる。RISC-V Rocket のマイクロコントローラを評価ボード FPGA に実装し、長さが異なる 3 種類の音声サンプルデータに対してアナログ・デジタル変換 (ADC)、フィルタリング、窓関数処理、FFT、およびメル特徴抽出の流れで処理を行った。そして、FFT の処理において、浮動小数点演算 FFT 処理と整数演算 FFT 処理を実装したハードウェア FFT で実験を行い、実行時間とハードウェアリソース使用率を比較した。結果として、整数演算 FFT アクセラレータの実装により、処理速度が 160 倍以上に向上した。MFCC 全体的な抽出パフォーマンスが 20.7 %向上した。ハードウェアリソース使用率として、マイクロプロセッサは System on Chip (SoC) の 19 %を占め、DDR3 コントローラーと FFT アクセラレータは、それぞれ 59 %と 16 %のリソースを占めた。FFT1024、FFT512、FFT256 は RISC-V プロセッサのハードウェアリソース使用率よりそれぞれ 16%、57%、77% と少なった。

浮動小数点演算を整数演算に置き換え、そしてハードウェア化によって効率が大幅に向上することができた。

6.2　今後の展望

実装されたプロセッサは整数演算のみをサポートする一方で、メル係数抽出には浮動小数点演算が必要である。よって、提案手法の FFT アクセラレータ実装後、主要なオーバーヘッドはメル係数抽出に表示されている。今後の展望として、メル特徴抽出処理の整数演算の置き換え及びハードウェア化の課題は将来の最適化方向になると期待できる。

参考文献

[1] I. Lopez-Espejo ´ et al., “ Deep Spoken Keyword Spotting: An Overview, ” IEEE Access, pp. 1–1, 2021.

[2] 中川聖一. 音声認識研究の動向 [J]. 電子情報通信学会論文誌 D, 2000, 83(2): 433-457.

[3] M. Hoy, “ Alexa, Siri, Cortana, and more: An introduction to voiceassistants, ” Medical Reference Services Quarterly, vol. 37, pp. 81–88,01 2018.

[4] H. Kou et al., “Optimized MFCC feature extraction on GPU,” in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 7130–7134.

[5] A. Jain et al., “ Evaluation of MFCC for speaker verification on various windows, ” in International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014). IEEE, may 2014. [Online].
<https://doi.org/10.1109$%$2Ficraie.2014.6909144>

[6] A. Waterman et al., “ The RISC-V Instruction Set Manual, Volume I: User Level ISA, Version 2.0, ” EECS Department, University of California,Berkeley, Tech. Rep. UCB/EECS-2014-54, May 2014.

[7] 青木直史, ゼロからはじめる音響学（講談社, 東京, 2014）.

[8] 青木直史. はじめての音声信号処理とサウンドプログラミング [J]. 日本音響学会誌, 2017, 73(4): 230-238.

[9] 堀部貴紀, 石原達馬, 白井暁彦, 等. 『転声こえうらない』 利用者の基本周波数分析 [J]. 研究報告音声言語情報処理 (SLP), 2020, 2020(18): 1-6.

[10] 鈴木雅之, 朝川智, 喬宇, 等. スペクトル領域特徴量を用いた音声の構造的表象に関する実験的考察 [J]. 電子情報通信学会技術研究報告. SP, 音声, 2008, 108(116): 73-78.

[11] 音響特徴量「メルスペクトル」と「MFCC（メル周波数ケプストラム係数）」の解説と実例紹介 [Online].
<https://fast-d.hmcom.co.jp/blog/melspectrum-mfcc/>

[12] バンドパスフィルタ [Online].
<https://aidiary.hatenablog.com/entry/20111030/1319895630>

[13] Slade G W. The fast fourier transform in hardware: A tutorial based on an FPGA implementation[J]. Mar, 2013, 21: 1-26.

[14] "There are three Address Generation Units (AGUs) for all load and store address generation." AMD. (2020). Software Optimization Guide for AMD EPYC™ 7003 Processors. rev. 3.00.

[15] SoCKit Development Kit Product Description. [Online].
<https://www.arrow.com/en/products/sockit/arrow-development-tools>

[16] RISC-V Foundation, " Rocket Chip Generator, " 2019. [Online].
<https://github.com/chipsalliance/rocket-chip>

[17] SiFive, Inc., " SiFive TileLink Specication, " Aug. 2019. [Online].
<https://www.sifive.com/documentation/tilelink/tilelink-spec/>

発表実績

・Wu Xiaoting, Ckristian Duran and Cong-Kha Pham, “Exploiting the FFT Acceleration for MFCC-Speech Recognition Using a RISC-V Microcontroller ”情報処理学会第 84 回全国大会, 2022.03.03.

謝辞

最後に、本卒業研究を行うにあたり、熱心にご指導くださった電気通信大学の範公可教授に感謝いたします。また、協力や助言を頂きました同室である範研究室所属のそのほかの方々、そして、合同ゼミでお世話になりました石橋孝一郎教授ならびに石橋 (孝) 研究室の皆様に感謝いたします。