Vol.2017-SLP-118 No.7
2017/10/13

組み込みシステム向け日本語テキスト音声合成ソフトウェア

西澤 信行[1,a)]　小原 朋広[1]　菅谷 史昭[1]

概要：KDDI 総合研究所が開発を進めている日本語音声合成ソフトウェア「N2」をベースに，組み込みシステム向け日本語テキスト音声合成ソフトウェアを開発したので報告する．また，ソフトウェアの動作検証用に小型のマイコンボードを試作したので併せて紹介する．これまでに KDDI 総合研究所では，HMM 音声合成における処理の高速化を目的として，HMM 音声合成におけるパラメータ軌跡計算処理の固定小数点演算化や疑似 QMF バンク上での正弦波合成に基づく音声波形生成等に取り組み，既にこれらの成果は現在のバージョンの N2 に反映されている．それにより N2 の基本的な処理が組み込みシステム向けテキスト音声合成ソフトウェアに求められる処理量に抑えられていることを確認したことから，組み込みシステム向けに特化した音声合成システムを作るのではなく，従来の N2 を PC，スマートフォンから組み込みシステムまでスケーラブルに動作するように改修した．音声合成処理のアルゴリズムは従来の N2 と同じであり，従来の PC やスマートフォンで用いている 24.4 万語の形態素辞書データと音響モデルデータを用いて，PC やスマ―フォン上の N2 と同じ合成音声波形をマイコンボード単体で得られることを確認した．

Japanese text-to-speech software for embedded systems

Nobuyuki Nishizawa[1,a)]　Tomohiro Obara[1]　Fumiaki Sugaya[1]

Abstract: This report describes our development of Japanese text-to-speech(TTS) software for embedded systems that is based on our commercial TTS software "N2." In addition, the report also introduces a small microcomputer (microcontroller) board to verify operation of the TTS software. KDDI Research, Inc. has worked on reduction of computational cost of HMM speech synthesis such as parameter trajectory calculation only by fixed-point arithmetic and speech waveform generation based on sinusoidal synthesis on pseudo QMF banks, etc. These efforts have already been reflected in the current version of N2. Since the processing performance required for text-to-speech synthesis software for embedded systems has been achieved by the methods, this development was mainly done by porting of N2 for embedded devices rather than creating a new speech synthesis system specialized for embedded systems. Consequently, N2 has acquired a wide range of scalability from PC to microcontroller. Experimental evaluation with the HMMs and 244k morphological dictionary for text analysis from the current version of N2 showed that the completely same results as the those by PCs and smartphones can be obtained by the microcomputer board alone.

1. はじめに

近年，ウェアラブル機器や小型の IoT(Internet of Things) 機器が普及しつつある．しかし，ウェアラブル機器では情報表示のための機構がしばしば機器形状や大きさの制約となっている．また IoT 機器も，無線タグやセンサデバイスといった超小型のデバイスの多くは，機器単体での情報出力機能が非常に限定的で，特にネットワークとの接続に問題が生じた場合に，その稼働状態を確認することが困難なことは少なくない．

このような場合，音声による情報出力が有効な場合は多いと考えられる．スピーカの追加やそれに伴う消費電力の増加も機器形状の制約とはなるものの，情報出力手段の選択肢が増えることで課題解決の可能性が高まる．また文字表示を行う場合でも，音声出力を併用することでユーザの利便性を高めることができる．さらに，より詳細な状況を音声で伝えるためには，任意文章の音声を出力できる，音声合成技術が不可欠である．

[1] 株式会社 KDDI 総合研究所
KDDI Research, Inc., Japan
[a)] no-nishizawa@kddi-research.jp

KDDI 総合研究所ではこれまで，高速処理や小フットプリントを特徴とする，日本語テキスト音声合成ソフトウェア「N2」[1] の開発を進めてきたが，この N2 をベースに，組み込みシステム向け日本語テキスト音声合成ソフトウェアを開発した．またソフトウェアの動作検証用に新規に開発した小型マイコン (micro controller unit, MCU) ボード上で，実際にソフトウェアの動作を確認したので報告する．併せて，この実現に不可欠であった HMM 音声合成のための処理量削減手法として，パラメータ軌跡計算における固定小数点演算化と，疑似 QMF バンク上での正弦波重畳による音声合成波形生成についてそれぞれ紹介する．

2. HMM 音声合成方式による組み込みシステム向け日本語テキスト音声合成

ウェアラブル機器や小型 IoT デバイスの多くは電池駆動であり，その上で音声合成処理を行う場合，小さいフットプリントと低消費電力動作の双方が求められる．このうち前者は，HMM 音声合成 [2] 方式により高い合成音声品質で実現できる．HMM 音声合成では, 音響的特徴に影響する要素を全て記述した音素ラベル情報 (フルコンテキストラベルと呼ばれ，音韻的な情報だけでなく韻律に関する情報も含む) から，決定木で対応する音素 HMM のパラメータが決まるように音素 HMM を学習することで，全体で数百キロバイトから数メガバイトのデータで任意の音声を合成できる．しかし，消費電力に大きく影響する要因である処理量の観点では，HMM 音声合成は必ずしも有利ではない．例えば，事前収集した音声波形の断片 (素片) を波形レベルで繋ぎ合わせて音声を合成する方式 (波形接続方式) の方が，合成音声品質をあまり考えなければ少ない処理量で音声を合成できる．

波形接続方式では PSOLA(pitch-synchronous overlap and add) 法 [3] による素片の基本周波数や継続長の変形が可能なものの，その変形を大きくすると音声品質が低下するため，大きな変形を避けるために多数の素片を事前準備しておく構成が用いられることが多く，その場合，1 発話分の滑らかな音声の合成に最適な素片列を探索処理 (素片選択処理) の処理量が非常に大きくなることはよく知られている [4], [5]．しかし仮に高い自然性を求めず，PSOLA 等による大きな波形変形を許容して素片選択に必要な処理量を大幅に削減する構成とすれば，音声変形に TD(time-domain)-PSOLA のような低コストな手法を用いた場合で，出力 1 サンプル当たりで数回程度の積和演算で音声波形生成処理を実現できる．これに対して HMM 音声合成では, 音響特徴量からそれに対応する音声波形を信号処理的に生成する必要があり，この処理には, インパルス列や白色雑音源といった白色の音源波形に対して音声のスペクトル特徴を生成するフィルタを適用する方法が一般に用いられる．例えば, メルケプストラムに基づく HMM 音声合成システムでは, メル対数スペクトル近似 (MLSA) フィルタ [6], [7] が広く用いられている．しかし, 音声のスペクトルの山や谷を表現するためには高次のフィルタが必要で, そのようなフィルタは通常, 出力 1 サンプル当たり数十回から数百回程度の積和演算を必要とする．

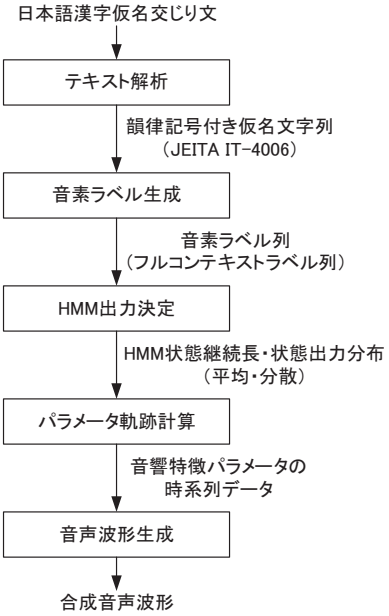

図 1 テキスト音声合成処理の流れ
Fig. 1 Flow of processing of our text-to-speech system.

そこで我々は，小フットプリント，高い音声合成品質，低消費電力のすべてを満たす，日本語テキスト音声合成ソフトウェアの開発を目指し，HMM 音声合成方式の高速化に取り組んでいる．

また音声合成の応用では, テキスト音声合成 (Text-to-speech, TTS) システムが必要な場合が多く，その場合，同じハードウェア上でテキスト解析処理を行う必要がある．このため，信号処理に特化したハードウェア構成より，汎用のプロセッサのみで高性能を達成できた方が広範な機器での利用が可能となりるので望ましい．後述する処理量削減手法も，汎用プロセッサ上での実現を前提としている．

3. HMM 音声合成の処理量削減

我々が開発した HMM 音声合成方式に基づく日本語テキスト音声合成ソフトウェアにおける一連の処理の流れを図 1 に示す．KDDI 総合研究所における HMM 音声合成ベースの音声合成ソフトウェア開発では，当初，波形生成処理に MLSA フィルタを用いていたが，当時 (2007 年頃) のターゲットである携帯電話端末には浮動小数点演算コプロセッサがほとんど搭載されておらず，ソフトウェアによる浮動小数点演算処理が極端に高コストであったことから，MLSA フィルタの固定小数点演算化をまず行っている．さらに，パラメータ軌跡計算の固定小数点演算化も行って，システム全体の固定小数点演算化を達成した．実際，N2

の最初のリリース (2011 年) 時点では MLSA フィルタベースの構成である．その後，波形生成処理を後述する現在の方法に変更している．

2017 年現在では，多くは単精度 (32 ビット浮動小数点数) のみのサポートだが，組み込み向けプロセッサでも FPU を備えた品種が増えており，固定小数点演算化の必要性は以前ほど高くない．しかし，後述する手法の改良の結果，固定小数点演算が容易になったことから，現在もソースコードレベルでは浮動小数点演算を完全に排除している．

以下では，パラメータ軌跡計算処理と音声波形生成処理で用いている処理量削減手法をそれぞれ説明する．

3.1 パラメータ軌跡計算の固定小数点演算化 [8]

ソフトウェアのみで浮動小数点演算を行うのは一般に高コストである．そこで，計算処理において，実数 x を整数 $\lfloor x \times 2^q \rfloor$ に近似的に変換してから整数演算を行うことを繰り返すことで，実数演算を近似的に行うことを考える．ここで，q は元の値の小数点以下を表すためのビット数である．また，$\lfloor x \rfloor$ は x 以下の最大の整数の値を表す． 値を 2^q 倍する処理は，実際には，q が正のときは左ビットシフト演算で，q が負の時は算術右シフト演算によりそれぞれ行う．パラメータ軌跡計算の固定小数点演算化では一連の処理で q の値を固定しており，一連の計算の入力を，最初にスケーリング (定数倍) しておくことで，丸め誤差と計算途中のオーバフロー回避を制御している．

HMM 音声合成では，フレーム t における音声の特徴ベクトルを $\boldsymbol{c}(t)$，また 1 発話全体のフレーム数を T とするとき，特徴ベクトルの時系列 $\boldsymbol{c} = [\boldsymbol{c}(1)^\top, \boldsymbol{c}(2)^\top, ..., \boldsymbol{c}(T)^\top]^\top$ により音声を合成する．しかし，$\boldsymbol{c}$ を直接 HMM でモデル化すると，HMM の状態が切り替わる箇所で対応する $\boldsymbol{c}(t)$ が大きく変化する．そこで滑らかな系列を得るために，$\boldsymbol{c}(t)$ の変化に相当する特徴も同時に HMM でモデル化する [2], [9]．以下，HMM の出力 $\boldsymbol{o}_t$ は，$\boldsymbol{c}(t)$ に加え，$\boldsymbol{c}(t)$ の 1 階差分 $\Delta\boldsymbol{c}(t)$ と 2 階差分 $\Delta^2\boldsymbol{c}(t)$ により，

$$\boldsymbol{o}(t) = [\boldsymbol{c}(t)^\top, \Delta\boldsymbol{c}(t)^\top, \Delta^2\boldsymbol{c}(t)^\top]^\top \tag{1}$$

$$\Delta\boldsymbol{c}(t) = -\frac{1}{2}\boldsymbol{c}(t-1) + \frac{1}{2}\boldsymbol{c}(t+1) \tag{2}$$

$$\Delta^2\boldsymbol{c}(t) = \boldsymbol{c}(t-1) - 2\,\boldsymbol{c}(t) + \boldsymbol{c}(t+1) \tag{3}$$

とする．HMM の各状態では $\boldsymbol{o}(t)$ の分布がモデル化されており，HMM 出力全体の尤度が最大となる系列を求めることでパラメータ軌跡は生成される．また，$\boldsymbol{o}(t)$ の各次元の要素は，HMM の各状態に結び付けられたそれぞれ独立な正規分布でモデル化されているとする．文献 [9] では特徴パラメータの次元間の相関を考慮した定式化が行われているが，組み込みシステムに限らず，特徴パラメータ次元間の独立性を常に仮定した方法は，他のシステム (例えば [10]) でも用いられている．

このとき，$\boldsymbol{c}$ の特定の次元に注目すると，処理における一連のパラメータは，

$$\boldsymbol{c} = [c(1), c(2), \ldots, c(T)]^\top \tag{4}$$

$$\boldsymbol{o} = [c(1), \Delta c(1), \Delta^2 c(1), c(2), \Delta c(2), \Delta^2 c(2), \ldots, c(T), \Delta c(T), \Delta^2 c(T)]^\top \tag{5}$$

$$\boldsymbol{\mu} = [\mu_0(1), \mu_1(1), \mu_2(1), \mu_0(2), \mu_1(2), \mu_2(2), \ldots, \mu_0(T), \mu_1(T), \mu_2(T)]^\top \tag{6}$$

$$\Sigma = \mathrm{diag}(\sigma_0^2(1), \sigma_1^2(1), \sigma_2^2(1), \sigma_0^2(2), \sigma_1^2(2), \sigma_2^2(2), \ldots, \sigma_0^2(T), \sigma_1^2(T), \sigma_2^2(T)) \tag{7}$$

と表すことができる．ただし，μ_0, μ_1, μ_2 および σ_0^2, σ_1^2, σ_2^2 はそれぞれ c, Δc, $\Delta^2 c$ の平均および分散である．

以下，$\boldsymbol{c}$ から $\boldsymbol{o}$ への $3T \times T$ の変換行列を W とする．つまり $\boldsymbol{o} = W\boldsymbol{c}$ とする．パラメータ系列の推定値 $\hat{\boldsymbol{c}}$ は

$$\hat{\boldsymbol{c}} = \operatorname*{argmax}_{\boldsymbol{c}} P(W\boldsymbol{c}|\boldsymbol{\mu}, \Sigma) \tag{8}$$

で得られるが，このとき $\hat{\boldsymbol{c}}$ は，

$$\frac{\partial}{\partial \boldsymbol{c}} P(W\boldsymbol{c}|\boldsymbol{\mu}, \Sigma) = 0 \tag{9}$$

を満たす．すなわち $\hat{\boldsymbol{c}}$ は方程式

$$W^\top \Sigma^{-1} W \hat{\boldsymbol{c}} = W^\top \Sigma^{-1} \boldsymbol{\mu} \tag{10}$$

を解くことで求まる．実際には $W^\top\Sigma^{-1}W$ はバンド幅 2 の対称帯行列で，実際のソフトウェアでは修正コレスキー分解により下三角行列，対角行列，上三角行列の積に分解した結果を用いて $\hat{\boldsymbol{c}}$ を求めている．

しかし，これを直接固定小数点演算化すると，特に F0 軌跡においては男女差の影響等から，HMM セットごとに入力のスケーリングを適切に設定しないと，計算誤差が可聴な水準で大きくなることがある．そこで値の取り得る範囲を抑えるために，上記の改良版として，適当な系列 $\boldsymbol{c}_0$ を予め設定し，$\boldsymbol{c}_0$ からの差分を計算することで，最終的に $\hat{\boldsymbol{c}}$ を求めることを考える．以下では，

$$\hat{\boldsymbol{c}} = \boldsymbol{c}_0 + \hat{\boldsymbol{c}}_1 \tag{11}$$

とする．このとき，式 (10) は，

$$W^\top \Sigma^{-1} W \hat{\boldsymbol{c}}_1 = W^\top \Sigma^{-1} \boldsymbol{\mu} - W^\top \Sigma^{-1} W \boldsymbol{c}_0 \tag{12}$$

と変形できる．右辺は全て定数なので式 (10) と同様に $\hat{\boldsymbol{c}}_1$ は計算できる．

$\boldsymbol{c}_0$ については，これを HMM の出力分布平均の系列 $[\mu_0(1), \mu_0(2), \ldots, \mu_0(T)]^\top$ とすることで，入力のスケーリング設定に対する誤差の感度を抑えることができ，スケーリング係数を用いる HMM によらず一定にしても誤差が小さいことを実験で確認している．

3.2 疑似 QMF バンク上での正弦波重畳ベース音声波形生成 [11]

HMM 音声合成では音声の特徴を音源特性 (基本周波数等) とスペクトル包絡特性とに分離する，ソースフィルタモデルに基づくモデル化と，それに直接対応した音声波形生成手法が用いられることが多い．我々の音声合成システムも音声のモデル化は従来同様のソースフィルタモデルに基いているが，音声波形生成処理では，疑似 (pseudo) QMF バンクによるサブバンド符号上での仮想的な正弦波重畳を用いている．なお，後処理としてのスペクトル強調処理が容易なことから，スペクトル包絡特性の表現には，従来と同様のメルケプストラム [7] を用いている．

以下，説明のためにまずは分割数 M の等帯域分割最大間引きフィルタバンクに基づくサブバンド符号化システムの構成を図 2 に示す. 符号化の際は, まずバンドパスフィルタ (BPF) により入力信号の帯域がそれぞれ制限され, 入力の $1/M$ のサンプリング周波数にダウンサンプリングされる. 現実には BPF の遷移域の周波数幅を 0 にすることはできないため, ダウンサンプリングとアップサンプリングに伴う折り返し雑音の考慮が必要だが, 生じた折り返し雑音を他のサブバンドで生じた折り返し雑音と互いに打ち消し合うような構造を有したフィルタバンクとして, 疑似直交鏡像フィルタ (Quadrature Mirror Filter, QMF) バンク [12] がある. フィルタバンクの k 番目のサブバンドの分析フィルタ・合成フィルタの伝達特性 $H_k(z)$, $F_k(z)$ は, 後述するプロトタイプフィルタの伝達特性を $H(z)$ とするとき,

$$H_k(z) = e^{j\theta_k} W_{2M}^{(k+1/2)\frac{N}{2}} H(zW_{2M}^{k+1/2}) + e^{-j\theta_k} W_{2M}^{-(k+1/2)\frac{N}{2}} H(zW_{2M}^{-(k+1/2)}) \quad (13)$$

$$F_k(z) = z^{-N} H_k(z^{-1}) \quad (14)$$

で与えられる. ただし θ_k は次式を満たす必要がある.

$$\theta_k = (-1)^k \pi/4 \quad (15)$$

プロトタイプフィルタは，フィルタバンクの基になるフィルタで，通常，遮断角周波数を π/M とする有限インパルス応答 (FIR) のローパスフィルタとして設計する．プロトタイプフィルタが FIR のとき，フィルタバンクの各フィルタのインパルス応答は，プロトタイプフィルタのインパルス応答をコサインで変調した形となる．このコサイン変調を高速離散コサイン変換 (DCT[13], [14]) により実行時に行うことで，高速なフィルタバンク処理を実現できる．

疑似 QMF バンクに基づくサブバンド符号化は MPEG Audio[15] で用いられており, MPEG Audio で用いられている 32 帯域分割のフィルタバンク設計をそのまま用いることもできる．フィルタ長は 512 タップで, プロトタイプフィルタのインパルス応答および周波数・振幅特性を図 3 のようになる.

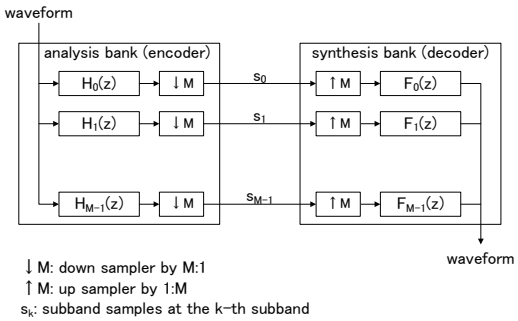

図 2 サブバンド符号化システムの構成図

Fig. 2 Block diagram of the subband coding system.

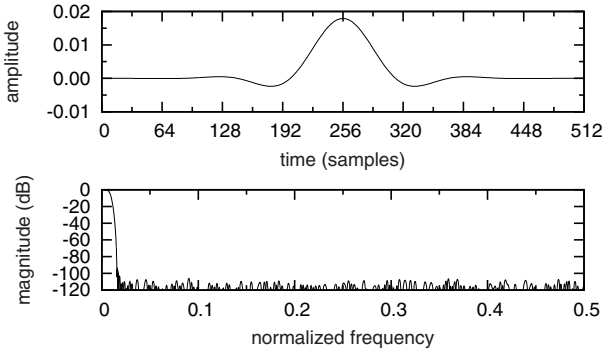

図 3 32 バンド疑似 QMF バンクのためのプロトタイプフィルタのインパルス応答および周波数-振幅特性

Fig. 3 Impulse response and frequency magnitude response of the prototype filter for a 32-band pseudo QMF bank.

先述のように, 隣接するサブバンドとの間以外では, BPF の遮断域で折り返し雑音を遮断する形となっているため, 仮にある周波数の定常な正弦波をサブバンド符号化すると，サブバンド符号 $\boldsymbol{s} = [s_0, s_1, \ldots, s_{M-1}]^\top$ は，最大でも隣接する 2 要素だけに 0 以外の値を持つスパースなベクトルになることが分かる．このベクトルは実際に分析バンクを使わなくても容易に計算可能であり，正弦波の振幅変更に必要な乗算回数も時間領域で行う処理の 2/M 倍に削減できる.

正弦波の重畳による周期波形生成は次式で表すことができる.

$$x(t) = \sum_j R(t, j \times \omega_0(t)) \cos(j \times \omega_0(t) \times (t - t_0)) \quad (16)$$

ここで, ω_0, $R(t,\omega)$，t_0 はそれぞれ角周波数表示の基本周波数, 時刻 t の角周波数 ω における振幅スペクトル特性, 位相が 0 となる基準の時刻である. $R(t,\omega)$ はメルケプストラムから計算する．この処理と等価な処理をサブバンド符号上で行うことで処理量を削減する. この処理量削減の効果が非常に大きく，サブバンド符号の復号処理を行ってもなお処理量的に有利であることが，本手法を用いる理由となっている.

まず，定常な正弦波は周波数領域では線スペクトルとなり, 正弦波をサブバンド符号化した結果の符号ベクトルは

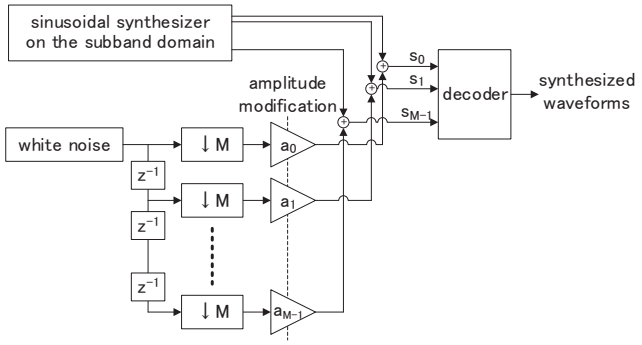

図 4 サブバンド符号上での処理による音声波形生成処理の構成
Fig. 4 Block diagram of speech waveform generation on the subband coding.

計算で直接求めることができる．具体的には, サブバンド符号化処理 (図 2 の左半分) における分析バンクの m 番目のフィルタ周波数-振幅特性を $H_k(\omega)$ とするとき, 角周波数を ω_j のコサイン波形をサブバンド符号化した結果のベクトルの k 番目の要素は,

$$s_{k,\omega_j}(t) = H_k(\omega_j)R(t,\omega_j)\cos(\omega_j(t-t_0)+\theta_k) \quad (17)$$

となる．$H_k(\omega)$ は音声合成時に直接計算すると高コストなため，事前に作成したテーブルを使って求める．そして，プロトタイプフィルタが FIR のとき，サブバンド符号の復号は線形演算だけで構成されることから，この方法ではサブバンド符号ベクトルに対して復号前に線形演算を行うことができる．つまり式 (16) はサブバンド符号上の演算で実現できる．また，これまでの議論では正弦波の定常性を仮定していたが，最終的には，プロトタイプフィルタのインパルス応答 $h(i)$ を窓関数とする M サンプルシフトの overlap-and-add となるため，特徴の変化が激しくなければこの仮定の影響は小さいと考えられる．

さらに実際のソフトウェアでは，正弦波の重ね合わせで合成される周期波形成分と，雑音成分とを別々に生成したうえで，サブバンド符号領域でベクトルを足し合わせてから復号する方法を用いている．サブバンド符号のベクトルの各要素にそれぞれ無相関の白色雑音を入力すれば，合成バンク側のアップサンプラで白色のまま帯域拡張され，それが BPF で帯域制限される形になるので，各サブバンドの周波数帯の雑音パワーと合うようにサブバンド符号ベクトルの各要素の振幅を調整することで合成音声の雑音成分を合成できる．つまり雑音成分に対しても符号ベクトルの要素を直接決めることができ，図 2 における左側の符号化部は実際には不要である．そのような処理の構成を図 4 に示す.

なお，波形生成処理全体で見ると，各フレームにおけるスペクトル包絡の計算，正弦波重畳処理，サブバンド符号の復号処理も必要で，それらの合計と従来のフィルタの処理量とを比較することになるが，例えば Linux ベースのマイコンシステム (ARM9E 400MHz コア) 上における MLSA フィルタ (5 次近似) ベースのシステムとの比較では，1/4 以下の処理時間となることを実験的に確認している [16]. さらに近年の HMM 音声合成では自然性改善等を目的に，インパルス列と雑音を混合した波形を MLSA フィルタや AR フィルタの入力とする混合励振が行われることが多いが，混合励振における雑音成分制御にはフィルタバンクが用いられることが多いが，その処理量も実際には小さくないため，追加の処理なしに混合励振と同様の結果が得られる本方法はさらに有利となる．

また AR フィルタや MLSA フィルタとは異なり，波形生成処理にフィードバックが全くないことから，各要素の値の範囲は明らかで，固定小数点演算化が容易である．実際に固定小数点演算化されていて，N2 は整数演算命令のみで高速に動作する．

4. 動作検証用マイコンボードの構成

本稿において，マイコン (MCU) とは，汎用のプロセッサコアを中心に，単体でコンピュータシステムとして動作させるために周辺回路まで含む形で構成された集積回路のことをいう．特に，組み込み用途をターゲットに，低消費電力のプロセッサコアを用いたものを MCU と呼ぶことが多い．本開発における MCU ベースの動作検証用マイコンボードの外観を図 5 に，またその仕様を表 1 に示す．本節で説明するマイコンボードは組み込みシステム向け日本語テキスト音声合成ソフトウェアの動作検証用に開発・製作したもので，以下ではその特徴等について説明する．

4.1 MCU

動作検証用マイコンボードでは，その処理性能やデバッグ機能の充実等から，ARM 社の Cortex-M4F コアを用いた MCU を用いた．Cortex-M4F ベースのマイコンは複数のメーカから多数の品種が提供されているが，今回は最初の検証であることから，(1) 内蔵 RAM 容量が大きい，(2) CPU コアの動作周波数が高い，(3) 外部フラッシュメモリと外部 RAM を同時に接続できる，といった基準を満たす中で，できるだけ高性能な MCU を選定した．

Cortex-M4F は 32 ビットの浮動小数点演算コプロセッサ (FPU) を備えているが，今回のベースとなる音声合成ソフトウェアは整数演算のみで実装されており，現在，FPU を積極的に用いていない．ただし，コンパイラによる最適化の結果として，整数除算等では FPU を用いた処理になることがある．

4.2 外部フラッシュメモリ

本稿の執筆時点で，MCU 内蔵フラッシュメモリの最大サイズは 4 MB 程度である (これは今回用いた MCU とは異なるベンダの製品である)．これに対し，今回の検討で用いたテキスト音声合成システムの形態素辞書データは

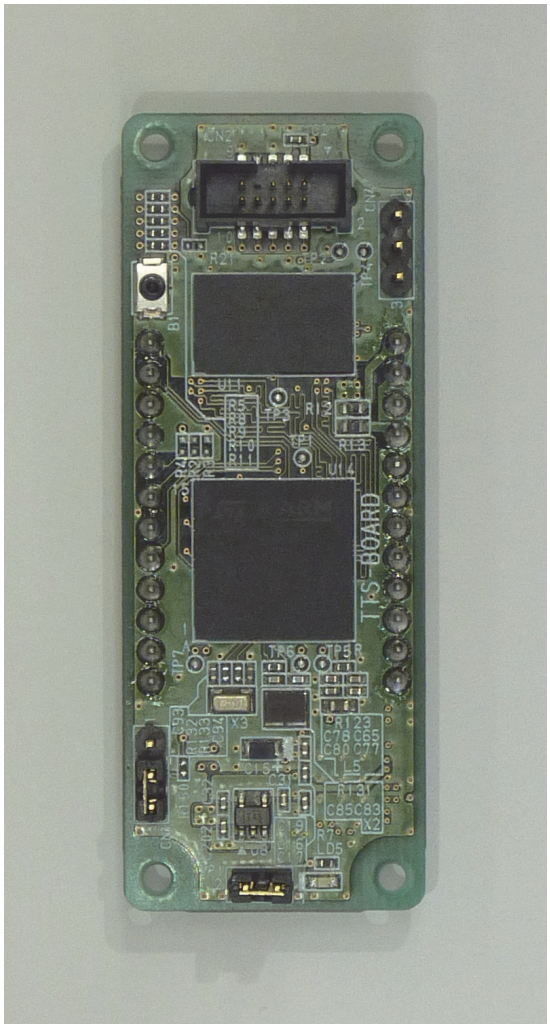

図 5 動作検証用マイコンボード
Fig. 5 Microcomputer board for our TTS software

表 1 動作検証用マイコンボードの仕様
Table 1 Specification of the microcomputer board

MCU	ST STM32F469NI
MCU コア	ARM Cortex-M4F (最高動作周波数 180 MHz)
内蔵 RAM	384 KiB (SRAM)
内蔵フラッシュ	2 MiB
外部フラッシュ	Micron N25Q512A (64MiB, QSPI 接続)
外部 RAM	SDRAM 16 MiB (32 ビット幅, 133 MHz)
オーディオ DAC	Cirrus Logic CS43L22
外部水晶振動子	8 MHz, 32.768 kHz
外部電源電圧	3.3 V
ボードサイズ	66 mm × 25 mm

5.4 MB，音響モデルデータは 3.3 MB である．音響モデルデータは，1 MB 以下のサイズに削減しても実用的な音声品質が得られるが，日本語テキスト解析に必要な形態素辞書データ等の削減は読み予測精度への影響が大きいことから，より現実的な想定として，外部フラッシュメモリを追加して，形態素辞書データや音響モデルデータはそちらに格納し，それを音声合成時に直接アクセスする方法を用いることとした．MCU 内蔵のフラッシュメモリには主にプログラムを配置している．

MCU にフラッシュメモリを接続する場合，配線数を削減するため，4 本の信号線を用いたシリアルインタフェースである serial peripheral interface (SPI) や，SPI のデータ線を双方向とし同時に 4 本用いる quad SPI(QSPI，信号線合計 6 本) をそのインタフェースとするフラッシュメモリが用いられることが多い．試作したシステムでも QSPI フラッシュメモリを用いた．現在の MCU には，QSPI フラッシュメモリをメモリアドレス空間にマップし，通常のメモリと同様に読み出せる機能を備えたものが多いが，今回開発したソフトウェアも，その機能の利用を前提としている．

なお，QSPI フラッシュはその通信プロトコルの特性として 1 回のアクセスのオーバヘッドが大きく，そのためにランダムアクセスがあまり速くないといった特徴がある．例えば今回用いた QSPI フラッシュでは，通信のクロックが 90 MHz の場合で，シーケンシャルな読み込みのスループットは 45 MB/s だが，仮にバイト単位で不連続なアクセスを行うと，3.5 MB/s (26 サイクル/バイト) にまで低下する．この要因の 1 つとしては，QSPI であってもリード等のコマンド発行 (8bit) を信号線 1 本だけで行っていることが挙げられる．連続リードに対しては最初のコマンドを省略できるモードを設けるといった拡張が各社で行われているものの，この拡張機能の利用については今後の課題である．

4.3 外部 RAM

用いた MCU の内蔵 RAM サイズは 384 KiB だが，後述する日本語テキスト音声合成ソフトウェアでは，ATR503 文の合成時で，最大 600KB 程度の RAM を必要とする．動作検証用マイコンボードの設計の時点で，そのような大容量の RAM を内蔵した MCU の入手が困難だったことから，MCU に 16 MiB の SDRAM を接続している．別の方法として SRAM の追加も考えられたが，大容量の RAM が必要になるような用途への応用可能性も考え，コスト面から SDRAM を採用した．

SDRAM ではメモリリフレッシュが必要だが，今回採用した MCU では，MCU 上のメモリコントローラによりリフレッシュ動作が行われ，実際に必要なのは初期設定のみである．特に MCU ベンダのマイコン評価ボードと同スペックの SDRAM を採用した場合など，初期設定に MCU のサンプルプログラムがほぼそのまま使える場合，ソフトウェア開発のコストあまり増えない．しかし，必要な配線数が多く (本試作ではデータ幅 32 ビットの SDRAM を採用したため，信号線だけで 56 本となった．ただしアドレス

表 2　動作検証用マイコンボードのピン配置
Table 2　Pin assignment of the external connectors

CN3 (ボード左側)		CN5 (ボード右側)	
1	UART_TX/D1	1	VDD(+3.3V)
2	UART_RX/D0	2	GND
3	#RESET	3	#RESET
4	GND	4	VDD(+3.3V)
5	I2C_SDA/D2	5	DAC1_OUT2/D11
6	ISC_SCL/D3	6	DAC1_OUT1/D10
7	D4	7	LINEOUT_B
8	D5	8	LINEOUT_A
9	D6	9	SPEAKER_BN
10	D7	10	SPEAKER_BP
11	D8	11	SPEAKER_AN
12	D9	12	SPEAKER_AP

とデータが別のピンのため，データ幅を狭くしてもある程度の信号線数は必要．)，MCU には多数の入手出力ピンを備えたパッケージが必要となる．また等長配線が必要だったりと，SDRAM の搭載は実際にハードウェアの大きなコスト上昇要因となった．なお，マイコンボードの動作電圧 (3.3V) は，SDRAM の動作電圧から決まった値である．

4.4　音声出力

今回用いた MCU には 12 ビットの DA コンバータ (DAC) が搭載されているが，バッファやスピーカアンプを別途接続する必要がある．搭載チップ数では変わらないことから，動作検証用マイコンボードでは，スピーカアンプ (D 級 1W×2) 内蔵の 24 ビットオーディオ DAC を音声出力に用いた．ただし，MCU 内蔵の DAC 出力も後述する外部接続端子から取り出せるような設計としている．

4.5　外部接続インタフェース

オープンソースハードウェアである Arduino Mini[17] のピン配列を参考に，12 ピンの外部接続端子 2 組 (電源，I/O，オーディオ出力等) をマイコンボード上に設けた．表 2 に配置を示す．汎用 I/O(GPIO) には最大 12 ピンが利用可能で (うち 6 ピンは外部通信端子やアナログ出力端子と共用)，その他，電源及び GND が 4 ピン，オーディオ DAC 出力であるスピーカ端子とオーディオライン出力がそれぞれ 4 ピン (2 線 ×2 チャネル) と 2 ピン，リセット信号 2 ピンの構成である．またピン間隔は 2.54 mm で，市販のソルダーレスブレッドボードにマイコンボードを直接搭載できる．さらにソフトウェア開発用に，ボード上に非同期シリアル通信用端子 (3 ピン) と MCU 搭載のデバッグ回路との通信用端子 (9 ピン) を設けている．

一般的な利用方法として，CN3 のシリアル通信端子 (UART_TX，UART_RX) 経由で他のデバイスから日本語テキストを取得し，その合成音声波形を CN5 のライン出力やスピーカ端子から出力するような利用を想定している．ただし，マイコンボード単独でも様々な処理を行えるように，I2C 端子や GPIO 端子も設けた．

5.　組み込みシステム向け日本語テキスト音声合成ソフトウェアの開発と動作

3 節にて説明した手法は，日本語テキスト音声合成ソフトウェア「N2」に導入済みである．今回は N2 をベースに組み込みシステム向けのソフトウェア開発を進め，前節にて説明した動作検証用マイコンボード上でその動作を確認した．以下ではその開発と動作についてそれぞれ説明する．

5.1　開発

N2 の主要部分は，同じソースコードを複数プラットフォームで用いるために，当初より移植性を重視して開発を進めてきた．このため移植作業は比較的容易で，実際の作業は，音声再生処理といった動作検証用マイコンボード特有の部分の開発がそのほとんどを占めた．

従って，メインの開発言語は従来のシステムと同様の C++である．今回の開発ではコンパイラに ARM 製コンパイラ (バージョン 5.06) を用いた．なお，C 言語で開発されたライブラリと同様に利用できるようにすることを目的として，C++実行環境 (ライブラリ等) に依存しないように，N2 のプログラムでは C++の機能である仮想関数や RTTI (実行時型情報) は全く用いていない．これが実際に可能かどうかはその処理系に依存するが，今回作成した ARM Cortex-M 用の実行形式では，C++ライブラリに全く依存していないことを確認している．

また，N2 では移植性を重視して API は全て同期動作するようにしているが，加えて，OS 等の支援がなくてもマルチタスク処理を疑似的に実現できるように，音声合成の内部処理を細かい単位で中断し (中断までの時間は API の呼び出し時に設定する)，それを後で再開できる機能を備えている．この機能をそのまま用いることで，今回の検証ではリアルタイム OS 等を用いずに，疑似的だがマルチタスクの処理を実現した．

5.2　動作

開発したテキスト音声合成ソフトウェアの形態素辞書データと音響モデルデータは PC やスマートフォン用のデータと互換で，24.4 万語の形態素辞書を用いて ATR503 文 [18] 全ての音声合成を行い，マイコンボード上での音声合成結果 (合成音声波形) が PC 版の結果と完全に一致することを確認した．また，処理時間を短縮するためにハードウェア的に設定可能な最大クロック設定 (MCU コア: 180 MHz，QSPI フラッシュおよび SDRAM: 90 MHz) とした場合の合計消費電力は，オーディオ DAC のスピーカアンプをオフにした状態で 0.37 W である．

また現在のバージョンでは，前述のクロック設定の場合，ATR503 文合成時の平均で，ほぼリアルタイムの処理速度である．一方，音声波形生成処理と音声再生処理はパイプライン処理による同時実行が可能なことから，テキスト音声合成の処理遅延に対応する，日本語テキスト解析処理からパラメータ軌跡計算処理までの合計待ち時間を測定したところ，実時間比 (RTF) で 0.8 程度であった．先行研究 [16] において，ARM9E 400MHz 上の Linux システムでの処理速度が RTF 0.15 程度で，そのうち合計待ち時間が RTF 0.04 程度だったことと比較すると，波形生成以外の処理で動作周波数比での速度性能が大幅に低下している．波形生成処理は MCU 上のメモリだけで処理を行うのに対し，その他の処理は外部フラッシュメモリを参照し，さらに先行研究のシステムとは異なり，今回用いた MCU にデータキャッシュメモリがないことから，この速度性能低下は，外部フラッシュメモリへのアクセスのオーバヘッドの大きさが主な原因と考えている．より詳細な調査については今後の課題である．

6. おわりに

MCU ベースのマイコンボード上で動作する HMM 音声合成ベースの日本語テキスト音声合成を開発した．また，本稿では，その実現に必要だった 2 つの手法，また開発したテキスト音声合成ソフトウェアの動作検証用マイコンボードについて紹介した．今回開発した日本語テキスト音声合成ソフトウェアは従来の PC・スマートフォン用のソフトウェアと同じ結果が得られる．しかし，プロセッサの動作周波数比の性能でみると，波形生成処理では高速な処理が実現できている一方で，その他の処理の部分で大幅な性能低下があり，その原因の調査および高速化について引き続き取り組む予定である．

参考文献

[1] KDDI 総合研究所: 音声合成ソフトウェア「N2」(オンライン), 入手先 ⟨http://www.kddi-research.jp/products/n2.html⟩.
[2] Tokuda, K., Nankaku, Y., Toda, T., Zen, H., Yamagishi, J. and Oura, K.: “Speech synthesis based on hidden markov models,” Proc. of IEEE, vol. 101(5), pp. 1234-1252 (2013.5).
[3] Moulines, E. and Charpentier F.: Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones, Speech Communication, vol. 9 (5,6), pp. 453–467 (1990.2).
[4] Black, A. and Campbell, N.: Optimising selection of units from speech databases for concatenative synthesis,Proc. EUROSPEECH ’95, vol.1, pp.581–584, Madrid, Spain (1995.9).
[5] 河井恒, 戸田智基, 山岸順一, 平井俊男, 倪晋富, 西澤信行, 津崎実, 徳田恵一: 大規模コーパスを用いた音声合成システム XIMERA, 信学論 (D), J89–D (12), pp. 2688-2698 (2006.12).
[6] 今井聖, 住田一男, 古市千枝子: 音声合成のためのメル対数スペクトル近似 (MLSA) フィルタ, 信学論, vol. J66–A (2), pp. 122–129 (1983.2).
[7] 今井聖: 音声信号処理, 森北出版 (1996)
[8] 西澤信行, 矢崎智基: HMM 音声合成における固定小数点演算によるパラメータ軌跡計算時の誤差削減, 信学論, vol. J100–D (2), pp. 230–243 (2017.2).
[9] 益子貴史, 徳田恵一, 小林隆夫, 今井聖: 動的特徴を用いた HMM に基づく音声合成, 信学論 (D), J79–D–II (12), pp. 2184–2190, (1996.12).
[10] hts_engine API (online), available from ⟨http://hts-engine.sourceforge.net/⟩.
[11] Nishizawa, N. and Kato, T.: Speech synthesis using a maximally decimated pseudo QMF bank for embedded devices, in Proc. 8th ISCA Speech Synthesis Workshop (SSW8), pp. 47–52 (2013.8).
[12] Rothweiler, J. H.: Polyphase quadrature filters – A new subband coding technique, in Proc. ICASSP ’83, Boston, MA, U.S.A., vol. 3, pp. 1280–1283 (1983.4).
[13] Chen, W. H., Smith, C. H, and Fralick, S. C: A fast computational algorithm for the discrete cosine transform, IEEE Trans. on Comm., vol. 25(9), pp. 1004–1009 (1977.9).
[14] Vetterli, M. and Nussbaumer, H. J.: Simple FFT and DCT algorithms with reduced number of operations, Signal Processing, vol. 6(4), pp. 267–278, (1984.8).
[15] ISO/IEC, JTC1/SC29/WG11 MPEG: Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s Part 3: Audio, IS11172-3 (1992).
[16] 西澤信行, 加藤恒夫: 疑似 QMF バンクを用いた合成音声波形生成手法の速度性能評価, 音講論 (春), 3–6–12, pp. 345–348 (2014.3).
[17] Arduino Official website (online), available from ⟨https://www.arduino.cc/⟩
[18] 阿部匡伸, 匂坂芳典, 梅田哲夫, 桑原尚夫: 研究用日本語データベース利用解説書 (連続音声データ編), TR-I-0166, ATR 自動翻訳電話研究所 (1990.8).