RC-008

マルチ SoC を適用した組み込みシステムの検討
Development of an embedded system with a multi-SoC

上村 敬志† 小池 正英† 野口 正雄†
Keiji Uemura Masahide Koike Masao Noguchi

1． はじめに

大型サイネージシステムや車載機器に代表されるような業務用の組み込みシステムでは，利用者の多種多様なニーズへ対応するために高機能化・多機能化が求められるようになり，それにともないシステムの複雑化が進んでいる．

業務用の組み込みシステムでは，ターゲットとなる機器の用途に特化した処理と低消費電力の実現のため，SoC（System-on-a-Chip）と呼ばれる，1 つのチップにプロセッサや周辺機能となる回路を統合した集積回路が活用されている[1]．SoC を採用することで，機器本体の小型化やコストの低減，部品点数の削減が可能になるなどの利点がある．

しかし，組み込みシステムにおいて多機能化の要求が高まることで，SoC に求められる処理負荷も増加しているが，このような高処理負荷に対応するために，動作周波数の高いプロセッサを選択することや，多岐の周辺回路を集積した SoC を選定するとコストが大幅に増加する問題があった．

このように多機能化する組み込みシステムに対して，1 つの基板の中でそれぞれ異なる処理に特化した SoC を組み合わせることで，ターゲットとなる機器に最適なプロセッサの処理能力と周辺回路を選定可能なシステムを検討した．本稿ではこれをマルチ SoC システムと呼ぶこととする．

本稿では，このマルチ SoC システムにおいて，システムの安定動作のために，汎用のインターフェースを用いて大容量データやデバッグ情報の通信を SoC 間で行う場合の検討および評価結果について述べる．

2． マルチ SoC システム

本章では，1 つのシステム上で複数の SoC が動作するマルチ SoC システムの基本的な構成と特徴および本システムを構成する上での検討課題について述べる．

2.1 システム基本構成

マルチ SoC システムは，ターゲット機器で要求される機能を複数の SoC に分担させて実行することが目的であり，全体としては 1 つのシステムとして機能する必要がある．そのため，各 SoC が有する機能を独立に実行しながら，全体のシステムとして機能するために互いに必要となる制御情報やデータ信号のやり取り行うことで，システムを実現する．本稿で取り上げるマルチ SoC システムの概要を図 1 に示す．本稿で述べるマルチ SoC システムは，一つの基板から構成されており，各 SoC 間は基板内の伝送を用いることを想定している．

†三菱電機株式会社,先端技術総合研究所

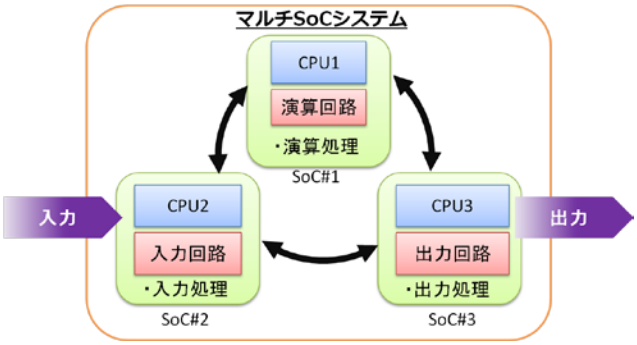

図 1．マルチ SoC システム概要

2.2 マルチ SoC システムの特徴

図 1 に示したマルチ SoC システムを構成することで，以下に示すような利点を得ることが可能となる．

(1) ターゲット機器に適した構成を幅広く選択可能
(2) 追加の周辺デバイス削減
(3) 高可用性
(4) 不具合発生時のログ取得

以下で，それぞれの利点について説明する．

(1) ターゲット機器に適した構成を幅広く選択可能

1 つの SoC でターゲット機器の機能を実現する場合，SoC に組み込まれた回路やプロセッサに求められる性能への要求が高まり，候補となる SoC が限定される．一方，マルチ SoC システムでは，求められる機能を複数の SoC で分担することで，1 つの SoC への要求機能が緩和され，SoC の選択肢を広げることが可能となる．そのため，SoC に含まれている回路の組み合わせを考慮し SoC の選定を行うことで，ターゲット機器の機能に対し，システム全体として過不足が小さくなるような構成を実現しやすくなると考えられる．

(2) 追加の周辺デバイス削減

複雑なシステム構成からなるターゲット機器では，単一の SoC に搭載された集積回路のみで要求される全ての機能を実現できるとは限らない．その場合，不足する機能を実現するためにシステム上に専用の集積回路を追加する必要があるが，回路の追加やそれに伴う配線の増加はコスト高の要因となる．ここで専用の集積回路とは，例えば Ethernet PHY や USB コントローラのようなチップデバイスを指す．

一方，マルチ SoC システムでは上記(1)で述べたように不足する機能を抑える構成をとることで，単一の SoC だけでは実現できない機能に対する専用の集積回路などのデバイスの追加を抑制することができる．さらに，このようにデバイスの追加を抑制することで，部品点数の削減だけでなく，省配線化による信頼性能向上や，デバイスの製造中止にともなう再設計の発生を抑えることができる．

(3) 可用性向上

組み込み機器はプラントや自動車の制御といったシステムの誤動作が人命にかかわる事態につながるものに適用されることも多く，不具合による動作停止が発生しない高い信頼性が求められる．また，家電製品などの民生機器の場合でも，システムの誤動作が重大な事態を引き起こす可能性を否定できない[2].

マルチ SoC システムでは複数の SoC が独立に動作していることから，どちらかの SoC に不具合が発生した場合に，他方の SoC によって重大な事態を引き起こす可能性のある処理を停止しないような制御を実行することや，ユーザに対してその旨を提示することが可能である．したがって誤動作や不具合に対してシステムが継続して使用できる能力（可用性）が高いと言える．

(4) 不具合時のログ取得

組み込みシステムは特定用途に特化した機器のため表示装置が不必要なものも多く，不具合発生時にどのような状況かを判断することが難しい．不具合解析のため，不具合発生時のメモリを記録するコアダンプ機能や，ネットワークを介してシステムのログを出力する syslog[3]などが用いられている．しかし，コアダンプ機能はメモリ内容をそのまま記録するため，その内容を解析するために高い専門性が必要であり，さらに SoC が動作不能の場合，記録されたメモリを読み出すことが困難となる．syslog は広く使用されており，アプリケーションのログ取得が容易に行えるが，サーバとなる機器にネットワークで接続されている必要がある．一方，マルチ SoC システムでは，SoC 間でログの送受信を行い，各 SoC で他方の SoC のログを記録することで，互いのログを記録することが可能である．したがって，いずれかの SoC に不具合が生じた場合でも，そのとき出力されたログを他の SoC にて記録可能であり，読み出しも容易となる．

2.3 マルチ SoC システム構築に対する課題

ここで，マルチ SoC システムを構築する上で検討すべき点を以下にまとまる．

(1) SoC 間接続の複雑化

(2) プログラム書込み作業の煩雑化

以下では各検討課題について詳細を述べる．

(1) SoC 間接続の複雑化

前節で述べたようにマルチ SoC システムでは各 SoC 間には制御信号およびデータ信号など様々な情報のやり取りが発生する．特に，大容量のデータを転送する場合，SoC 間で高速に動作する伝送方式を用いるためには，SoC 間の伝送においても基板内での伝送路を考慮してハードウェア設計を行う必要がある[4]．また，一方の SoC に不具合が発生した場合でも，基板内での接続のため外部から接続を切り離すことができない．つまり，不具合が発生した SoC から意図しない信号が発せられることにより他方の SoC に影響がおよび不具合が連鎖するおそれがある．したがって，SoC 間接続において，一方の SoC の不具合が他方の SoC に影響を与えない伝送方式が求められる．

(2) プログラム書込み作業の煩雑化

業務用途の組み込みシステムのように長期間運用される機器では，運用中にセキュリティ対策等の理由から機器の保守点検作業においてソフトウエアのアップデートを行うことがある．ソフトウエアアップデート作業は，ネットワークが接続されていない機器の場合，作業者が機器に対して USB メモリや SD カードなどの記憶媒体経由でアップデートプログラムの書き込みを行う必要がある．対象となる SoC が一つであれば，一回の書き込み作業で完了するが，複数の SoC の場合，各 SoC に対して記憶媒体を接続し，書き込み作業を行う必要がある．つまり，作業者が一つのシステムに対して SoC の個数分のデータ書き込み作業が発生する．そのため，作業時間の増加に加えて，作業工数の増加で作業誤りが発生する可能性が高まることにつながり，保守点検作業の非効率化の要因となる．

3. 実装検討

本稿では，ターゲットシステムとして映像を記録・再生するシステムを想定して検討を行った．ターゲットとなる本システムの機能は，映像の記録機能と再生機能の二つに大別され，それぞれの処理は独立性が高く，各 SoC にそれぞれの機能を分担させやすい．また，映像の入出力など必要となる周辺回路も多く，これらを過不足なく備えた単体の SoC の選択も難しい．以上のことから，今回の対象システムとしてマルチ SoC システムに適したターゲットであると言える．

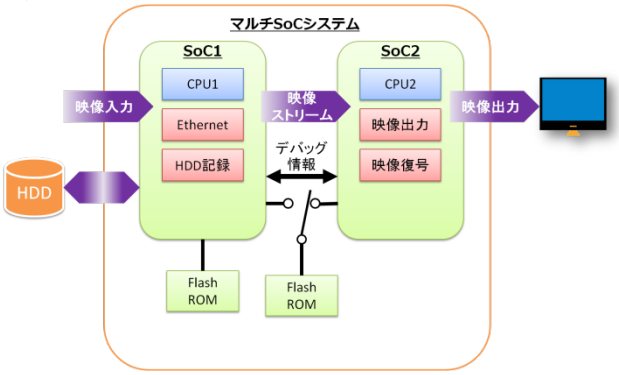

図 2. ターゲットシステム概要

本稿で検討するシステムの概要を図 2 に示す．本システムでは主に映像記録機能が動作する系に SoC#1，主に映像再生機能が動作する系に SoC#1 とは異なる SoC#2 を割り当てた．SoC#1 には映像入力のためのネットワーク接続や録画のための HDD（Hard Disk Drive）が接続されている他，SoC#1 上で動作する OS やアプリケーションプログラムが書き込まれた Flash ROM（Read Only Memory）を接続した．

また，SoC#2 には映像出力インターフェースの他，OS やアプリケーション等のプログラムが書き込まれた Flash ROM，3.2 節で述べるスイッチ回路が Flash ROM に接続されている．

本システムの OS には両 SoC ともに Linux を採用した．組み込み機器では高機能化に対応するため OS を用いたシステムが増加しており，特に，本システムのような多機能化要求の強い映像処理を扱う組み込みシステムでは，2000 年頃から機能拡張ライブラリが充実した汎用 OS である Linux が採用される例が増えている[5][6][7].

次節から，本システムにおいて実装した機能の詳細について述べる．

3.1 SoC 間接続

2.3(1)も述べたように，マルチ SoC システムでは SoC 間の通信が複雑化する問題があった．そこで，本節では，本システムに適切な SoC 間通信の接続方式を検討した．

SoC 間の通信は，実際に通信が行われるデータやシステム構成によって異なるが，今回ターゲットとするシステムでは，以下の点を考慮して接続方式の検討を行った．

(ⅰ) 一方の SoC が動作していない場合に，通信相手の SoC に影響しない
(ⅱ) 一方の SoC のリセット後，再接続可能
(ⅲ) 相互に対等な通信が可能

本システムでは，SoC 間で図 2 に示すように，(1)映像ストリームおよび(2)デバッグ信号の通信が必要となる．

(1) 映像ストリーム

映像ストリームでは，送信側となる SoC#1 は確実に受信側である SoC#2 へのデータ転送が完了したことを判断する必要がある．また，映像ストリームは，必要とされる通信量が多く，今回のターゲットでシステムでは少なくとも 10Mbps 以上の帯域を想定した．これらの要件を満たすインターフェースとして，Ethernet[8]による通信を行うこととした．Ethernet を用いることで通信の接続/再接続が容易に行え，他方の SoC が起動していない場合でも，正常な動作が期待できる．さらに，正常に受信が行われるまで再送することが可能である．同じ高速伝送可能な接続として PCIe（PCI Express, [9]）や USB（Universal Serial Bus, [10]）が考えられるが，PCIe や USB では(ⅱ)の再接続性や(ⅲ)の対応な通信の問題が解決しない．

(2) デバッグ情報

ここでデバッグ情報とは SoC 上で動作する Linux のシリアルログ出力を指す．シリアル通信の接続方式には，一般に組み込みシステムで多く使用されている UART（Universal Asynchronous Receiver Transmitter, [11]）に代表される調歩同期方式や SPI（Serial Peripheral Interface, [12]），I^2C（Inter-Integrated Circuit, [13]）などがある．デバッグ情報は上記(ⅰ)~(ⅲ)に加えて，Linux システムにおけるカーネル起動完了以前，ブートローダによる処理から実行できる必要がある．SPI や I^2C はブートローダでの実装も比較的容易ではあるが，SPI や I^2C では一方の SoC がマスターとなり，通信を開始しなければスレーブ側は通信が行えないマスター-スレーブ方式のため，(ⅲ)に適しない．

実際に，デバッグ情報の取得では他方の SoC からの出力タイミングが分からないため，SPI や I^2C といったマスター-スレーブ方式の通信ではスレーブとなった SoC のデバッグ情報の確実な取得が行えない．

また，一般的に SoC のデバッグ情報の出力は UART から行われるように設計されているため，本開発においてもデバッグ情報の通信には UART を用いた．

3.2 プログラム書き込みの簡素化

2.3(2)で述べたように，複数の SoC が同一システム上にある場合，プログラムアップデート時のプログラム書き込みは各 SoC に接続された Flash ROM に対して行う必要があった．SoC ごとに Flash ROM へのプログラム書き込みを行えば，SoC の数だけ作業工数が増加し，プログラム書き込みにかかる時間が増加するだけでなく，操作に誤りが発生する可能性が高まる．そこで，プログラムアップデート時におけるプログラム書き込みの簡素化方法を検討した．

プログラム書き込み作業の増加は USB メモリなどの外部記憶媒体を各 SoC に対して機器に挿入することが要因であった．そこで，本システムでは，プログラムの書き込み時に一方の SoC にのみ外部記憶媒体を接続する方式を検討した．具体的には SoC のプログラムが書き込まれる Flash ROM の接続先を選択出来るマルチプレクサを搭載し，片方の SoC から他方に書き込みが行える構成をとった．本システムでの適用例を以下に示す．

まず，図 3-1 にあるように，外部記憶媒体が接続された SoC#1 は，外部記憶媒体内に記憶された自身のアップデートプログラムである CPU1 プログラムと SoC#2 に対するアップデートプログラムである CPU2 プログラムを読み出し，一旦自身に接続されている Flash ROM に書き込む．

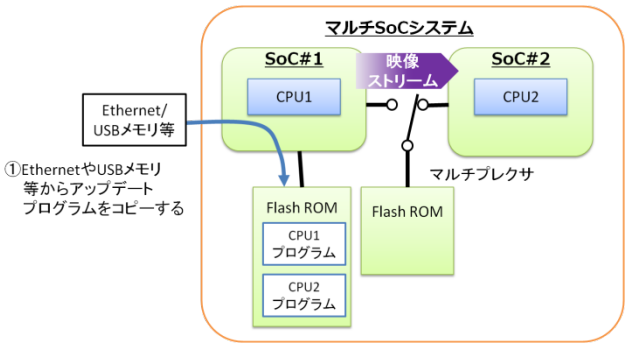

図 3-1. SoC#1 による書き込み

次に，SoC#2 のプログラムが記録される Flash ROM に接続されたスイッチ回路であるマルチプレクサを制御し，SoC#1 に接続されるようにする．SoC#2 は動作に必要なプログラムを主記憶装にコピーして処理を実行しているため，Flash ROM が接続されていない状態でも正常に処理を続けることができる．

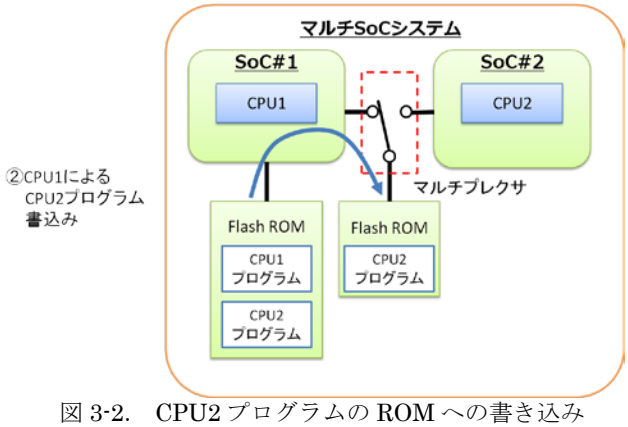

図 3-2. CPU2 プログラムの ROM への書き込み

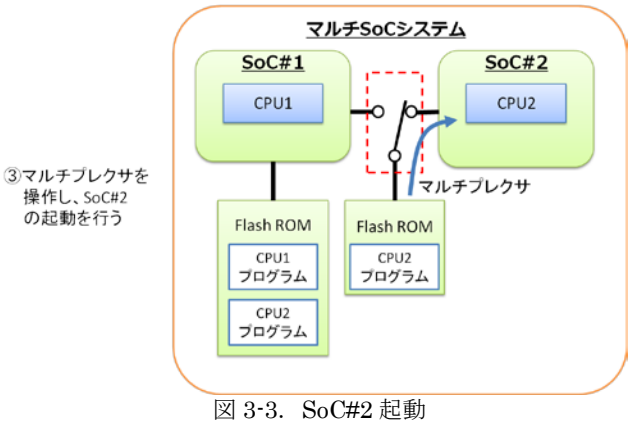

図 3-3. SoC#2 起動

マルチプレクサを切り替えた後に，SoC#1 は自身の Flash ROM に記録された CPU2 プログラムを SoC#2 の Flash ROM に直接書き込むことで，外部記憶媒体を接続していない SoC に対してアップデートプログラムを書き込むことが可能となる（図 3-2）．

Flash ROM への書き込み後，再びマルチプレクサを操作し，Flash ROM を SoC#2 に接続されるようにすることでシステムリセット後，更新されたプログラムで起動することが可能になる（図 3-3）．

ここで，本手法に適した Flash ROM のインターフェースについて考察する．本手法では，Flash ROM との信号線をマルチプレクサによって切り替える方法を採った．マルチプレクサにより信号線の接続自体を切り替えるため，

(ⅰ)信号線の本数が少なく

(ⅱ)信号線の切り替えによる波形の乱れが少ない

ことが必要となる．(ⅰ)を満たすものとして，シリアル信号線を持つ Flash ROM がある．代表的なものとして，I^2C，SATA(Serial ATA)，SPI などがある．この中で，SATA の場合は伝送速度が高速であり，マルチプレクサによる信号線自体の切り替わりによって伝送路長の不整合やマルチプレクサ内のトランジスタ特性により波形が乱れることが考えられる．そこで I^2C と SPI が適することになるが，本システムで使用されるプログラムの容量や転送速度の観点から本開発では SPI Flash ROM を選択した．

3.3 相互起動ログ取得

3 章において，マルチ SoC システムでは各 SoC で相互にデバッグ情報を記録できることを述べた．これは，本システムにおいては，OS である Linux が起動することで，メインのアプリケーションの実行と並列してデバッグ情報である SoC のシリアルログを記録する処理をマルチタスクとして処理を実行できるためである．

一方，システム電源投入後の Linux の起動段階では，Linux の本体である Kernel が起動する以前に Kernel の RAM への展開や，各周辺デバイスの初期化を行うブートローダが処理を行う．しかし，組み込み用途で広く使用されているブートローダでもマルチタスクに対応していないことが多く，ブートローダが起動処理を実行している間は他方の SoC から送信されてくるデバッグ情報を取得することができない．しかし，ブートローダが処理を行う起動処理時のデバッグ情報としてのログは重要であり，品質向上の面からも取得の要求は高かった．単一の SoC からなるシステムでも同様にブートローダのログ取得は課題となるが，本システムでは，複数の SoC を時間差で起動するように起動順を制御することで，シングルタスクでもブートローダの起動ログを取得することが可能となる手法を検討した．

図 4 で，本起動ログ取得手法の処理の流れを示す．まず，システムへの電源投入後，各 SoC はそれぞれに接続された Flash ROM からブートローダを読み出し，起動を開始する．各ブートローダはシリアル通信などのデバイスの初期化を行い，ログを出力できる状態となったら，まず SoC#2 のブートローダは，起動処理を一旦停止し，SoC#1 からログが出力されるまで待機する．

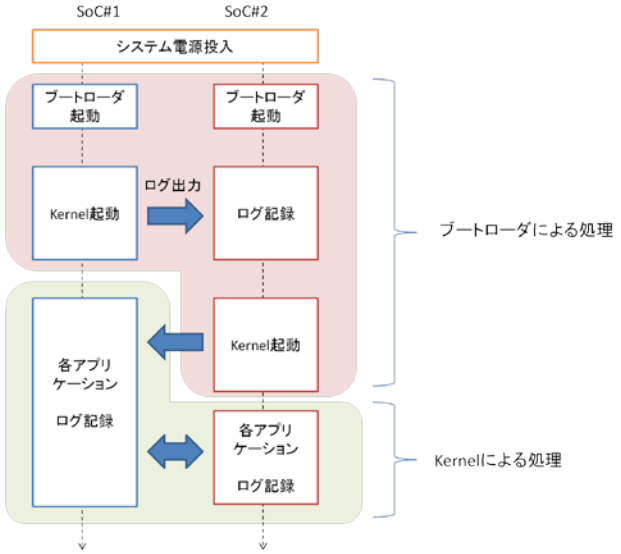

図 4. 相互起動ログ取得シーケンス

一方，SoC#1 は起動処理を続け，Kernel を Flash ROM から読み出し，Kernel の起動処理を実行する．この間，SoC#1 は起動ログを SoC#2 に対して出力する．SoC#2 は，SoC#1 から受け取ったログを Flash ROM に記録する．

SoC#1 の Kernel 起動が完了すると，続いてもう一方の SoC#2 の Kernel 起動処理が開始され，SoC#1 に対しログを出力する．SoC#1 は SoC#2 から出力されるログを取得しなければならないが，既に Kernel が起動しているため，マルチタスク処理が可能となっている．したがって，記録処理を実行しながらログ取得も同時に実行できる．

さらに，本起動ログ取得手法では，Kernel 起動時のログが相互に取得できるだけでなく，ブートローダの起動を同時に行うことから，各 SoC を順に起動した場合より起動時間を短縮することができる．

また，各 SoC が互いのログ記録をせず，それぞれ独立して同じタイミングで起動させた場合と比較すると，本手法では SoC#2 の起動時間は SoC#1 の Kernel 起動ログを取得する時間だけ増加するが，先に起動する SoC#1 の起動時間は他方の SoC の起動ログ取得処理に影響されず，起動時間は増加しない．

4. 動作検証

3 章で述べた手法を実装したシステムを用いて動作検証を行った．

4.1 SoC 間通信動作検証

3.1 節で述べた SoC 間通信についての動作検証を行った．まず，本システムの機能である記録映像の再生機能が動作するかを検証するため，SoC#1 に接続した外部記憶媒体に表 1 に示す映像データの再生をおこなった．マルチ SoC システムを実装した本システムにおいて外記憶媒体に記録した映像データを SoC#1 で読み出し，Ethernet により SoC#2 に送出，SoC#2 によってデコード，映像再生が正常に動作するかを検証した．

表 1. 検証用映像データ

コーデック	H.264
コンテナ形式	MPEG-4
解像度	1920×1080
ビットレート	Max.8Mbps
フレームレート	60Hz

実際に映像再生を行ったところ，目視により正常な映像再生が行えていることを確認できた．

また，映像再生処理を実行中でのデバッグ情報の取得についても動作検証を行った．動画再生処理実行中に UART 出力したシリアルログを相互に受信し，各 SoC に接続された Flash ROM に記憶した．動画再生処理が一定時間経過したところで起動ログを取り出し，シリアルログが正常に取得できたことが確認できた．

4.2 プログラム書き込み検証

次に，3.2 節で述べたプログラムアップデート時の Flash ROM へのプログラム書き込み手法の検証を行った．本プログラム書き込み手法を適用することでアップデートプログラムを一方の SoC にのみ接続することで他方の SoC のプログラムも正常にアップデートされているかを検証した．

本システムのアプリケーション動作中に外部記憶媒体を直接接続しない SoC#2 のアプリケーションプログラムの更新を行った．3.2 節で述べたプログラム書き込み手順を実行し，SoC#1 から Flash ROM にバージョンの異なるプログラムデータの書き込みを行った．

書き込んだ CPU#2 のプログラムはメインアプリケーションプログラムであり，プログラム内に記述されたプログラムのバージョン番号を参照することでアップデートされたプログラムであることが確認できる．

プログラムアップデート動作を実行した結果，SoC#2 の再起動後，書き込んだプログラムを実行できるかを検証した．その結果，SoC#2 は正常に Flash ROM からプログラムを読み出し，再生プログラムを実行されることを確認した．したがって，スイッチ回路であるマルチプレクサを利用した SoC#1 からの書き込み，および，マルチプレクサの接続の再変更による SoC#2 からのプログラムデータ読み出しが正常に行えたことを確認した．

4.3 相互起動ログ取得起動検証

本システムを用いて，起動ログの相互取得処理の動作検証を行った．各 SoC が独立して起動した場合，本相互起動ログ取得手法を実施した場合，各 SoC を順に起動した場合を比較した．図 5 に SoC の起動時間の測定結果を示す．それぞれの条件で 5 回動作させた場合の平均を取った．本検証での起動時間は，ブートローダのシリアルログが Windows PC 上のターミナルソフトウエアに表示されてから，ログインプロンプトが表示されるまでの時間とした．

独立して起動した場合，二つの SoC の起動時間のうち，起動時間の長い SoC の起動時間が全体の起動時間となる．一方，順に起動した場合では，互いの起動時間の合計が全体の起動時間となる．

図 5 に示すように本手法を用いた場合は，SoC を同時に起動させた場合より約 6.9 秒遅く，準に起動させた場合より約 1.4 秒早い結果となった．

それぞれの SoC が独立して起動した場合，起動ログを他方の SoC で取得することはできない．一方，順に起動する場合，先に起動した SoC によって他方の起動ログは取得できるが，先に起動した SoC の起動ログを他方の SoC によって取得することができない．したがって，本提案手法である相互ログ取得手法を用いることで SoC 間の相互に起動ログ取得および起動時間の短縮を実現したといえる．

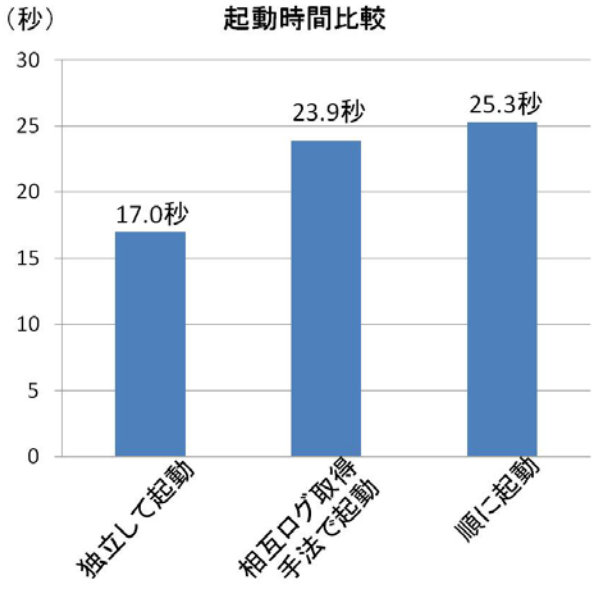

図 5. 起動時間比較

5. 考察・今後の課題

4 章における実機での動作検証の結果，今回検討した手法が実際に有効であることを動作検証により明らかにした．このことから，今回のシステムのような処理負荷の高い組み込みシステムにおいて，システムを機能ごとに分割し SoC を複数用いることで，SoC 自体のコストを下げるとともに周辺回路を省くことができるマルチ SoC システムの有効性を確認することができた．今回，マルチ SoC システムを構築する上で発生する SoC 間通信の複雑化も，通信するデータ量や必要な速度を考慮しインターフェースを選択することで対応した．

一方，マルチ SoC システムを採用することで，コストの面だけでなく，起動時のブートローダの起動ログの相互取得のように，複数の SoC を用いることによる新たな機能を実現することができた．

今後は，本稿で検討した要件（記録，再生）だけでなく，様々な組み込みシステムへの応用を考えている．また，今回 2 つの SoC での実証であったが，3 つ以上の SoC によってマルチ SoC システムを構成した場合に，最適なインターフェース選択や，1 対 1 ではない接続方式を検討が課題となる．

さらに，本稿で述べたマルチ SoC システムを適用する際の基準を検討し，定量的な指標を得る必要があると考えている．定量的な指標を得ることで，ターゲットとする機器においてマルチ SoC システムの採用を決定する判断基準を確立し，組み込みシステム開発の効率化を実現することができると考えている．

6. おわりに

本稿では，高機能化が進む組み込みシステムにおいて，複数の SoC で構成することで全体のコストを削減可能なマルチ SoC システムでの，相互接続や相互ログ機能の取得に関する考察と実機での動作検証を行った．今回動作検証を行ったシステムでは，マルチ SoC システムを採用することで，コスト面だけでなく，起動ログの相互取得などマルチ SoC システム特有の機能によりシステムとしての信頼性を向上可能であることを検証した．

参考文献

[1]内山邦夫.”システムLSIにおけるプロセッサ技術”, 電子情報通信学会誌, Vol.95, No.7,pp.582 ~ 588(2012).
[2]高田 広章”リアルタイムOSと組み込み技術の基礎”, Interface 増刊TECH. Vol.17(2003)
[3] “RSYSLOG THE ROCKET-FAST SYSTEM FOR LOG PROCESSING”, http://www.rsyslog.com/
[4]浅井 秀樹. “高速電子設計のためのSI/PI/EMIシミュレーション技術―過去，現在，そして未来―”, Fundamentals Review Vol.5 No.2, pp146~154(2011).
[5]中島 達夫. “組込みオペレーティングシステム概論”, 映像情報メディア学会誌, Vol.63, No.5, pp.633～637 (2009).
[6]田丸 喜一郎. “5.組み込みプラットフォームの動向”, 情報処理,45(7),699-703 (2004-07-15).
[7]木内 志朗. “Linux を使った組み込みシステム開発”, Interface 増刊TECH I. Vol.16「組み込みLinux 入門」第1 章. (2003)
[8]” IEEE 802.3 ETHERNET WORKING GROUP”, http://grouper.ieee.org/groups/802/3/
[9]” PCI Special Interest Group - PCI Express”, http://pcisig.com/specifications
[10]” USB Implementers Forum, Inc”, http://www.usb.org/home
[11]” RL78/G13 シリアル・アレイ・ユニット（UART 通信）”, http://documentation.renesas.com/doc/products/mpumcu/apn/rl78/r01an0459jj0300_rl78g13.pdf
[12]” シリアル ペリフェラル インターフェイス (SPI)”, http://ww1.microchip.com/downloads/jp/DeviceDoc/70206C_JP.pdf
[13]”I²C MANUAL”, http://www.nxp.com/documents/application_note/AN10216.pdf