Vol.2017-ARC-225 No.34
Vol.2017-SLDM-179 No.34
Vol.2017-EMB-44 No.34
2017/3/10

ヘテロジニアスプロセッサ向け通信ライブラリMDCOM

大竹 史紘[1]　本田 晋也[2]　高田 広章[2]

概要：近年，組込み向けプロセッサは，高パフォーマンスに加えて低消費電力を求められている．一般的に，高パフォーマンスと低消費電力はトレードオフの関係にあり，単一のコアで両方の要件を満たすことは難しい．そこで，この問題を解決するアーキテクチャとしてヘテロジニアスプロセッサがある．ヘテロジニアスプロセッサとは，1 つのチップ上に複数の異なるアーキテクチャのコアが乗るプロセッサのことで，それぞれのコアが高パフォーマンスと低消費電力を実現する．ヘテロジニアスプロセッサを利用する場合，それぞれのコアが協調して処理を行うためにコア間通信が必要になる．これを実現するライブラリとして，MDCOM（Multi Domain COmmunication Module）を提案する．本研究では，MDCOM と既存技術である OpenAMP との比較を行う．また，TOPPERS の API との実行時間，MDCOM と OpenAMP の通信時間を評価する．これらの結果から，MDCOM の優位性を示す．

MDCOM, communication library for heterogeneous processor

Fumihiro Ohtake[1]　Shinya HONDA[2]　Takada Hiroaki[2]

Abstract: Recently, the processors for embedded system are required high-performance and low-energy. Generally, they are trade-off, it is difficult to fullfill them with uni-core. There is a heterogeneous processor which solve this problem. Heterogeneous processor has various architecture cores on a chip. Some cores realize high-performance, another cores realize low-energy. When we utilize heterogeneous processor, inter-core communication is necessary for each cores to cooperate. We propose a inter-core communication library, MDCOM. In this research, MDCOM and OpenAMP are compared. Additionally, TOPPERS API, commucation time of MDCOM and OpenAMP are evaluated. Then we indicate that MDCOM is superior to OpenAMP.

1. はじめに

近年，組込み機器の高機能化が進み，要求される機能がより多く，複雑になった．これまでは，リアルタイム性の保証が重要な機能だったが，現在ではそれに加えて，リッチな GUI やメディアの再生機能などが必要となっている．そのため，組込み向けプロセッサには，低消費電力に加えて高パフォーマンスが求められるようになった．一般的に，これらはトレードオフの関係にあり，単一のコアで両方の要件を満たすことは難しい．そこで，この問題を解決するアーキテクチャとしてヘテロジニアスプロセッサがある．

ヘテロジニアスプロセッサとは，1 つのチップ上に複数の異なるアーキテクチャのコアが乗るプロセッサである．メリットとして，様々なコアを利用することによる高効率なコンピューティングが挙げられる．すなわち，処理の特徴によって振り分けるコアを選択することにより，プロセッサ全体のスループットが上昇する．一方，デメリットとして開発コストが増加が挙げられる．ヘテロジニアスプロセッサでは，異なるアーキテクチャのコアが混在するため，それぞれのコアに対して適切な処理を記述する必要がある．

ヘテロジニアスプロセッサを利用する場合，それぞれのコアが協調して処理を行うためにコア間通信が必要になる．既存技術として，Multicore Association が策定した OpenAMP がある．これは，Linux の機能である remoteproc と RPMsg を利用しており，他のプロセッサに対して，制御と通信を可能とする．NXP 社による実装と拡張があり，

[1] 名古屋大学 工学部電気電子情報工学科
Department of Information Engineering, Nagoya University
[2] 名古屋大学 大学院情報科学研究科
Graduate School of Information Science, Nagoya University

Vol.2017-ARC-225 No.34
Vol.2017-SLDM-179 No.34
Vol.2017-EMB-44 No.34
2017/3/10

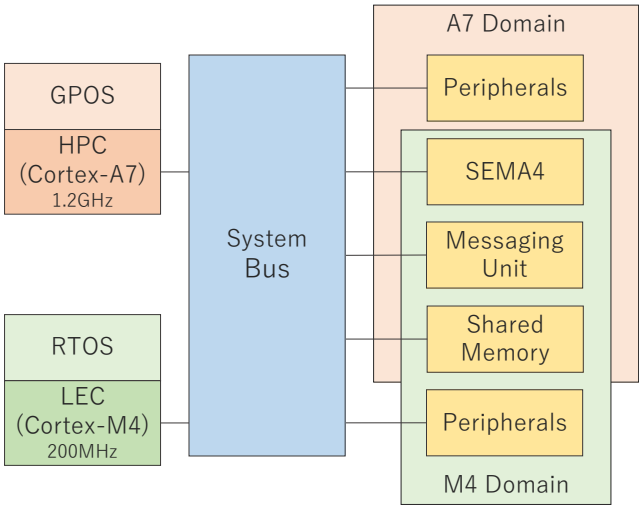

図 1: HAP（i.MX7Dual）のアーキテクチャ

RTOS でのゼロコピー送受信を実現している．しかし，OpenAMP，特に RPMsg については，リアルタイムシステムへの利用を考えた時，いくつかの問題点がある．

そこで，本論文では MDCOM（Multi Domain COmmunication Module）を提案する．現在，MDCOM は i.MX6SoloX と i.MX7Dual，ZynqMP ZCU102 において動作する．

本研究では，MDCOM の評価と既存技術である OpenAMP との比較を行う．また，TOPPERS カーネルの API 実行時間，MDCOM と OpenAMP の通信時間を比較する．

2. ヘテロジニアスプロセッサ

2.1 概要

ヘテロジニアスプロセッサとは，1 つのチップの上に異なるコアを実装したプロセッサのことである [1]．コアの組み合わせには様々な構成が存在する．本研究では，**図 1** に示すように，，高機能な汎用 OS（GPOS）を動作させる高パフォーマンスコアと高信頼や高いリアルタイム性を実現する RTOS を動作させる低消費電力コアを組み合わせたヘテロジニアスプロセッサを対象とする．高パフォーマンスコアと低消費電力コアをそれぞれ High Performance Core（HPC），Low Energy Core（LEC）と略す．また，このようなタイプのヘテロジニアスプロセッサを HAP（Heterogeneous Architecture Processor）と呼ぶことにする．HAP の例として，NXP 社の i.MX や Xilinx 社の Zynq Ultrascale+　MPSoC やルネサスエレクトロニクス社の R-Car H3 等がある．ヘテロジニアスプロセッサの利用例として，HEMS とカーナビゲーションシステムが挙げられる．

図 1 のとおり，i.MX7Dual[2] は Cortex-A7 と Cortex-M4 からなる HAP である．Cortex-A7 が HPC，Cortex-M4 が LEC にあたり，それぞれ GPOS と RTOS を動作させるのが一般的である．

また，i.MX7Dual では，CPU ごとにドメインという概念を持つ．これは，それぞれの CPU からアクセスできるメモリ領域を表しており，RDC（Resource Domain Controller）によって管理される．MDCOM では，排他処理を実現する SEMA4 と MU（Messaging Unit）が持つコア間割り込み機能，情報の送受信に利用される共有メモリを使用する．

2.2 コア間通信の機構

ヘテロジニアスプロセッサを用いたシステム開発における課題としてコア間通信が挙げられる．ヘテロジニアスプロセッサを利用してあるシステムを構築する場合，各プロセッサで協調処理が必要となり，コア間通信のしくみが必要となる．コア間通信を行う場合，共有メモリを利用する方法が容易である．その場合は，共有メモリの読み書きを行う際にロックを取得するようにしないとメモリの一貫性が保てない [3]．

3. 関連研究

マルチプロセッサ向けの通信仕様として，MCAPI[4]，MRAPI，OpenAMP がある．このうち，HAP を対象としている OpenAMP について説明する．

3.1 OpenAMP

3.1.1 概要

OpenAMP とは，Multicore Association が策定したコア間通信に関する仕様であり，実装がオープンソースとして公開されている [5]．OpenAMP は HPC で動作する Linux と，LEC で動かす FreeRTOS と呼ばれるリアルタイム OS の OS 間通信をサポートする．機能は 2 種類あり，1 つは remoteproc，もう 1 つは RPMsg である．

remoteproc はリモートプロセッサの制御を行う仕組みであり，Linux から FreeRTOS が動作するコア（リモートプロセッサ）を制御する機能である．具体的な制御としては，リモートプロセッサの電源のオン・オフ，ファームウェアのロードとスタートが挙げられる．

RPMsg はコア間通信を実現する機能であり，その概要を**図 2** に示す．通信はチャネルとエンドポイントを利用して行われる．ユーザーはあらかじめカーネルによって用意されているチャネルに対して，エンドポイントと呼ばれる通信インターフェースを作成し，それを通じて通信を行う．このとき，同時に通信できるチャネルは 1 本のみである．また，通信の開始や終了は，remoteproc と同様に Linux 側からのみ行うことができる．

RPMsg でサポートしている API を**表 1** に示す．印がついている API は OpenAMP で定められている標準の API である．データ送信は送信 API（rpmsg_send()）によって行う．一方，データ受信は，データ受信用のコールバック

Vol.2017-ARC-225 No.34
Vol.2017-SLDM-179 No.34
Vol.2017-EMB-44 No.34
2017/3/10

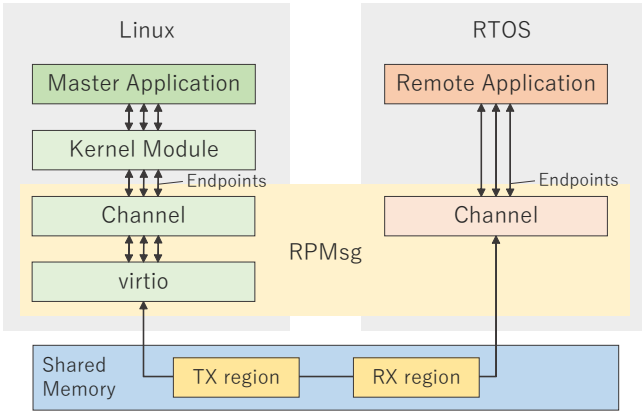

図 2: OpenAMP RPMsg の構成

表 1: OpenAMP の API

	API	機能
○	rpmsg_create_ept()	エンドポイントの作成
○	rpmsg_destroy_ept()	エンドポイントの破棄
○	rpmsg_send()	データを送信．空きバッファがない場合，15 秒後タイムアウト
○	rpmsg_trysend()	データを送信．空きバッファがない場合，ただちにエラーを返す
	rpmsg_rtos_alloc_tx_buffer()	送信バッファの割り当て
	rpmsg_rtos_send()	データを送信
	rpmsg_rtos_send_nocopy()	データを nocopy で送信

関数が用意され，データを受信するとその関数が呼ばれる．

RPMsg の実装に関しては，Linux が標準で備えている仮想マシン向けの送信機構である virtio を利用している．通信バッファには virtio による上限があり，これを超えた場合，通信に失敗する．現状公開されている実装では，OpenAMP の各機能はカーネルモジュールに対してのみ提供されている．

3.1.2 RTOS 向け拡張

前述の RPMsg の仕様は Linux に向けた仕様であり，リアルタイム OS で実現するには以下の課題がある．

- 割り込みコンテキストでのデータ処理
- ブロッキング受信 API が未実装
- ゼロコピー送受信が未実装
- RTOS でのタイムアウト受信の未実装
- Linux アップストリーム非互換

これらの課題を解決した RPMsg の API の拡張仕様とその実装が存在する．これを表 1 の印のついていない API として示す．

送受信関数は大きく分けて 2 種類ある．nocopy で行う関数と内部コピーを行う関数である．nocopy で送信を行う場合，あらかじめ rpmsg_alloc_tx_buffer() などを呼び，送信用のバッファを確保する必要がある．

3.1.3 OpenAMP の問題点

現状の OpenAMP には仕様と実装に以下の問題がある．

- 通信の開始や終了を Linux しか担えないこと
- エラー時のタイムアウト時間が 15 秒の固定値
- 同時に通信できるチャネルが 1 つのみ
- ユーザーアプリケーションレベルの通信ができない
- virtio による通信バッファサイズの制限

例えば，組込みシステムにおいてヘテロジニアスプロセッサを採用する場合，LEC にリアルタイム性の必要な処理，HPC にメディアなどの重たい処理を割り当てる場合がある．このとき，制御は信頼系に実行させる方が自然だが，仕様として，OpenAMP では RTOS での制御はできない．

コア間通信を行っている以上，非信頼系が停止したとしても信頼系は変わらず処理を続ける必要がある．しかし，OpenAMP では送信に失敗した場合，15 秒処理をブロックしたのち制御が戻る API があるため，問題がある．OpenAMP には様々な送信関数があり，即座に制御が戻る rpmsg_trysend() を利用すれば信頼系でのコントロールを失わないが，RTOS ではそのような API が実装されていないため，十分な実装ができているとは言い難い．

RPMsg の仕様では，同時に通信できるチャネルは 1 本のみである．そのため，複数のプロセス・タスク間で同時に通信を行うことはできない．しかし，アプリケーションはチャネルに対して複数のエンドポイントを扱うことができるので，1 対のプロセス・タスク間で複数の通信路を利用できる．

Linux で RPMsg を利用する場合，ユーザーアプリケーションでの利用はできず，カーネルモジュールとして実装する必要がある．そのため，実際のアプリケーションに応用するときには不都合が生じうる．virtio にはバッファ制限があり，512byte を超えたデータを送信すると通信に失敗する．そのため，Linux から 512byte を超えるデータを送信したい場合は，通信を複数に分割する必要がある．

4. MDCOM

前述の OpenAMP の問題を解決するため，我々は独自のヘテロジニアスプロセッサ向けの通信仕様と実装である MDCOM を開発した．

4.1 仕様

ドメインとは，システム上の OS を意味する．通信は，ドメインは単一の Master と 1 つ以上の Remote に分類される．Master は通信の開始や終了を制御を行い，一般的には LEC で動作する RTOS である．一方，Remote は HPC で動く Linux もしくは RTOS である．サポートする環境は，Master として TOPPERS カーネル，Remote として Linux と TOPPERS カーネルが使用できる．ドメイン間の通信は，共有メモリとコア間割り込みとコア間排他制御

表 2: MDCOM の API

API	機能
mdcom_init()	MDCOM の初期化
mdcom_term()	MDCOM の終了
mdcom_sense_process()	Remote のプロセス数を取得
mdcom_sense_fifoch()	FIFOCH の使用状況の取得
mdcom_fifoch_alloc()	FIFOCH からブロックを割当て
mdcom_fifoch_free()	FIFOCH へのブロックの解放
mdcom_fifoch_get()	FIFOCH のブロックのポインタの取得
mdcom_fifoch_enqueue()	FIFOCH の FIFO キューへのブロック ID の送付
mdcom_fifoch_dequeue()	FIFOCH の FIFO キューからのブロック ID の取得
mdcom_fifoch_notify()	FIFOCH のイベント送付
mdcom_fifoch_wait()	FIFOCH のイベント受信
mdcom_filter_set()	FIFOCH へフィルターをセット
mdcom_fifoch_init()	FIFOCH の再初期化
mdcom_smemch_get()	SMEMCH の共有メモリのポインタの取得
mdcom_smemch_trylock()	SMEMCH のロック取得の試行
mdcom_smemch_unlock()	SMEMCH のロックの解放

機構を利用して行われる．この時，メッセージや MDCOM の制御情報は共有メモリ上に配置される．

MDCOM では，チャネルと呼ばれるオブジェクトによりコア間の通信を実現する．チャネルは以下の 2 種類が用意されている．

- SMEM チャネル
- FIFO チャネル

それぞれのチャネルの詳細については後述する．これらのチャネルを用いて，複数のプロセスと複数のタスクの間で同時に通信することができる．さらに，Linux においてはユーザープロセスに対してドライバとライブラリの組み合わせでこれらの機能を提供する．

MDCOM のコンフィギュレーションは専用の API を記述し，コンフィギュレータに読ませることで生成される．

また，MDCOM の API を**表 2** に示す．ただし，この表においてチャネルを CH と略すことにする．

4.1.1 SMEM チャネル

SMEM チャネルは共有メモリを提供するチャネルである．**図 3**(a) にあるように通信に利用する領域に対してロックを取得し，共有メモリの読み書きを行い，ロックを開放することで他のドメインにアクセスさせる．

4.1.2 FIFO チャネル

FIFO チャネルは，FIFO を利用した通信を提供するチャネルである．図 3(b) のように，メッセージの読み書きは Block を使用して行い，FIFO キューに Block の ID を送付することによって，複数のメッセージを送信することができる．SMEM チャネルの場合は，単一のメッセージしか送受信できない．つまり，FIFO チャネルと SMEM チャネルは，一度に送信できるメッセージの数が異なる．

また，ユーザーアプリケーションでの通信を低減する仕組みとしてフィルターがある．特定の条件を満たしたときに初めて通信を行うことで通信を低減する．例えば，偶数回目の通信要求を受けた時，通信を行うなどである．

4.1.3 基本的な通信フロー

MDCOM の基本的な通信フローを**図 4** に示す．図において，Master と Remote 間の矢印は共有メモリを利用した通知とその方向を表す．

まず, Master と Remote とそれぞれで mdcom_init() を呼び，MDCOM の初期化を行う．このとき，初期化は Master から行われるが，関数呼び出しのタイミングはどちらが先でも構わない．

次に，FIFO チャネル上に送信用の領域を確保する．これを実現するのが，mdcom_fifoch_alloc() である．この関数を実行すると FIFO チャネル上に Block を確保し，Block の ID が分かる．

メッセージの書き込みを行うためには，Block のポインタを取得するために，mdcom_fifoch_get() を実行する．

Block に対して memcpy() などでメッセージを書き込んだのち，Remote が Block の ID を受け取れるように mdcom_fifoch_enqueue() を呼び出し，FIFO キューへ Block の ID を入れる．

その後，mdcom_fifoch_notify() を実行することで，Remote へとイベントの通知を行う．

イベントの通知を受け取るため，Remote ではあらかじめ mdcom_fifoch_wait() を実行しなくてはならない．このとき，待ち時間は任意時間，無限待ち，ポーリングから選択できる．

通知を受け取ったら，mdcom_fifoch_dequeue() を呼び，FIFO キューから Block の ID を受け取る．Master 同様に，mdcom_fifoch_get() と memcpy() などのメモリアクセス関数でメッセージを受け取ることができる．

最後に，Remtoe から Master に対してイベントの通知を行い，送信が完了したことを伝える．

4.2 実装

MDCOM の実装は，特定のアーキテクチャに依存しない共通部と，アーキテクチャに依存したターゲット依存部に分かれている．

4.2.1 共通部

ファイル構成としては，ユーザー向けインクルードファイル，ライブラリ本体とハードウェア依存部，Linux では加えてキャラクタデバイス，カーネルモジュールが実装されている．Linux における通信の仕組みとしては，ユーザーが API を叩くと，キャラクタデバイスを通してカーネルモジュールに命令が渡され，共有メモリへの読み書きが行わ

Vol.2017-ARC-225 No.34
Vol.2017-SLDM-179 No.34
Vol.2017-EMB-44 No.34
2017/3/10

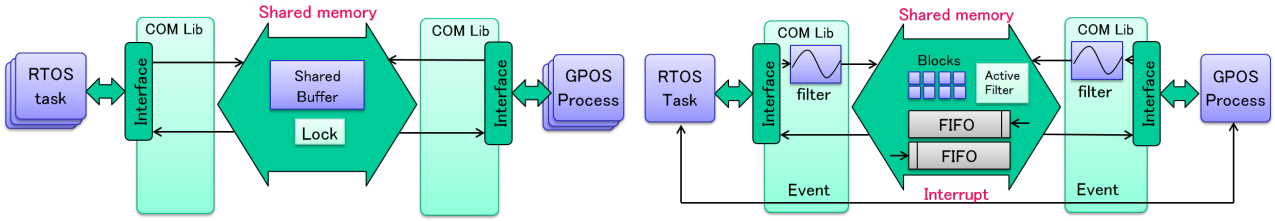

図 3: チャネルの設計

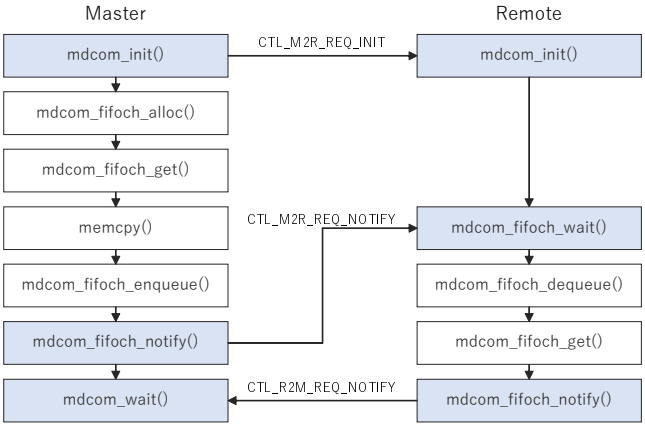

図 4: MDCOM の通信フロー

れる．TOPPERS カーネルでは，ユーザー API が低レベル API を呼び出し，ハードウェアを直接制御する．

4.2.2 ターゲット依存部（i.MX7Dual）

ターゲット依存部の例として i.MX7Dual の依存部を説明する．i.MX7Dual のターゲット依存部では，図 1 のうち，排他処理を実現する SEMA4 と MU（Messaging Unit）が持つ割り込み機能，情報の送受信のために共有メモリを使用する．

SEMA4 は排他のための仕組みで，それぞれの CPU が”Hardware gate”に対してロックを取得することにより排他を実現する．”Hardware gate”は 16 個あり，MDCOM ではそのうちの 1 つを利用する．

Messaging Unit は，i.MX7Dual が持つハードウェアでプロセッサ間通信を実現する．メッセージを受信すると CPU に対して割り込みをかける機能があり，MDCOM ではこれを利用する．

また，共有メモリはメッセージの送受信に使われるが，加えて，MDCOM の状態を記憶したり，mdcom_init() や mdcom_fifoch_notify() などを実行した際に，相手の CPU に対して通知を行うためにも使用される．

4.3 OpenAMP との定性的な比較

OpenAMP との定性的な比較を**表 3** に示す．

まずリアルタイム性に関しては，OpenAMP では送信キューに空きがないとき，Linux やベアメタルにおいては，rpmsg_trysend() 関数を利用することにより即座にエラーが返るが，RTOS 向けの拡張 API ではブロックしタイムアウトするのは 15 秒後であり，リアルタイム性が低い．一方，MDCOM では任意のタイムアウト時間を指定して API を実行することが可能である．

次に，同時通信に関しては，OpenAMP では同時に 1 本のチャネルしか作成できないのに対して，MDCOM では，任意のチャネルを作成することができる．

OpenAMP での下位レイヤーの実装は virtio を利用している．そのため，バッファの最大数が 512byte となっている．一方，MDCOM にはこのような制約はなく，任意のサイズにチャネルをコンフィギュレーションすることが可能である．

実行環境は，OpenAMP は Linux と RTOS の組み合わせのみをサポートしているのに対して，MDCOM は，RTOS と RTOS の組み合わせもサポートしている．

Linux における API の提供先は，OpenAMP ではカーネルモジュールであるのに対して，MDCOM はユーザープログラム API を提供する．

5. 評価

実行環境は，MDCOM では，HPC で Linux，LEC で TOPPERS カーネルを実行した．また，RPMsg では，HPC で Linux，LEC で FreeRTOS とした．

5.1 API 実行時間

MDCOM の API 実行時間を**表 4** に示す．また，TOPPERS カーネルの API 実行時間は，act_tsk()，wup_tsk()，chg_pri() の実行時間を測定し，最頻値はそれぞれ，900，500，1000nsec であった．

他の関数と比較して，mdcom_fifoch_enqueue() と mdcom_fifoch_dequque() の実行時間が長く，FIFO チャネルを使用する場合，これらの関数を必ず使用する．一方で，SMEM チャネルを使用する場合は，その必要はない．つまり，FIFO チャネルと SMEM チャネルは，実行速度と複数のメッセージを送れるかという点でトレードオフといえる．

Vol.2017-ARC-225 No.34
Vol.2017-SLDM-179 No.34
Vol.2017-EMB-44 No.34
2017/3/10

表 3: MDCOM と OpenAMP の比較

	OpenAMP	MDCOM
リアルタイム性	RTOS では処理をブロックし，15 秒でタイムアウト	即座にエラーが返る
同時通信	1 本のみ	複数チャネルで同時に通信できる
下位レイヤーの実装	Linux では virtio を利用，RTOS では自前で実装	自前で実装
実行環境	Master: Linux / Remote: RTOS，ベアメタル	Master: RTOS / Remote: RTOS，Linux
API の提供先（Linux）	カーネルモジュール	ユーザープログラム

表 4: MDCOM の API 実行時間 (nsec)

API	最頻値	平均	標準偏差
mdcom_fifoch_alloc()	1800	1778	61
mdcom_fifoch_free()	700	669	28
mdcom_fifoch_get()	600	618	48
mdcom_fifoch_notify()	1000	928	29
mdcom_fifoch_enqueue()	1800	1806	68
mdcom_fifoch_dequeue()	1900	1823	77

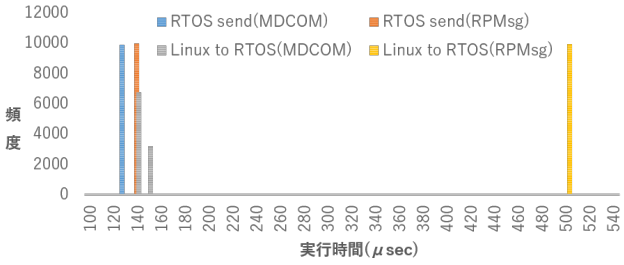

図 5: 通信時間の評価

5.2 RTOS 側の送信時間

MDCOM における RTOS 側の送信時間を図 **5** の RTOS send(MDCOM) に示す．また，RPMsg における RTOS 側の送信時間を図 5 の RTOS send(RPMsg) に示す．ただし，測定には送信バッファを確保する処理を含めていない．

測定した範囲は，それを除いた，RTOS 側で送信完了するまでに必要な処理である．すなわち，MDCOM は送信バッファの書き込みを完了し，FIFO キューにブロック ID を送付したのちにイベントの通知を行うまで，RPMsg は送信バッファの書き込みを完了し，nocopy 送信関数を呼ぶまでである．これは，RPMsg に確保した送信バッファを開放する関数が存在しないためである．

5.3 Linux 送信から RTOS 受信までの時間

MDCOM における Linux 送信から RTOS 受信までの時間を図 5 の Linux to RTOS(MDCOM) に示す．また，RPMsg における Linux 送信から RTOS 受信までの時間を図 5 の Linux to RTOS(RPMsg) に示す．このとき，RPMsg では拡張によって追加された nocopy 送受信関数ではなく，標準の送受信関数を利用した．さらに，MDCOM はユーザープログラムとして，RPMsg はカーネルモジュールとして実装した．

RPMsg に比べて，MDCOM の方が送信時間にばらつきが生じた．これは，プログラムの実行優先度の違いによるものと考えられる．

また，RPMsg に比べて MDCOM が約 3.6 倍速い結果になった．これは，RPMsg 内部でデータのコピーが複数回発生したためだと考えられる．

6. おわりに

本論文では，MDCOM の評価と既存技術である RPMsg の比較を行った．評価内容は，MDCOM の API 実行時間，TOPPERS カーネルの API 実行時間，MDCOM と RPMsg の RTOS 側が送信完了までの時間，Linux から RTOS への送信時間であった．

評価により，各チャネルのトレードオフの関係性，RTOS 側の送信完了までの時間，Linux から RTOS への送信時間についての MDCOM の優位性を示した．

参考文献

[1] F. Baum, A. Raghuraman, ”Making Full use of Emerging ARM-based Heterogeneous Multicore SoCs”, Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Jan 2016.
[2] i.MX7D: i.MX 7Dual Processors - Heterogeneous Processing with dual ARM Cortex-A7 cores and Cortex-M4 core
[3] O. Tomoutzoglou, D. Mbakoyannis, G. Kornaros, M. Coppola, ”Efficient Communication in Heterogeneous SoCs with Unified Address Space”, Proc. Int. Symp. on Reconfigurable Communication-centric Systems-on-Chip - ReCoSoC, May 2016.
[4] Thiago Raupp da Rosa, T. Mesquida, R. Lemaire, F. Clermidy, ”MCAPI-compliant Hardware Buffer Manager mechanism to support communication in multi-core architectures”, Design, Automation & Test in Europe Conference & Exhibition (DATE), March 2016
[5] OpenAMP 入手先 ⟨https://github.com/OpenAMP/open-amp⟩（参照 2016-01-24）