情報処理学会論文誌 Vol.59 No.8 1464–1476 (Aug. 2018)

組込みミックスドクリティカルシステム向けイーサネットコントローラ共有機構

石野 正敏[1,a)] 本田 晋也[1]

受付日 2017年11月16日, 採録日 2018年5月10日

概要：近年，自動車等の組込みシステムの多機能化が進み，求められる信頼度が異なる機能が混在しているシステムが増加している．信頼度の異なる機能間は影響を防ぐため，パーティショニングが必要となる．プロセッサやメモリに関しては，RTOS の提供するメモリ保護機能等により，パーティショニングすることが可能である．一方，周辺回路（デバイス）に関しては，信頼性確保のために機能ごとに用意し，それぞれの機能に占有して使用させる方法が一般的である．しかしながら，ネットワークデバイス等はコスト等の問題で機能ごとに用意することがきわめて困難であり，信頼度の異なる機能間で共有する必要がある．そのため，単一のデバイスを信頼度の異なる機能間で安全に共有する手法が必要となる．本論文ではイーサネットコントローラを機能間で安全に共有する手法を提案する．提案手法では，既存のイーサネットコントローラに対して，ProtectionWrapper というデバイス保護機構の追加および共有のためのハードウェア拡張を行うことで，パーティショニングを維持したまま，それぞれの機能が直接イーサネットコントローラを操作して送受信を行うことを可能とした．提案手法を既存のイーサネットコントローラに対して適用し評価した結果，それぞれの機能に対してイーサネットコントローラを割り付けた場合と比較して，ハードウェア面積が 23%の減少となり，実行オーバヘッドを 1.7%に抑えられた．

キーワード：組込みシステム，ミックスドクリティカル，車載制御システム，機能安全

Ethernet Controller Sharing Mechanism in Embedded Mixed-criticality Systems

Masatoshi Ishino[1,a)] Shinya Honda[1]

Received: November 16, 2017, Accepted: May 10, 2018

Abstract: In recent years, systems having different reliability functions are increasing because of an increase in functions in cars. In such systems, partitioning is necessary in order to prevent the influence between functions having different reliability. Regarding the processor and memory, partitioning can be realized by using the memory protection function of RTOS. On the other hand, regarding peripheral devices, it is common to prepare for each function to ensure reliability. However, it is difficult to prepare network devices for each function due to its cost, therefore, it is necessary to share network devices between functions having different reliability. Hence, it is necessary to have a method for securely sharing a single network device between such functions. In this paper, we propose the method to share Ethernet controller between functions. We applied the device protection mechanism called Protection Wrapper to the existing Ethernet controller, and expand controller for sharing, then it becomes possible for each function to use one Ethernet controller independently. As a result of using the method we propose, compared to the case of assigning Ethernet controller to each function, echo back time was increased by 1.7%, and hardware area was decreased by 23%.

Keywords: embedded systems, mixed-criticality systems, automotive systems, functional safety

1. はじめに

近年，自動車は多機能化が進み，求められる信頼度の異なる機能が混在しているシステムが増加している．このよ

[1] 名古屋大学大学院情報学研究科
Graduate School of Informatics, Nagoya University, Nagoya, Aichi 464–8601 Japan
[a)] ishino@ertl.jp

うに求められる信頼度が異なる機能が混在しているシステムをミックスドクリティカルシステムという [1], [2]．システムの機能はソフトウェアによって実現され，そのソフトウェアは電子制御ユニット（ECU）上で実行される．機能を追加するごとに ECU を追加するとその数が膨大になり，限られた自動車の空間内では ECU を搭載するスペースが不足してしまう．そのため，将来的にマルチプロセッサを利用する等して，複数の機能を 1 つの ECU に実現したいという要求がある．しかし，そのような ECU では信頼度の異なる機能が混在することになる．

特に，高い信頼性が求められる機能に関しては，ISO26262 に基づく安全規格に従う設計が必要である．安全規格に従うために，求められる信頼度が異なる機能を信頼度ごとにパーティショニングするような手法がとられている．シングルプロセッサであればメモリ保護機能付きの RTOS を用いることによって，使用するメモリ領域を機能ごとに制限することでパーティショニングすることが可能である．マルチプロセッサであれば信頼度の異なる機能を別々の CPU で実行し，使用するメモリ領域をバス機構等で分離することによってパーティショニング可能である．

汎用システムと異なり，車載システムは多くのデバイスを制御する必要があり，デバイスについては，信頼度ごとに割り付けて使用する方法が最も容易であるが，ハードウェア面積や消費電力といった点から共有せざるを得ない場合がある．その最たる例としてネットワークデバイスがあげられ，複数のネットワークデバイスを使用すると，前述の点に加えて，ケーブルやスイッチのポートの数がより多く必要になるという問題がある．

信頼度の異なる機能間でネットワークデバイスを共有する方法として，それぞれの機能が直接ネットワークデバイスを操作する直接操作方式があげられるが，この方式では，低信頼機能によって高信頼機能の信頼性が損なわれてしまう危険性が存在する．そのため，高信頼機能のみがネットワークデバイスにアクセス可能とし，低信頼機能がデバイスを使用したい場合は処理を高信頼機能に依頼する方式（依頼方式）が一般的手法としてある．しかし，この方式では，高信頼機能が低信頼機能からの処理依頼を受け取る必要があることや，高信頼機能と低信頼機能間でのデータの受け渡し処理等が必要になるため実行オーバヘッドが増加してしまう．そのほかにも高信頼機能が低信頼機能からの連続的な送信依頼により DoS 攻撃を受ける可能性があるという問題がある．

これらの問題を解決し，車載システム向けのネットワークコントローラの一種である CAN コントローラを複数の機能間で安全に共有する方式がこれまで検討されている [3], [4], [5]．また，我々も文献 [6] において，直接操作方式において機能間のパーティショニングを実現する方法として，ProtectionWrapper（PWrap）と呼ぶアクセス保護機構を提案し，CAN コントローラに適用してきた．

本論文では，昨今の車載システムにおけるイーサネットによるネットワークの需要が高まっていることから [7], [8]，直接操作方式によってイーサネットコントローラを共有しつつパーティショニングを実現する機構を提案する．直接操作方式では，各機能がイーサネットコントローラに直接アクセスして送受信を行う．そのため，低信頼機能がイーサネットコントローラを不正に操作しないように，イーサネットコントローラに PWrap を適用しアクセス制御を行う．しかしながら，CAN コントローラと異なり，イーサネットコントローラは PWrap の適用だけでは，パーティショニングされた直接操作方式を実現できなかったため，イーサネットコントローラを拡張した．機能拡張したイーサネットコントローラを用いた直接操作方式を実現し，実行オーバヘッドやハードウェア面積を評価し，提案手法の優位性を提示する．

本論文のコントリビューションは以下のとおりである．

- 既存のイーサネットコントローラを依頼方式や直接操作方式で共有する場合の問題点を提示．
- 直接操作方式でイーサネットコントローラを安全に共有するための機構の提案．

2. 前提と要件

本論文で対象とするシステムの前提と，イーサネットコントローラの共有に対する要件について説明する．

2.1 システムの前提

本論文で扱うシステムの前提について説明する．

（**前提 1**）システムは，1 つの高信頼機能と 1 つの低信頼機能という 2 つの機能で構成されている．

（**前提 2**）それぞれの機能は独立したバイナリとして実行される．

（**前提 3**）それぞれの機能は独立した IP アドレスを持つ．

（前提 1）に関しては，組込みシステムは一般にリソース制約が厳しく，3 個以上の異なる信頼度の機能を持つことはまれである．そのため，信頼性の高い高信頼機能と低い低信頼機能を 1 つずつ持つことを前提とする．このようなシステムを実現する方法として**図 1** に示すような実現方法が考えられる．1 つはシングルプロセッサで信頼度の異なる機能ごとにパーティショニングして実現する方法で，もう 1 つはマルチプロセッサで CPU ごとに信頼度の異なる機能を実現する方法である．本論文では両方のケースを想定するが，説明を簡略化するため，これ以降は図 1 (2) のマルチプロセッサを例に説明する[*1]．マルチプロセッサにおいて，高信頼機能を実行する CPU を高信頼 CPU，低信

[*1] 物理的に分割されているか，仮想化等により論理的に分割されているだけの違いであるため，提案手法は両方のケースに適応できるため．

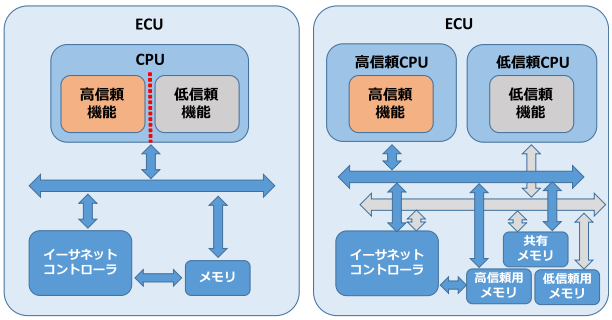


図 1 信頼度の異なる機能の実現方法
Fig. 1 How to realize system having different reliability.

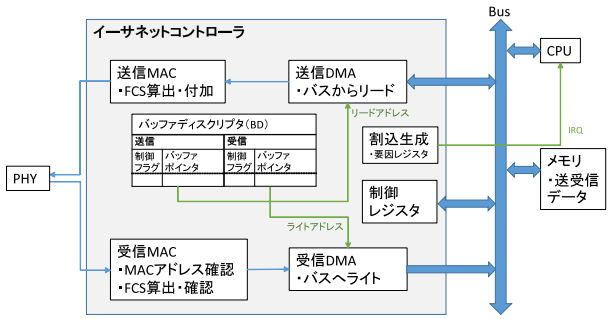


図 2 イーサネットコントローラの構成
Fig. 2 Design of Ethernet Controller.

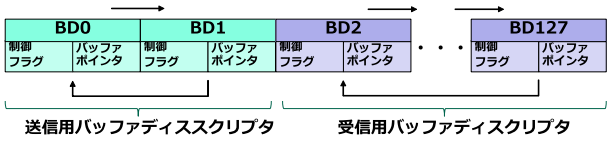


図 3 バッファディスクリプタ（BD）の構成
Fig. 3 Sets of Buffer Descriptor.

頼機能を実行する CPU を低信頼 CPU と呼ぶ．

（前提 2）に関しては，それぞれの機能は異なる安全度水準で開発する必要があるため，独立性を高める目的で独立したバイナリとして実行する．

（前提 3）に関しては，それぞれの機能は独立しており，受信するフレームも共有しないため，別々の IP アドレスを割り付けて，ネットワーク外からは異なるノードとして見えるようにする．

2.2 イーサネットコントローラの共有に対する要件

前述のシステムにおいて，イーサネットコントローラを機能間で共有する際の要件について説明する．

（**要件 1**）低信頼機能の動作不良により，高信頼機能の送受信が阻害されないこと．

（**要件 2**）イーサネットコントローラを共有することにより生じる実行オーバヘッドの増加を抑えること．

（**要件 3**）高信頼機能と低信頼機能間でのインタラクションを少なくすること．

（**要件 4**）低信頼機能がネットーワーク帯域を占有しないこと．

（**要件 5**）プロトコルスタックとドライバの変更量を少なくすること．

（要件 1，3）はパーティショニングの実現のために定めた．（要件 2）は，実行オーバヘッドが増加すると同性能の機能を実現するために，高性能な CPU が必要となり，コストの増大を招くためである．（要件 4）はシステム全体の安全性を担保するために必要である．なお，車載システムのネットワークはスイッチを用いたスター型であり，構成は静的であるため，外部から送られてくるフレームに対してはイーサネットスイッチの機能で，低信頼宛のフレームで帯域を占めないように送信頻度を抑制することによって，制御が可能である．そのため，本研究では，高信頼 CPU からの送信に対する帯域保証についてのみ扱う．（要件 5）は，開発工数を増大させないために定めた．

イーサネットコントローラを 2 つ用意して，高信頼 CPU と低信頼 CPU にそれぞれ占有させた場合は上記の要件のうち（要件 1，2，3，5）は満たすが，低信頼 CPU が割り当てられたイーサネットコントローラを自由に使用可能であるため，（要件 4）については充足しない．また，そのほかにも消費電力やハードウェア面積の増大，ケーブルやスイッチのポート等が倍必要となり，コストが増加してしまう問題がある．

3. ハードウェア仕様

この章では，本論文で使用したイーサネットコントローラと ProtectionWrapper（PWrap）について説明する．

3.1 イーサネットコントローラ

OpenCores で公開されている “Ethernet MAC 10/100 Mbps” [9] を利用した．ハードウェア構成は**図 2** のようになっており，送受信フレームはイーサネットコントローラの外部のメモリに保持され，バッファディスクリプタ（BD）によって管理される．以降は本研究に関連する仕様について説明する．

3.1.1 バッファディスクリプタ（BD）

BD は送受信の際に使用する機構で，128 個あり，**図 3** に示すように，0 から任意の数までの BD を送信用，残りを受信用に割り付けることが可能である．BD は順番かつ循環して使用する機構となっている．図 3 の割当ての例では，送信では，まず BD0 を使用し，次には BD1 が使われ，その次は BD0 が使用される．受信では，フレームを受信するとまず BD2 が使われ，次に BD3 が使われ，BD127 が使われると，BD2 が再び使われる．

各 BD は制御フラグ部とバッファポインタ部で構成されている．バッファポインタ部では送受信のフレームを格納するメモリアドレスを保持する．制御フラグ部では送受

信の制御と状態を管理しており，送信用の BD であれば READY フラグがあり，セットすると，イーサネットコントローラがバッファポインタの指すメモリアドレスから送信フレームを読み込んで送信する．受信用の BD であれば EMPTY フラグがあり，イーサネットコントローラが BD のバッファポインタの指すメモリアドレスに受信フレームを書き込んだらフラグに 0 をセットする．

3.1.2 制御レジスタ

制御レジスタには送受信開始/停止を設定するレジスタや送信 BD 数を設定するレジスタ等が存在する．これらのレジスタ設定値が不正に書き換わると通信に問題が発生する可能性があるため，低信頼機能からのアクセスを制限する必要がある．

3.1.3 割込み

割込みは 1 本あり，受信フレームを受信して外部メモリに書き込んだ後に割込みを発生させる．

3.1.4 プロトコルスタック

プロトコルスタックとしては，オープンソースの uIP [10] を使用した．本論文に関連する uIP による初期化処理および送受信処理を示す．

- BD に関する初期化処理
  (1) 送信/受信 BD 数の設定を行う．
  (2) 各送信/受信 BD について，バッファポインタに各 BD で使用するメモリアドレスを割り当てる．
  (3) 各送信/受信 BD について，制御フラグの値を設定し，BD を使用可能にする．
  (4) 送受信フレームを格納するメモリをクリアする．
- 送信処理
  (1) 使用する BD のバッファポインタの指し示すアドレスに，送信フレームを書き込む．
  (2) 制御フラグの READY フラグに 1 をセットする．
  (3) イーサネットコントローラがフレームを送信完了する（READY フラグを 0 にする）のを待つ．
  (4) 制御フラグの READY フラグが 0 になったのを確認して送信終了する．
- 受信処理
  (1) 受信割込みをイーサネットコントローラから受け取り，受信処理を開始する．
  (2) 使用する受信 BD の制御フラグの EMPTY フラグが 0 であることを確認する．
  (3) BD のバッファポインタの指し示すアドレスに格納されているフレームを読みこんで受信する．
  (4) 受信が完了したら，BD の制御フラグの EMPTY フラグに 1 をセットする．

3.2 ProtectionWrapper（PWrap）

PWrap は文献 [6] で提案したデバイス保護機構であり，CPU からデバイスのデバイスレジスタへのアクセス制御を行うハードウェア機構である．PWrap を用いることにより，デバイス自身を変更することなく，アクセス保護を実現可能である．

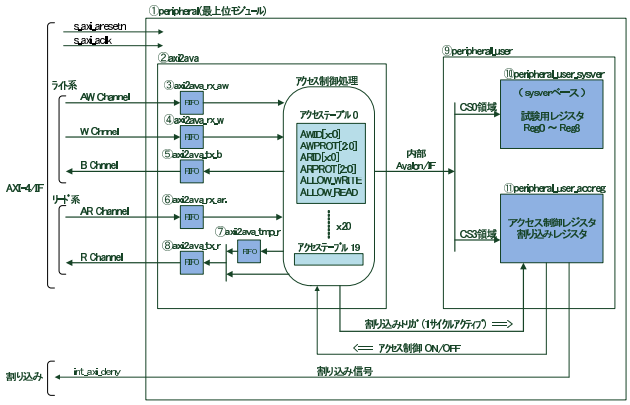


図 4 ProtectionWrapper のハードウェア構成
Fig. 4 Design of Protection Wrapper.

3.2.1 ハードウェア構成

PWrap のハードウェア構成を**図 4** に示す．アクセス制御レジスタで保持されるアクセステーブルの設定によって，アクセス制御処理でアクセスを制御する．アクセステーブルは複数あり，それぞれ連続したアドレス領域（リージョン）へのアクセス制御を設定可能である．AXI バスから送られてくるデータにはアクセス元の ID が付与されており，この ID とアクセス先のアドレスに対応するアクセステーブルを比較することで，アクセス可能/不可能の制御やアクセス頻度の制限を行うことができる．アクセス違反が起きた際にはアクセス元にアクセス違反を割込みで通知する．

3.2.2 アクセステーブル

アクセステーブルの設定はアクセス制御レジスタで設定する．アクセスを許可/拒否の判定の基準となるアクセス属性には READ/WRITE のほか，通常/特権アクセス，セキュア/非セキュアアクセス，データ/命令アクセス，アクセス元パーティション，アクセス周期，書き込み値マスク，書き込み値範囲を使用可能である．

4. 依頼方式

ミックスドクリティカルシステムでイーサネットコントローラを共有する手法として既存のイーサネットコントローラを使用した依頼方式について説明し，この方式の要件充足を評価する．

依頼方式は低信頼 CPU が送受信を行う際に，高信頼 CPU に処理を依頼する方式である．ハードウェア構成を**図 5** に示す．低信頼 CPU はイーサネットコントローラと接続されておらず，直接操作することができない．低信頼 CPU が通信を行う際には，送受信フレームを高信頼 CPU と低信頼 CPU の間で受け渡す処理が必要になるため共有メモリが用意されている．高信頼用メモリは送受信フレームが格納されるため，高信頼 CPU だけでなくイーサネッ

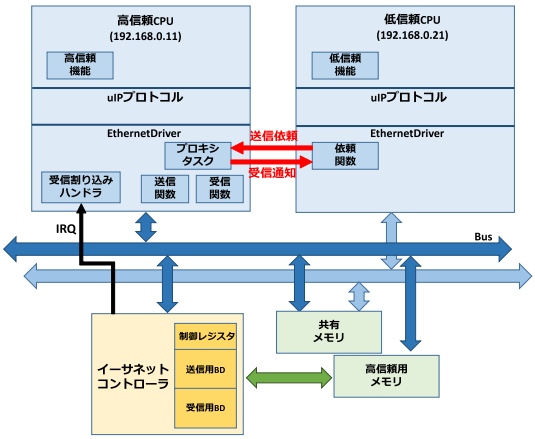


図 5 依頼方式のハードウェア構成

Fig. 5 Hardware design using conventional device sharing method.

トコントローラからもアクセスされる．以降に具体的な低信頼 CPU の送受信の手順を述べる．

4.1 低信頼 CPU の送信方法

(1) 低信頼 CPU が共有メモリに送信フレームを書き込む．
(2) 低信頼 CPU が高信頼 CPU に割込みを入れて送信を依頼する．
(3) 高信頼 CPU のプロキシタスクが起床し，低信頼 CPU の送信フレームを共有メモリから高信頼用メモリにコピーして，送信関数を呼び出して送信を行う．

4.2 低信頼 CPU の受信方法

(1) イーサネットコントーラが受信したフレームを高信頼用メモリに格納する．
(2) 高信頼 CPU がイーサネットコントローラから受信割込みを受け，受信関数により高信頼用メモリにある受信フレームを確認する．
(3) 受信フレームの宛先 IP アドレスを確認し，低信頼 CPU 宛もしくはブロードキャストであれば受信フレームを高信頼用メモリから共有メモリにコピーする．
(4) 高信頼 CPU のプロキシタスクが低信頼 CPU に割込みで受信を通知する．
(5) 低信頼 CPU が共有メモリに格納された受信フレームを読み込み受信する．

4.2.1 要件充足

前述の依頼方式により，2 章であげた要件を充足するか確認する．

（**要件 1**）低信頼 CPU がイーサネットコントローラにアクセスできないことから，低信頼 CPU がイーサネットコントローラを不正に操作する可能性がないために要件を満たす．

（**要件 2**）低信頼 CPU の送受信の際に高信頼 CPU に処理を依頼する必要があり，その際に実行オーバヘッドが生じてしまうため要件を満たさない．

（**要件 3**）低信頼 CPU が送受信を行うたびに，高信頼 CPU とやりとりを行う必要があり，インタラクションの頻度が高い．

（**要件 4**）低信頼 CPU の送信処理はすべて高信頼 CPU に依頼するため，高信頼 CPU によって，低信頼 CPU の送信頻度を制限することが可能であり，低信頼 CPU によってネットワーク帯域が占有されることはなく，要件を満たす．

（**要件 5**）受信処理の際に宛先 IP アドレスの確認処理が高信頼 CPU の受信関数内で必要になるため，若干のソフトウェア変更が必要になる．

このように依頼方式では，（要件 2，3）を満たすことができない．どちらの要件についても，低信頼 CPU の処理を高信頼 CPU が代行することに起因しており，依頼方式では満たすことができない．

5. 直接操作方式

前述のとおり，依頼方式では（要件 2，3）を満たすことができない．そのため，直接操作方式によってイーサネットコントローラを共有する方式の検討を行った．直接操作方式はそれぞれの CPU がイーサネットコントローラを直接操作して送受信を行う方式である．しかしながら，単に低信頼 CPU がイーサネットコントローラにアクセスできるようにしただけでは安全な共有は実現できない．安全な共有を実現するため，次の 2 ステップでハードウェア拡張を実施した．

- イーサネットコントローラへの PWrap の適用
  文献 [6] の CAN コントローラと同様に PWrap を適用してアクセス制御を行う．
- イーサネットコントローラの機能拡張
  CAN コントローラとは異なり，PWrap の適用だけでは満たせない要件があるため，イーサネットコントローラの機能を拡張する．

2 ステップに分けた理由は，CAN コントローラの場合は PWrap の適用のみで安全な共有が実現できたため，イーサネットコントローラに対しても同様に PWrap の適用のみでどの程度要件を満たせるか検討するためである．そのうえで，満たせない要件に関して，イーサネットコントローラを拡張して対応する．

以降ではまず，既存のイーサネットコントローラを用いた場合の問題点を明らかにし，前述の 2 ステップの拡張と問題点への対応について説明する．

5.1 既存のイーサネットコントローラによる直接操作方式

既存のイーサネットコントローラを用いた直接操作方式のハードウェア構成を**図 6** に示す．低信頼 CPU がイーサネットコントローラを利用できるようにバスで接続する．

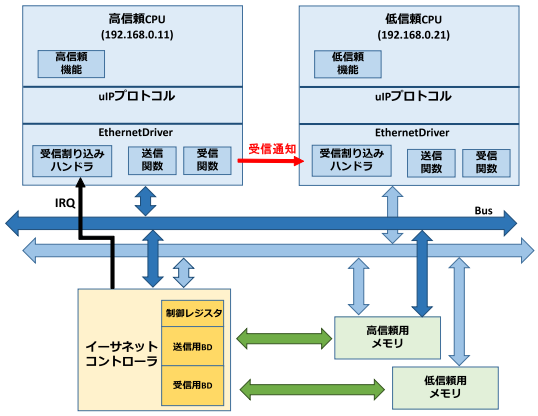


図 6 既存のイーサネットコントローラを用いた直接操作方式のハードウェア構成

Fig. 6 Hardware design using proposing device sharing method with existing ethernet controller.

イーサネットコントローラが使用するメモリは，高信頼用と低信頼用の 2 種類を持つが，後述の理由により，低信頼 CPU が高信頼用メモリにアクセスできるよう接続する．割込みは高信頼 CPU に入力する．

5.1.1 送信方法

各 CPU が同一の機能を持つ個別の送信関数を呼び出し，BD を直接操作して送信を行う．その際，各 CPU が同時に同じ BD を使用しないように，スピンロックによる排他制御を実施する．送信フレームはそれぞれのメモリに用意して，そのアドレスを BD に指定することで，送信を行う．

5.1.2 受信方法

初期化時に，高信頼用メモリに受信フレームを格納する領域を設けて，そのアドレスを受信 BD のバッファポインタに設定する．これは，受信時にどちらの CPU 宛の受信フレームか，ハードウェアで判断する方法がないためであり，割込みを受け付ける高信頼用 CPU のメモリに受信するようにする．

以下に具体的な受信処理の手順を示す．

(1) イーサネットコントーラが受信したフレームを高信頼用メモリに格納する．
(2) 高信頼 CPU がイーサネットコントローラから受信割込みを受け，受信関数を呼び出す．
(3) 受信関数内で宛先 IP アドレスを確認し，ブロードキャストもしくは低信頼 CPU 宛であれば低信頼 CPU に割込みで受信を通知する．
(4) 低信頼 CPU が割込みを受けて受信関数を呼び出し，BD のバッファポインタで設定されているメモリアドレス（高信頼用メモリ）から受信フレームを読み込んで受信する．受信が完了したら制御フラグを操作して受信完了に設定する．

このように，受信したフレームを低信頼 CPU が読み込めるように，高信頼用メモリは低信頼 CPU からアクセス可能とする必要がある．受信時に高信頼 CPU が低信頼用メモリにコピーする方法もあるが，依頼方式と同様となり，実行オーバヘッドが増加するため，採用しない．

5.1.3 要件充足

既存のイーサネットコントローラによる直接操作方式の要件充足について述べる．

（**要件 1**）この要件は直接操作方式を用いることによって満たさなくなる．その原因として次の 4 つの問題点があげられる．

1 つ目は，イーサネットコントローラの制御レジスタに低信頼 CPU からも自由にアクセスできるという問題である（**問題点 1-1**）．低信頼 CPU が不正にイーサネットコントローラの制御レジスタ等を書き換えることにより，高信頼 CPU の通信が阻害される可能性がある．

2 つ目は，BD を信頼度の異なる機能間で共有している問題である（**問題点 1-2**）．低信頼 CPU が BD の制御フラグを不正に書き換えて使用できないようにすると，高信頼 CPU の通信が阻害される可能性がある．

3 つ目は，低信頼 CPU が受信 BD に高信頼 CPU が使用するメモリのアドレスを指定した場合，受信時にその BD が使用されると，高信頼 CPU が使用するメモリを破壊することや，送信 BD に高信頼 CPU が使用するメモリアドレスを指定して，高信頼 CPU のデータをネットワークに送ることが可能であるという問題である（**問題点 1-3**）．この問題は，PWrap により，低信頼 CPU から受信用 BD のバッファポインタに書き込む値を制限して防ぐ方法もあるが，図 3 に示すように BD は制御フラグとバッファポインタが交互に並んでおり，各バッファポインタへの書き込み値の制限ごとに PWrap のリージョンが必要となるため，非常に多くのリージョンが必要となり，実現は現実的ではない．

4 つ目は，高信頼用メモリを低信頼 CPU がアクセスできるという問題である（**問題点 1-4**）．

（**要件 2**）依頼方式と比較して，低信頼 CPU による送信の実行オーバヘッドは発生しないが，低信頼 CPU 宛てのフレームを受信する際には高信頼 CPU がいったんフレームを受信してから宛先 IP アドレスを確認して，低信頼 CPU に通知する処理が必要になるため，実行オーバヘッドが生じる（**問題点 2**）．

（**要件 3**）送信の際には高信頼 CPU と低信頼 CPU 間で排他制御以外のやりとりを行う必要がなくなったため，依頼方式と比べて CPU 間のインタラクションの頻度は低くなった．しかし，受信の際には高信頼 CPU と低信頼 CPU 間でやりとりを行う必要があるという問題がある（**問題点 3**）．

（**要件 4**）低信頼 CPU が自由にイーサネットコントローラにアクセスすることができるため，ネットワーク帯

域の制限はできないという問題がある（**問題点 4**）．

（**要件 5**）ドライバの受信関数で宛先 IP アドレスの確認と低信頼 CPU への通知が必要になるため，若干のソフトウェア変更が必要になる．

依頼方式と比較すると，（要件 2）の実行オーバヘッドに関する問題と（要件 3）の機能間のインタラクションの頻度が高い問題は，ある程度は解決できるが完全には解消されていない．また，直接操作方式により，（要件 1，4）を満たさなくなる．

5.2 PWrap を適用したイーサネットコントローラによる直接操作方式

既存のイーサネットコントローラによる直接操作方式の問題を解決するため，まず，PWrap のみを適用し，どの程度問題が解決するか検討する．PWrap をイーサネットコントローラに適用した後のハードウェア構成を**図 7** (1) に示す．低信頼 CPU からの制御レジスタへのアクセスは PWrap により制限する．既存のイーサネットコントローラによる手法と同様に，高信頼用メモリは，低信頼 CPU からもアクセス可能である必要がある．PWrap を適用することによって，以下の 2 つの問題については解決可能である．

（**問題点 1-1**） イーサネットコントローラの制御レジスタは主に初期化時に値が設定されるが，この処理は高信頼 CPU によってのみ行われる．そのため，低信頼 CPU はコントローラの制御レジスタにアクセスできる必要がなく，不正に操作するとコントローラの機能が不能になるレジスタも存在する．このことから，PWrap のアクセステーブルの設定により，低信頼 CPU がアクセスできるコントローラのメモリ領域を BD 部分のみに限定し，制御レジスタにはアクセスできないようにする．これにより（問題点 1-1）は解消される．

（**問題点 4**） PWrap は CPU からイーサネットコントローラのデバイスレジスタへのアクセス頻度を制限することができる．そのため，低信頼 CPU からの送信 BD の READY フラグの書き込み頻度を制限することによって，低信頼 CPU からの送信頻度を制限できるため，高信頼 CPU からの送信に対する帯域保証が可能となる．これにより（問題点 4）は解消される．なお，6 章において，帯域保証の効果を評価する．

残りの問題点には関しては，次の理由により，PWrap のみでは解決できない．

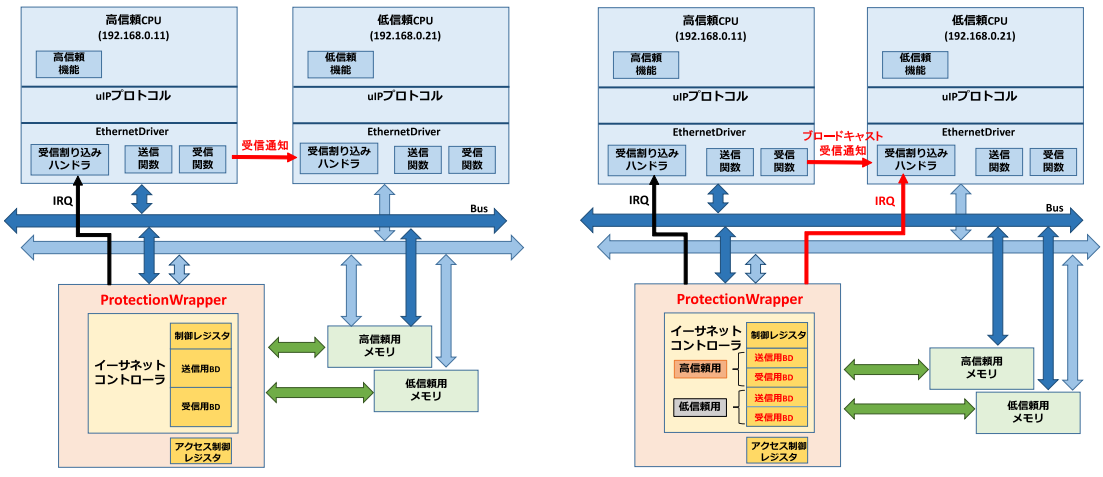
（問題点 1-2）に関しては，既存のイーサネットコントローラでは BD を高信頼用と低信頼用に静的に分けることができないため解決できない．具体的には，送受信ともに BD は循環して使用するため，低信頼 CPU もすべての BD にアクセスできる必要があり，結果的に保護が実現できない．

（問題点 1-3）に関しては，現状の PWrap ではデバイスのマスタインタフェースに対する保護機構がないこと，保護機構があったとしても，前述のとおり，ある BD は，高信頼 CPU も低信頼 CPU も使用する可能性があるため，保護を適用するかどうか判断することができない．

（問題点 1-4）に関しては，受信のために，高信頼メモリも低信頼 CPU からアクセス可能とする必要があり，この点でも保護が実現できない．

（問題点 2）に関しては，現状のイーサネットコントローラは，受信割込みが 1 本しかなく，低信頼 CPU 向けのフレームを受信したとしても，高信頼側に割込みを入れる必要があるため，解決できない．

（問題点 3）に関しては，前述のように低信頼 CPU 宛のフレームを受信した場合も高信頼側に割込みが発生するため，高信頼 CPU は低信頼 CPU に何らかの方法でフレームの受信を伝える必要があるためである．



(1) ProtectionWrapper適用後　　(2) イーサネットコントローラ機能拡張後

図 7 拡張後の直接操作方式のハードウェア構成

Fig. 7 Expanded hardware design using proposing device sharing method.

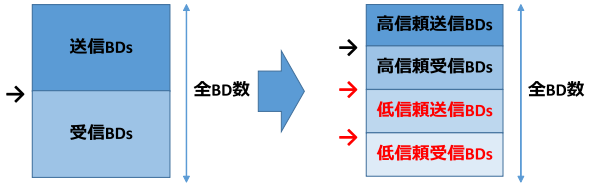


図 8 バッファディスクリプタ分割
Fig. 8 Buffer Descriptor Partitioning.

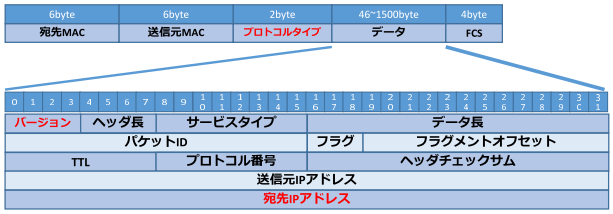


図 9 イーサネットフレーム
Fig. 9 Ethernet Frame.

5.3 イーサネットコントローラの機能拡張

前述の PWrap の適用だけでは充足できない問題点を解決するため，イーサネットコントローラの機能を拡張した．

イーサネットコントローラの機能拡張後のハードウェア構成を図 7 (2) に示す．高信頼用メモリは低信頼 CPU からアクセスできないように変更されている．

イーサネットコントローラ対して具体的に行った拡張は以下に記す 2 点である．

- バッファディスクリプタ構成の変更
- 宛先 IP アドレス判別処理の追加

5.3.1 バッファディスクリプタ構成の変更

（問題点 1-2，1-3）を解決できない原因であった BD の共有問題を解決するため，**図 8** に示すように，BD を高信頼用と低信頼用に分割した．また，送信を循環で行うのではなく，READY フラグをセットした BD があれは即座に送信するように変更した．この変更により，低信頼 CPU が自身の BD を不正に操作しても，高信頼 CPU の通信には影響しない．また，高信頼用と低信頼用で BD 領域が分離されたため，PWrap によるアクセス制御によって低信頼 CPU が高信頼用の BD にアクセスできないように設定することが可能となる．なお，（問題点 1-3）に関しては，この変更だけでは解決しないため，後述の PWrap 拡張で対応する．

5.3.2 宛先 IP アドレス判別処理の追加

（問題点 2，3）は，すべての受信で高信頼側に割込み要求を発生させることが原因であるため，まず受信割込みを高信頼側と低信頼側に分け，そのうえで，イーサネットフレームを受信した際の宛先 IP アドレスの判別をイーサネットコントローラで行い，低信頼宛のフレームを受信した場合は，低信頼用の BD を用いてフレームを受信した後，低信頼側の割込みを発生させ，それ以外の場合は，高信頼用の BD を用いて受信して高信頼側に割込みを発生させる機構を追加した．

具体的には，**図 9** に示すように，イーサネットフレームの 3 つの部分について値の確認を行う．まず受信したフレームのプロコトルタイプが IP プロトコル（0x0800）であるかどうかの確認を行う．そして受信したフレームが IP プロコトルであれば，IP プロコトルのバージョンが IPv4（0x4）であるかの確認を行う．そして最後に，宛先 IP アドレスが低信頼 CPU の IP アドレス（レジスタ設定値）と一致するかの比較を行う．受信したフレームの宛先が低信頼 CPU 宛であった場合は低信頼用の BD を使用し，それ以外の宛先であれば高信頼用の BD を使用する．

この変更により，低信頼 CPU 宛のフレームを受信した場合に，高信頼側に割込みが入らず，（問題点 2）が解決される．さらに，ブロードキャストフレーム受信時以外は，高信頼 CPU は低信頼 CPU とやりとりを行う必要がないため，（問題点 3）は緩和される．

（問題点 1-4）に関しては，高信頼用メモリの低信頼 CPU からのアクセスを不可能とすることで解決した．これは，BD を高信頼用と低信頼用に分割すること，受信時に使用する BD を自動的に選択することにより，低信頼 CPU 用の受信フレームは，低信頼用メモリに直接格納できることから，低信頼 CPU から高信頼用メモリへのアクセスが不要になったためである．なお，両方の CPU で受信する必要があるブロードキャストのフレームに関しては，各 CPU 用の BD を用いてフレームを同時に保存する方法もあるが，ハードウェアの改変が多いため，依頼方式と同様に，いったん高信頼 CPU で受信して，低信頼 CPU にコピーを渡す．

5.3.3 PWrap のマスタ側の保護の追加

前述のとおり（問題点 1-3）は，イーサネットコントローラの機能拡張だけでは，解決しないため，PWrap に次の機能拡張を行った．

イーサネットコントローラを機能拡張する前の状態では，BD が信頼度ごとに分割されていなかったため，マスタ側の保護を実現することができなかった．機能拡張により，BD が信頼度ごとに分割されたため，使用している BD の信頼度によってアクセス可能なメモリ領域の制限を可能とした．

5.3.4 ドライバの変更

低信頼 CPU からの制御レジスタへのアクセスは PWrap により制限されるため，低信頼 CPU の IP アドレスの設定や，高信頼用 BD 数・低信頼用 BD 数の割当て処理は，高信頼 CPU で実施する．

5.3.5 送信方法

BD のパーティショニングによって機能間で排他的に送信を行う必要がなくなった．それぞれの CPU は割り付けられた BD を用いて独立に既存のイーサネットコントロー

ラと同様の手順で送信を行う．

5.3.6 受信方法

イーサネットコントローラの機能拡張にともない，受信の手順は以下のようになる．

宛先 IP アドレスが低信頼 CPU 宛の場合

(1) イーサネットコントローラが受信したフレームの宛先 IP アドレスを確認し，宛先 IP アドレスが低信頼宛であれば，低信頼用 BD を使用し，低信頼用メモリに受信したフレームを書き込む．
(2) イーサネットコントローラが低信頼 CPU に受信割込みを入れる．
(3) 低信頼 CPU が受信割込みを受け，受信関数を呼び出して受信を行う．受信が完了したら，BD の制御フラグを操作して受信完了に設定する．

宛先 IP アドレスがブロードキャストの場合

ブロードキャストのフレームは次のように受信する．

(1) イーサネットコントローラが受信したフレームの宛先 IP アドレスを確認し，ブロードキャストであれば高信頼用 BD を使用し，高信頼用メモリに受信したフレームを書き込む．
(2) イーサネットコントローラが高信頼 CPU に受信割込みを入れる．
(3) 高信頼 CPU が割込みを受け，受信関数を呼び出して受信を行う．
(4) 高信頼 CPU の受信が完了したら，受信したフレームを低信頼用メモリに書き込み，BD の制御フラグを操作して受信完了に設定する．
(5) 高信頼 CPU が低信頼 CPU に割込みでブロードキャストの受信を通知する．
(6) 低信頼 CPU が割込みを受け，受信フレームを読み込む．

6. 評価

前述の各方式についてハードウェアを FPGA 上に実装し，定量的な評価の後，提案手法の要件充足，他の環境への適用可能性について評価する．定量的な評価は，通信における実行オーバヘッド，帯域保証，ハードウェア面積，ソフトウェア変更量，ハードウェア変更量について行った．評価に用いた環境は次のとおりである．

- 評価ボード：Terasic 社 DE2-115（Cyclone IV）
- CPU：NiosII/f プロセッサ（50 MHz）× 2
- OS：TOPPERS/ATK2-SC1-MC（AUTOSAR 仕様）
- 論理合成：Quartus Prime 16.1

実行オーバヘッドおよびハードウェア面積の評価の際には以下の 3 パターンについて測定を行った．

- 依頼方式
  図 5 で示したハードウェアによる依頼方式．

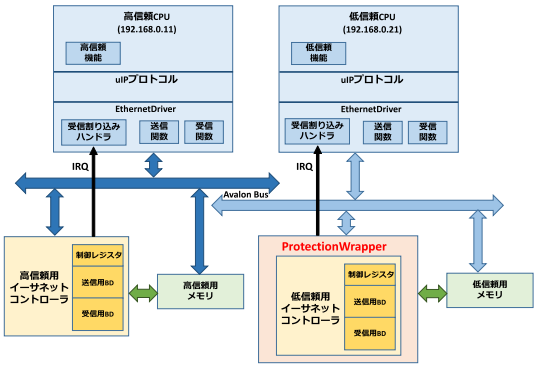


図 10 非共有方式のハードウェア構成
Fig. 10 Hardware design using device non-sharing method.

- 直接操作方式
  図 7 (2) で示したハードウェアによる直接操作方式．
- 非共有方式
  上に示す 2 つの共有方式と比較するために，**図 10** に示す，イーサネットコントローラを共有しない方式を用意した．各 CPU にイーサネットコントローラを割り当て，高信頼機能の帯域保証のために低信頼 CPU のイーサネットコントローラには PWrap を適用して BD に対するアクセス頻度の制限を行うことで，送信頻度を制限している．

6.1 実行オーバヘッド

各方式における通信時の実行オーバヘッドについて評価を行った．評価は，ユニキャスト通信とブロードキャスト通信における実行オーバヘッドを測定した．

6.1.1 ユニキャスト通信

ユニキャスト通信における実行オーバヘッドの評価では，PC と評価ボード間を 100 Mbps で通信を行い，PC から評価ボード間上の各 CPU の IP 宛に PING 要求を送りエコーバックするまでの時間を測定することで評価を行った．測定はネットワーク・アナライザ・ソフトウェアである Wireshark [11] を使用した．具体的な処理の内容は次のとおりである．PC 上の C 言語で記述されたプログラムから，ICPM プロトコルの echo request パケットを評価ボード上の CPU に送信して，このパケットを受け取った評価ボードの CPU で動作するプロトコルスタックから，PC に echo reply パケットが送られる．PING 要求は一定周期で繰り返し送信した．なお，直接操作方式は，ブロードキャスト受信時には通常の受信と異なる処理となるが，エコーバック時間の計測においては，PC 側の ARP キャッシュに評価ボード側の情報がない場合，ICMP プロトコルによる通信を行う前に，ARP リクエストが PC 側から 1 度ブロードキャストされる．そのため，PING 要求の送信時にブロードキャストが発生する可能性がある．しかしながら，上記の echo request パケットの送信から，echo reply パケット受信の間を計測しているため，この ARP リクエ

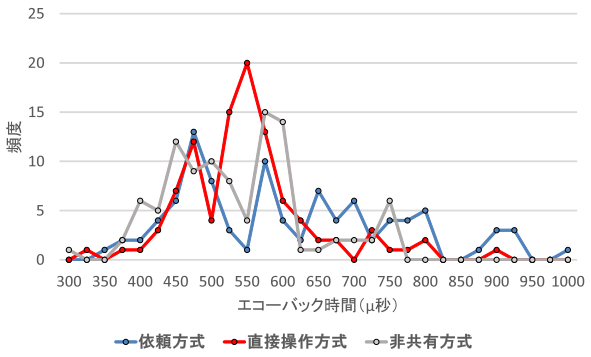


図 11 高信頼 CPU のエコーバック時間測定結果
Fig. 11 Echo back time from high reliability CPU.

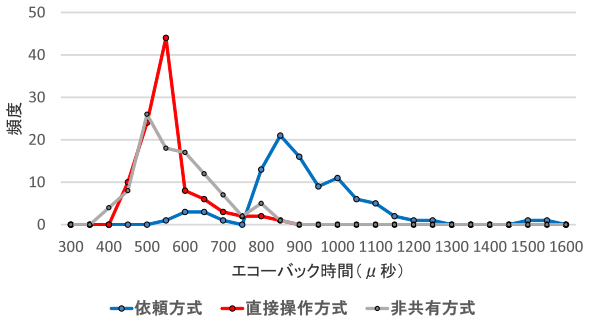


図 12 低信頼 CPU のエコーバック時間測定結果
Fig. 12 Echo back time from low reliability CPU.

表 1 ユニキャスト通信の平均エコーバック時間（μ 秒）
Table 1 Average echo back time of unicasting.

	高信頼 CPU	低信頼 CPU
依頼方式	627	933
直接操作方式	544	533
非共有方式	524	524

スト（ブロードキャスト）は計測結果には含まれていない．

高信頼 CPU の測定結果を**図 11**，低信頼 CPU の測定結果を**図 12**，それぞれの平均値を**表 1** に示す．測定した結果から非共有方式と各共有方式を用いたときのエコーバック時間の変化に関する考察を行う．

高信頼 CPU の送受信の処理は，依頼方式で行う場合も直接操作方式で行う場合も非共有方式と同じ手順によって行われる．しかし，受信の際に宛先 IP アドレスの確認処理が依頼方式および直接操作方式の両方で必要になるためいくらか実行オーバヘッドが生じている．

低信頼 CPU のエコーバック時間については，依頼方式を用いた場合は非共有方式と比較して 78%増加している．この増加分が依頼処理ために発生するオーバヘッドに相当する．

依頼方式を用いることによって生じる実行オーバヘッドを送信時と受信時について評価ボード上のハードウェアタイマを使用して計測した．具体的には，送信については 4.1 節の (1) から (3) までの処理に要する時間を，受信については 4.2 節の (2) から (5) までの処理に要する時間をそれぞれ測定した．測定結果は，送信時に発生する実行オーバヘッドは 149.6 μ 秒になり，受信時については 138.6 μ 秒となった．

表 2 ブロードキャスト通信の平均エコーバック時間（μ 秒）
Table 2 Average echo back time of broadcasting.

	高信頼 CPU	低信頼 CPU
依頼方式	463	610
直接操作方式	466	477
非共有方式	373	373

また，非共有方式と比較して，エコーバック時間の分散が大きくなっている．これは低信頼 CPU が処理を依頼する際に CPU 間で割込みが発生し，割込み先 CPU の動作状況によっては処理を待たされるためである．

一方，直接操作方式を用いた場合の低信頼 CPU のエコーバック時間は非共有方式と比較して 1.7%増加で抑えられている．送信に関してはそれぞれの CPU が独立して行うことが可能になった．受信に関しては，イーサネットコントローラの機能拡張により，ブロードキャスト以外は低信頼 CPU のみで独立して受信が行えるようになったため，エコーバック時間の増加を抑えることができた．

6.1.2 ブロードキャスト通信

ブロードキャスト受信時に発生する実行オーバヘッドを測定した．測定は PC から ARP 要求を送信して，PC が ARP 応答を受信するまでの時間を測定した．評価対象は，非共有方式および依頼方式，直接操作方式提における高信頼 CPU に向けた ARP と低信頼 CPU に向けた ARP について測定した．結果を**表 2** に示す．高信頼 CPU に関しては，非共有方式と比較して，依頼方式と直接操作方式の両者ともに，低信頼 CPU 側に対してもフレームを送るため，同様の実行オーバヘッドが発生する．一方，低信頼 CPU に関しては，受信は依頼方式と直接操作方式の両者ともに高信頼 CPU を経由するが，直接操作方式では送信は直接行うため，依頼方式より低い実行オーバヘッドで実現できている．

6.2 帯域保証

帯域保証の評価として，提案手法を用いることによって，高信頼側の送信に関して通信帯域が保証可能か評価した．

評価は，低信頼 CPU のプログラムに不具合があり連続送信を行っている想定として，ARP リプライを送り続けている状態で，PC から高信頼 CPU に対する 6.1 節のエコーバック時間計測と同様の測定を実施した．そのうえで，PWrap によって低信頼 CPU からイーサネットコントローラの送信 BD へのアクセス周期が制限されていない場合と，されている場合について比較した．アクセス周期の制限では送信 BD への書き込み可能周期を 100 ms に設定し，低信頼 CPU が一度送信 BD に書き込んでから 100 ms

表 3 ハードウェア面積比較
Table 3 Comparison of hardware area.

	LUT 数
依頼方式	2,941
直接操作方式	6,794
非共有方式	8,790

以上経過しないと，再び書き込めないように制限した．

評価の結果，アクセス周期の制限がない場合では，高信頼 CPU の平均エコーバック時間は 6,787 μ 秒であったのに対して，制限がある場合の平均値は 622 μ 秒であった．低信頼による不正連続送信がない状態での高信頼 CPU のエコーバック時間が表 1 より 544 μ 秒のため，どちらの場合でもエコーバック時間は増加している．しかし，アクセス周期の保護がない場合ではエコーバック時間は通常の場合と比べて 11.6 倍なのに対して，アクセス周期の保護がある場合では 1.06 倍に抑えられており，有用性を確認した．なお，1.06 倍と増加している理由としては，低信頼 CPU が連続してイーサネットコントローラの BD や制御レジスタに対してバスアクセスを行っているため，高信頼 CPU からの BD や制御レジスタへのアクセスが遅延しているためだと考えられる．

6.3 ハードウェア面積

ハードウェア面積の評価は FPGA に実装する際に使用されたルックアップテーブル数（LUT 数）の値を比較することにより行った．それぞれのイーサネットコントローラのモデルについて測定を行った結果は**表 3** のようになった．

測定した結果から非共有方式と各共有方式でのハードウェア面積を比較すると，依頼方式では約 67%，直接操作方式では約 23%減少した．

6.4 ソフトウェア変更量

各方式での送受信を実現する際に行ったソフトウェア変更量および変更箇所の詳細は以下のとおりである．変更箇所はプロトコルスタック内部ではなく，ドライバの箇所だけである．

(1) 依頼方式：38 行
- 受信フレームの宛先 IP アドレス確認
- 依頼にともなう低信頼 CPU と高信頼 CPU 間のフレーム受け渡し処理

(2) 直接操作方式：34 行
- イーサネットコントローラ拡張にともなう初期化処理
- 受信フレームがブロードキャストかの確認処理

6.5 ハードウェア変更量

前述のイーサネットコントローラの機能拡張を実現するための変更量は 343 行であった．具体的な変更箇所は，受信フレームの宛先 IP アドレス判別を行う回路の追加と BD 分割にともなう回路変更である．既存のイーサネットコントローラには宛先の MAC アドレスをチェックする機構があるため，その機構を拡張して，IP パケットの場合は宛先 IP アドレスをチェックする機構を追加した．

6.6 提案手法の要件充足

機能拡張したイーサネットコントローラを用いた直接操作方式に対する，2 章であげた要件の充足を確認する．

（**要件 1**）BD が信頼度ごとにパーティショニングされており，PWrap により低信頼 CPU が高信頼 BD を不正に操作することがないこと，制御レジスタと高信頼用メモリは低信頼 CPU からアクセスできないことより，高信頼 CPU の送受信が低信頼 CPU の動作不良により阻害される危険性はない．

（**要件 2**）低信頼 CPU 宛のフレーム受信時に高信頼 CPU が行う処理はいっさいないため，実行オーバヘッドは発生しない．ただし，ブロードキャストフレーム受信時には高信頼用メモリから低信頼用メモリへのコピーが発生するため，実行オーバヘッドが生じる．

（**要件 3**）ブロードキャストフレーム受信時以外で高信頼 CPU と低信頼 CPU 間のやりとりはない．

（**要件 4**）低信頼 CPU が低信頼用 BD にアクセスする頻度を PWrap によって制限することによって，低信頼 CPU がネットワーク帯域を占有しないことを保証できる．

（**要件 5**）ドライバの初期化処理および受信処理に変更の必要があるが，変更量は小さい．

以上のように，提案手法はほぼすべての要件を満たしているといえる．

6.7 適用可能性評価

他の OS やプロトコルスタック，IP プロトコルのバージョン，イーサネットコントローラに対する提案手法の適用可能性について評価を行う．

6.7.1 他の OS やプロトコルスタック

提案手法を他の OS やプロトコルスタックに適用可能か評価する．まず，他の OS として，Linux について，文献 [12] の情報を元に見積もった結果，次の変更が必要であることが分かった．

- 初期化・停止処理の調停
- 送信関数の変更
- 受信関数の変更
- コア間のメモリの用意
- コア間通知の実現
- そのほかイーサネットコントローラの状態変更の調停

初期化・停止処理の調停は，Linux ではネットワークドライバの open・close で任意のタイミングでデバイスの開

始と終了が可能であるため，Linux 間で調停が必要となる．送信関数の変更は，uIP のドライバと同様に使用するディスクリプタを変更する必要がある．受信関数も uIP のドライバと同様に使用するディスクリプタの変更と，マルチキャストへの対応が必要となる．マルチキャストへの対応として，共有メモリを用意してそれぞれのドライバでマッピングする必要がある．さらに，マルチキャストデータ受信時に低信頼側への通知として，コア間割込みハンドラを用意して，ブロードキャスト受信時は受信関数で，データをコピーしてハンドラを呼び出す変更を加える必要がある．また，ハンドラを用意して高信頼側から受け取ったパケットを受信関数と同様にネットワークコードの上位層に送る必要がある．そのほかイーサネットコントローラの状態の変更の調停して，リンクステートの変更等を Linux 間で調停する必要がある．

このように uIP と比較すると変更量が大きくなる．しかしながら，Linux の場合，すでに多くの仮想化環境が提供されており，ソフトウェア的にイーサネットコントローラを仮想化して共有可能であるため，本機構を使う必要性は低いと考えられる．

次に，他のプロトコルスタックとして，RTOS 向けかつ uIP と比較して複雑な LWIP [13] や TINET [14] について検討する．これらのプロトコルスタックのドライバは uIP のドライバと同様に，初期化関数，送信関数，受信関数，割込みハンドラで構成されている．そのため，変更量に関しても uIP と大きく変わらないと予想される．

6.7.2 IPv6

IPv6 に適用する場合，イーサネットコントローラの機能拡張の宛先 IP アドレスの判別処理を拡張することで対応可能である．具体的には受信したフレームが IP プロトコルでバージョンが IPv6 で宛先 IP が低信頼 CPU 宛であるかを判別する必要がある．この際，IPv4 では IP アドレスのデータサイズが 32 bit なのに対し，IPv6 は 128 bit となっており，また，宛先 IP を読み込むまでのデータサイズが IPv4 では 128 bit なのに対し，IPv6 では 192 bit である．宛先 IP アドレスを判別する際にはイーサネットコントローラの内部バッファに宛先 IP アドレスの情報が含まれる部分までのフレームを保存する必要があるため，IPv6 の方が内部バッファをより多く使用することになる．

6.7.3 他のイーサネットコントローラ

提案手法を NE2000 仕様および組込み向けマイコン内蔵のイーサネットコントローラに対して適用可能か検討する．

NE2000 仕様のイーサネットコントローラでは送受信フレームをメモリに格納してリングバッファとして使う仕様となっている．そのため，今回使用したイーサネットコントローラと同様に高信頼用と低信頼用とでメモリの使用領域を分け，それぞれにリングバッファを用意して多重化する必要がある．それに加えて，イーサネットコントローラで宛先アドレスを確認して，どちらのリングバッファを使用するか判別する機構の追加が必要となる．

一方，組込みシステム向けのマイコンである RX62N に搭載されているイーサネットコントローラ（EHERC）では，受信フレームの宛先 IP アドレスを確認する機能の追加およびディスクリプタのパーティショニングを行う必要がある．ETHERC ではフレームを格納するメモリにアクセスするために，コントローラ内部にイーサネットコントローラ用ダイレクトメモリアクセスコントローラ（EDMAC）が搭載されている．そのため，EDMAC を高信頼用と低信頼用とに多重化する必要がある．

両イーサネットコントローラともにデバイスレジスタにより制御を行うため，それぞれのイーサネットコントローラに上記の変更を適用し，PWrap を用いて低信頼 CPU からのデバイスレジスタへのアクセスを制限することにより，今回用いた “Ethernet MAC 10/100 Mbps” と同等の保護された共有が実現可能である．

7. 関連研究

ミックスドクリティカルシステムにおいて，デバイスの共有を実現する取り組みとしては，車載システムで広く使われている CAN コントローラに関する研究がなされている．文献 [3], [4], [5] では，車載システムにおいて仮想マシン間やマルチプロセッサ間で CAN コントローラを共有する方式として，ソフトウェアレベルとハードウェアレベルの手法をそれぞれ提案している．ソフトウェアレベルでは，依頼方式と同様に，ホストのみが CAN コントローラを操作しゲストはホストに依頼する共有方法となっており，実行オーバヘッドが大きくなってしまうが，既存の車載システムで使用されているプラットフォームでも利用可能であると述べられている．それに対してハードウェアレベルでは，CAN コントローラを変更し，仮想マシンごとに独立したレジスタを提供する機構を追加している．各仮想マシンからの送信フレームをハードウェアでスケジューリングすることにより，帯域の保証を可能としている．本論文の提案手法と異なり，この方法ではデバイスドライバの変更がないというメリットがある．一方，この手法は，ハードウェアの変更量が大きいことや，CAN コントローラの仕様に大きく依存しており，5.3.3 項で述べたマスタ側の保護も提案はされていないため，イーサネットコントローラにはそのまま適用はできない．

車載システムや組込みシステムを対象としたイーサネットの共有に関する研究は現時点では報告されていない．一方，サーバでの仮想化においては，PCI Express の仮想化機能である SR-IOV を用いたデバイスが使用されている．組込みシステムでは，イーサネットコントローラは PCI Express ではなく，直接バスに接続されていることが一般的であるため，これらの機構を用いることはできない．文

献 [15] では，SR-IOV を用いて仮想マシン間でネットワークを共有した場合の問題点として，イーサネット通信のフロー制御で用いられるポーズフレームに関する問題が取り上げられている．この問題は低信頼機能がポーズフレームを外部のイーサネットスイッチに送ることによって，外部のイーサネットスイッチからフレームがいっさい送られてこなくなり，高信頼機能も受信ができなくなる問題である．ポーズフレームは受信ノードのバッファが一杯になったときに送信ノードに対して送られるフレームで，高い信頼性の求められる車載システムでは受信バッファがそもそも溢れないように設計するべきであるとは考えられるが，そのことを保証することは難しいと考えられる．そのため，車載イーサネットでフロー制御を用いることを想定した場合に同じ問題が発生する可能性を考え，車載システムにおけるこの問題に対する解決を今後の課題とする．

8. まとめ

本論文では求められる信頼度の異なる機能が混在しているシステムで，機能間で安全にイーサネットコントローラを共有する手法として直接操作方式向けのイーサネットコントローラを拡張する手法を提案した．提案手法では，従来手法の依頼方式と比べて実行オーバヘッドを大幅に削減することができ，機能間でのインタラクションの頻度を下げることができた．

今後の課題としては，関連研究で述べたポーズフレームの問題への対応や，現状の PWrap は設定を動的に変更可能であるため面積が大きいという問題があるため，静的な設定のみ可能として面積を減少させる等があげられる．

謝辞 本研究の一部は，（株）半導体理工学研究センターとの共同研究による．

参考文献

[1] Baruah, S., Li, H. and Stougie, L.: Towards the design of certifiable mixed-criticality systems, *Real-Time and Embedded Technology and Applications Symposium*, IEEE (2010).
[2] Crespo, A., Alonso, A., Marcos, M., de la Puente, J.A. and Balbastre, P.: Mixed criticality in control systems, *IFAC Proceedings Volumes*, Vol.47, No.3, IFAC (2014).
[3] Herber, C., Richter, A., Rauchfuss, H. and Herkersdorf, A.: Self-virtualized CAN Controller for Multi-core Processors in Real-Time Application, *Architecture of Computing Systems*, *ARCS 2013*, ARCS (2013).
[4] Herber, C., Richter, A., Rauchfuss, H. and Herkersdorf, A.: Spatial and temporal isolation of virtual can controllers, *ACM SIGBED Review*, Vol.11, No.2, ACM (2014).
[5] Herber, C., Reinhardt, D., Richter, A. and Herkersdorf, A.: HW/SW Trade-offs in I/O Virtualization for Controller Area Network, *52nd ACM/EDAC/IEEE Design Automation Conference*, DAC (2015).
[6] 加藤祐輔，本田晋也，高田広章：ミックスドクリティカルシステム向けペリフェラル共有機構，SCIS (2016).
[7] Hank, P., Vermesan, O., Muller, S., Van Den Keybus, J.: Automotive ethernet: In-vehicle networking and smart mobility, *Advanced Microsystems for Automotive Applications 2012*, IEEE (2012).
[8] Bello, L.L.: The case for Ethernet in Automotive Communications, *Special Issue on the 10th International Workshop on Real-time Networks*, RTN (2011).
[9] OpenCores: Ethernet_MAC_10/100Mbps, available from ⟨https://opencores.org/project/ethmac⟩ (accessed 2017-11-14).
[10] Dunkels, A.: *The uIP Embedded TCP/IP Stack*, OpenCores (2006).
[11] Wireshark, available from ⟨https://www.wireshark.org/⟩ (accessed 2017-11-14).
[12] Corbet, J., Rubini, A. and Kroah-Hartman, G.：Linux デバイスドライバ第 3 版，オライリー・ジャパン (2005).
[13] Dunkels, A.: Design and Implementation of the lwIP TCP/IP Stack, Swedish Institute of Computer Science (2001).
[14] 阿部 司, 吉村 斎, 久保 洋：組込みシステム用 TCP/IP プロトコルスタックの実装と評価，情報処理学会論文誌，Vol.44, No.6, pp.1583–1592 (2003).
[15] Smolyar, I., Ben-Yehuda, M. and Tsafrir, D.: Securing Self-Virtualizing Ethernet Devices, *USENIX Security Symposium*, USENIX (2015).

石野 正敏

2018 年名古屋大学大学院情報学研究科修士課程在学中．車載組込みシステムの研究に従事．

本田 晋也 （正会員）

2002 年豊橋技術科学大学大学院情報工学専攻修士課程修了．2005 年同大学大学院電子・情報工学専攻博士課程修了．名古屋大学大学院情報科学研究科附属組込みシステム研究センター助教等を経て，2014 年より名古屋大学大学院情報科学研究科情報システム学専攻准教授，リアルタイム OS，ソフトウェア・ハードウェアコデザインの研究に従事．博士（工学）．2002 年度情報処理学会論文賞受賞．ACM，IEEE，電子情報通信学会，日本ソフトウェア科学会各会員．