Vol.2016-OS-138 No.11
2016/8/8

Solid State Server “Olive” におけるeMMCおよびNIC高速化

追川 修一[1]　中村 孝史[2]　飯塚 拓郎[2]　岡田 優美[3]　田村 陽介[3]　三木 聡[3]

概要：Solid State Server “Olive” は，2.5 インチ HDD のフォームファクタに，ARM CPU と FPGA を搭載した ZYNQ および eMMC ストレージ，ネットワークインタフェースをおさめ，Linux を動作させることのできる省電力サーバである．ネットワークストレージサーバとして，小型かつ高密度のファイルサーバや，FPGA 回路によりデータの加工を行うことのできるサーバとしての用途などが検討されている．ZYNQ は，FPGA がチップ上の構成要素を接続する．そのため，様々な構成をとることが可能であり，その構成が性能に大きく影響を与える．例えば，デバイスへの接続がキャッシュコヒーレントでない構成の場合，キャッシュフラッシュ処理が必要となり，その処理が性能低下の原因となるため，キャッシュフラッシュ処理を削減することのできる構成をとることで性能向上を図ることができる．さらに，ネットワークストレージサーバとして，ストレージ・ネットワークインタフェース間の転送を高速化する手法について実験を行い，その有効性を確かめた．

1. はじめに

Solid State Server “Olive” は，2.5 インチ HDD のフォームファクタに，ARM CPU と FPGA を搭載した ZYNQ および eMMC ストレージ，ネットワークインタフェースをおさめ，Linux を動作させることのできる省電力サーバである．ネットワークストレージサーバとして，小型かつ高密度のファイルサーバや，FPGA 回路によりデータの加工を行うことのできるサーバとしての用途などが検討されている．

Olive の仕様を表 1 に示す．FPGA は Xilinx Zynq-7030 であり，これは CPU として ARM Cortex A9 Dual Core を搭載している．Gigabit Ethernet コントローラはオンチップに実装されている．また，メインメモリとしては DRAM を 512MB 搭載している．eMMC コントローラは FPGA に実装されており，eMMC は 512GB から 13TB までの容量を搭載することができる．

ZYNQ は，FPGA がチップ上の構成要素を接続するため，様々な構成をとることが可能であり，その構成が性能に大きく影響を与える．最初に実装した構成では，想定以下の性能となった．その主な理由は，デバイスへの接続がキャッシュコヒーレントではないことで必要となるキャッシュフラッシュ処理であり，その削減を可能にする構成へ変更を行った．さらに，プロセッサの処理の仕組みが性能に与える影響を調べるため，プロセッサアフィニティを設定した実験，sendfile システムコールによるユーザプログラム介在を削減した実験を行った．最後に，ネットワークストレージサーバとして，ストレージ・ネットワークインタフェース間の転送を高速化する手法について実験を行った．その結果，CPU 利用率を削減し，Gigabit Ethernet の帯域幅を使い切った性能を達成することができた．

本稿では，これらの Olive における eMMC および NIC 高速化手法について述べる．

表 1　Olive の仕様

Chassis	2.5 inch HDD Form Factor
FPGA	Xilinx Zynq-7030
CPU	ARM Cortex A9 Dual Core
RAM	DDR2 512MB
NIC	1Gb Ethernet
Storage	eMMC NAND Flash Memory (512GB∼13TB)
Power Consumption	6.5W

2. 初期構成

ZYNQ-7000 は，ARM CPU コアが実装された Processing System (PS) と FPGA を実装する Programmable Logic (PL) からなる．Olive は，PL 部に eMMC コントローラを

[1] 筑波大学システム情報系
University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[2] Fixstars Solutions Inc.
Irvine, California 92618, USA
[3] (株) フィックスターズ
Fixstars Corporation, Shinagawa, Tokyo 141–0032, Japan

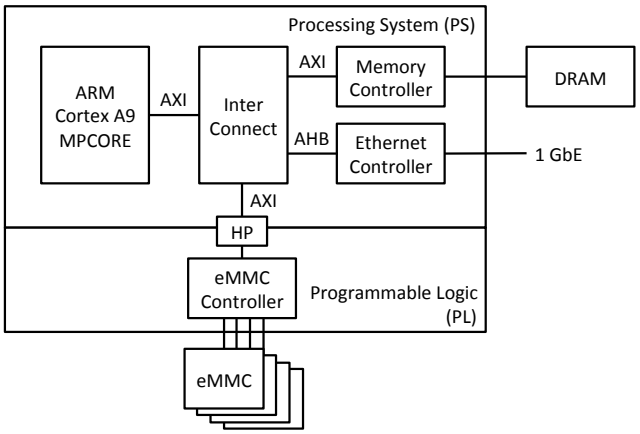


図 1 初期構成の概略

表 2 初期構成でのネットワーク性能の測定結果 (TCP)

	TX (Mbps)	CPU	RX (Mbps)	CPU
4KB	312.8	99%	605.3	98%
64KB	607.0	99%	619.9	95%
4MB	679.7	42%	485.4	86%

表 3 初期構成でのファイル I/O 性能測定結果 (MB/s)

	Read	Write
Sequential (64KB)	144.1	96.9
Sequential (4MB)	191.3	101.6

実装している．eMMC コントローラは High Performance Port (HP) に接続され，HP は AXI バスを介して，ARM CPU コアに接続されている．Gigabit Ethernet コントローラは PS 部に実装され，Advanced High-performance Bus (AHB) を介して，ARM CPU コアに接続されている．図 1 に初期構成の概略を示す．

この構成で，ネットワーク性能とファイル I/O 性能の測定を行った．ネットワーク性能は，nuttcp コマンド [1] を用い，TCP によりデータ転送を行った結果である．通信相手は，Intel Core i7-4790 3.6GHz CPU, Intel X540 10GbE NIC を搭載した PC 互換機である．ファイル I/O 性能は，fio コマンド [2] を用い，ioengine に libaio，iodepth は 32 として計測を行った結果である．ファイルシステムには ext4 を用いた．その結果を，表 2 と表 3 に示す．

どちらも，想定以下の結果となった．ネットワーク性能は，バッファサイズを 4KB に設定した時の送出性能が特に低い．その他も，理論値の 6 割程度にとどまっている．ファイル I/O 性能も，4MB のシーケンシャルアクセスで 500MB/s 以上の性能を持つ SSD がベースであることを考えると，低い値である．

3. キャッシュフラッシュを削減した構成

初期構成は十分な性能が出なかったため，初期構成での問題点を考察し，改良した構成を実装，実験を行った結果を示す．

3.1 初期構成における問題

初期構成では，eMMC コントローラが HP を介して PS に接続されている．ZYNQ-7000 では，HP 経由での DMA 転送はキャッシュコヒーレントではない．また，Gigabit Ethernet コントローラは PS 部に実装されているが，AHB で接続されているため，この DMA 転送もキャッシュコヒーレントではない．そのため，メインメモリと eMMC コントローラとの間，また Gigabit Ethernet コントローラとの間で DMA 転送を行う場合，ARM CPU のキャッシュフラッシュを行う必要がある．図 2 に，初期構成におけるデータパスと，DMA 転送にあたり必要となるキャッシュフラッシュ処理を示す．図は，eMMC のデータをネットワークに送出，またその逆を想定したデータパスであるが，eMMC，Ethernet へのアクセスのそれぞれに，キャッシュフラッシュ処理を行う必要があることを示している．

ZYNQ-9000 の ARM Cortex-A9 のキャッシュラインは 32byte である．例えば 4KB ページのフラッシュには，4KB は 128 ラインにわたるため，キャッシュラインのフラッシュを 128 回繰り返し行う必要がある．これを，eMMC，Ethernet へのアクセスのそれぞれに対して行う必要があるため，CPU への負荷が大きい．表 2 に示したネットワーク性能における CPU 使用率が高いのも，キャッシュフラッシュ処理の負荷が高いこと一因となっていると考えられる．

3.2 キャッシュフラッシュ処理の削減

キャッシュフラッシュ処理を削減することで，それぞれの処理を高速化できると考え，そのための構成変更を行った．変更点は，eMMC コントローラ接続先ポート変更と，ネットワーク受信時の ARM CPU 内メモリの利用の 2 点である．

まず，初期構成では eMMC コントローラを，キャッシュコヒーレントではない HP に接続していたが，これを Accelerator Coherency Port (ACP) に接続するように変更した．ACP は，ARM CPU 内の snoop キャッシュコントローラに接続されているため，ACP に接続したデバイスとメインメモリ間の DMA 転送はキャッシュコヒーレントになる．従って，eMMC コントローラとの間の DMA 転送に，キャッシュフラッシュは不要となる．

Ethernet コントローラの AHB への接続は固定されているため変更できないが，On Chip Memory (OCM) を用いることで，受信時のキャッシュフラッシュ処理を削減することを可能にした．OCM は，ARM CPU 内の 256KB のメモリである．Ethernet コントローラの受信バッファとして OCM を指定することで，ネットワークからのデータ受信時に，Ethernet コントローラはデータを OCM に DMA 転送する．受信による割り込みにより起動されたデバイスドライバは，受信データを OCM からメインメモリへコピーする．この時，OCM はキャッシュコヒーレントであ

表 4　改良した構成でのネットワーク性能の測定結果 (TCP)

	TX (Mbps)	CPU	RX (Mbps)	CPU
4KB	348.5	99%	923.7	99%
64KB	667.4	99%	665.2	99%
4MB	690.1	83%	656.2	84%

表 5　改良した構成でのファイル I/O 性能測定結果 (MB/s)

	Read	Write
Sequential (64KB)	176.7	121.9
Sequential (4MB)	212.4	132.2
Random (64KB)	63.1	101.4
Random (4MB)	93.6	112.6

表 6　改良した構成での NFS におけるファイル I/O 性能測定結果 (MB/s)

	Read	Write
Sequential (64KB)	88.5	70.1
Sequential (4MB)	88.2	88.3
Random (64KB)	37.4	55.6
Random (4MB)	84.5	82.0

るため，キャッシュフラッシュ処理は不要となる．

図 3 に，キャッシュフラッシュ処理を削減した構成におけるデータパスと，DMA 転送にあたり必要となるキャッシュフラッシュ処理を示す．eMMC のデータをネットワークに送出する想定では，キャッシュフラッシュ処理は 1 回削減され，逆にネットワークから受信したデータを eMMC に書き込む想定では，キャッシュフラッシュ処理は不要となる．

3.3　実験結果

キャッシュフラッシュ処理を削減した構成で，ネットワーク性能とファイル I/O 性能の測定を行った．その結果を，表 4 と表 5 に示す．また，図 3 のデータパスでの性能を測定するため，Olive のストレージを NFS でマウントし，NFS 上のファイル I/O 性能の測定を行った結果を，表 6 に示す．

表 4 から，バッファサイズが 4KB の時のネットワーク受信性能が，52.6%向上していることがわかる．バッファサイズが 64KB の時は 7.3%の向上にとどまっているが，4MB の時は 35.2%向上している．改良された構成では，図 3 (Write & RX) に示すとおり，キャッシュフラッシュ処理の代わりに，OCM からメインメモリへのデータコピー処理が行っている．ARM CPU 上の高速アクセス可能なメモリである OCM からのコピーではあるが，受信性能の向上から，キャッシュフラッシュ処理の負荷が高いことがわかる．

表 5 から，シーケンシャルアクセス時には，ブロックサイズ 64KB の読み込みが 22.6%，書き込みが 25.8%，ブロックサイズ 4MB ではそれぞれ 11.0%, 30.1%向上していることがわかる．ランダムアクセス時にも，書き込みはシーケンシャルアクセスに近い性能となっている．自作のベンチマークプログラムを作成し，同条件で実行したところ，最大で約 4 倍の性能をブロックサイズ 4MB のランダム読み出しに対し計測した．平均では約 2 倍の性能となった．そこで，fio を tmpfs メモリファイルシステム上で実行したところ，例えばブロックサイズ 4MB のシーケンシャル読み出しは 230.8MB/s となり，fio 自体の実行コストによる性能の上限に，ある程度近い性能が出ていることがわかる．表 6 からは，NFS の性能はネットワーク性能で制限されていることがわかる．

ACP 経由の DMA 転送は，キャッシュフラッシュを不要とするが，必ずしも高速化につながるとも限らないことに注意が必要である．データサイズがキャッシュサイズに十分に収まっている場合は，キャッシュフラッシュが不要であり，DMA 転送も高速である．しかしながら，キャッシュサイズ以上のデータを転送しようとすると，結局メインメモリへのデータ転送が必要となり，キャッシュを経由することで HP 経由よりも半分以下の転送速度となることが報告されている [4]．そのため，キャッシュフラッシュ処理の負荷との兼ね合いで，接続先ポートを選択する必要がある．

4.　プロセッサの影響

キャッシュフラッシュ処理を削減することにより，ネットワーク性能が約 7~52%向上，ファイルアクセス性能が約 11~30%向上する結果となった．しかしながら，ネットワーク性能は，バッファサイズ 4KB の受信性能以外の性能は，理論値よりは低くとどまっており，バッファサイズ 4KB の送出性能が特に低い．そこで，プロセッサの処理の仕組みが性能に与える影響を調べる．

4.1　プロセッサアフィニティの影響

ARM 用の Linux カーネルは，デバイスからの割り込みはブート CPU に割り振るのがデフォルトの設定となっている．そのため，Gigabit Ethernet コントローラからの割り込みも，全て CPU 0 で処理されることになる．ユーザプログラムの処理中に割り込みが発生すれば，ユーザプログラムの実行が中断され，割り込み処理が行われるため，当然，ユーザプログラムの処理は遅延することになる．ネットワーク性能計測時のベンチマークプログラムの CPU 利用率は 100%に近いため，割り込みによる処理の中断の影響は大きいと考えられる．そこで，`taskset` コマンドにより，ベンチマークプログラムを実行する CPU を CPU 1 に固定して実行した．

表 7 に示した結果からわかるように，ベンチマークプログラムの実行を CPU 1 に固定したことで，ネットワーク性能は大幅に向上した．特に，受信時には，CPU によ

表 7 CPU を固定した場合のネットワーク性能の測定結果 (TCP)

	TX (Mbps)	CPU	RX (Mbps)	CPU
4KB	524.7	99%	941.5	97%
64KB	867.7	99%	941.5	95%
4MB	791.0	51%	941.4	95%

表 8 sendfile を用いたネットワーク送出性能 (TX) の測定結果 (TCP)

	write (Mbps)	CPU	senfile(Mbps)	CPU
4KB	577.5	99%	713.1	99%
64KB	851.0	99%	941.7	10%
1MB	785.0	50%	941.5	12%

る OCM から DRAM へのデータコピーを行うため，割り込みによる処理の中断がなくなったことの影響が大きく，キャッシュフラッシュ処理が不要なこともあり，ほぼ Gigabit Ethernet の帯域幅を使い切った性能となっている．一方で，送信は，キャッシュフラッシュ処理が介在するため，約 15∼51%の性能向上となっている．

4.2 ユーザプログラム介在の削減

sendfile システムコールは，ページキャッシュ上のファイルのデータを，他のバッファにコピーすることなく，直接ネットワークに送出することができる．ユーザプログラムはネットワークに送出するデータのあるファイルと，送出先のソケットを指定するだけで，ファイルからの読み込みからネットワークへの送出まで，sendfile の処理は全てカーネル内で行われ，ユーザプログラムは介在しない．

ベンチマークプログラムとして iperf3 コマンド [3] を用い，write および sendfile システムコールによりネットワーク送出した結果を，表 8 に示す．iperf3 コマンドは，CPU 1 に固定して実行した．sendfile を用いることで，write によるネットワークへの送出よりも，約 11∼20%の性能向上となった．ファイルサイズが 64KB, 1MB の場合は，ほぼ Gigabit Ethernet の帯域幅を使い切った性能となっており，かつ CPU 利用率がそれぞれ 10%, 12%と非常に低い値となっている．

5. キャッシュ不可領域を介した転送による高速化

Linux カーネルには，sendfile と同様の機能を持つシステムコールで，ファイルディスクリプタ間のデータ移動を行う splice システムコールがある．splice システムコールを用いることでも，ユーザプログラムの介在なしに，ファイルからネットワークへの送出を行うことができるが，間に pipe を挟むなど，splice の方がより柔軟な構成をとることができる．

そこで, pipe システムコールで確保するバッファをキャッシュ不可領域とする設定を追加し，キャッシュフラッシュ処理を削減する手法を実装，実験した．この手法による処理の流れは，以下になる．まず，pipe システムコールにオプションが指定された時，確保するバッファをキャッシュ不可領域とする．splice によりその pipe バッファ領域にファイルからのデータを読み込み，また別の splice により pipe バッファ領域からネットワークへの送出を行う．Direct I/O を用いることで，ファイルから pipe バッファ領域へは，直接データ転送が行われる．そして，ネットワークへの送出へも，pipe バッファ領域から直接転送される．この時，pipe バッファ領域はキャッシュ不可領域となっているため，キャッシュフラッシュ処理を行う必要がなくなる．eMMC からの転送は，キャッシュへの転送を避ける必要がある．現状では，HP 経由で転送を行っている．この手法によるデータパスを，図 4 に示す．

表 9 splice を用いたネットワーク送出性能の測定結果 (TCP)

uncached pipe	無効 (Mbps)	CPU	有効 (Mbps)	CPU
4KB	397.4	99%	897.6	22%
64KB	819.4	28%	898.0	8%
1MB	815.9	30%	898.7	7%

デバイス間の通信バッファをキャッシュ不可領域とすることによるキャッシュフラッシュ処理の削減は，ACP を消費しないという利点がある．使用できる ACP の数には限りがあるため全てのデバイスを接続することはできず，また Ethetnet コントローラのように接続先が固定されている場合もある．また，前述のようにデータ転送量が多い場合はキャッシュ経由の方が転送速度が低下する．そのような場合に，キャッシュ不可領域を経由したデータ転送手法は有用となる．

キャッシュ不可領域を介し splice システムコールによりネットワーク送出した結果を，表 9 に示す．経由するバッファを通常のキャッシュ可能とした場合（uncached pipe 無効）とキャッシュ不可とした場合（uncached pipe 有効）の両方を測定した．キャッシュ不可とすることで，キャッシュフラッシュ処理が不要となり，ファイルサイズが 4KB の場合は 2 倍以上，64KB, 1MB の場合は 10%性能が向上した．キャッシュ不可の場合，ほぼ Gigabit Ethernet の帯域幅を使い切った性能となっているため，帯域幅により測定値が制限されている可能性がある．また，キャッシュフラッシュ処理が不要となったことで，CPU 利用率が大幅に減少している．

6. 関連研究

[4] は，ACP の性能を解析している．ACP はキャッシュコヒーレントであるため，高性能であり使いやすいが，必ずしも万能ではなく，例えばデータ転送量が多い場合はキャッシュ経由の方が転送速度が低下することなどを示している．

[5] は，Ethernet コントローラからの出力を，PL 部に実装したパケット処理機能に接続し，そこから ACP に接続する方法を提案している．この方法は ACP を経由することで，高速化を可能としている．一方，本論文で述べたキャッシュ不可領域を介した転送では，ACP を消費せず，高速化を実現している．

udmabuf [7] は，ユーザ空間における IO (UIO) を行うため，連続したメモリ領域を DMA 領域として確保可能とするデバイスドライバである．udmabuf は，事前にメモリ割当の上限を決める必要が有り，また UIO ドライバを対象としているため，その仕様には管理者権限が必要となる．一方，本論文で述べたキャッシュ不可領域を介した転送では，そのどちらの制限もない．

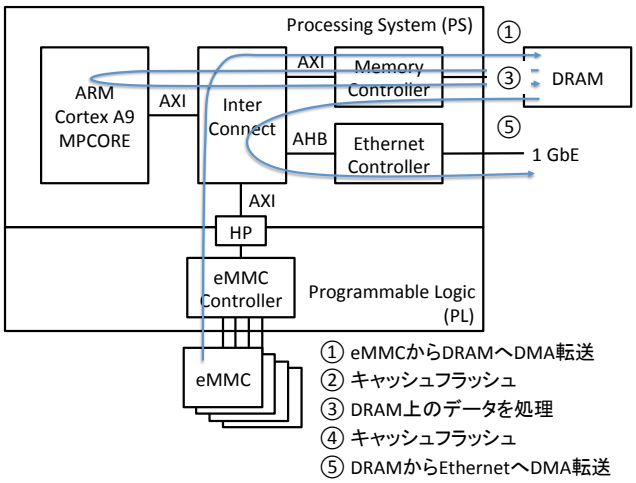
7. まとめと今後の課題

ZYNQ により構成されるシステムの場合，FPGA で様々な構成をとることが可能であるため，その構成が性能に大きく影響を与える．例えば，デバイスへの接続がキャッシュコヒーレントでない構成の場合，キャッシュフラッシュ処理が性能低下の原因となる．また，ZYNQ-7000 の CPU は近年の Intel プロセッサと比較すると低速であるため，デバイスの性能を発揮させるためには，プロセッサの処理の仕組みを把握した上で，プログラムを実行する必要がある．

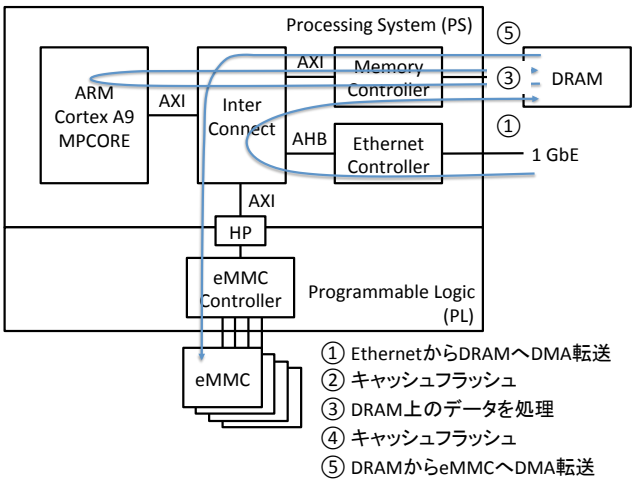
今後の課題は，Olive 上で実用的なプログラムを実際的な速度で動作させることである．本論文で述べたように，ZYNQ-7000 には様々な制約がある．工夫をすることで，単純なベンチマークプログラムでは Gigabit Ethernet の帯域幅を使い切ることができるようになった．実用的なプログラムでは，同様の工夫を適用することは単純ではない．そのような条件下で，十分な性能を出すことが課題となる．

参考文献

[1] nuttcp: a network performance measurement tool. http://www.nuttcp.net/ (2015).
[2] fio: Flexible I/O Tester. https://github.com/axboe/fio (2016).
[3] iperf3: a tool for performing network throughput measurements. http://software.es.net/iperf/ (2016).
[4] Mohammadsadegh Sadri, Christian Weis, Norbert Wehn, and Luca Benini. Energy and performance exploration of accelerator coherency port using Xilinx ZYNQ. In *Proceedings of the 10th FPGAworld Conference (FPGAworld '13)*, 2013.
[5] Zynq-7000 AP SoC - Performance - Ethernet Packet Inspection - Linux - Redirecting Packets to PL and Cache Tech Tip, http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+-+Performance+-+Ethernet+Packet+Inspection+-+Linux+-+Redirecting+Packets+to+PL+and+Cache+Tech+Tip (2013).
[6] Zynq-7000 AP SoC - Performance - Ethernet Packet Inspection - Bare Metal - Redirecting Packets to PL Tech Tip, http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+-+Performance+-+Ethernet+Packet+Inspection+-+Bare+Metal+-+Redirecting++Packets+to+PL+Tech+Tip (2015).
[7] udmabuf (User space mappable DMA Buffer). https://github.com/ikwzm/udmabuf (2016).

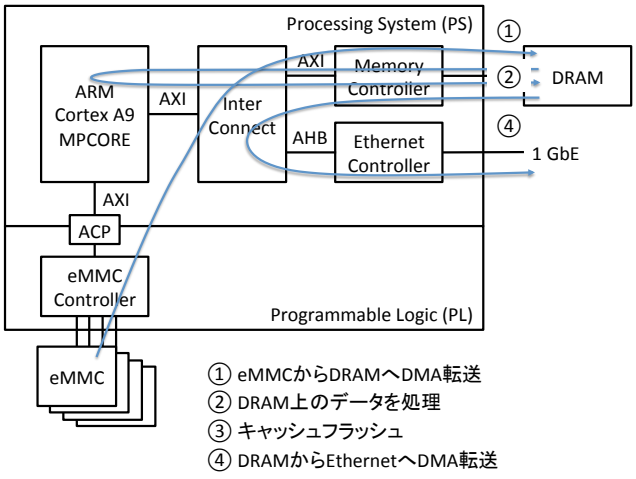


(Read & TX)

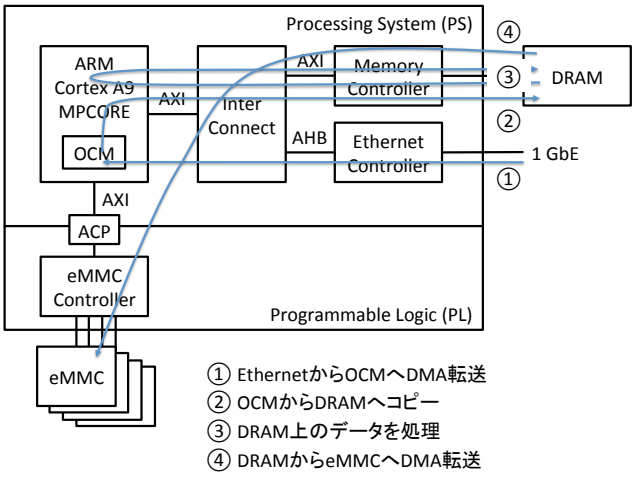


(Write & RX)

図 2　初期構成におけるデータパス

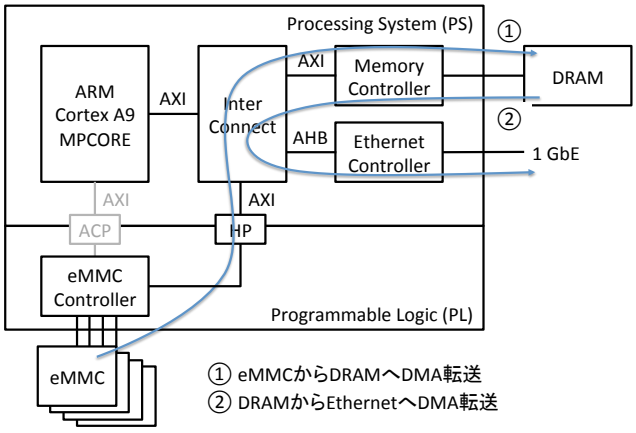


(Read & TX)



(Write & RX)

図 3　キャッシュフラッシュ処理の削減した構成におけるデータパス



(Read & TX)

図 4　デバイス間転送を行う場合におけるデータパス