一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-91,CPSY2018-101,RECONF2018-65(2019-01)

# 非整列ストリームデータ処理向け マルチコアプロセッサシステムの検討と評価

三好　健文†

† (株) トヨタ IT 開発センター

**あらまし**　組み込み機器からアップロードされるデータは，通信リソースと機器上の作業メモリスペースの都合によりサイズを小さくすることが求められ，CAN データのようにビットレベルでデータに意味を持たされる非整列データとなる．一方で，アップロードされたデータを活用するサーバ側では，組み込み機器ほどメモリスペースを節約する必要はなく，メモリ使用量を犠牲にしてもソフトウェアでの扱いやすさが重視される．このギャップを解消するため，データを受け取る側でデータ変換処理が必要になる．しかし，多種多様の組み込み機器のデータを変換するための大規模な分岐処理は，汎用プロセッサ上で効率良く実行できない．本稿では，まず，汎用プロセッサによる非整列データの処理時間について簡単な予備実験を行った結果について述べ，次に，これを解決するために検討したマルチプロセッサシステムについて述べる．提案するプロセッサは，演算ユニットに対して入力ストリームのデータを供給するとともに，命令メモリから命令を読み出し，後からそのデータに対する命令を演算ユニットにセットして実行する．この機構により，命令メモリ読み出しレイテンシを隠蔽して，間断なく入力ストリームを処理することができる．FPGA 上に実装した場合のリソース使用量と動作周波数の結果より，平均的なシナリオを想定する場合に 40Gbps 程度の入力スループットが達成できること，および，メモリレイテンシの隠蔽が可能である見積りが得られた．

**キーワード**　FPGA，ストリームデータ処理，マルチプロセッサ

## 1. はじめに

ロボットや自動車など様々な組み込み機器からアップロードされるデータをサービスに活用する試み [1] は，データの迅速かつ大容量な生成・流通・蓄積・分析・ 活用に支えられる．たとえば，ICT 端末としての機能を有する自動車から車両の状態や周囲の道路状況などの様々なデータを取得し利用するコネクテッドカーの世界市場は，2014 年に 1,300 万台以上，2025 年は 6,500 万台を超えると予測されており，これを支える通信およびデータ処理基盤が求められる．

一般に，組み込み機器からアップロードされるデータは通信リソースと機器上の作業メモリスペースの都合によりデータサイズを小さくすることが求められる．たとえば，自動車をはじめとした組み込み機器内のデータ転送フォーマットとして広く利用されている Controller Area Network (CAN) [2] [3] は，図 1 に示す 11 ビットあるいは 29 ビットの識別子と 64 ビット以下のデータフィールド，および，いくつかの制御ビットから構成される．データフィールドの 64bit のフォーマットは設計者が自由に決めることができる．データサイズを小さくするためにビットレベルで意味を持たされることが一般的である．情報をアップロードする組み込み機器とデータを受け取り利用する側でビット割当て表を共有し，アップロードする側はセンサデータをエンコード，受け取る側ではデコードする必要がある．

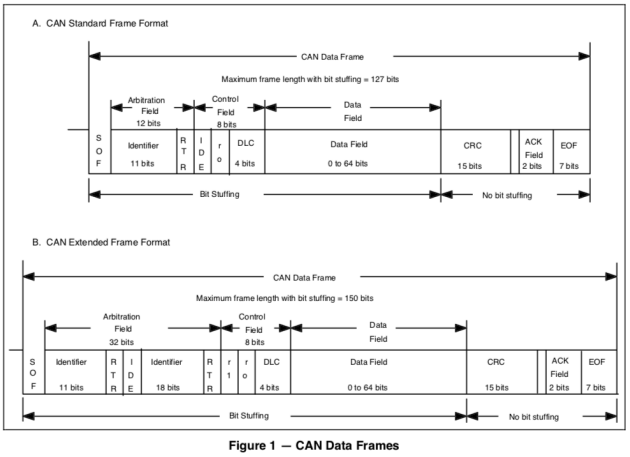


**Figure 1 — CAN Data Frames**

図 1　CAN データフレーム ( [2] より引用)
Fig. 1　CAN Data Frames (cf. [2])

一方でアップロードされたデータを活用する側には，組み込み機器ほどにメモリスペースを節約する必要はない．逆に，汎用プロセッサでの演算ユニットの構成やプログラミングコストの観点から，バイト単位あるいは 32bit などのワード単位，あるいは文字列など，メモリ使用量を犠牲にしても，ソフトウェアでの扱いやすさが重視される．このギャップを埋めるために，ビットレベルで意味を持たされた生のデータを，処理の前に扱

```
public int test(int k, int v){
  int kk = k & 0x000000FF;
  switch(kk){
  case 0: return test_0(k>>8, v)
  case 1: return test_1(k>>8, v)
  ...
  case 256: return test_256(k>>8, v)
  default: return v & 0xffffffff;
}
public int test_0(int k, int v){
  switch(k){
  case 0: return v & 0x29042ca3;
  case 1: return v & 0x1002f993;
  case 2: return v & 0x7c191c28;
  ...
  default: return v & 0xffffffff;
}
public int test_1(int k, int v){
...
```

図 2　予備評価用のプログラム
Fig. 2　A toy function to do pre-evaluation

いやすいデータ形式に変換する ETL(extract-transform-load) 処理が必要である．

ETL 処理の負荷やメンテナンスコストを考えると，すべての機器で同じビット割当てフォーマットを共通できることが望ましい．しかしながら，ロボットや自動車など長い期間利用される機器においては，開発時に想定されていた計算資源および通信資源や取り扱えるセンサの種類や個数がそれぞれ大きく異なる．そのため，すべての機器で共通のビット割当てフォーマットが利用できることを期待することは簡単ではない．そのため，データ受け取るサーバ側で，アップロードされたデータを，それぞれの機器にあわせたビット割当てフォーマットに従ってデコードすることが求められる．データはビットレベルで意味を持たされた非整列データであるため，細々としたデータに対するビットレベルでの処理と多数の候補に対する分岐が必要になる．

本稿では，まず，予備評価として，多種多様なフォーマットのデータを解析するソフトウェア処理が汎用プロセッサ上で効率良く実行できないことを示す．次に，その対策として FPGA 上に実装可能なマルチコアプロセッサシステムを提案する．最後に，提案するプロセッサシステムのリソース使用量と性能見積りの結果を示す．

## 2. 予 備 評 価

多種多様なフォーマットのデータをデコードするための計算負荷を簡単な実験で示す．デコードプログラムの実装方法には幾つか考えられるが，一つの実装方法は，キーとなるフィールドに応じて分岐することである．この実行負荷を計測するために，図 2 に示す，キーとして与えられた値で分岐するプログラムを考える．分岐先では，適当な負荷の模擬として，乱数で生成した異なる値との論理積命令を実行する．C と Java でこのようなプログラムを用意し，分岐先の数が異なる場合の実行時間を計測した．計測に用いた計算機の諸元は表 1 の通りである．

図 3 に，C および Java で，分岐先の個数が 256 個から 65,536 個まで異なるプログラムを実行したときの時間を計測した結果を示す．いずれも，分岐を含む関数に乱数で生成した引数を与えながら $10^9$ 回呼び出して測定した．図 3 の結果から，同じ回数だけ関数が呼び出されたにも関わらず，存在する分岐先の数によって，処理時間が増加していることがわかる．

表 1　予備実験に用いた計算機の諸元
Table 1　Specification of computer for pre-evaluation

| | |
|---|---|
| CPU | Intel(R) Xeon(R) Gold 6126, 2.6GHz |
| メモリ | 256GB, DDR4 26666MHz |
| OS | CentOS 7.5 |
| C コンパイラ | 4.8.5 (-O3 でコンパイル) |
| JVM バージョン | 1.8.0 |
| perf | 3.10.0 |

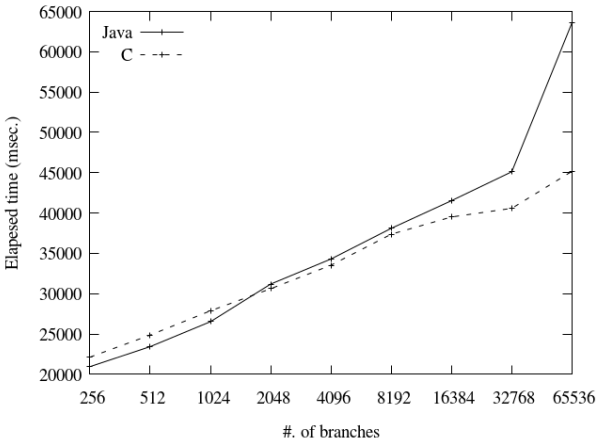


図 3　分岐数と実行時間の予備調査の結果
Fig. 3　#. of branches and processing time

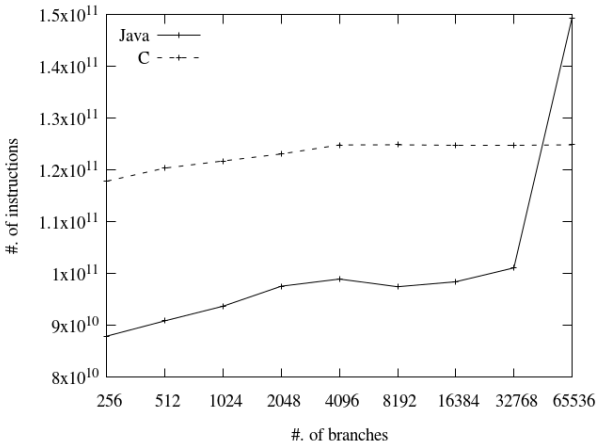


図 4　分岐数と実行命令数
Fig. 4　#. of branches and #. of executed instructions

perf コマンドを利用し，実行命令数，分岐予測ミス，L1 命令キャッシュミスを計測した結果を，それぞれ図 4，図 5，および図 6 に示す． C プログラム，および，Java プログラムの 65,536 以外では，実行命令数にはほぼ変わりがない．一方で，L1 命令キャッシュミスおよび分岐予測ミスが増加しており，汎用プロセッサの高速化機構がうまく活用されていないことがみてとれる．

実際に非整列データを整形する処理では，分岐だけではなくある程度の演算を伴なう．分岐先で適当な演算を行なう場合を想定して，XOR 命令を 16，32，64 回実行した場合の処理時間を計測した結果を，図 7 に示す．同様に，分岐数の増加に伴って実行時間が増加している傾向がみてとれる．

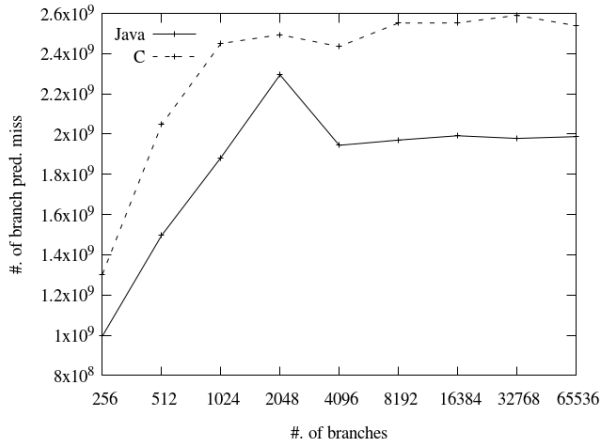


図 5　分岐数と分岐予測ミス率
Fig. 5　#. of branches and miss ratio of branch prediction

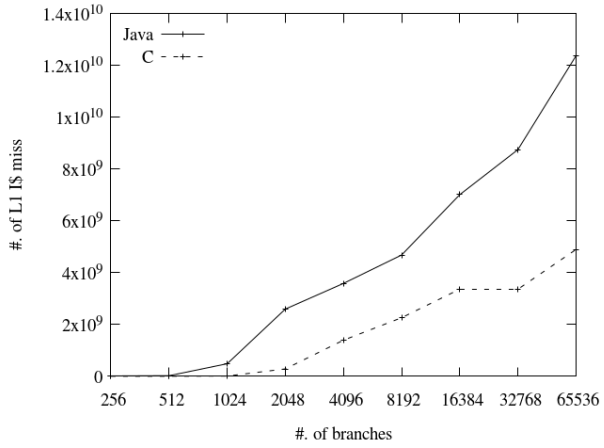


図 6　分岐数と L1 命令キャッシュミス
Fig. 6　#. of branches and miss ratio of L1 instruction cache

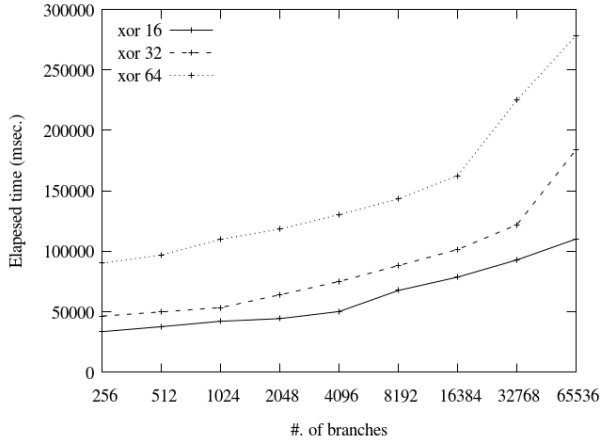


図 7　分岐先で命令実行がある場合の分岐先数と実行時間の評価結果
Fig. 7　#. of branches and #. of executed instructions with some XOR instructions

図 3 の数値の意味について考えてみる．Java プログラムでは，256 分岐と 65,536 分岐の実行時間は 20.76 秒と 63.61 秒であった．これは，それぞれ 48.17M/秒および 15.72M/秒の関数呼び出しスループットに相当する．仮に機器から送られてくる CAN データの 64bit がすべて使用される場合であっても，それぞれ，データを必要な処理に振り分けるだけの処理のスループットは，約 3.1Gbps および 1.0Gbps 程度になる．情報処理基盤を構築するネットワークが，40Gbps あるいは，さらに 100Gbps のイーサネットと高速化していることを考えると，分岐数の増加に伴う処理時間の増加が処理のボトルネックとなると考えられる．

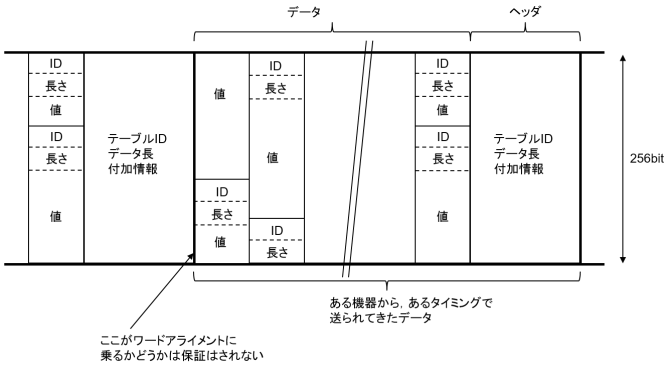


図 8　対象データストリーム
Fig. 8　Target data stream

## 3. ストリーム処理プロセッサ

多種多様な機器からアップロードされるデータの解析処理において，分岐先数の増大による演算時間増加の問題を解決するためのプロセッサシステムを考える．このプロセッサに求められる要件は，

（1） 分岐処理で性能が低下しないこと
（2） ビットレベルの処理との親和性が高いこと
（3） ユーザが処理内容を設定できること

である．(1) は，実験で示したような汎用プロセッサにみられる処理低下をおこさないことを意味する．(2) は専用プロセッサならではのメリットを活用してビットレベルの演算を効率良く処理できること，(3) は，多種多様な機器からのアップロードを受け付けるというアプリケーションの仕様上，処理に汎用性をもたせる必要があること，による．

処理性能は，少なくとも対象システムにおけるワイヤレートでの処理ができることを想定する．すなわち 40Gpbs のスループットで処理できるような演算処理機構を考える．レイテンシについては，収集したデータがアプリケーションプログラムで解析される時間に比べれば無視できるため，ここでは考慮しない．

### 3.1 対象ストリームデータ

対象データストリームの具体例を図 8 に示す．ストリームデータは，テーブル ID，データ長などからなるヘッダと，ID，長さ，および，値からなるデータが複数連なったペイロード，で構成されるパケットの連続である．各パケットヘッダのテーブル ID が機器の情報に紐づいており，テーブル ID に応じたフォーマットによってペイロードのデータを解析する．扱う対象のデータは，CAN データのようにビット単位で長さが可変であるため，ストリームは，パケットの先頭が必ずしも都合の

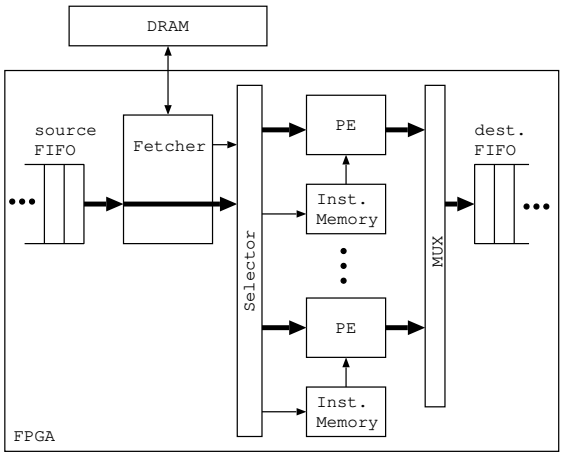


図 9　プロセッサシステムの概要図
Fig. 9　Architecture overview

よい境界でアライメントされているとは限らない非整列データである．

対象とするストリームと，そのデータに対する処理は，

- ストリームが到着するまで適用処理が分からない
- 適用対象となる処理の候補が多い
- 適用対象の処理が決定した後の処理自体は重くない

という特徴をもっている．そこで，ストリームが到着してすぐに適用対象の処理を選択できる命令フェッチ機構と，その命令フェッチの読み出しレイテンシを十分に隠蔽できる程のプロセッサと演算機構を搭載することで，ワイヤレートでの処理の実現を考える．

### 3.2　プロセッサの全体構成

提案するプロセッサシステムの全体構成を図 9 に示す．このプロセッサは，入力データを `source FIFO` から受け取り，演算結果を `dest. FIFO` 経由で出力する．演算処理は内部に持つ複数の演算ユニット (`PE`) で実行される．各演算ユニットで実行すべき命令列は `DRAM` に格納し，必要に応じて `Inst. Memory` にロードされ `PE` に供給される．

与えられたストリームのパケットの切れ目を `Fetcher` が検出し，パケットヘッダ内のテーブル ID に応じた命令列を読み出し，適切な演算ユニットに供給する．先行するパケットの処理を他の演算ユニットが演算している間に，`Fetcher` が命令列を読み出すことで，ストリームデータの処理を間断なく実行することができる．

### 3.3　命令フェッチユニット

命令フェッチユニットのタスクは，(1) 入力されるデータ中に新しいパケットのはじまりが含まれているかどうか判定すること，(2) 新しいパケットが含まている場合に該当する命令例を DRAM から読み出すこと，の二つである．図 10 に，命令フェッチユニットの内部構造を示す．入力データストリームは，ストリーム中からパケットを切り出す `Detector` と FIFO に供給される．`Detector` は，先行するパケットヘッダ中の長さフィールドによるカウンタを使って新しいパケットのはじまりを待つ．新しいパケットが到着すると，パケットヘッダ中のテーブル ID に紐づいた変換処理を DRAM から読み出す．命令

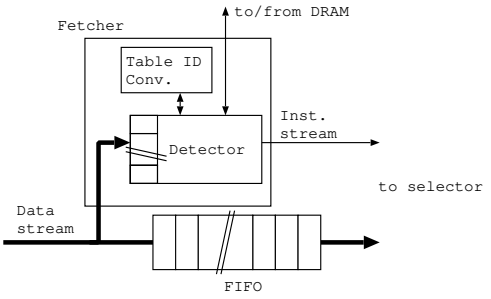


図 10　命令フェッチユニット
Fig. 10　blockdiagram of fetcher

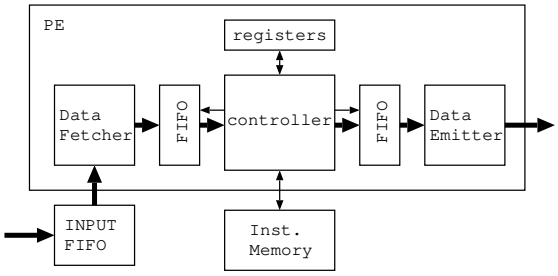


図 11　演算ユニットの内部構造
Fig. 11　Blockdiagram of PE

の読み出し先アドレスは変換テーブル `Table ID Conv.` で算出する．FIFO に供給されたデータは，後段のセレクタ `Selector` に供給される．

### 3.4　セレクタ

セレクタは，アイドル状態にある演算ユニットとデータストリームの行き先を管理する．セレクタもフェッチユニット同様にストリームからパケットを切り出す `Detector` を内部に持ち，パケットの切れ目を検出するたびに新しいアイドル状態演算ユニットにデータの供給先を切り換える．データを供給していた演算ユニットの履歴を FIFO で保持し，命令フェッチユニットが読み出した命令は，データ供給先履歴に基づいて書き出し先演算ユニットを決める．

### 3.5　演算ユニット

演算ユニットでは，与えられた命令列に沿って，入力されたストリームデータを処理する．図 11 に，演算ユニットの内部構造を示す．演算ユニットは，内部に入力 FIFO と出力 FIFO を持つ．演算ユニットの待機中にセレクタに選択されると，入力 FIFO`INPUT FIFO` に入力データが書き込まれる．その後，命令メモリに命令列が格納された後で，演算処理が開始される．演算ユニットは，加減算，乗除算，論理演算，比較および分岐命令をサポートする．オペランドにはレジスタだけではなく，入力および出力 FIFO をオペランドに指定できる．これにより 1 命令で入力ストリームを処理して出力ストリームに書き出す処理を実行できる．また，非整列データの処理を効率良く実行できるように，その演算で消費する bit 数を命令列の中で指定する．これにより，ストリームから任意のビットデータを取り出し演算することができる．

入力データの流れと命令フォーマットを図 12 に示す．処理対象のデータは ID と長さ，値から構成される．処理対象のデー

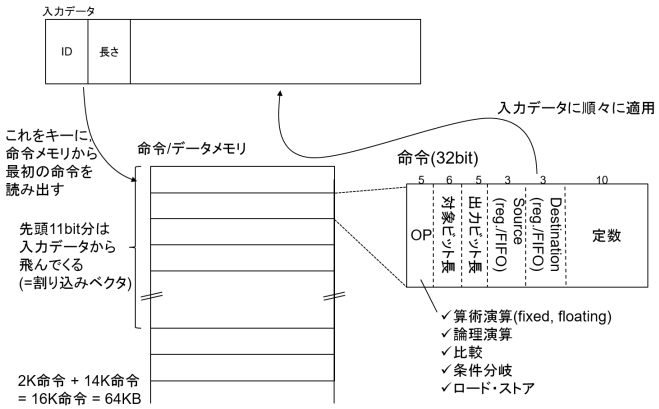


図 12　入力データと命令フォーマット
Fig. 12　Input data and instruction format

```
parser test{
  id : 11;
  value : <64;
}

task test(id=1){
  v0:int<10> { return<int> v0 + 10; }
  v1:int<7> { return<int> > v1 * 2; }
}

task test(id=2){
  v0 : int<5> {}
  v1 : int<32> {return<int> v1 * 314 / 100 + v0; }
}
```

図 13　設計言語の例
Fig. 13　Example of program

タはプロセッサの期待するワードに整列れされているわけではないため，データフェッチャ `Data Fethcer` によって整列してコントローラに投入される．演算ユニットは処理対象の ID に応じた命令を命令メモリから取り出し，値に処理を適用する．

### 3.6 記 述 方 式

提案するプロセッサシステムでのプログラミングは，(1) 機器の種類に相当するテーブル ID 毎の処理と，(2) 各機器で定義される ID 毎の処理，の 2 種類を記述する必要がある．この二種類の記述を見通しよく記述できるよう，`parser` と `task` の二種類を導入する．プログラムの例を図 13 に示す．この例は，テーブル ID が 11 の機器に対する処理を定義であり，この機器では ID が 1 と ID が 2 のデータが来ることが想定されている．

- ID が 1 の場合

（1） 0〜9bit 目までの 10bit を整数として読み出し 10 を足して出力する

（2） 10〜16bit 目までの 7bit を整数として読み出し 2 倍して出力する

- ID が 2 の場合

（1） 0〜5bit 目までの 6bit を整数として読み出す

（2） 6〜37bit 目までの 32bit を整数として読み出し 3.14 倍し，(1) で読み出した値を足して出力する

という処理を行なう．

## 4. 性 能 評 価

提案プロセッサシステムを FPGA に実装した評価結果と性能見積りについて述べる．評価に使用した FPGA の諸元を表 2 に示す．合成には Intel Quartus Prime Pro 16.1 を利用した．

表 2　評価に用いた FPGA 環境
Table 2　Specification of the FPGA board for evaluation

| | |
|---|---|
| FPGA | Intel(R) Arria10 GT FPGA 10AT115S2F45E2SG<br>1150K LEs, 53Mb(M20K) |
| メモリ | 8GB, DDR4 |
| I/F | PCIe x8 Gen3<br>QSFP 4x10G |

表 3　ハードウェアリソース
Table 3　Resource usage

| | ALM | M20K | DSP |
|---|---|---|---|
| ペリフェラル | 75342(17%) | 137(5.0%) | 2(0.13%) |
| 命令フェッチユニット | 4133(0.27%) | 30(1.1%) | |
| セレクタ | 1042(0.24%) | 5(0.18%) | |
| 演算ユニット (1 個) | | | |
| 演算ユニット | 3073(0.72%) | 30(1.1%) | 4(0.26%) |
| データフェッチ | 2076(0.49%) | 20(0.74%) | |
| データエミッタ | 1124(0.26%) | 20(0.74%) | |

表 4　演算ユニット数が 20 個と 30 個のときのリソース使用量
Table 4　Resource usage

| | ALM | M20K | DSP |
|---|---|---|---|
| 20 個 | 206925(49%) | 1572(57%) | 82(5.4%) |
| 30 個 | 269634(63%) | 2272(83%) | 122(8.0%) |

### 4.1 ハードウェアリソース

必要なユニットを FPGA 上に実装する際に必要なハードウェアリソースを評価した結果を表 3 に示す．各演算ユニットに実装した命令は，加減算，乗除算，論理演算，比較および分岐命令であり，各命令メモリを 64KB とした．また，各演算ユニットのデータ入出力 FIFO を 256bit 幅で深さ 1024+$\alpha$ で構成した．

対象 FPGA 上に 20 個あるいは 30 個のコアを実装する時のリソース使用量を表 4 に示す．表 3 の結果より，対象 FPGA に 30 個のコア数を搭載できることがわかる．また，コア数の増加に伴ない，やや使用 ALM は増加するが，ほぼコア数に比例すると考えられる．また 30 コアの構成は 200MHz のタイミング制約に対して制約違反はなく実装できた．

### 4.2 性能見積り

提案プロセッサシステムで処理する場合のスループットについて考える．ターゲットの CAN データ (ID などを含めて 128bit) のデータ 64bit にすべてデータが詰められていて，かつ，64bit をそのまま出力できる場合，最も高速に入力データを出力できる．この場合，命令をフェッチしてデータ入出力処理の実行に 3 サイクルかかるので，入力データに対する処理のスループットは 8.5Gbps になる．考えられる平均的なシナリオとしては 64bit データを 16bit ずつの 4 つのデータに分割して，それぞれに定数を加算するという処理が考えられる．この場合，の入力データに対する処理スループットは，約 2Gbps になる．ワイヤレート 40Gpbs での処理を行う場合には 20 並列できればよいが，これは，表 4 の結果より実現可能であるとい

える．

複数のコアで演算を実行させるためには，命令フェッチユニットによる命令読み出しレイテンシが十分短い必要がある．さもなければ，データが演算ユニットの FIFO に格納されても演算ユニットは動作を開始することはできない．ここで，命令を DDR メモリから読み出すレイテンシを $L$ とする．各パケットのサイズが平均して $l \times 256bit$ とすると演算ユニット数が $n$ に対して $l \times n$ サイクル分のレイテンシが命令フェッチユニットに許される．すなわち，演算ユニット数が 30 の時，メモリ読み出しレイテンシ 200 サイクルを隠蔽するとすると，各パケットのサイズが $7 \times 256 = 1792$bit 必要になる．これは CAN データ 13〜14 個程度，224Byte に相当する．

## 5. 関 連 研 究

汎用プロセッサの代わりに，FPGA を利用したストリーム処理の高速化する手法には，文献 [4] [5] などがある．FPGA ではデータの移動と計算を同時に行なうことができるため，ストリーム処理と相性が良いことが知られている．特に，決められたデータに対する演算器を並べておくことで，データ並列性，パイプライン並列性を活用して効率良い処理ができる．一方で，本稿で対象とする処理も同様のストリーム処理ではあるが，どのような処理を行なうかはデータが到着するまでは決められない．また，データに対して行なう処理の種類が多いことから，これをすべて FPGA 上に展開しておくことは現実的ではない．そのため，何かしらのプロセッシング機構が必要になると考えられる．

汎用プロセッサで ETL 処理を高速化するためのマイクロプロセッサアーキテクチャに UDP [6] がある．UDP は複数の命令を発行する機構を備えることで複雑な分岐処理を効率良く処理できる．しかしながら FPGA 上に，この機構を導入しようとすると複数ポートのメモリと配線リソースが必要となり ASIC を作るように効率良く実装することは簡単ではない．

パケット処理のためのマルチ VLIW プロセッサの検討に V-PMP [7] がある．しかし，V-PMP では命令の読み込み機構については考慮されていない．ストリームに対して多種多様な処理を選択して，間断なく処理を実行するためには，本稿で考えたような命令読み出し機構が必要である．

## 6. ま と め

組み込み機器からアップロードされるストリームデータの処理では入力データに応じて多種多様な処理を行なう必要があることから，汎用プロセッサでは実行時間が増加することを示し，これを解決するためのプロセッサシステムを検討した．検討したプロセッサは，ストリームデータを解析することで必要な命令を読み出す命令フェッチユニットと，複数の演算ユニットで構成される．複数の演算ユニットに対して，入力データをワイヤレートで供給し，後からその演算ユニットに必要な命令をセットして実行を開始する．そのため命令メモリからの読み出しレイテンシを隠蔽することができ，間断なく入力ストリームを処理することができる．FPGA 上に実装した場合のリソース使用量と動作周波数の結果より，平均的なシナリオを想定する場合に 40Gbps 程度の入力スループットが達成できること，および，メモリレイテンシの隠蔽が可能である見積りが得られた．

今後の課題として，実データに適用することで本プロセッサシステムの有効性を確認する必要がある．また，CAN データ以外のデータフォーマット，たとえば msgpack などへの適用が考えられる．

## 文　　献


[1] 総務省，“平成 27 年版情報通信白書 第 4 章 暮らしの未来と ICT,” 2015. http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h27/pdf/n4100000.pdf

[2] ISO 11783-3:2014, “Tractors and machinery for agriculture and forestry – serial control and communications data network – part 3: Data link layer,” May 2014. https://www.iso.org/standard/57554.html

[3] ISO 11898-1:2015, “Road vehicles-controller area network (can) -part 1 : Data link layer and physical signalling,” Dec. 2015. https://www.iso.org/standard/63648.html

[4] J. Hou, Y. Zhu, L. Kong, Z. Wang, S. Du, S. Song, and T. Huang, “A Case Study of Accelerating Apache Spark with FPGA,” 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp.855–860, Aug. 2018.

[5] K. Nakamura, A. Hayashi, and H. Matsutani, “An fpga-based low-latency network processing for spark streaming,” 2016 IEEE International Conference on Big Data (Big Data), pp.2410–2415, Dec. 2016.

[6] Y. Fang, C. Zou, A.J. Elmore, and A.A. Chien, “Udp: A programmable accelerator for extract-transform-load workloads and more,” Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, pp.55–68, MICRO-50 ’17, ACM, New York, NY, USA, 2017. http://doi.acm.org/10.1145/3123939.3123983

[7] M.S. Brunella, S. Pontarelli, M. Bonola, and G. Bianchi, “V-PMP: a VLIW Packet Manipulator Processor,” 5G PPP@EUCNC_2018, June 2018. https://5g-ppp.eu/5g-ppp-at-eucnc-2018/