平成 29 年度

修士学位論文

データ駆動センサハブを用いた IoT 向け SoC に関する研究

A Study on IoT SoC integrated with Data-Driven Sensor Hub

1205081　　渋田 広樹

指導教員　　岩田 誠

平成 30 年 2 月 28 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報システム工学コース

要 旨

データ駆動センサハブを用いた IoT 向け SoC に関する研究

渋田 広樹

IoT（Internet of Things）の端末数の増加に伴い，クラウドの負荷の増加が問題となっている．そのため，クラウドとエッジ（辺縁部）で負荷分散を行うエッジヘビーコンピューティングが各所で研究されている [1]．この各エッジ部においては，SoC の性能向上が求められている．一般的なエッジは，多様なセンサを統合するセンサハブ，アプリケーションプロセッサ（AP），通信モジュールが主な構成となる．センサハブは，センサとの通信処理を担うため，AP が常時駆動する必要がなくなり，高性能かつ高消費電力な AP の稼働時間を低減できる．しかし，一般的なセンサハブはデータの到着毎に割り込み処理を必要とし，データの退避や復帰が頻繁に繰り返される．また，多数のデバイスと接続されデータの I/O の発生頻度が増加すればする程，処理時間や消費電力の浪費が増大する．

本研究では，割り込み処理なしに動作できるデータ駆動型プロセッサ（DDP：Data-Driven Processor）[2] を応用したデータ駆動型センサハブ（DDSH : Data-Driven Sensor Hub）[3] のアーキテクチャを提案し，DDSH が IoT 向け SoC として有効であることを明らかにする．DDSH は，複数のセンサデータを統合してより高度な情報を生成するセンサフュージョンの処理も可能とした DDSH コア，多種多様なセンサや AP と接続可能にする I/F で構成される．

また，DDSH を 65nm CMOS 標準セルライブラリで設計し，性能評価を行った．その結果，DDSH は一般的なセンサハブの約 50 倍の性能を達成できることを確認した．

キーワード　IoT (Internet of Things)，センサハブ，データ駆動型プロセッサ

Abstract

A Study on IoT SoC integrated with Data-Driven Sensor Hub

Hiroki SHIBUTA

With rapid increase of Internet of Things (IoT) devices, processing load on the cloud server would be tremendously increasing. In order to suppress such heavy load, edge-heavy computing has been studied at various communities. The typical edge device is generally composed of multiple sensors, a sensor hub to integrate sensed data, an application processor (AP), and a communication module. When equipped with a sensor hub, AP is free from sensor management process. Whenever the interrupt will occur in the hub, its processing resource and power will be wasted.

This study focuses data-driven processor (DDP) that can operate without any interrupt process. This paper presents an architecture of data-driven sensor hub (DDSH) processor as an extended version of DDP. DDSH is composed of the DDSH core and I/F units which convert data/packet format for adapting to individual peripheral device interface. Furthermore, the proposed DDSH supports hub instructions to realize sensor fusion on edge devices.

In this study, the DDSH circuit is synthesized using 65nm CMOS standart cells. The result indicated its throughput had achieved about 50 times faster than that of typical sensor hub, ARM Cortex-M0+.

key words IoT (Internet of Things), sensor hub, data-driven processor

目次

図目次

表目次

第 1 章

序論

IoT（Internet of Things）技術 [5] の急速な発展により、IoT デバイスの数が爆発的に増加しており，図 1.1 に示すように，2020 年には約 300 億個に達すると予測されている．IoT

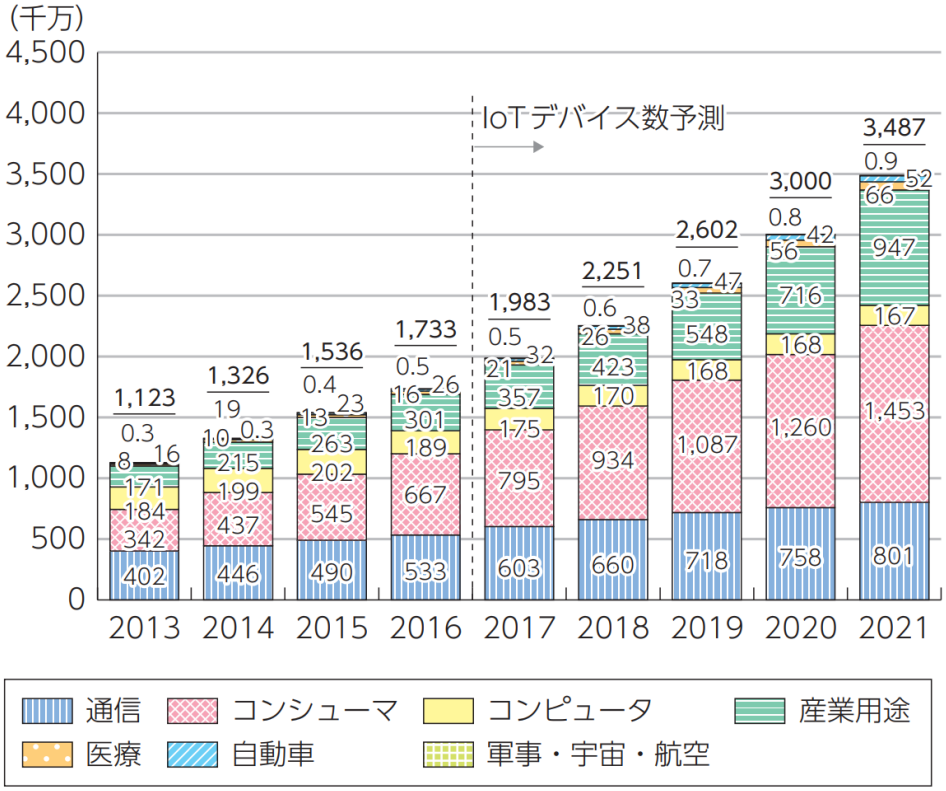

図 1.1　世界の IoT デバイス数の推移及び予測（文献 [4] より）

デバイスの増加に伴い，クラウドへ集積されるデータが増加するため，クラウドの処理負荷が高くなっている．近年は各方面でビッグデータを利用した機械学習によるデータ解析や認識などの高負荷な処理が求められており，クラウドの高性能化に対する需要が高まってい

るが，クラウドの高性能化にはコストがかかるため，コストが限られている場合は高性能化にも上限が発生する．そこで，負荷分散の手法の一つとして，センサなどを搭載した末端（エッジ）側のシステムで一次的な処理を施し，クラウドの負荷を分散させるエッジコンピューティングあるいは，フォグコンピューティングに関する研究が盛んに行われている [1]．エッジに搭載される SoC は基本的にセンサ，アプリケーションプロセッサ（AP），通信モジュールで構成されている．本稿における AP は，組み込みシステムで導入されるマイコンなどの IoT デバイスとしては比較的高性能かつ多くの消費電力を必要とする SoC のことを示す．SoC の構成として，AP と多種多様なセンサが直接接続される構成の場合，AP はセンサインターフェース処理のため常時稼働してしまうため，システム全体の消費電力が大きくなってしまう．そこで，センサハブと呼ばれるセンサ信号処理専用の超低消費電力プロセッサを導入することが近年注目されている [6]．センサデータの受信などの信号処理をセンサハブに担わせ，AP の稼働時間を減らすことでシステム全体の消費電力を低下させることが可能となる．

センサハブの役割としては，先述した最小限の消費電力でセンサデータを取得することに加え，多種多様なセンサを含むシステムを簡素化する目的もある．これは，AP とセンサ類を一対多に接続する必要があったシステムが，AP とセンサハブを一対一で接続するだけで良くなったためである．その結果，AP は機械学習などのより高度な処理のために性能のリソースを割くことが可能となる．

センサハブを搭載した一般的な SoC は，基本的に図 1.2 のようにセンサハブが多種多様なセンサと接続され，AP と一対一で接続された構成となる．センサハブは多種多様なセンサインターフェースの処理によってセンサデータを取り込み，取得したデータに対して一次的な処理を施し，処理後のデータを AP へと送信する動作が基本となる．AP ではセンサハブから得られたデータに対して，意味のあるデータに変換するため，より高度な処理を行い，通信モジュールを介してクラウドにデータを送信する．また，エッジコンピューティングの初歩的な例として，2 つのイメージセンサの画像から視差を計算する処理が自動運転などの場面で必要とされる．このような複数のセンサの情報を組み合わせてより有意な情報に

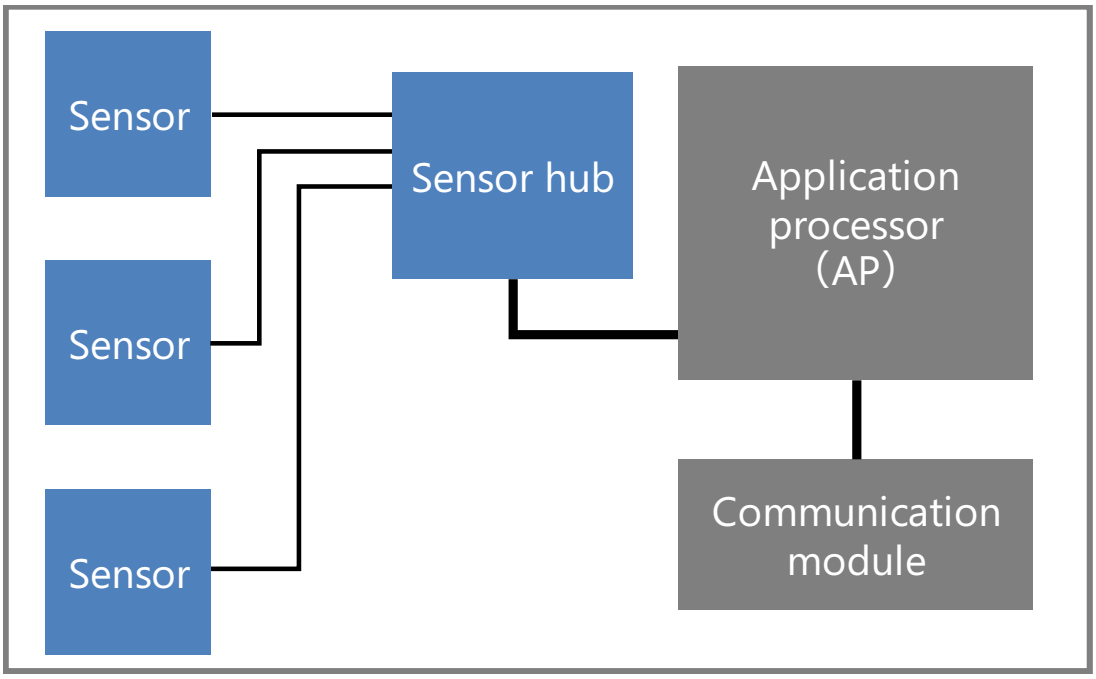

図 1.2　一般的な IoT で使われる SoC の構成

統合する処理は，センサフュージョン [7] と呼ばれることもある．

センサハブの例として，米 Apple 社の専用センサハブ IC「モーション・コプロセッサ」や米 Atmel 社の「SAM D20」などがある．これらは，主にスマートフォンやタブレットに搭載されており，加速度センサ，コンパス，ジャイロスコープ，気圧計といったセンサ類から取得したデータを処理して AP へデータを出力する．センサハブを搭載することで，性能が高く消費電力の大きいプロセッサの稼働時間を減らして，処理分散と低消費電力化を実現している．これらの一般的なセンサハブは，消費電力の小さい ARM Cortex-M シリーズをセンサハブとして動作させることが主流となっている．

一般的なセンサハブは ARM アーキテクチャなどのノイマン型の同期回路であり，複数のセンサデータを取得する場合，割り込み処理等の前処理を必要とする．その結果，割り込み処理の前処理として，処理中のデータを退避させる命令や，元の処理に復帰させる命令を実行する必要がある．具体的には，割り込み時には復帰アドレスの作成，現在の作業レジスタや状態レジスタの退避の処理が必要となり，復帰時にはスタックから作業レジスタや状態レジスタを復帰し，復帰アドレスを基に復帰する処理が実行される．このような割り込み処

理は，組み込みシステムなどではデータの入出力時に発生する．複数のセンサと接続され．データの I/O が頻繁に発生するセンサハブでは，総処理時間の増加や，それに伴った割り込み処理自体で消費する電力の無駄が問題となる．近年では，山や川等の自然環境で稼働する IoT システムも増えてきており，安定した電力供給が見込めない場合もあるため，なるべく消費電力を抑える必要がある．

この問題に対して，アルゴリズム，ソフトウェア，ハードウェア等，多方面から研究が行われている．我々の研究室では，受動的にパイプライン処理を開始するデータ駆動型プロセッサ (以下，DDP : Data-Driven Processor) の研究を行っており [2]．DDP は転送制御信号のやり取りで動作するため，データの到着をトリガとして，並列的にパイプライン処理を開始することができる．そのため，割り込み処理を必要とせず，一般的なセンサハブと比較して，性能向上並びに低消費電力化が期待できる．

本研究では，DDP を利用したデータ駆動型センサハブ（以下，DDSH : Data-Driven Sensor Hub）[3] のアーキテクチャについて検討している．特に，DDSH では，センサからデータの入力を受けて AP に出力する I/O の動作が主な動作となるので，I/O に注目して DDP を拡張したデータ駆動型センサハブの設計を行い，センサハブとして必要とされる要件を満たしたアーキテクチャについて検討を行った．

本稿において，第 2 章ではエッジコンピューティングの特性から IoT 向けの SoC の要件をまとめ，IoT 向けの SoC に組み込まれるセンサハブの要求条件について議論する．この要求条件を基に一般的なセンサハブの問題点を踏まえ，DDP のアーキテクチャをベースとした DDSH を含んだ SoC の設計方針を述べる．

第 3 章では，第 2 章で述べたセンサハブの要求仕様から DDSH の命令セットアーキテクチャ（ISA）を提案する．DDSH は複数 I/O デバイスと接続されることや，センサフュージョンなどの処理を実行する．その ISA を基に DDSH のマイクロアーキテクチャについて述べる．具体的には DDSH のアーキテクチャとして DDSH コアとセンサや AP の I/F の仕様について述べ，従来の DDP との差分を明らかにする．

第 4 章では DDSH と DDP を 65nm CMOS 標準セルライブラリを用いて論理合成ツー

ル，Synopsys 社 Design Cpmpiler で論理合成を行い，回路面積について比較した．また，一般的な同期回路のセンサハブと DDSH の性能評価を行った結果を考察する．

第 5 章では，DDSH の有効性をまとめるとともに，今後の課題について述べる．

第 2 章

IoT 向け SoC の要件

2.1 緒言

近年の IoT 技術の発展により，高機能な SoC に対する需要が高まっている．その中で，センサハブと呼ばれる超低消費電力なプロセッサをセンサ信号専用に導入されている．多様なセンサと接続されるセンサハブは，各データのフォーマットや転送タイミングを考慮してヘテロジニアスに動作する必要がある．そこで，異なるタイミングの信号を柔軟に処理することができるデータ駆動型プロセッサ（DDP）をセンサハブとして動作させることで，従来のセンサハブのような割り込み処理が不要になる．本章では，センサハブの要件をまとめ，DDP をセンサハブとして拡張した DDSH の設計方針を述べる．

2.2 エッジコンピューティングにおける SoC

IoT システムは，数多くの IoT デバイスから収集したデータをクラウド上で統計処理を行う中央集中型のシステムが多い．しかし，近年の IoT デバイスの増加に伴い，クラウドの負荷が増大している．そこで，エッジコンピューティングと呼ばれるエッジ（末端）部で一次的な処理を行い，クラウドの処理負担を軽減させる技術が注目されている．エッジコンピューティングはクラウドの負荷を分散させる技術の一つである．しかし，図 2.1 のようにエッジコンピューティングにおけるエッジは正確に定義されておらず，クラウドと別に簡易的な処理を行うサーバを「エッジサーバ」と呼び，エッジサーバとセンサシステムを含んだものをエッジと呼ぶ場合もある．また，Raspberry pi のようなシングルボードコンピュータ

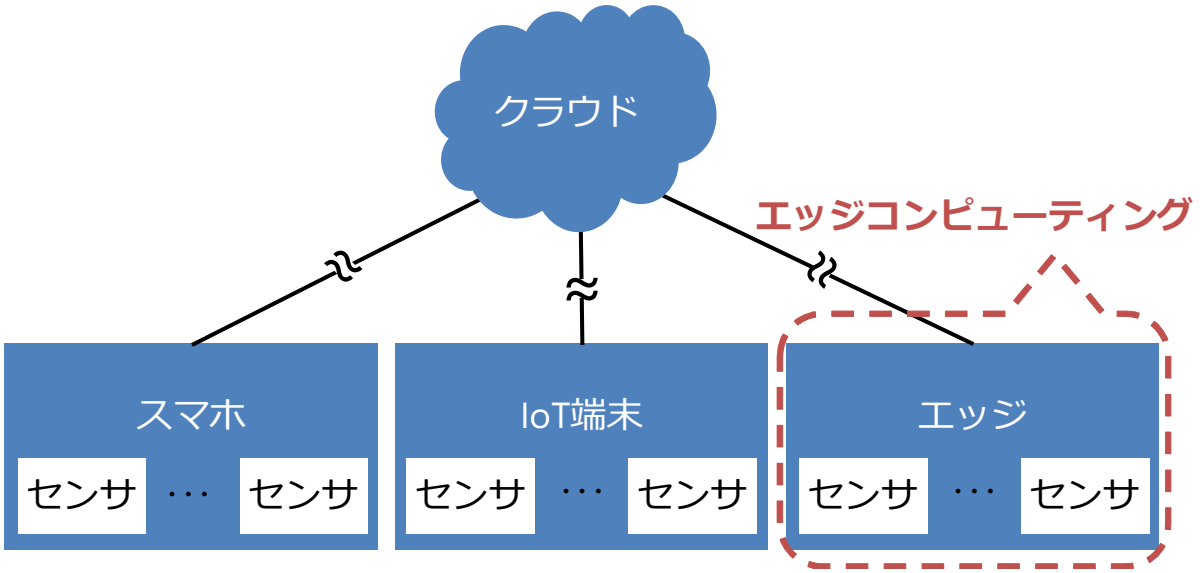

図 2.1　エッジコンピューティングの概略図

とセンサだけを含んだシステムをエッジと呼ぶ場合もある．本論では，センサとアプリケーションプロセッサ（AP）と通信モジュールを含んだ組み込みシステムのことをエッジと定義する．

エッジの主な役割は，クラウドで実行していた処理の一部を担い，センサのデータ量を圧縮することでクラウドの負荷を軽減させることと，通信遅延を減少させレスポンス性を高めることである．センサのデータ量を圧縮する技術の一つとして，複数のセンサデータを組み合わせてより高次な情報を作りだすセンサフュージョンといった技術が注目されている．例えば，自動車の周囲の状況を把握するため，カメラや赤外線センサをセンサフュージョンによって組み合わせ，より正確に周囲の状況を把握することが可能となる．エッジでは，このようなセンサフュージョンの処理を実行することが求めらている．

近年のエッジコンピューティングでは，機械学習などの高度で処理負荷の高いアプリケーションはクラウドで実行される．そのため，エッジ側の処理負担の比率が高まっており，よりエッジに搭載される SoC の性能が求められる．しかし，エッジは安定した電力を得られる場所で利用されるとは限らないため，低消費電力での動作が望まれる．SoC では高性能であることと低消費電力であることはトレードオフの関係であり，両立は難しいとされている．

このことから，高性能・高消費電力な AP を高度な処理に専念させるため，低消費電力なセンサハブを搭載した SoC が増加している．センサハブを搭載することで AP はセンサとのインターフェースの処理を省略することができ，稼働時間を短縮することが可能となる．以降，IoT システム向けのセンサハブについて議論する．

2.3　センサハブの要求条件

センサハブの主な役割は，多種多様なセンサを含むシステムを簡素化することと最小限の消費電力でセンサデータを取得することである．AP が多種多様なセンサと接続する場合，センサハブを導入することにより，センサハブとの一対一の接続で構成することができる．また，高性能・高消費電力な AP の稼働時間を減少させることができるので，システム全体の消費電力を抑えることができる．

一般的なセンサハブは図 1.2 のように構成される．複数のセンサと接続され，AP と一対一で接続されている．基本的な動作は，AP から実行命令を受け取り，センサからデータを取得して，センサフュージョンなどの処理を施してから，AP へ送信する．

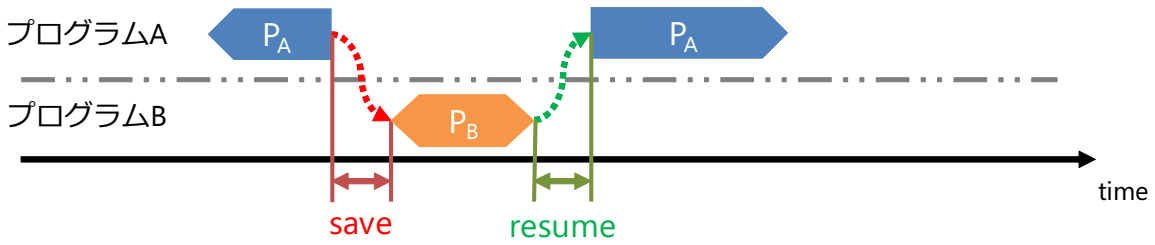

図 2.2　ノイマン型センサハブで発生する I/O オーバヘッド

一般的なノイマン型のセンサハブであれば，複数のセンサがバスに繋がれ，センサハブのプロセッサが常時稼働して，入力が発生するとポーリングや割り込み処理でデータの処理を開始する．ポーリングは入力機器ごとの状態ビットを定期的にチェックする方式で，プロセッサの多くの時間が無駄に浪費される欠点がある．プロセッサよりも格段に低速な入出力機器の状態ビットの変化を周期的にチェックするため，状態が変化しなくても無駄にチェッ

クする可能性がある．また，割り込み処理の場合，図 2.2 に示すように割り込み前に使用していたレジスタを退避（save）させ，割り込み後に退避させたレジスタを復帰（resume）させる処理が必要である．そのため，割り込み処理によるデータの退避と復帰の処理を繰り返し実行することで誘発される性能低下や消費電力の増加を引き起こすという問題があった．例えば，ARM Cortex-M0+の場合，下記アセンブリプログラムのような処理が必要とされており，save に 24 clock cycle，resume に 15 clock cycle が必要である [8].

```
;; save
SUB     lr, lr, #4              ; construct the return address
PUSH    {lr}                    ; and push the adjusted lr_IRQ
MRS     lr, SPSR                ; copy spsr_IRQ to lr
PUSH    {R0-R4,R12,lr}          ; save AAPCS regs and spsr_IRQ
BL      identify_and_clear_source
MSR     CPSR_c, #0x9F           ; switch to SYS mode, IRQ is
                                ; still disabled. USR mode
                                ; registers are now current.
AND     R1, sp, #4              ; test alignment of the stack
SUB     sp, sp, R1              ; remove any misalignment (0 or 4)
PUSH    {R1,lr}                 ; store the adjustment and lr_USR
MSR     CPSR_c, #0x1F           ; enable IRQ
BL      C_irq_handler

;; resume
MSR     CPSR_c, #0x9F           ; disable IRQ, remain in SYS mode
POP     {R1,lr}                 ; restore stack adjustment and lr_USR
ADD     sp, sp, R1              ; add the stack adjustment (0 or 4)
MSR     CPSR_c, #0x92           ; switch to IRQ mode and keep IRQ
                                ; disabled. FIQ is still enabled.
POP     {R0-R4,R12,lr}          ; restore registers and
MSR     SPSR_cxsf, lr           ; spsr_IRQ
LDM     sp!, {pc}^              ; return from IRQ.
END
```

以上から，センサハブは，多種多様なセンサと接続可能であり，それらのセンサからセンサフュージョンの処理が可能で，I/O のオーバヘッドが少ないことが要求条件となる．

2.4　データ駆動型センサハブ DDSH の構想

上記の I/O オーバヘッドが発生する問題を解決するには，割り込みのようにデータが発生次第動作し，割り込み処理を不要とする受動的に動作するプロセッサが望ましい．マルチスレッド実行が可能なスーパスカラアーキテクチャは一般には High-End のエンタープライズ用の大型 CPU に向いたアーキテクチャであるため，本研究で対象とする低電力小型センサハブには向いていない．そこで，本研究では，データの到着をトリガとしてパイプライン動作する DDP に着目して，DDP アーキテクチャをベースとしたセンサハブを設計し，その有用性を明らかにすることを目的とした．

DDP は 1 入力 1 出力であり，センサハブの要件の複数入力複数出力を満たしていないという問題がある．また，データの入出力時に DDP のパケットフォーマットと外部機器のフォーマットを相互変換する必要がある．また，DDP はセンサなどの識別情報を書き換える機能を備えておらず，2.2 節で議論したようなセンサフュージョンなどの複数のデバイスの情報を統合した処理が不可能となっている．

DDP をセンサハブとして活用する際に不足している機能を受けて，提案回路は入出力部を拡張する必要がある．センサからの入力時に，センサデータを DDP のパケットフォーマットに変換する機構と，出力時に出力先情報を読み取り経路選択を行い，外部機器のフォーマットに準拠させる機構が必要である．また，センサフュージョンなどの処理を可能にした命令セットアーキテクチャの設計を行う．

2.5　結言

本章では，エッジコンピューティングにおける SoC の需要が高まっていることに対し，エッジ部の SoC でセンサハブを導入するシステムの増加を受け，センサハブの要求条件をまとめた．その要求条件を満たすため，DDP を採用したセンサハブであるデータ駆動センサハブ（DDSH）の設計方針をまとめた．DDSH は多種多様なセンサと接続可能であり，センサフュージョンの処理を可能にすることで実現できる．

2.5 結言

次章では，本章で設定した設計方針を踏まえた上での詳細な I/O 部のアーキテクチャについて述べる．具体的には，命令セットアーキテクチャや入出力機構の詳細部の設計を行う．

本章では，一般的な IoT システムとして，センサのデータを基にクラウドなどで高度なアプリケーションを実装するシステムを想定している．しかし，IoT システムの一例として，車載システムなどのリアルタイム処理を求めるシステムについては議論していないため，それらのシステムを想定する場合，リアルタイム処理についても議論する必要がある．

第 3 章

DDSH の構成

3.1　緒言

一般的なセンサハブでは，I/O オーバヘッドとして，割り込み処理などが必要である．そのため，I/O オーバヘッドを低減させるデータ駆動型センサハブの構成としてを考案した．本章では，DDSH の基本構成に加え，センサハブとして必要な処理であるセンサフュージョンなどの処理を実行可能な命令セットアーキテクチャや，多種多様なセンサや AP と接続可能な I/F について述べる．

3.2　命令セットアーキテクチャ（ISA）

センサハブはセンサや AP と通信するため，データの I/O に関する処理が多用される．一般的なセンサハブは基本的に割り込みや DMA（Direct Memory Access）によってデータ入出力を実現するため，割り込みの前処理として，データの退避や復帰に命令リソースを割く必要がある．DDP はデータを含んだパケットが入力されるとそのパケットに対して処理を施すため，そのような I/O オーバヘッドが必要ない．しかし，DDP は 1 入力 1 出力の構成であり，複数デバイスと接続するための命令セットアーキテクチャISA（Instruction Set Architecture）は定義されていない．そのため，複数デバイスと接続されることを前提とした ISA を定義する必要がある．

複数デバイスを扱う場合，どのデバイスからの入力で，どのデバイスへ出力するかを判断しなければならないため，デバイスの識別子が必要となる．入力時には，その識別子をデー

タに付加する必要がある．出力時には，デバイスの識別子と別に宛先情報も必要となるため，これらの情報を保持しなければならない．そのため，識別子と出力先情報を同時に保持可能なアーキテクチャを設計する．

また，センサフュージョンを可能とするため，種類の違うデバイス同士のデータを計算可能にする必要がある．デバイスの識別子が異なるデータ同士を計算するため，識別子を変更する命令が必要となる．さらに，輝度センサの値によって CMOS カメラと赤外線カメラを切り替える，といった出力先も計算結果によって変更させる処理も想定できるため，出力先情報を書き換える命令も必要となる．

以上のことから，識別子を変更する等の命令の設計が必要となるため，基本的な演算やメモリアクセス命令が定義されている DDP の ISA からセンサハブ向きに拡張した命令について述べる．

- out

  パケットを外部に出力する命令である．その際，dest の値に応じて出力先を決定する．
- chgdest

  出力先情報である dest をデータメモリからロードした値に書き換える命令である．その際，c フラグによって出力先を分岐させることができる．
- chgcol

  デバイスの識別子である color を第 2 オペランドの値に書き換える命令である．センサフュージョンなどの異なる種類のデバイス同士のデータについて計算する際に実行する．
- addgen

  gen に対して，符号付き加算を行う命令である．順番の前後したデータ同士を計算に使用する場合に実行する．

これらのセンサハブ向き命令以外にも，四則演算やシフト演算などの ISA の詳細は付録 A に記載した．以降では，上記の命令を実行可能なマイクロアーキテクチャについて議論していく．

3.3　DDSH の基本構成

提案回路のデータ駆動型センサハブは図 3.1 のように構成される．DDSH コアでは，IN と OUT というステージが入出力機構として増設されており，IN では複数入力で，外部から入力したデータを DDP のパケットフォーマットに変換するステージである．OUT は DDP 内で指定した宛先情報を元に外部機器への経路選択や外部機器とのデータフォーマットを変換するステージである．また，各種センサや AP の I/F として同期回路と非同期回路を変換可能な Syn-Asyn I/O を取り付けることで接続可能にしている．以降，センサフュージョンなどの処理を可能としたコアアーキテクチャと，センサや AP と接続可能な I/F について祥細を述べる．

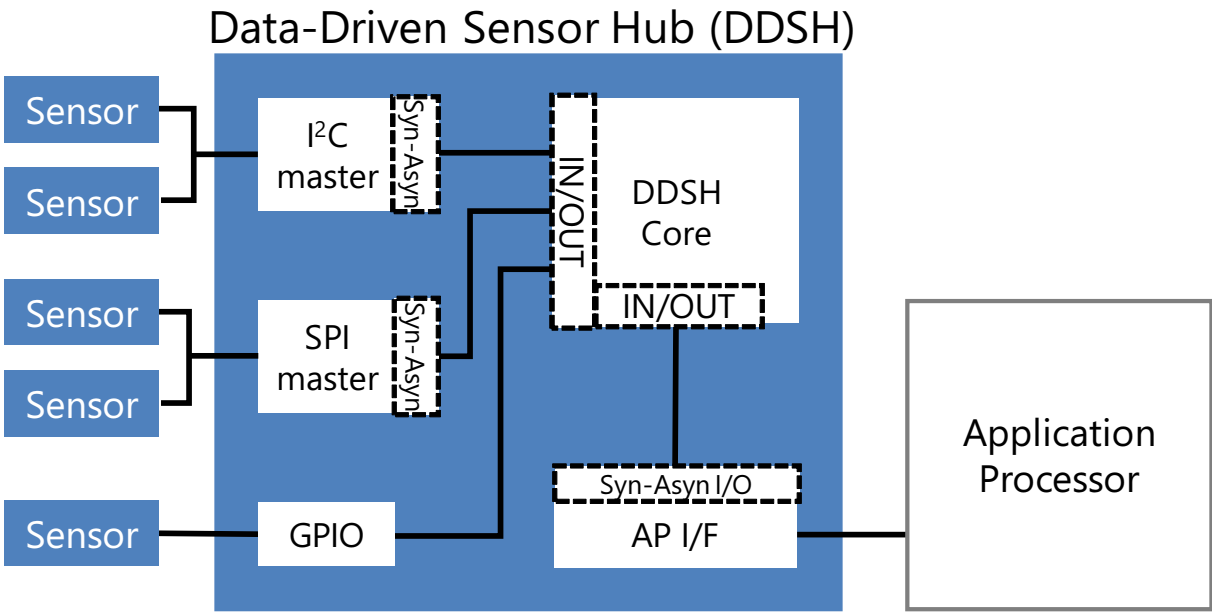

図 3.1　データ駆動センサハブの構成

3.4　コアアーキテクチャ

DDP はセルフタイム型パイプライン（以下，STP：Self-Timed Pipeline）[2] で構成されている．STP は図 3.2 のように構成されており，複数の転送制御回路 C（以下，C 素子：

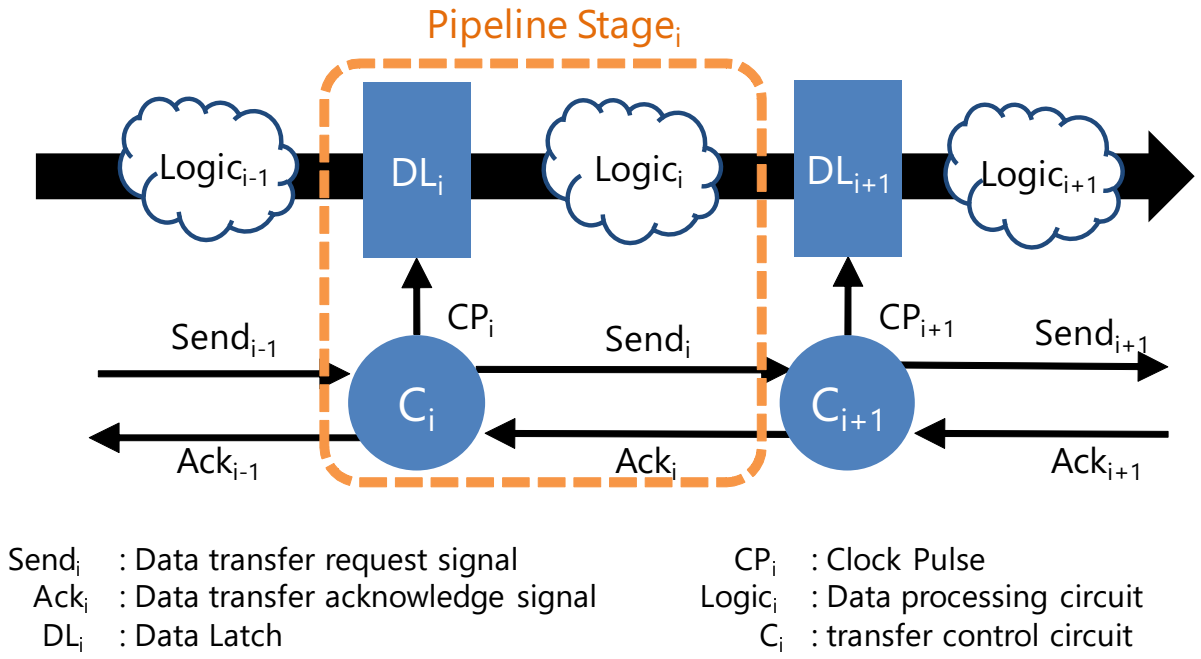

図 3.2　セルフタイム型パイプライン（STP）機構

Coincidence flip-flop）で構成される．STP の動作は基本的に C 素子間でのハンドシェイクで動作する．隣接するパイプラインステージ間の C 素子がデータ転送要求信号（send）とデータ転送許可信号（ack）をやりとりする事で，パイプラインステージ間でパケットの転送制御を行う．隣接するパイプラインステージ間でのみハンドシェイクで局所的に動作するため，配線や電力消費を極小化できるとともに，データ配送時にのみ電力を消費する自律的な省電力機能を有している [2].

STP は，図 3.3 のタイミングチャートに基づき動作している．パケットを転送開始する場合，send 信号を STP へ送信すると，転送が可能であれば ack 信号が返信される．そして，自己同期クロックである CP 信号をデータラッチに送信することで，データラッチを解放し，ステージにパケットを転送することができる．その後，処理が終了すると次ステージへパケットを転送するため，次ステージの C 素子に send 信号を送信して転送を行う．このように send 信号さえ受け取ることができれば，動作を開始することが可能なパイプライン機構となっている．

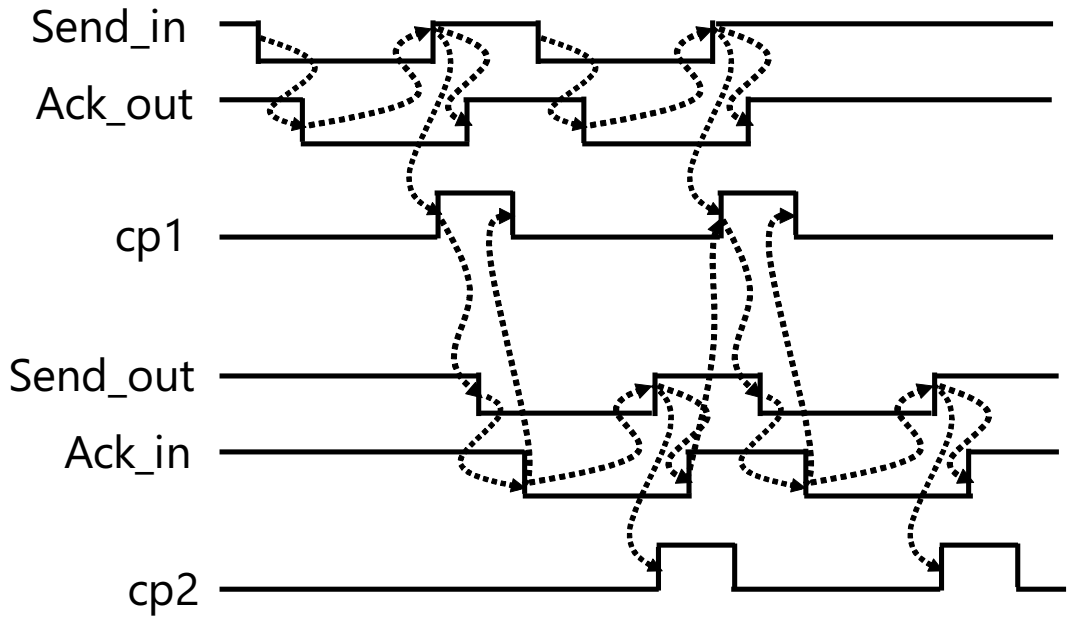

図 3.3　STP のタイミングチャート

STP を採用した DDP はセンサデータの到着をトリガに並列的にパイプライン処理を行う．DDP の構成を図 3.4 に示す．DDP を構成する各ステージの機能と役割の概略を以下に記す.

- パケット合流機構（M：Merge）

  DDSH コアは環状パイプラインを周回するパケットに対して各ステージで処理を施すアーキテクチャのため，周回する経路と外部から入力される経路が必要となる．そのため，パケット合流機構ではパケットを外部から入力されるパケットと，周回してきたパケットの合流を調停し，CST へパケットを出力する.

- 定数読み出し機構（CST：Constant Memory）

  定数演算を実行する場合，定数を格納しておくメモリ領域が必要となるため，定数読み出し機構内のメモリに，演算に用いる定数が格納されている．パケットが持つ情報により，定数を読みだす/読み出さないを判別する．読みだした場合は，その情報をパケットに付加して MM へパケットを出力する．読み出さない場合は，入力されたパケット

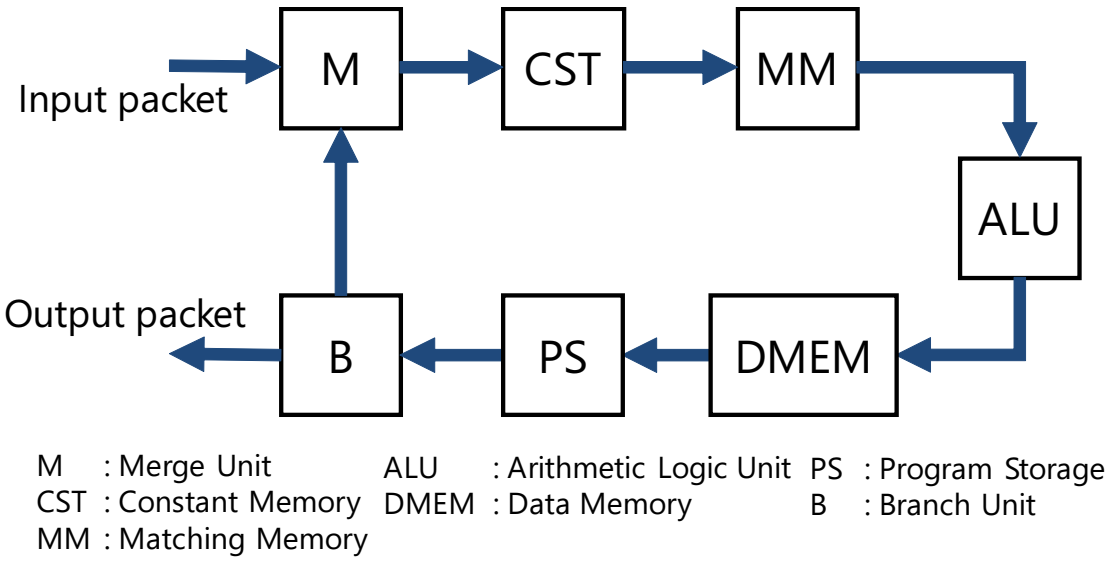

図 3.4　データ駆動型プロセッサ（DDP）の構成

をそのまま通過させる．

- パケット待ち合わせ機構（MM：Matching Memory）

  DDSH コアで 2 項演算を実行する場合は，2 つのパケットに保有されているオペランド同士の演算となるため，2 つのパケットを 1 つのパケットに変換する機構が必要である．パケット待ち合わせ機構では，CST からパケットを受け取り，一時的に内部メモリに保持する．そして，演算実行の際にペアとなるパケットとの待ち合わせを行う．待ち合わせ対象となるパケットが到着すると，パケットの処理が開始（発火）され，ALU へオペランドが対のパケットを出力する．

- 演算機構（ALU：Arithmetic Logic Unit）

  演算機構では，発火パケットの保持するオペレーションコードに従い，演算処理を行う．また，その後，演算結果をパケットに書き込み，DMEM へデータを出力する．その際にキャリー（C）フラグやゼロ（Z）フラグもパケットに書き込む．

- メモリアクセス機構（DMEM：Data Memory）

  メモリアクセス機構では，ALU からパケットを受け取り，ロード/ストア命令が実行されると，データメモリに対して実行する．読みだしたデータは，パケットに付加して

PS へパケットを出力する．また，外部デバイスの出力先情報も格納しており，ALU で計算したアドレスを基に出力先情報を読み出し，パケットに付加する．

- 命令フェッチ機構（PS：Program Storage）

  実行するプログラムが格納されているメモリで，DMEM から送られたデータの次の宛先ノード番号，及びオペレーションコードのフェッチを行い，パケットの持つ内容を書き換え，B にパケットを出力する．また，不要となったパケットを削除する機能も備わっており，命令コードが削除命令であれば，削除する．

- 分流制御機構（B：Branch）

  分流制御機構では，データの持つ宛先情報から，出力先を判別し，外部もしくは内部にパケットを転送する．

DDSH コアのベースとなる DDP での動作過程は，投入されたパケット（図 3.5，表 3.1）の持つ宛先情報から PS ステージで次の命令を発行し，上記ステージを周回しながら処理する．処理結果を保持したパケットは B ステージで外部へ出力されるか，引き続き DDP 内を周回して命令を実行するかを判断する．このように，DDSH コアはパケットに対して処理を施していくため，環状パイプライン上で周回するパケットが詰まらない程度であれば，多重処理が可能となる．

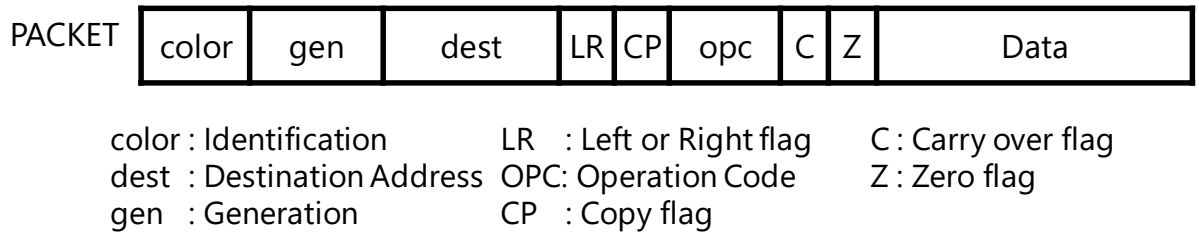

図 3.5　DDP のパケットフォーマット

また，センサハブではセンサフュージョンなどのような複数センサを統合する処理が多用されるため，多対一の構成となる．しかし，DDP は複数デバイスとの通信を想定しておらず，複数デバイスを統合した処理が不可能であった．そのため，デバイスの識別子を変更で

表 3.1　パケットが保持する情報

フィールド名	名称	情報	ビット数
color	Color	パケットの識別情報	3bit
gen	Genalation	パケットの世代（順番）	8bit
dest	Destination	パケットの宛先	7bit
LR	Left or Right	パケットの左右の識別	1bit
CP	Copy Flag	コピーの有無	1bit
opc	Operation Code	命令	3bit
C	Carry Flag	キャリーフラグ	1bit
Z	Zero Flag	ゼロフラグ	1bit
Data	Data	演算データ	16bit

きるように ALU を拡張した．

また，センサハブは一般的に複数のセンサからデータを受信し，AP と接続され，通信機器を利用してクラウドサーバへ送信したり，LED やスピーカなどの出力機器に送信する．そのため，入力されるデータや出力先のフォーマットに多様性がある．また，データ駆動型プロセッサ（DDP）には図 3.5 に示すようにパケットフォーマットが定められており，入力したデータをそのフォーマットに準拠させる必要がある．

このパケットフォーマットから各タイプの入出力機器で，どの情報が必要かを表 3.2 に示す．まず，入出力機器の分類としては，同期回路で入力順にデータを扱う機器，同期回路で入力順にデータを扱わない機器，非同期回路でデータを入力順に扱う機器，非同期回路でデータを入力順に扱わない機器で分類した．同期回路で入力順にデータを取り扱わない機器は一般的に存在しないと判断し，除外した．A/D 変換器や MCU（Micro Control Unit）などの同期回路からの同期的入力データには，センサの識別子を示す color と，センサデータの順番を示す gen を付属して，color を基にメモリである I/O PS から残りのヘッダ情報を読み出す．近接センサなどの非同期イベントでデータを発生するが，順序の

表 3.2　タイプ別 I/O フォーマット

		Synchronized I/O Devices		Asynchronized I/O Devices	
		ordered	non-ordered	ordered	non-ordered
IN	color	Input ID	–	Input ID	✓
	gen	(order)		(order)	✓
	dest	I/O PS[color]		I/O PS[color]	✓
	opc	I/O PS[color]		I/O PS[color]	✓
	flags	I/O PS[color]		I/O PS[color]	✓
	ex.	A/D, MCU	–	HES	DDNP
OUT	color	– / 2nd data	-	–	✓
	gen	(sort) / 3rd data		(sort)	✓
	dest	Output ID		Output ID	✓
	opc	–		–	✓
	flags	–		–	✓
	ex.	D/A, MCU	–	LED	DDNP

flags: LR, CP, C, Z　　✓ : transfer it with data.　　– : null

あるデータを扱う機器も同様である．非同期回路で入力順にデータを扱わない機器として，DDNP(Data-Driven Networking Processor) を想定すると，DDP と同様のパケットフォーマットで動作できるので，入出力どちらとも DDP 準拠のパケットを送受信する．

出力の場合の D/A 変換器や MCU などの同期回路で順序のあるデータを取り扱う機器は，基本的に dest のみ出力し，宛先アドレスとして扱う．color が必要な場合は 2nd Data として送信し，gen は出力前にソートしてから出力するか，3rd Data として送信する．LED などの非同期回路でも順序通りにデータを扱う機器は，dest のみ付加し，ソートして送信する．

以上のことから，DDSH コアを 3.6 のように構成した．追加したステージは入力機構の

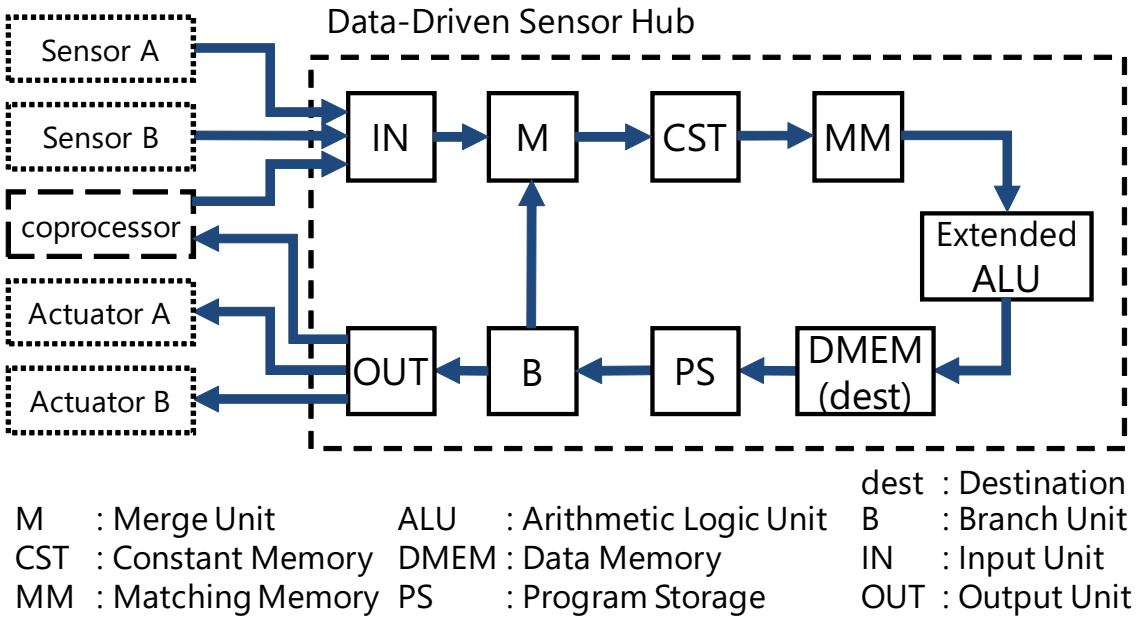

図 3.6　DDSH コアの構成

IN と出力機構の OUT であり，変更したステージは ALU と DMEM である．Extend ALU は，ISA に基づき DDP から追加された命令を実行できるように拡張された．DMEM では，従来の DDP では出力先情報を保持するステージがなかったため，DMEM に保持できるように変更した．以降，追加されたステージである入力機構と出力機構について議論し，ALU で実行可能にした命令について述べる．

入力機構

表 3.2 より，入力機構を図 3.7 のように構成した．センサなどのデータのみを入力するデバイスには，入力の ID から color を付加する Color Generator と，入力順にカウンタで数えて gen を付加する Generation Generator を取り付け，残りのヘッダは color を参照して，メモリである I/O PS から読み出す．また，ヘッダを付加して入力する機器に対しては，そのままのフォーマットで入力する．このそれぞれの入力を CM 素子と呼ばれる調停回路で，到着順にデータを受け入れる．

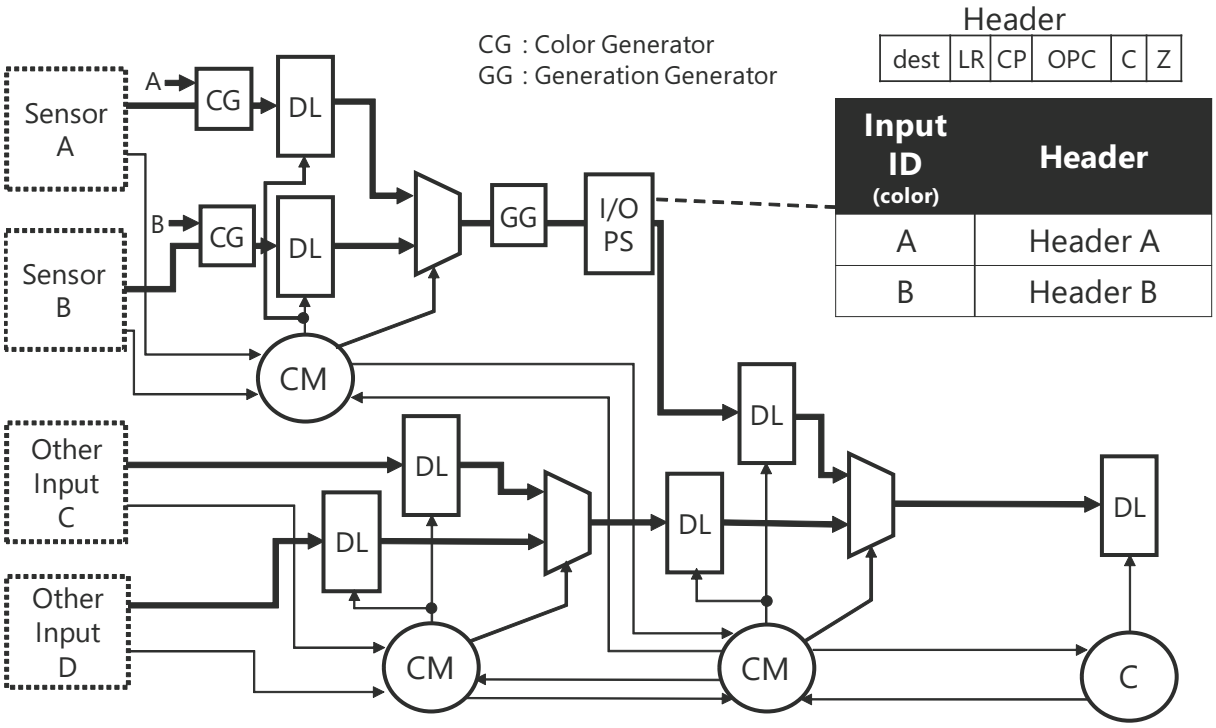

図 3.7　入力機構

出力機構

次に，パケットの経路選択を行う出力機構を図 3.8 のように構成した．DDP から出力されたパケットは，dest を基に Address Decoder で経路情報を読み出し，CB 素子と呼ばれる分岐回路で出力機器へ出力する．また，Data のみを必要とする機器などは Data Maker で gen と Data のみのパケットに変換する．そして，入力順にデータを扱う機器では，gen を基に Sorter でソートしてから出力機器へ出力する．STP ベースの Sorter は本研究の研究対象ではないが，既提案の Sorter[9] を利用することを想定する．また，出力機器には複数のデータの入力を基本とした機器もあるので，出力機器に出力する前に Matching Memory のような待ち合わせ回路を実装しておく．

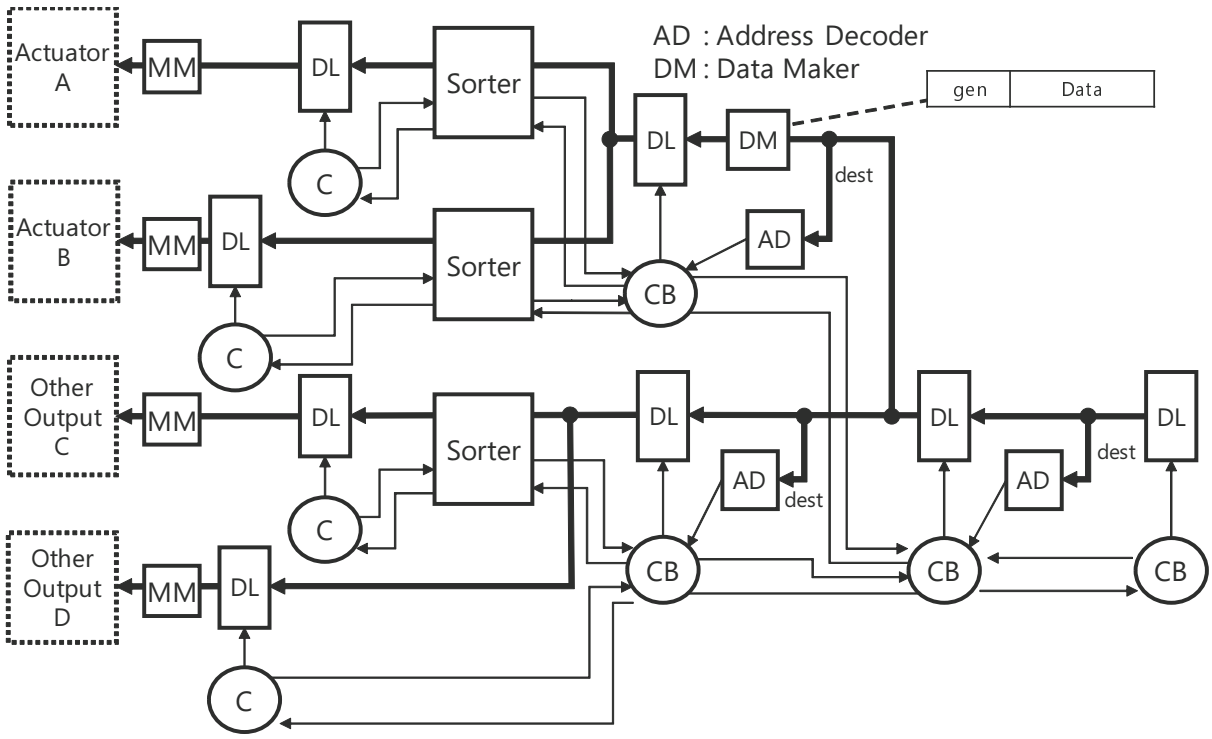

図 3.8　出力機構

3.5　センサインターフェース

センサハブは多種多様なセンサと接続されるため，多種多様な I/F プロトコルに対応する必要がある．一般的に使用されるセンサのインターフェースは，GPIO や UART などの非同期インターフェースと，I^2C や SPI などのシリアルバスを使った同期インターフェースに分類される．以降，非同期センサや同期センサの分類を行い，それぞれに適した I/F について記述していく．

3.5.1　IoT システムのセンサ

IoT システムで使われるセンサは，温度，加速度，地磁気，照度，流量などの様々なセンサがある．この多種多様なセンサの I/F は，種類によって違い，同じ温度センサでも GPIO や I^2C であったりと様々な I/F によってデータが転送される．よって，センサハブはこれらの I/F を想定しなければならない．本研究では，GPIO などの非同期 I/F と I^2C や SPI な

どの同期 I/F に分類して設計していく．

3.5.2　DDSH センサ I/F

非同期センサのインターフェースは，A/D 変換器でデジタル値に変換されていれば，直接 DDSH コアの入力機構と接続することが可能である．しかし，同期センサと接続する場合，DDSH の I/F は STP であり，I^2C や SPI などのシリアルバスを使った I/F と相性が悪く，図 3.7 や図 3.8 と直接的に接続することができない．例として，I^2C は，SCL と SDA の二本のバスで通信をやり取りする．基本的には，SCL をクロック信号として，各スレーブは SCL に同期して動作するため，I^2C のマスタはクロック信号に同期して動作する必要がある．

STP は，非同期に動作するため，I^2C と通信しようとすると，内部 DFF のセットアップホールド時間を守れなくなる恐れがある．その要因として，標準セルライブラリの DFF は面積も小さく，遅延を最小限に抑えられている．しかし，その影響でセットアップホールド時間が厳しく設定されていることが多い．そこで，図 3.9 のように論理ゲートを組み合わせた DFF を採用した．この Gate DFF には，調停機能が備わっているが，動作は比較的低速になってしまう．しかし，非同期回路の STP と接続する際に発生するタイミング違反を吸収することが可能である．よって，この Gate DFF を I^2C や SPI のマスタ回路と組み合わせることで接続が可能となる．

この Gate DFF を採用したセンサインターフェースは 3.10 のように構成される．クロックと非同期に動作する send 信号と ack 信号は，Gate DFF によってタイミング違反を防ぎ，バスマスタへ Read もしくは Write の要求を伝搬する．

3.6　アプリケーションプロセッサインターフェース

センサハブは基本的に，AP から発行された命令に従い，センサからデータを取得し，センサフュージョンなどの処理を施してから AP へデータを送信する，といった動作を行う．

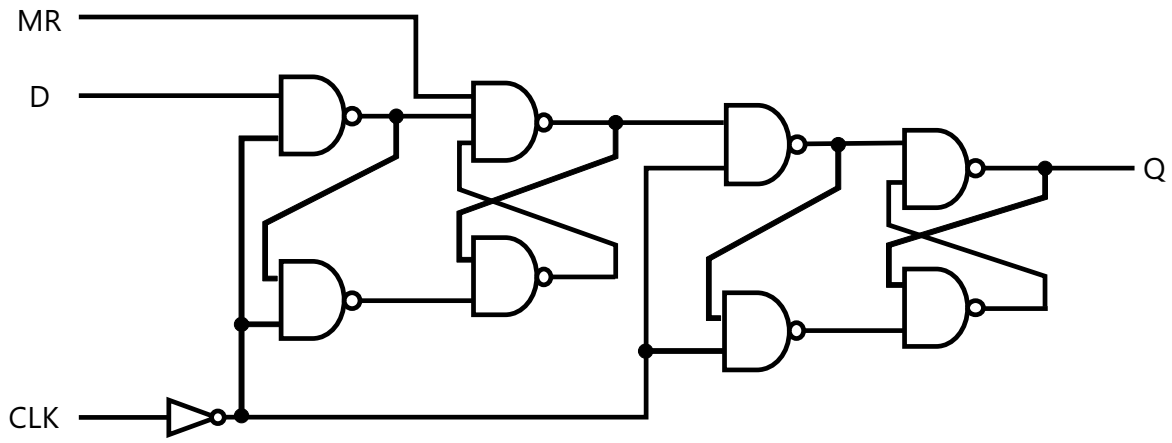

図 3.9　Gate DFF の構成図

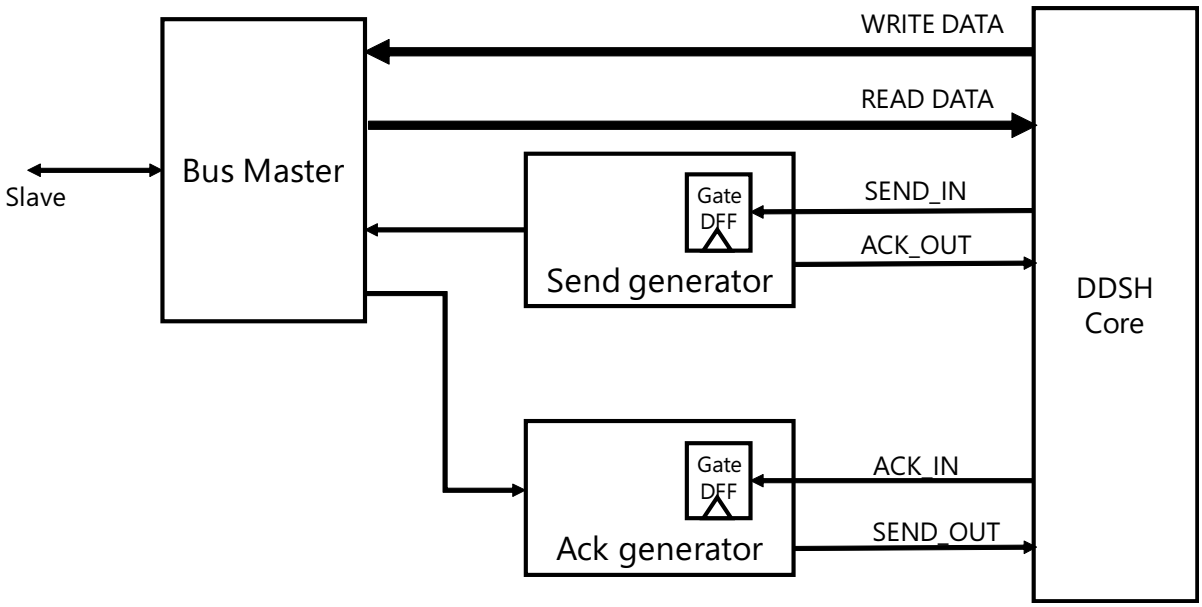

図 3.10　センサインターフェースの構成図

つまり，AP とセンサハブは，センサハブが実行すべき命令データやセンサハブが取得・加工したデータを I/F によって送受信する．よって，AP との I/F はこれらのデータを送受信可能でなければならない．

DDP の ISA には，外部から受け取った命令を各メモリに書き込むこと動作が想定されていないため，外部から入力されたパケットから書き込めるように，init_prog 命令を設計した．init_prog 命令は color が一定パターンの場合，各メモリに書き込みを実行する特殊な

フォーマットとなっている．

また，一般的な AP はバス I/F で外部機器と通信してるため，DDSH は AP とバスインターフェースによって通信する必要がある．例えば，ARM プロセッサのインターフェースは AMBA というバスであり，AMBA はスレーブデバイスのアドレスと各フラグによって通信する．AMBA を代表とした AP の I/F は図 3.11 のように構成される．STP はハンドシェイクによって通信するため，DFF を用いて擬似的にハンドシェイクを実現した．その際に発生するタイミング違反を防止するために，図 3.9 の Gate DFF を採用し，STP のハンドシェイクとバス I/F のプロトコルを相互変換するように構成した．

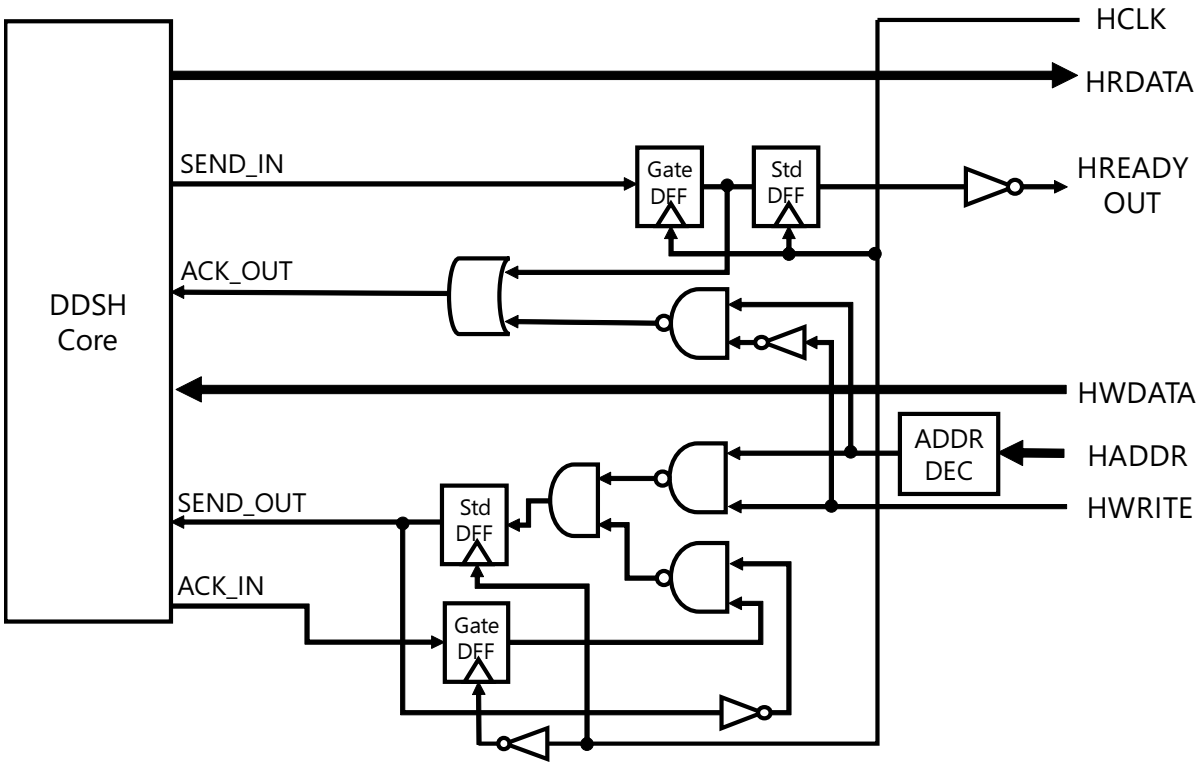

図 3.11　アプリケーションプロセッサインターフェースの構成

3.7　結言

本章では，DDP アーキテクチャを基に DDSH の構成について述べた．DDSH は DDSH コアとセンサ I/F，AP I/F で構成されている．DDSH コアはセンサフュージョンなどの処

理を可能にするため，パケットの識別情報を書き換えることを可能にした．さらに，複数 I/O デバイスとの接続を可能にするため，入力機構と出力機構を拡張ステージとして設計した．センサ I/F と AP I/F は，非同期回路の STP と同期回路を接続するために，タイミング違反の発生しにくい Gate DFF を採用することで，接続を可能にした．センサ I/F は I^2C や SPI などが存在するが，それらの通信手順は基本的に同じなので，バスマスタの構成を I^2C や SPI に適用すれば，通信可能となっている．

次章では，DDSH の面積と性能について評価した結果について述べる．

第 4 章

設計・評価

4.1 緒言

本章では，第 2，3 章で述べた DDSH の評価を行うために，65nm CMOS 標準セルライブラリを用いて DDSH 回路を設計，論理合成と配置配線を行った．設計回路の回路シミュレーションを行い，その結果と一般的なセンサハブの性能を比較する．その比較結果から，DDSH の I/O オーバヘッドが一般的なセンサハブより少ないことを確認する．

また，IoT システムで DDSH が有効であることを証明するため，2 章で述べた IoT システム上での要求条件から DDSH の評価条件を設定し，その目標性能を満たしているかを確認する．

4.2 IoT における DDSH の評価条件

DDSH を評価するに当たって，センサハブを含んだ典型的な IoT システムから評価条件を設定した．図 4.1 は一般的な IoT システムの概略図であり，基本的な要素はセンサとエッジデバイス（図内では例として Raspberry Pi），そしてクラウドサーバである．センサハブの性能として考慮すべき要素は，センサから出力されるデータ量，対象とする AP，そしてエッジデバイスからクラウドへ送信する通信規格である．以降，これらの評価条件を定義していく．

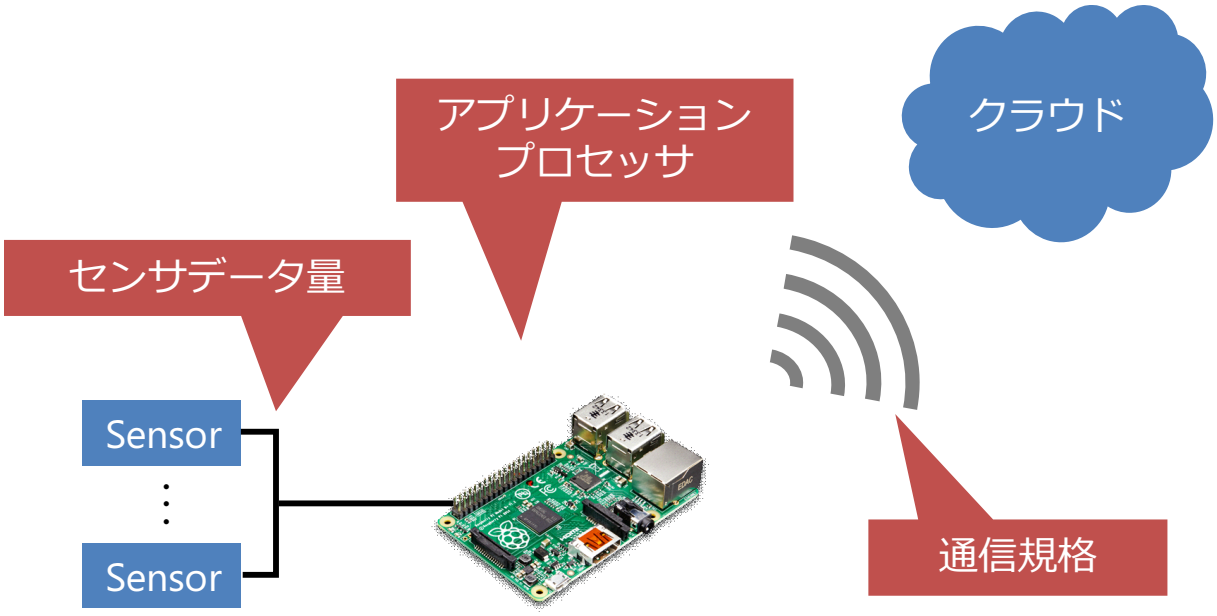

図 4.1　一般的な IoT システムの概略図

4.2.1　通信規格

近年，IoT 向け無線技術として Low Power Wide Area Networks （LPWAN）に関する研究が盛んに行われており [10]，現在 IoT 向けとされている無線技術を表 4.1 にまとめた．IoT システムでは高速通信を求められず，基本的に低周波帯が使用され，通信速度も 1Mbps 程度の規格が多い．そのため，センサから大量のデータが発生すると，ネットワークが輻輳してしまうため，エッジの SoC がデータを圧縮する必要がある．そのため，センサハブで大量のデータを 1Mbps に圧縮することができれば，AP がクラウドに直接データを送信する処理のみで済む．

4.2.2　センサから取得されるデータ量

センサハブは多数のセンサと接続されることを想定されており，少ない消費電力でそのデータを処理しなければならない．米 FLOWSERVE[11] では，ポンプシステムを制御するシステムとして大量のセンサを導入している．そのポンプシステムでは 2.5Mbps のデータが発生したと記されている．そのため，2.5Mbps 以上のセンサデータを処理することがで

表 4.1 IoT の無線ネットワーク技術

	規格	周波数	距離	速度
近距離無線	Wi-Fi	2.4/5GHz (ISM)	50m	150 ～ 200Mbps
	Bluetooth	2400Hz (ISM)	50m ～ 150m	1Mbps
	ZigBee	2400Hz (ISM)	10 ～ 100m	250Mbps
	Z-Wave	900MHz (ISM)	30m	9.6/40/100kbps
	ISA100	2400Hz (ISM)	10 ～ 100m	250kbps
	WirelessHART	2400Hz(ISM)	10 ～ 100m	250kbps
	Wi-SUN	900MHz (ISM)	500m	250kbps
	Thread	2400Hz (ISM)	N/A (Home)	250kbps
	NFC	13.56MHz	10cm	100 ～ 250kbps
LPWAN	NB-IoT (3GPP)	700 ～ 3500Hz	5/10m	～ 1Mbps
	SIGFOX	900MHz	30 ～ 50m	10 ～ 1000bps
	Neul (Weightless)	900(ISM),458(UK), 470 ～ 790(HS)Hz	10km	～ 100bps
	LoRaWAN	900, 2400Hz (ISM)	2 ～ 5km	0.3 ～ 50kbps
	Ingnu	2400Hz (ISM)	30km	60 ～ 3800bps

きれば，センサハブの性能として十分条件であると言える．

4.2.3 IoT 向きアプリケーションプロセッサ

IoT システムでは，消費電力の低いプロセッサが望まれる．そのため，IoT システムで採用されているプロセッサは ARM プロセッサが多く採用されている．ARM プロセッサには，消費電力を最小限に抑えた Cortex-M シリーズやリアルタイム向けの Cortex-R シリーズなどがある．その中で，IoT システムで多用される Raspberry Pi にも搭載されており，Linux

などの OS も動作させることの可能な Cortex-A シリーズを対象の AP に設定する．そのため，DDSH は ARM プロセッサの ARM Cortex-A シリーズのプロセッサと通信することを想定し，設計する．ARM アーキテクチャは AMBA バスを I/F としており，AMBA の AHB で通信していることが多いため，AHB バスと通信することを想定する．

4.3　65nm CMOS による設計

提案回路の有効性を示すために，DDSH 回路を 65nm CMOS 標準セルライブラリを用いて設計した．この提案回路は，Verilog-HDL を用いて設計し，Synopsys 社の Design Compiler により論理合成，IC Compiler により配置配線を行った．

4.3.1　回路仕様

DDP 及び，DDSH を 65nm CMOS 標準セルライブラリを用いて論理合成を行い，両回路を比較した．その際の設定条件は表 4.2 のように設定した．入力データは 16bit のデータを保持した 41bit のパケットとなっており，入力パケットに包含される各データの内訳は表 4.3 のように設定した．

センサハブは高性能化より低消費電力化の需要が高いため，64bit や 32bit のデータは必要なく，センサデータも 8bit や 16bit のデータサイズが多用されるため，DDSH のデータサイズは 16bit に設定した．また，各メモリも大容量のメモリは，高性能化を可能とするが，回路面積の増大やそれに伴う高消費電力化を招くため，各メモリサイズを表 4.2 のように設定した．

表 4.3 内の入力パケットの color は，DDSH 内で各外部デバイスの識別子を示す．表 4.2 に示すように，入力デバイスを 4 つ，出力デバイスを 4 つに設定して設計したため，color を 3bit に設定した．また，dest はノイマン型センサハブにおけるプログラムカウンタのような役割であり，プログラムの依存関係を示す際に使用する．DDSH では，演算結果の出力先を指定する際にも使用するため，color より大きいサイズであり，100 ノード以上のプロ

表 4.2　論理合成時の設定条件

Process	65nm CMOS
# stages of STP ring	8 stages
# I/O devices	4 inputs, 4 outputs
Constant memory (CST)	19bit × 128 word
Matching memory (MM)	37bit × 16 packets
Data memory (DMEM)	16bit × 512 word
Program storage (PS)	16bit × 128 word
I/O Program storage (I/O PS)	14bit × 8 word

表 4.3　入力パケットフォーマット

color	3 bit
gen	8 bit
dest	7 bit
LR	1 bit
CP	1 bit
opc	4 bit
C	1 bit
Z	1 bit
Data	16 bit

グラムを実行できるよう 7bit として設計した.

以上の条件を基に論理合成及び配置配線を行い，面積評価及び性能評価を経て DDSH の有用性を評価する.

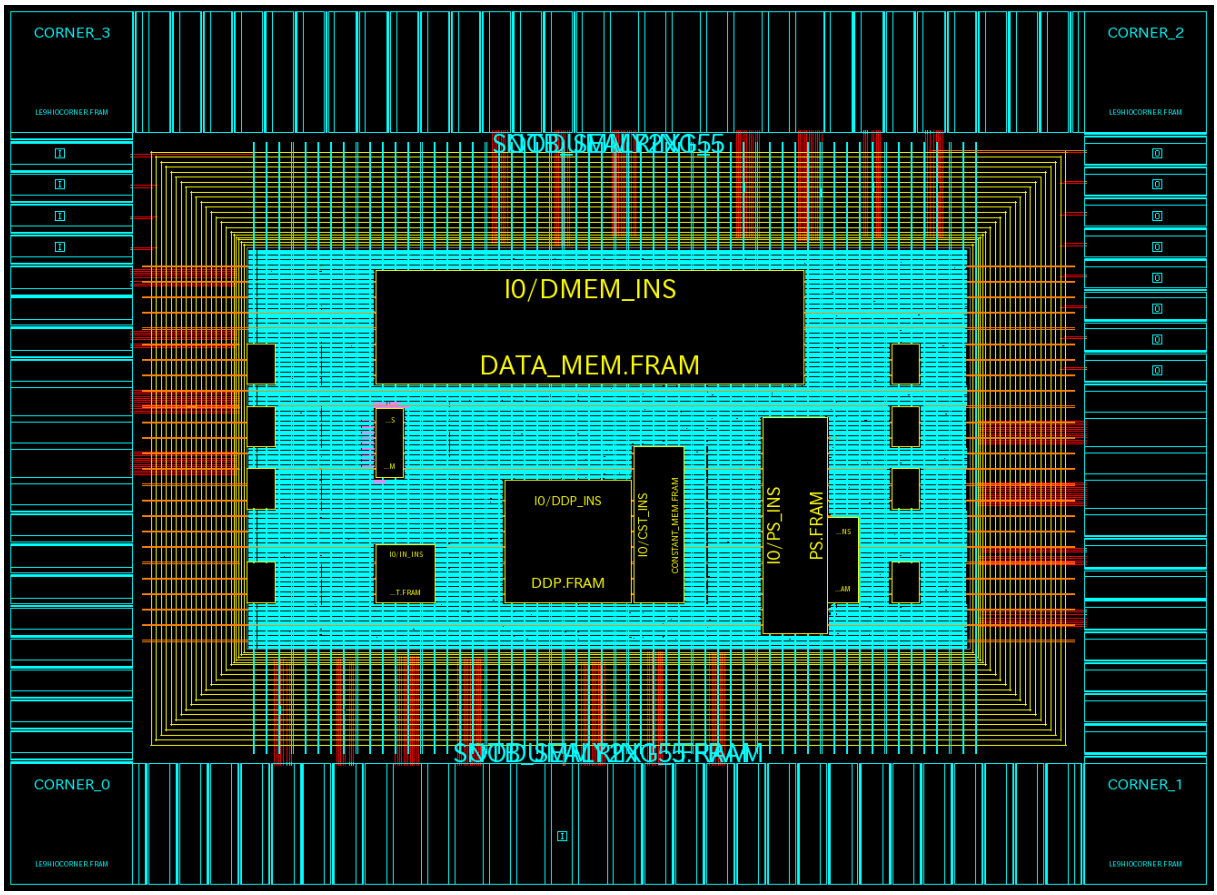

図 4.2　DDSH 回路のレイアウト（検証中）

4.3.2　Layout 設計

提案回路の配置配線を行った結果を図 4.2 に示す．

提案回路は，2mm × 1.5mm のダイにレイアウトしている．各マクロは，DDSH コアと入力部，出力部，そしてメモリに相当している．チップ上の大部分を占めているマクロは各メモリであり，DFF によって構成されている．DFF を採用した理由は，より面積の小さい SRAM は動作が遅く，提案回路の仕様を満たすものが提供されていないためである．しかし，提案回路の仕様上，DFF によってレイアウトしても 2mm × 1.5mm のダイに収まる大きさにすることができた．現在，配置配線後の DDSH 回路の LVS，DRC を検証中である．

4.4　基本性能及び面積評価

提案回路を 65nm CMOS 標準セルライブラリを用いて設計・論理合成を行った．その結果を表 4.4 に示す．ASIC を設計する場合，各マクロ内やマクロ間に電源や配線領域を考慮する必要があるため，総セル面積は，セル密度を 80%に換算した結果を示している．メモリを除いた提案回路の総論理セル数は，7.7k 個（面積：0.0359mm^2）となり，I/O のモジュールは全体の約 20%であった．DDSH の合計面積は，ARM Coretex-M0+を搭載した米 Atmel 社の SAM D20[12] の面積が 0.04mm^2 であることから，一般的なセンサハブと同等かそれ以下の面積である．

表 4.4　DDSH の回路コスト/面積

	core	I/O	total
standard cells	6.2k	1.5k	7.7k
area [mm^2]	0.0287	0.0072	0.0359

4.5　性能評価

DDSH の性能評価方法として，タスクの条件を設定する必要がある．条件として，最大処理性能近辺を計測可能であり，DDSH の有用性である，I/O オーバヘッドが顕著に確認することのできるタスクを設定する．また，評価条件として設定した，2.5Mbps のセンサデータを 1.0Mbps に圧縮する処理が可能であるかを評価する．

I/O オーバヘッドの影響を顕著に確認でき，最大処理性能近辺の処理負荷が発生するタスクとして，動画処理を評価タスクに設定した．CMOS カメラから一定のフレームレートで得られた画像に対して，バイリニア法で 1 フレームの画像を 1/4 に縮小する処理によって性能評価を行った．

4.5.1　比較評価

比較対象としては，割り込みによって通信開始し，DMA でデータを取得する ARM 社の Coretex-M0+を対象とした．動画のフレームレートを変化させ，処理限界の合計画素数の比較結果が表 4.5 のようになり，フレームレートが 30fps から 120fps の時，Cortex-M0+の約 50 倍の性能を達成することが確認できた．

表 4.5　画像縮小を伴う CMOS センサハブ機能の性能比較

フレームレート	DDSH	ARM Cortex-M0+
30fps	158,730	3,185
60fps	79,365	1,590
120fps	39,682	793

※評価条件：30MHz 動作，フレームバッファを想定

※評価指標：処理可能な最大フレームサイズ [画素/フレーム]

DDSH は，割り込み処理などの前処理を必要としないため，DDSH コアの環状パイプライン内を周回するパケットを詰ませずにパケットの転送を行うことができれば良いため，I/O オーバヘッドは性能に影響しない．しかし，Cortex-M0+は割り込み処理として，データの退避に 24 clock cycle 必要であり，データの復帰に 15 clock cycle 必要である．また，DMA 転送で 1 データ転送するあたり 6 clock cycle 必要である．そのため，これらの I/O の処理に処理リソースを大きく割くため，このような結果となった．センサハブは I/O 処理を多用するため，DDSH は I/O オーバヘッドが少ないため，センサハブとして有用であることが分かる．

実際の IoT システムのアプリケーションでは，画像サイズを細分化して変更することは一般的でないため，多面的な評価として，画像サイズを一定に設定した際の処理限界のフレームレートを比較した結果が表 4.6 である．画像処理は一般的には AP で実行することを想定しているため，一般的なセンサハブでは，QVGA の画像サイズでも 1.2fps の一般的にはありえない程低フレームレートの動画しか処理することができないことに対し，DDSH は

QVGA の画像であれば 62.0fps のフレームレートまで処理可能であることが分かった．

表 4.6　各画像サイズにおけるセンサハブ機能の性能比較

画像サイズ	DDSH	ARM Cortex-M0+
QVGA（320 * 240）	62.0 fps	1.2 fps
VGA（640 * 480）	15.5 fps	0.3 fps
HD（1280 * 720）	5.1 fps	0.1 fps

※評価条件：30MHz 動作，フレームバッファを想定

※評価指標：処理可能な最大フレームレート [fps]

4.5.2　最大処理性能評価

上記の性能評価は比較対象と公平な比較を行うため，DDSH の各ステージのパケット伝搬時間を 30MHz に設定して評価した．しかし，DDSH は各ステージごとに遅延時間を調整することが可能であるため，各ステージの遅延時間を計測し，最大性能を計測する必要がある．

表 4.7　論理合成後の DDSH の各ステージのパケット転送時間

ステージ名	パケット転送時間 (ns)
M	0.8
CST	7.4
MM	4.3
ALU	3.8
DMEM	6.5
PS	8.0
B	0.4

表 4.7 は，論理合成後の DDSH の各ステージのパケット転送時間である．最大性能を計

測する場合は，この遅延時間をパケット転送時間に設定し，計測する．この転送時間を基に表 4.6 の評価を計測した表が 4.8 である．QVGA 時の最大フレームレートを計算すると，1302fps の動画まで処理可能であることが分かった．さらに，Full HD の画像サイズでも 48fps の動画まで処理可能であるため，監視カメラなど高いフレームレートを必要としないシステムなら適用できることが分かった．

表 4.8　各画像サイズにおける DDSH の最大性能

画像サイズ	DDSH
QVGA（320 * 240）	1302.0 fps
VGA（640 * 480）	325.5 fps
HD（1280 * 720）	108.5 fps
Full HD（1920 * 1080）	48.2 fps
4K（3840 * 2160）	12.0 fps

※評価条件：30MHz 動作，フレームバッファを想定

※評価指標：処理可能な最大フレームレート [fps]

4.6　結言

本章では，DDSH の評価を行うなため，65nm CMOS 標準セルライブラリを用いて Verilog-HDL による回路設計を行った．IoT システムの要求条件として，2.5Mbps のデータを 1.0Mbps に圧縮することを十分条件とし，AP は IoT システムで多用される ARM プロセッサを想定した．面積評価の結果，DDSH は一般的なセンサハブとして妥当な面積であることが分かった．

DDSH の評価として最大処理性能近辺を確認でき，I/O オーバヘッドの影響が顕著に確認できるタスクとして，動画処理にて性能評価を行った結果，DDSH は一般的なセンサハブの約 50 倍の性能を達成することが確認できた．一般的なセンサハブでは処理することのできなかった画像処理も単純な処理であれば処理可能であることが分かったため，画像処理

の一次的な処理を DDSH が実行することで，より AP の負担を減らすことも可能である．

また，最大処理性能を計測すると，30fps で Full HD の動画も一次的な処理であれば処理することが可能であることが分かった．そのため，DDSH は一般的なセンサハブでは不可能であった画像処理の一部担うことが可能であり，AP の負担削減やシステム全体の消費電力削減に貢献できることが分かった．

本研究では，65nm CMOS のプロセスにて性能計測を実施したが，今後，微細化が進むにつれ，20nm CMOS などのプロセスで DDSH を設計することができれば，さらなる高速化が見込める．

第 5 章

結論

情報通信技術の発展に伴い，IoT としてインターネットに接続するデバイスが年々増加の一途を辿っており，2020 年には約 300 億個に達すると予測されている．その結果，クラウドの処理負担の増加が問題となっている．その対抗策として負荷を分散させるエッジコンピューティングが注目されており，エッジ側で一次的な処理を施し，データ量を減らすことで，クラウドの負担や通信トラフィックを減らすための研究も盛んに行われている．

一般的なエッジの基本的な構成として，複数のセンサデータを統合するセンサハブ，アプリケーション（AP），通信モジュールで構成されている．AP は高性能が求められる一方，消費電力が高いものが多い．そのため，AP がセンサデータを受信するため常時稼働すると，システム全体の消費電力が増加してしまう．そこで，センサハブを搭載することでセンサとの通信処理をセンサハブに担わせることで，AP がセンサデータを常時受信する必要がなくなり，高性能かつ高消費電力な AP の稼働時間を低減でき，システム全体の消費電力を抑えることができる．

しかし，一般的なセンサハブはノイマン型プロセッサであり，多数のセンサデータを取得する際，入力のタイミングによっては割り込み処理を実行する必要がある．そのため，データの退避や復帰が頻繁に繰り返し必要となり，その分のスループットの低下や消費電力の増加などの問題が生じる．

本研究では，割り込み処理等の I/O オーバヘッドを削減するため，割り込み処理を必要とせず動作可能なセルフタイム型パイプライン（STP）によるデータ駆動型プロセッサ（DDP）に注目し，DDP を採用したデータ駆動型センサハブ（DDSH）のアーキテクチャを提案し，DDSH が IoT 向け SoC として有効であることを明らかにした．具体的には，DDP

と外部機器のデータフォーマットを変換するための入力機構と出力機構を増設し，従来のDDP の ISA では実現できなかったセンサフュージョンを実現するための命令を設計した．

また，DDSH は AP や多種多様なセンサと接続させることを目的としているため，それらの I/F についても考えなければならない．センサは I^2C や SPI のシリアルバスが多用されており，AP の I/F は基本的にバス I/F である．そのため，本研究では DDSH に STP とバス I/F を変換するための回路を設計した．STP は非同期回路であるため，同期回路であるバス I/F と相性が悪く，タイミング違反を起こす可能性が高いため，タイミング違反を起こしにくい遅延の多い DFF を採用した．

本稿では，IoT で多用されるセンサハブについて，データの受信をトリガに動作可能な DDP を活用したセンサハブであるデータ駆動型センサハブ（DDSH）の構成法を述べた．また，DDSH の LSI 設計を通した評価，ならびに，一般的なセンサハブと比較した性能評価結果を述べた．

本稿において，第 2 章では IoT 向け SoC の要件を定義した．IoT ではエッジコンピューティングが盛んに際ようされており，エッジ部に搭載される SoC は低消費電力と高性能を両立させることが求められている．そのため，センサハブを搭載した SoC を採用するシステムが増加している．一般的なセンサハブは，I/O 処理として割り込み処理でデータの入出力を実現しており，その前処理に伴う性能低下や消費電力の増加を引き起こすという問題があった．そこで，割り込み処理なしに動作可能な DDP をセンサハブへと拡張した DDSH の構想について議論した．

第 3 章では，第 2 章で議論した DDSH の構想から DDSH の設計について述べた．DDSH はセンサフュージョンなどを処理するため，複数 I/O デバイスとのインターフェースを搭載しており，センサフュージョンなどの処理を実行可能にした ISA に基づき動作する．その ISA を実行可能なマイクロアーキテクチャは，DDSH コアとセンサや AP とのインターフェースで構成されている．DDSH コアは，従来の DDP で実現不可能であった複数入出力を可能としており，センサフュージョンを実行するため，複数デバイスのデータを組み合わせて処理することが可能となっている．センサのインターフェースは，DDSH コアと直接接

続可能な GPIO などの非同期センサインターフェースと，直接接続することのできない同期センサインターフェースが存在する．本研究では，同期インターフェースと非同期回路である STP を直接接続させた際に発生するタイミング違反を防ぐことのできる，調停機能を備えた DFF を採用することで，接続を可能にした．AP のインターフェースは，バスインターフェースが多用されているため，同期センサインターフェースと同様に，調停機能付き DFF によって接続を可能にした．

第 4 章では，第 3 章で設計した DDSH の面積及び性能の評価について述べた．典型的な IoT システムから評価条件としてセンサから得られるデータ量，IoT の通信規格，IoT 向き AP の 3 つを設定した．その結果，DDSH の性能目標として 2.5Mbps のデータを 1.0Mbps に圧縮可能であれば十分条件と言えることが分かった．その評価条件を基に 65nm CMOS 標準セルライブラリを用いて設計した結果，総面積が $0.0359mm^2$ であったため，DDSH は一般的なセンサハブとして妥当な面積であることが分かった．また，性能評価として，最大処理性能近辺を確認でき，I/O オーバヘッドの影響が顕著に確認できるタスクとして，動画処理にて性能評価を行った．その結果，一般的なセンサハブの約 50 倍の性能を達成することができた．

以下に本研究においての今後の課題を示す．

- AP I/F の機能拡張

  本研究で設計した AP I/F は基本的なバス I/F を変換する機能のみとなっている．今後はバースト転送などの高速なデータ転送を可能とした通信にも対応するための機能拡張をすることが求められる．

- DDSH のレイアウトの高速化

  今回，DDSH の性能評価は，論理合成の結果を元に行った．そのため，配線遅延を含めた性能評価を行っていない．よって，65nm CMOS 標準セルライブラリで論理合成した結果を配置配線し，より詳細な厳密に性能評価を行う．

- 電力に関する評価

DDSH は IoT を想定して設計しているため，面積や性能以外にも消費電力の評価を行う必要がある．よって，SPICE シミュレーションで電力評価を行い，一般的なセンサハブに比べて電力の面でも DDSH の有効性を示す．

今後，以上の課題を解決することで，更なる性能向上を図ることができると考えられる．

提案方式の今後の展望として，消費電力を抑えるために DVS（Dynamic Voltage Scaling）[13] と STP を組み合わせて実現することが考えられる．STP は DVS によってパイプラインステージ間の遅延量が変化しても柔軟に動作することができるため，クロック信号を用いた同期回路と比較して，誤動作を起こす恐れがすくない．そのため，STP に DVS を組み合わせると，少ないリスクで消費電力を下げることが可能である．

謝辞

本研究に際して，様々なご指導を頂きました岩田誠教授に深く感謝いたします．若輩者である私を指導教員として厳しくも優しい指導していただいたおかげで本論文を完成させることができました．学部 1 年次から修士 2 年次までの間，長いようで短い期間でしたが，勉学以外の事でも大変お世話になりありがとうございました．ここに感謝の意を表します．

ご多忙な中，本研究の副査をお引き受け下さり，梗概および様々な疑問点や改善点等を指摘していただいた横山和俊教授，栗原徹准教授に心より感謝致します．

また上記の主査，副査の先生をはじめとした高知工科大学の教職員の皆様方，特に情報システム工学コースの教職員の皆様方には研究，勉学だけでなくより充実した学生生活を送ることができました，厚く感謝のほどを申し上げます．

研究室に配属されてから今まで，時間を惜しむことなく，回路設計のノウハウ，研究に関して相談に乗っていただいた大学院生の宇野則文氏，岡宗祥平氏に心より感謝いたします．研究に行き詰った時には，いつも親身になって協力して頂いたおかげで，この論文を完成させることができました．これからも，先輩たちを目標に，日々を全力で精進していきます．

日頃からコンピュータ構成学研究室同輩として全力で切磋琢磨してくれた，梅嵜佑樹氏，小川友暉氏に心より感謝します．これからもよき仲間として共に頑張りましょう．

コンピュータ構成学研究室の後輩として，日頃からご支援，御協力を頂いた，修士 1 年の齋藤あかね氏，田原匡浩氏，福田和馬氏，学部 4 年の穴井志穂氏，奥井里永子氏，山崎尚之氏，吉本大輔氏，和田悠伸氏，学部 3 年の楠田健太氏，汐見興明氏，下出千晴氏，長野寛司氏，柳田海志氏に心より感謝します．これからも技術者として，高い志をもって日々を大切に頑張ってください．

最後になりましたが，日頃からご支援頂いた関係者の皆様に心より御礼を申し上げます．

参考文献

[1] Li Da Xu, Wu He, and Shanchang Li. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, Vol. 10, No. 4, pp. 2233–2243, February 2014.

[2] Hiroki Terada, Souichi Miyata, and Makoto Iwata. DDMP's: Self-Timed Super-Pipelined Data-Driven Multimedia Processors. *Proceedings of the IEEE*, Vol. 87, No. 2, pp. 282–296, November 1999.

[3] Hiroki Shibuta and Makoto Iwata. Self-Timed I/O Architecture of Data-Driven Sensor Hub. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 323–328, July 2016.

[4] 総務省｜平成 29 年版 情報通信白書｜爆発的に増加する IoT デバイス. http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h29/html/nc133100.html, 2017.

[5] Special Issue on Advancing the Internet of Things. *IEEE Computing edge*, Vol. 2, No. 4, pp. 9–45, April 2016.

[6] Perera Charith, Jayraman Prem, and Zaslavsky Arkady. Dynamic Configuration of Sensors Using Mobile Sensor Hub in Internet of Things Paradigm. *IEEE ISSNIP*, pp. 473–478, April 2013.

[7] D. L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, Vol. 85, No. 1, pp. 6–23, Jan 1997.

[8] RealView Compilation Tools デベロッパガイド - 6.2.7 割り込みハンドラ. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0203ij/Cacbhjjj.html, 2018 年 1 月 26 日参照.

[9] Kazuhiro Komatsu, Shuji Sannomiya, and Makoto Iwata. Interaction Self-Timed

Pipelines and Elementary Couping Control Modules. *Transactions of IEICE*, Vol. E92-A, No. 7, pp. 1642–1651, July 2009.

[10] Raza Usaman, Kulkarni Parag, and Sooriyabandara Mahesh. Low power wide area networks: An overview. *IEEE Communications Surveys and Tutorials*, Vol. 19, pp. 855–873, January Secondquarter 2017.

[11] FLOWSERVE Corporation. https://www.flowserve.com/en, 2018 年 1 月 26 日参照.

[12] SAM D ARM Cortex M0 MCUs - Atmel. http://www.atmel.com/products/microcontrollers/arm/sam-d.aspx, 2018 年 1 月 24 日参照.

[13] Kei Miyagi, Shuji Sannnomiya, Makoto Iwata, and Hiroaki Nishikawa. Low-Powered Self-Timed Pipeline with Variable-Grain Power Gating and Suspend-Free Voltage Scaling. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, pp. 618–624, July 2013.

付録 A

DDSH の ISA

DDSH の ISA は演算命令，シフト命令，条件分岐命令，メモリアクセス命令などが存在する．オペレーションコード（opc）が 8bit の場合の ISA 一覧が表 A.1，A.2，A.3，A.4，A.5，A.6，A.7，A.8 となっている．

表 A.1　パケット間演算命令

命令	opc	Left opc	Right opc	動作
add	0000 000X	0000	000X	2 つのオペランドの加算
addc	0000 001X	0000	001X	C フラグを考慮した 2 つのオペランドの加算
sub	0000 010X	0000	010X	2 つのオペランドの減算
subc	0000 011X	0000	011X	C フラグを考慮した 2 つのオペランドの減算
and	0000 100X	0000	100X	2 つのオペランドの論理
or	0000 101X	0000	101X	2 つのオペランドの論理
xor	0000 110X	0000	110X	2 つのオペランドの排他的論理和
mul	0000 111X	0000	111X	2 つのオペランドの乗算

表 A.1 はパケット同士のオペランドに対して演算を実行する命令の一覧である．DDSH は基本的にパケットが opc の半分を保持しており，組となるパケットがもう片方の opc を保持している．その 2 つのパケットが保持する opc を組み合わせることで，演算が実行可能となる．

表 A.2　パケット定数間演算命令

命令	opc	Left opc	Right opc	動作
add	0001 000X	0001	000X	2 つのオペランドの加算
addc	0001 001X	0001	001X	C フラグを考慮した 2 つのオペランドの加算
sub	0001 010X	0001	010X	2 つのオペランドの減算
subc	0001 011X	0001	011X	C フラグを考慮した 2 つのオペランドの減算
and	0001 100X	0001	100X	2 つのオペランドの論理積
or	0001 101X	0001	101X	2 つのオペランドの論理和
xor	0001 110X	0001	110X	2 つのオペランドの排他的論理和
mul	0001 111X	0001	111X	2 つのオペランドの乗算

表 A.2 はパケットが保持するオペランドと定数の演算を実行する命令の一覧である．これらの命令は，パケットに保持されるオペランドと CST ステージのメモリに格納される定数との演算を行う．

表 A.3　シフト演算命令

命令	opc	Left opc	Right opc	動作
shl	0010 000X	0010	000X	右パケットの値だけ左に論理シフト
shr	0010 001X	0010	001X	右パケットの値だけ右に論理シフト
rol	0010 010X	0010	010X	右パケットの値だけ左にローテート
ror	0010 011X	0010	011X	右パケットの値だけ右にローテート

表 A.3 はシフト演算命令の一覧である．シフト演算は左パケット（LR フラグが L を意味するパケット）が保持するオペランドの値を右パケット（LR フラグが R を意味するパケット）の値だけシフトまたはローテートする．

表 A.4 は条件分岐命令の一覧である．左パケットの宛先（dest）を右パケットが保持して

表 A.4　条件分岐命令

命令	opc	Left opc	Right opc	動作
bz	0011 000X	0011	000X	Z=0 : dest = dest, Z=1 : dest = dest + 1
bnz	0011 001X	0011	001X	Z=0 : dest = dest + 1, Z=1 : dest = dest
bc	0011 010X	0011	010X	C=0 : dest = dest, C=1 : dest = dest + 1
bnc	0011 011X	0011	011X	C=0 : dest = dest + 1, C=1 : dest = dest

いる C フラグまたは Z フラグで制御する命令である．分岐条件が成立した場合，左パケットの宛先は本来の宛先に 1 加算した値となり，条件が成立しない場合は左パケットの宛先は変更されない．この宛先は次の命令となるので，分岐させる場合は PS メモリに隣接したアドレスに格納することで分岐させることができる．

表 A.5　メモリアクセス命令

命令	opc	Left opc	Right opc	動作
ldm	0100 000X	0100	000X	メモリ（DMEM）からデータをロード
stm	0100 001X	0100	001X	データをメモリ（DMEM）にストア

表 A.5 はメモリアクセス命令の一覧である．ldm 命令は，2 つのパケットが保持しているオペランドの値の和をアドレスとしてデータメモリにアクセスし，データをロードする命令である．stm 命令は，右パケットが保持しているデータを直接アドレスとしてデータメモリにアクセスし，左パケットが保持するオペランドの値をデータメモリにストアする命令である．

表 A.6　absorb 命令

命令	opc	Left opc	Right opc	動作
absorb	1111 XXXX	1111	XXXX	パケットを消去

表 A.6 の absorb 命令はパケットを消去する命令である．PS メモリから absorb 命令をフェッチした場合，PS ステージに備わっているパケット消去機能にてパケットを消去する．プログラム上で不要になったパケットを削除するための命令であり，パケットを消去しないと DDSH 内に不要なパケットが巡回し，処理リソースの無駄を引き起こすため，absorb 命令にて消去する．

表 A.7　out 命令

命令	opc	Left opc	Right opc	動作
out	XXXX XXX1	XXXX	XXX1	パケットを DDSH 外部に出力する

表 A.7 の out 命令はパケットを DDSH 外部に出力する命令である．out 命令は他の命令と併用され，opc の LSB によって演算後のパケットを外部へ出力するかを B ステージにて判断する．その際，出力されるパケットの dest が外部デバイスのアドレスとなるため，OUT ステージで dest の値を基に出力先へ分岐させ，出力する．

表 A.8　識別子変更命令

命令	opc	Left opc	Right opc	動作
chgdest	1000 000X	1000	000X	dest を DMEM からロードした値に変更
chgcol	1000 001X	1000	001X	color を右オペランドの値に変更
addgen	1000 010X	1000	010X	gen と右オペランドを符号付き加算

表 A.8 は識別子を変更する命令の一覧である．chgdest 命令は，外部デバイスへの出力先として dest を変更する場合に使用し，データメモリからロードした値を dest に上書きする．chgcol 命令は，種類の異なるデバイス同士の演算を行う場合に用いる命令である．color を右オペランドの値に変更することで，color の異なるパケット同士でも演算が可能となる．addgen 命令は，gen が異なるパケット同士の演算を行う場合に用いる命令であり，gen と右オペランドを符号付き加算を実行する．