令和 4 年度

修士学位論文

リアルタイム処理可能な
データ駆動型センサノードの構成法の検討

A Study on Data-Driven Sensor Nodes
Capable of Real-Time Processing

1255100　　岡野 秀平

指導教員　　岩田　誠

2023 年 2 月 3 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報学コース

要 旨

リアルタイム処理可能な
データ駆動型センサノードの構成法の検討

岡野 秀平

近年、Internet of Things (IoT) 端末は様々な分野で活用され，それに伴い，高性能化が求められている．特に，スマートシティやスマート工場，自動運転技術に用いられるセンサノードには，安全な動作の保証のためのリアルタイム性の確保が要求される。このためには，周期的/非周期的タスクの時間制約 (デッドライン時刻) を保証できるように，タスクの優先度に基づいた動的スケジューリング方式が必要になる．中でも、Earliest Dedline First (EDF) は最も簡易な方式であり，先行研究では，複数タスクの多重処理が可能なデータ駆動型プロセッサ Data-Driven Processor (DDP) にハードウェア EDF スケジューラを搭載する基礎的検討がなされている [1]．しかし，この構成では，単項演算や定数演算のみからなるプログラムしか処理できないという課題があった．

本研究では，EDF スケジューラを搭載しセンサデータ処理が可能な DDP（以下 DDSH：Data-Driven Sensor Hub と呼ぶ）を Digilent 社 ZyboZ7-10 ボードに組み込んだ，リアルタイムデータ駆動型センサノードの構築を目指す．そのため，プログラムをマクロモジュール化し観測するパケットを限定した上で，従来のハードウェア EDF スケジューラの微修正でより汎用的なプログラムをリアルタイム処理可能な方式を提案する．

提案方式を導入した DDSH を ZyboZ7-10 の FPGA 上に実装し，EDF-DDP[2] と回路規模を比較した結果，レジスタ数が約 0.01%増加した程度であった．また，典型的なプログラムタスク群を対象に動作検証し，確実に動作することを確認した．

キーワード　Earliest Deadline First，リアルタイムデータ駆動型センサノード，FPGA

Abstract

A Study on Data-Driven Sensor Nodes Capable of Real-Time Processing

Shuhei Okano

In recent years, the Internet of Things (IoT) devices have been widely used so that they are required to support higher performance. Especially, the sensor nodes used in smart cities, smart factories, and automated driving technology must be capable of real-time processing to guarantee safe operation. To realize real-time processing, it is necessary to observe the temporal status of each task execution with its time constraints, and each task execution must be dynamically scheduled based on task priority. Earliest Deadline First (EDF) is one of the dynamic scheduling scheme based on task priority. Fukuda, et al. [1] proposed to employ EDF scheduling circuit in the data-driven processor (DDP) capable of multiprocessing tasks. However, this circuit deals with only the program task including unary-operations and constant-operations.

This study aims to realize a real-time data-driven sensor node by integrating a DDP and a EDF scheduler, called Data-Driven Sensor Hub (DDSH), on Digilent's ZyboZ7-10. Therefore, we proposed to make the programs into macro-modules and limited the number of packets to be observed on the EDF-scheduler circuit. As a result, the previous scheduler circuit can deal with the program task including binary-operations and branch. The proposed DDSH was implemented on the FPGA-part of ZyboZ7-10. The utilization of FPGA resource was slightly increased 0.01% compared to the EDF-DDP [2]. After that, we confirmed that the implemented DDSH can manipulated

typical task group in real-time according to each time constraints.

key words Earliest Deadline First, Real-Time Data-Driven Sensor Node, FPGA

目次

図目次

図目次

表目次

第 1 章

序論

近年，IoT (Internet of Things) の普及により，様々な分野で IoT デバイスが利用されている．その中でも，スマート家電や IoT 化された電子機器が増加するコンシューマ，スマート工場やスマートシティが拡大する産業用途，コネクテッドカーの普及により IoT 化の進展が見込まれている自動車の分野での IoT デバイス数の高成長が予想されている [3]．これらの分野のシステムでは，リアルタイム性を要求することが多く，複数のタスクを制御するためのスケジューリングが必要である．リアルタイムシステムのスケジューリングでは，制御対象の時間制約 (デッドライン時刻) を守る必要があり，そのためには，タスクに優先度を設定し，優先度に基づいた制御とエラーを起こさず継続的に動作する信頼性が必要である [4].

リアルタイムスケジューリングの例を図 1.1 に示す．TaskB の時間制約を相対デッドライン時間と呼び，実行要求時刻から相対デッドライン時間を足し合わせた値を絶対デッドライン時刻と呼ぶ．図 1.1 では，TaskB の実行要求時点でまだ TaskA が実行しているため，TaskB は TaskA の実行が完了するまで実行を停止する．TaskA の実行が完了すると，TaskB の実行を開始する．このとき，TaskB の実行開始から実行終了までの時間を実行時間と呼び，絶対デッドライン時刻から実行時間を差し引いた値を余裕時間と呼ぶ.

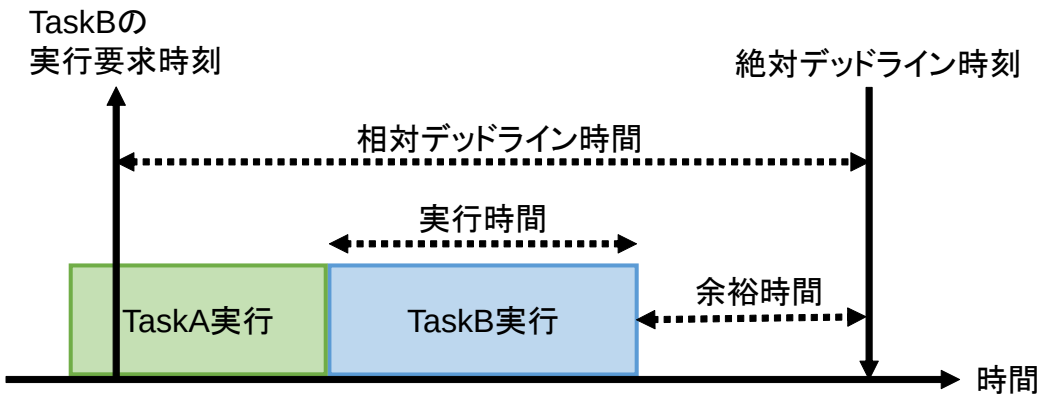


図 1.1　リアルタイムスケジューリングの動作例

リアルタイムスケジューリングのアルゴリズムとして，ラウンドロビン方式，静的優先度方式，動的優先度方式が挙げられる．

ラウンドロビン方式は，実行する時間単位 (タイムスライス) を決め，順番に実行するアルゴリズムである．各タスクの実行時間を均等に割り当てることができる．リアルタイムシステムでは，タスクの優先度に基づいたスケジューリングを行う必要があるため，リアルタイムシステムの要件を満たすには，優先度の高いタスクを先行して実行するアルゴリズムである優先ベースのプリエンティブ方式と組み合わせる必要がある [5]．また，時間単位を短くすると，より多くのタスクを均一に実行することができるが，頻繁なタスク切り替えによってオーバーヘッドが大きくなる問題がある．

静的優先度方式は，代表的なアルゴリズムとして，Rate Monotonic (RM) が挙げられる．RM は実行周期の短いタスクから順に高い優先度を割り当てるアルゴリズムである [6]．リアルタイムシステムでは，非周期的にタスクを実行することもあり，周期的なタスクでしかスケジューリングできない RM はリアルタイムシステムの要件を満たすことができない．

動的優先度方式は，代表的なアルゴリズムとして，Earliest Deadline First (EDF) と Least Slack Time (LST) が挙げられる．EDF は絶対デッドライン時刻が近いタスクに高い優先度を割り当てるアルゴリズムである [6]．LST は，余裕時間が短いタスクに高い

優先度を割り当てるアルゴリズムである [7]．EDF と LST は，シングルコアにおいて，リアルタイムシステムで最適なスケジューリングである．その一方で，マルチコアでは，EDF は絶対デッドライン時刻に基づいてスケジューリングを行うため実行時間を考慮せず，デッドラインミスを予想できないという問題がある．LST は余裕時間に基づいてスケジューリングを行うため実行時間を考慮しており，デッドラインミスを予想することができる．しかし，余裕時間の近いタスクが複数存在する場合，それぞれのタスクの優先度が短い期間で変化するためスラッシングが発生し，スケジューリングオーバーヘッドが大きくなる可能性がある [8]．

オーバーヘッドが大きくなる問題を解決するために，スケジューリングを搭載するプロセッサに着目する．ノイマン型プロセッサは逐次処理であるため，タスクの切り替え時にコンテキストスイッチが発生する [5]．コンテキストスイッチの発生時間は短時間であるものの，頻繁なタスク切り替えが起こるリアルタイムシステムでは，大きなオーバーヘッドとなる可能性がある．

タスクを切り替えることなく実行する方法として，複数タスクを多重に実行することが考えられる．多重処理能力を有する計算モデルとして，データ駆動が挙げられる．データ駆動では，各のノードの演算は，その演算に必要なデータが揃うことで実行するため，データ間に依存関係がないノードの演算を多重に実行することができる．そのため，データ駆動の動作をハードウェアに実装したデータ駆動型プロセッサ (Data-Driven Processor：DDP) は多重処理が可能であり，DDP の多重処理能力に余力がある場合はコンテキストスイッチによるオーバーヘッドが発生することなく複数タスクの実行が可能である [9]．

先行研究では，DDP と動的優先度方式に着目し，シングルコア DDP に動的優先度方式のスケジューリング回路を搭載する方法が提案されている [1][10]．

先行研究の提案により，ハードウェアによるリアルタイムスケジューリングが実現可能であるという見通しが得られている．また，先行研究のスケジューリング回路を搭載した DDP の回路規模が大きく，EDF スケジューリング回路を搭載した DDP は DDP

と比べて約 6 倍，LST スケジューリング回路を搭載した DPP は約 7 倍にまで増大するといった課題も，EDF スケジューリング回路の回路規模の課題については，最適化を検討する研究 [2] によって改善の見通しが得られている．

しかし，上記の研究のスケジューリング回路はどれも単項演算か定数演算のみで構成されたプログラムのタスクを対象としている．リアルタイムシステムでスケジューリング回路を搭載した DDP を実用的に利用するためには，二項演算や条件分岐を含む汎用性のあるプログラムのタスクを対象として処理できる方法を見出す必要がある．

以上のことから，本研究では，汎用的なプログラムのタスクでもリアルタイム処理可能な方式を提案する．センシングを用いたシステムを対象として，EDF スケジューリング回路を搭載しセンサデータ処理が可能な DDP (以降 DDSH：Data-Driven Sensor Hub と呼ぶ) を Digilent 社の ZyboZ7-10 に組み込んだ，リアルタイムデータ駆動型センサノードの構築を目指す．

本論文においては，第 2 章ではリアルタイムデータ駆動型センサノードの構成について述べ，汎用的なプログラムのタスクの処理を可能にする方式の方針について述べる．

第 3 章では，第 2 章で述べた要件から，DDSH におけるリアルタイム処理方式についてとその方式をアルゴリズムとして定量化したことについて述べる．

第 4 章では，提案方式を取り入れた DDSH を ZyboZ7-10 の PS 上に実装し，デッドライン時刻が異なる典型的なタスクを複数実行し，スケジューリング可能かを確認する．また，EDF-DDP[2] との回路面積の評価を行う．

第 5 章では，本研究で提案したスケジューリング回路の最適化についてまとめ，今後の課題を述べ，本論文を総括する．

第 2 章

リアルタイムデータ駆動型センサノードの構成

2.1 緒言

データ駆動処理方式は，データ依存関係がない演算を同時並列に実行できるという計算原理を基礎としている．このことから，DDP はリアルタイム性が要求される IoT システムに有用である．しかし，DDP の多重処理能力には限界があり，限界を超える演算数の実行が要求された際に，スケジューリング制御が必要となる．

本章では，まず，データ駆動の動作と DDP のアーキテクチャについて述べ，動的スケジューリング方式である EDF を搭載した DDP のスケジューリング制御について述べる．

その後，リアルタイム性が要求される IoT システムとして，センシングを行い，データ分析するシステムを対象とした，スケジューリング制御可能な DDP をセンサデータ処理のための DDSH として活用する方針について述べる．

2.2 データ駆動処理方式

データ駆動はデータを基に演算を行う計算モデルであり，データに依存関係がない演算の多重処理が可能である．図 2.1 のように，データ駆動の処理において，プログラム中の各ノードは，必要なデータが揃うことで実行可能状態になる．実行可能状態になる

と，ノードは発火して演算を実行し，演算結果を次のノードへ転送する．このように，データ駆動では各ノード間でデータの受け渡しが行われるため，演算の実行順序はデータの受け渡しに依存する．データの受け渡しの依存関係はデータフローグラフで表すことができ，$(A+B)-(C+D)$ の演算結果を出力するプログラムを実行したときのデータフローグラフを図 2.2 に示す．この場合，$E-F$ は $A+B$，$C+D$ と依存関係にあるため，$A+B$，$C+D$ よりも先に演算を実行することはできない．$A+B$ と $C+D$ については，データに依存関係がないため，多重に実行することが可能である．

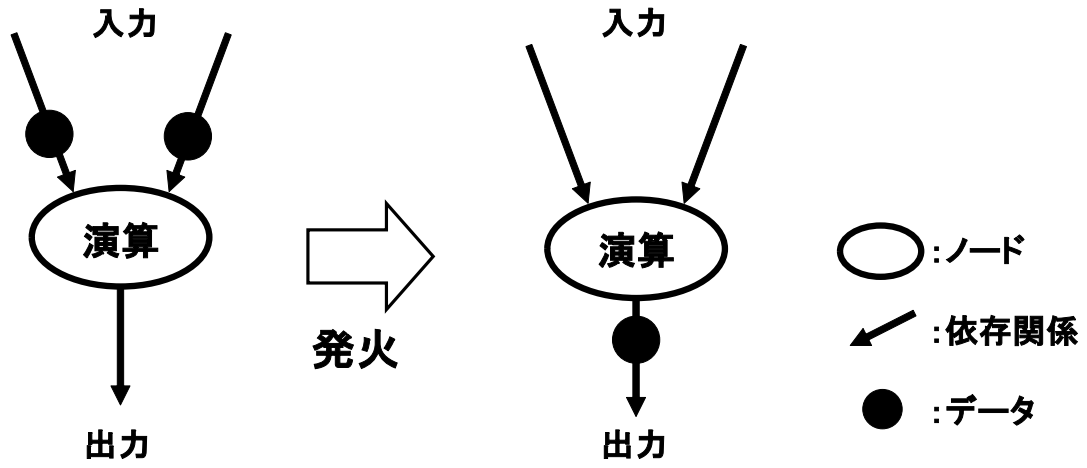


図 2.1　データ駆動の動作原理

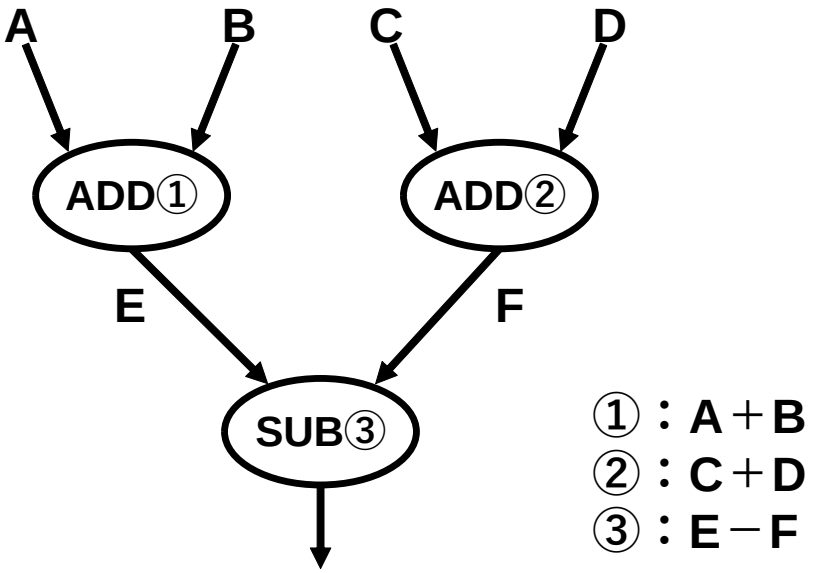


図 2.2　データフローグラフの例

2.3　データ駆動型プロセッサ (DDP) アーキテクチャとそのセルフタイム型パイプライン (STP) 実現

データ駆動型プロセッサ (DDP) はプログラムをパケット単位で受け取り，データ駆動の動作で処理を行う．先行研究 [2] で扱う DDP のアーキテクチャを図 2.3 に示し，DDP が受け取るパケットのフォーマットを図 2.4，表 2.1 に示す．DDP は 8 ステージで構成されており，受け取るパケットの長さは 42bit である．その内，26bit はヘッダ長であり，16bit がデータ長である．DDP の各ステージの詳細を以下に示す．

- パケット合流機構 (Merge Unit：M)

  外部から入力されるパケットとパイプライン内を周回する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 (Constant Memory：CST)

  定数命令を実行する際に用いる定数が定数メモリに格納されている．定数命令を実行するパケットである場合は，パケットが保持している宛先情報をアドレスとして，

演算に必要な定数を定数メモリから読み出し，パケットに付与する．定数命令を実行するパケットでない場合は，そのまま MM へ出力する．

- パケット待ち合わせ機構 (Matching Memory：MM)

  CST から受け取ったパケットが待ち合わせを行うパケットである場合，対応するパケットと待ち合わせをするために一時的に内部メモリへと格納する．対応するパケットが到着すると，2 つのパケットの情報を結合して ALU へ出力する．待ち合わせを行わないパケットの場合は，そのまま ALU へ出力する．

- 演算機構 (Arithmetic Logic Unit：ALU)

  パケットが保持している命令コードを基にして，算術演算または論理演算を実行する．ロード命令やストア命令の場合は，DMEM でデータメモリにアクセスする際に使用するアドレスを算出してから DMEM へパケットを出力する．

- データメモリ機構 (Data Memory：DMEM)

  ALU から受け取ったパケットがロード命令を実行する場合は，データメモリから読みだしたデータをパケットに付与する．ストア命令を実行する場合は，パケットに保持している演算データをデータメモリへと格納する．このとき，データメモリのアクセスにはパケットが保持している演算データ (data) をアドレスとして用いる．

- コピー機構 (Copy Unit：COPY)

  パケットが保持しているコピーフラグを基にパケットの複製を行う．複製の際，命令フェッチ機構で複製元と次に実行する命令コードが同じものにならないように，パケットの宛先情報は複製元とは異なる値に変更する．

- 命令フェッチ機構 (Program Storage：PS)

  次に実行する命令の命令コードとパケットの宛先情報が PS メモリに格納されている．DMEM から受け取ったパケットが保持している宛先情報をアドレスとして，パケットが保持している情報を PS メモリから読み出した情報に書き換える．その後，B へパケットを出力する．

- 分岐機構 (Branch Unit：B)

パケットが保持している情報を基にして，パケットを外部に出力するか再びパイプラインを周回するかを判断し，パケットを出力する．

DDP はセルフタイム型パイプライン (STP) 回路で実現されている．STP は図 2.5 のように，複数の転送制御回路 (Coincidence flip-flop：C 素子) で構成されており，パイプラインステージ内に有効なデータが到着したときに，隣接するパイプラインステージの C 素子間でデータ転送要求信号 (Send) とデータ転送許可信号 (Ack) のやり取りをし，データラッチ (DL) にデータラッチ開放信号 (CP) を送信する．CP を受け取った DL は，ロジック (Logic) へとデータを転送し，データの処理を行う．また，有効なデータを持たないパイプラインステージは電力を消費しないため，低消費電力で動作することが可能である．

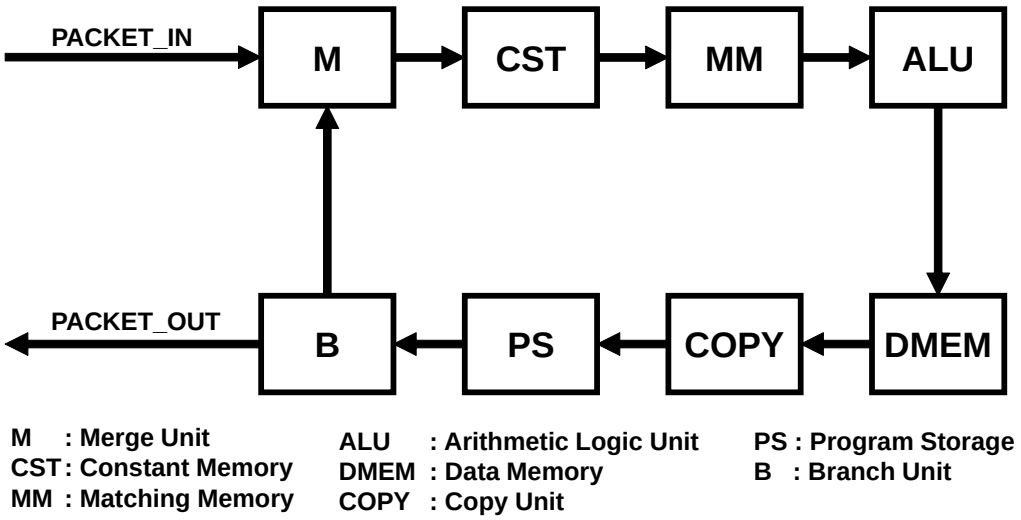


図 2.3　DDP アーキテクチャ [1]

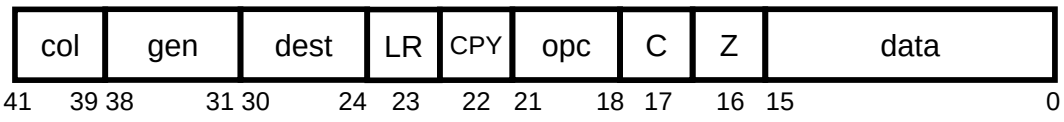


図 2.4　DDP の入力パケットフォーマット [1]

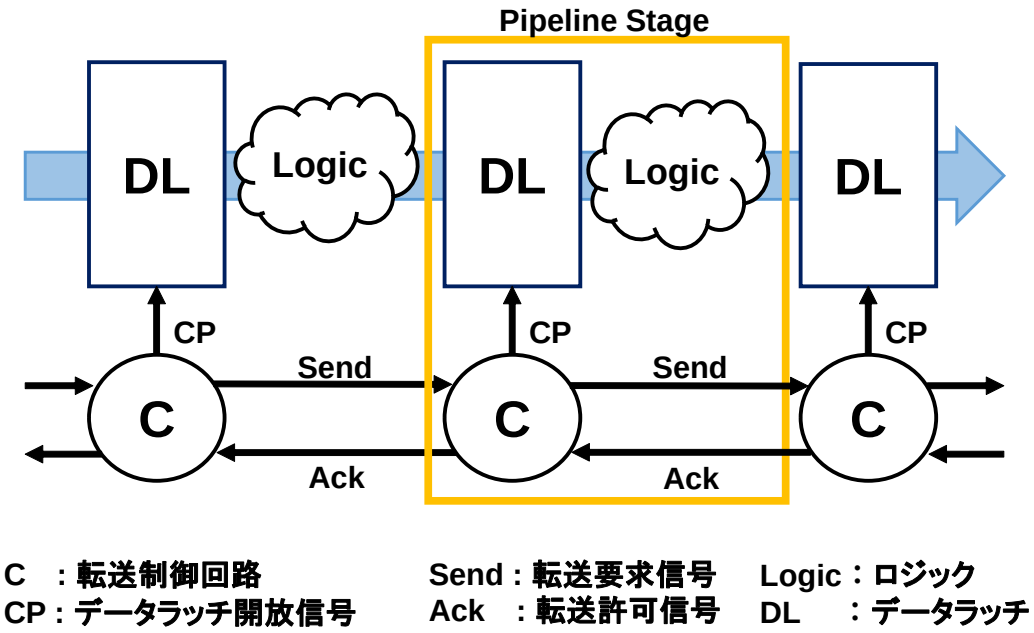


図 2.5　セルフタイム型パイプライン (STP)

表 2.1　パケットフォーマットの詳細 [1]

フィールド名	名称	情報	ビット数
col	color	パケットの識別	3bit
gen	generation	パケットの世代	5bit
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	コピーフラグ	1bit
opc	operation code	命令コード	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

2.4　動的スケジューリング方式

動的スケジューリング方式とは，タスクの実行順序を優先度に基づいて割り当てるスケジューリング方式である．タスクとは，節 2.2 で示すデータ駆動のプログラムを指す．代表的な動的スケジューリング方式の 1 つとして，EDF が挙げられる．EDF とは，各タスクの絶対デッドライン時刻を参照して優先度を付与するスケジューリング方式であり，絶対デッドライン時刻が近いタスクほど高い優先度が付与される．

EDF の動作例として，表 2.2 に示す 3 タスクを逐次処理するときの様子を図 2.6 に示す．タスク B の実行要求時に，タスク A が実行中であるため，EDF によるスケジューリングを行う．タスク B の方が絶対デッドライン時刻が近いため，タスク B に高い優先度が割り当てられ，実行を開始する．絶対デッドライン時刻がタスク B よりも遠いタスク A とタスク C は，タスク B の実行が完了するまで実行を停止する．タスク B の実行が完了すると，タスク A とタスク C のスケジューリングを行う．タスク A の方が絶対デッドライン時刻が近いため，タスク A に高い優先度が割り当てられ，実行を開始する．タスク A の実行が完了すると，タスク C の実行が開始・完了し，全てのタスクの実行を完了する．

表 2.2　タスクの詳細情報

タスク	実行要求時刻	実行時間	相対デッドライン時間
A	0	4	10
B	1	5	7
C	2	3	10

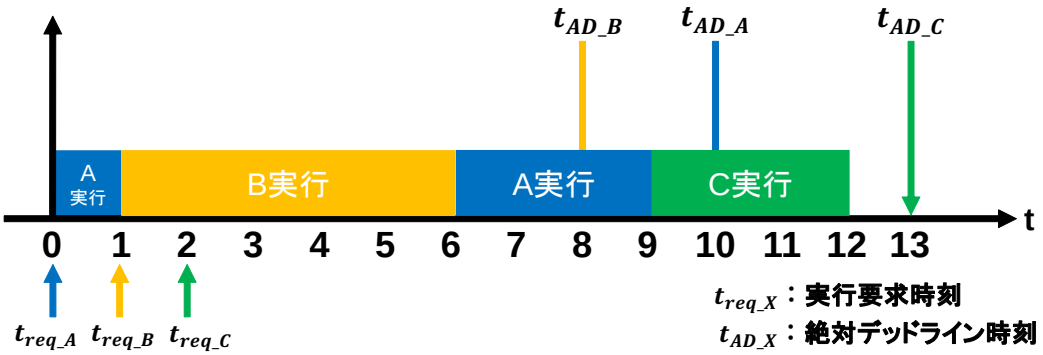


図 2.6　逐次処理における EDF の動作

2.5　DDP におけるスケジューリング制御

DDP は多重処理能力を有しており，低優先度のタスクを実行中に高優先度のタスクの実行が要求されても，それぞれのタスクを多重に実行することが可能であるためスケジューリング制御は不要である．しかし，DDP の多重処理能力には限界があるため，限界を超えるタスクの実行が要求された場合には，高優先度のタスクを低優先度のタスクよりも先に実行するためのスケジューリング制御が必要となる．DDP における EDF の動作例として，表 2.2 に示す 3 タスクを多重処理するときの様子を図 2.7 に示す．このとき，DDP の最大多重度は 2 と仮定する．

図 2.7 より，タスク B の実行要求時に，タスク A が実行中であるが，DDP は多重処理能力を有しているため，タスク A とタスク B を同時に実行することが可能である．しかし，DDP の最大多重度は 2 と仮定しているため，3 つのタスクを同時に実行することはできない．そのため，タスク C の実行要求時に EDF によるスケジューリングを行う必要がある．EDF により，絶対デッドライン時刻が近いタスク B とタスク A にそれぞれ高い優先度が割り当てられて実行を開始し，タスク C は実行を停止する．タスク A の実行が完了すると，DDP の多重処理能力に余力ができるため，タスク C の実行を開始し，タスク B と同時に実行する．タスク B，タスク C の実行が完了すると，全てのタスクの実行を完了する．

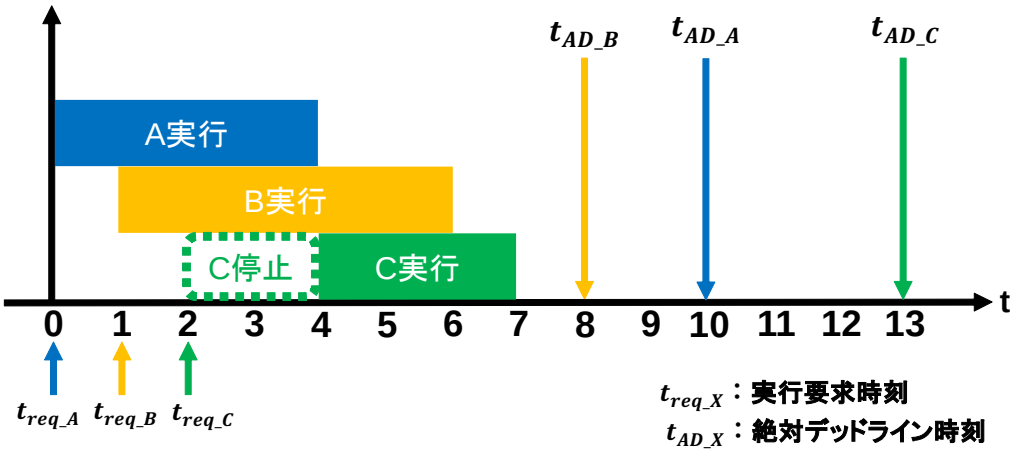


図 2.7　多重処理における EDF の動作

2.6　Earliest Dedline First (EDF) スケジューリング回路を搭載した DDP

DDP におけるスケジューリング制御は，DDP の構成にデッドライン時刻更新機構，デッドライン時刻保存機構，負荷観測機構，キューイング機構を搭載することで可能となる．先行研究 [2] の EDF スケジューリング回路を搭載した DDP (以下，EDF-DDP と呼ぶ) のアーキテクチャを図 2.8 に示し，デッドライン時刻更新機構，デッドライン時刻保存機構，負荷観測機構，キューイング機構の詳細を以下に示す．

- デッドライン時刻更新機構 (Deadline Fix：DF)

  タスク優先度と相対デッドライン時刻がタスクメモリに格納されている．パケットが保持しているパケットの識別情報をアドレスとして，タスク優先度と相対デッドライン時刻をタスクメモリから読み出す．このとき，パケットを受け取った時刻と相対デッドライン時刻を足し合わせることで絶対デッドライン時刻を算出する．その後，パケットにタスク優先度と絶対デッドライン時刻を付与して，M へ出力する．
- デッドライン時刻保存機構 (Deadline Time Memory：DT)

  パケットが保持している絶対デッドライン時刻をメモリに一時保管する．外部から

入力されたパケットは，パケットが保持しているパケットの識別情報をアドレスとして，メモリに書き出す．DDP 内を周回してきたパケットは，メモリから絶対デッドライン時刻を読み出す．

- 負荷観測機構 (Load Monitor：LM)

  DDP 内を周回するパケット数を観測する．多重処理にかかる負荷と DDP 内のパケット数には相関があるため，パケット数から負荷を観測することが可能である．パケット数が増減する要因を以下に示し，これらの要因が発生したかどうかを検出することでパケット数を観測する．パケットに観測結果を付与して Q へ出力する．

  - パケット数が増加する要因
    - パケットが外部から入力される
    - パケットの複製が実行される
  - パケット数が減少する要因
    - パケットが外部へ出力する
    - 待ち合わせをしているパケットが MM で発火する
    - パケットの削除命令が実行される

- キューイング機構 (Queueing Unit：Q)

  LM の観測結果を基にして，受け取ったパケットを管理する．パケット数が EDF-DDP の多重処理能力の限界を超える場合は，パケットをキューに保管してパケットの実行を中断する．このとき，キュー内に保管しているパケットを絶対デッドライン時刻が近い順に並べ替え，絶対デッドライン時刻が最も近いパケットを M へ出力する．他のパケットは多重処理能力に余力ができるまでキューに保管しておく．パケット数が多重処理能力の限界を超えていない場合は，そのまま M へ出力する．

パケットが DF を通過した時点での，先行研究 [2] のパケットフォーマットを図 2.9，表 2.3 に示す．EDF-DDP は，DDP と比較して，入力パケットのパケット長は同じであるが，DF を通過した後，パケットにタスク優先度と絶対デッドライン時刻の情報が付与

されるため，パケット長は 64bit にまで拡張する．

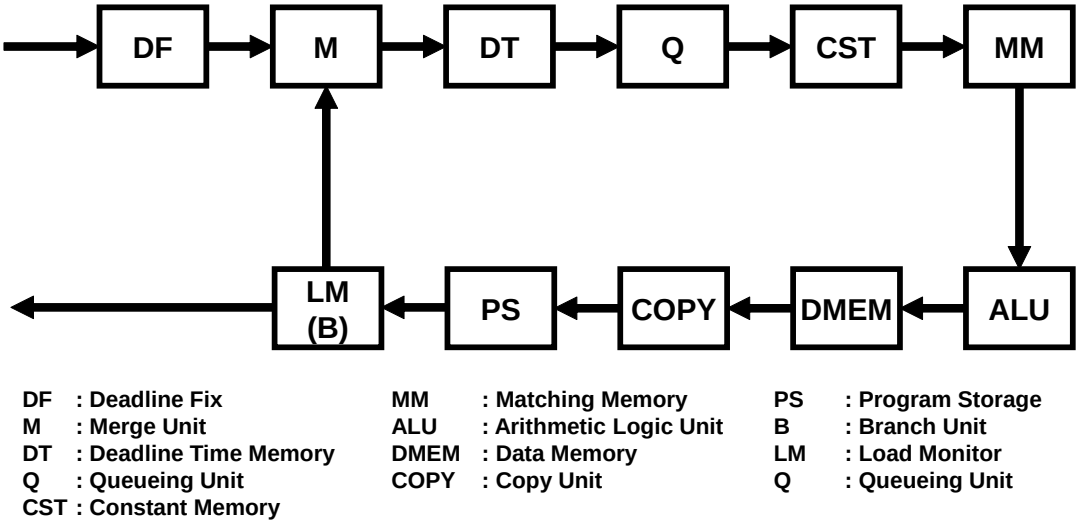


図 2.8　EDF-DDP アーキテクチャ [2]

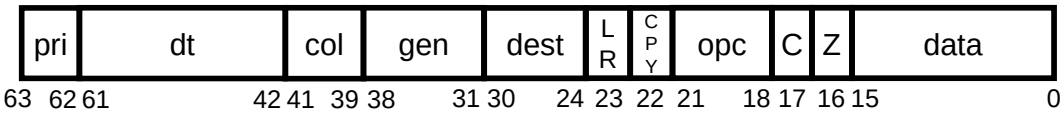


図 2.9　DF のパケットのフォーマット [2]

表 2.3　DF のパケットフォーマットの詳細 [2]

フィールド名	名称	情報	ビット数
pri	priority	タスクの優先度	2bit
dt	deadline time	絶対デッドライン時刻	20bit
col	color	パケットの識別	3bit
gen	generation	パケットの世代	5bit
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	コピーフラグ	1bit
opc	operation code	命令コード	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

2.7　リアルタイムデータ駆動型センサノード

EDF-DDP のリアルタイム IoT システムにおける実用性を検討するためには，対象となるアプリケーションを設定する必要がある．本研究では，アプリケーションとして，センシングを行いデータ分析するシステムを対象とする．それに伴い，EDF-DDP を DDSH として活用する．

2.7.1　リアルタイム駆動型センサノードの概要

リアルタイムデータ駆動型センサノードを図 2.10 に示す．リアルタイムデータ駆動型センサノードを構築するにあたり，前提として，自己充電型のセンサノードを想定している．そのため，間欠動作モードを導入し，センシングをしない間はリアルタイムデー

タ駆動型センサノードの電源を切っておくことで省電力化を図る．

また，目標として消費電力量の削減を目指す．消費電力量は消費電力と稼働時間によって決まるが，本研究では，稼働時間を短くすることで消費電力量の削減を図る．具体的には，リアルタイム性能の向上を研究課題として位置づけ，1 回の稼働で効率良くセンサデータの分析をさせるスケジューリングアルゴリズムを取り入れることで稼働時間の短縮を図る．

リアルタイムデータ駆動型センサノードのプラットフォームは Digilent 社の ZyboZ7-10 と呼ばれる FPGA ボードを想定しており，FPGA ボード内の Zynq7010 と呼ばれる Xilinx 社製プラットフォームのプロセッサ領域に DDSH を，FPGA 領域に各種センサや BLE，組み込み用 OS である PetaLinux を組み込む．プロセッサ領域と FPGA 領域の詳細を以下に示す．

- プロセッサ領域 (Processing System：PS)

  Cortex-A9 の ARM プロセッサが内蔵されている領域であり，各種センサや BLE とのインタフェースを制御する．具体的には，取得したセンサデータを SPI/I2C と AXI を経由して PetaLinix に送信し，DDSH 用のパケットに成形する．パケットが一定量貯まると，それをタスクとして FPGA 領域にある DDSH に送信し，タスクを基に温度や流量データの統計量を求め，UART を介した BLE 通信で制御用端末に統計量データの送信を行う．

- FPGA 領域 (Programmable Logic：PL)

  設計者がプログラム可能な論理回路を集積している領域であり，PS から送信されたタスクの温度や流量データの統計量を DDSH で処理する．DDSH では，割り当てられたタスクの優先度に従ってリアルタイム処理を実現する必要がある．例えば，統計量には複数の算出方法が用意されており，重要な統計量のタスクを処理する場合は High Priority Task として優先的に処理され，重要でない統計量のタスクの処理は Low Priority Task として DDSH の多重処理能力に余裕がある時に処理される．

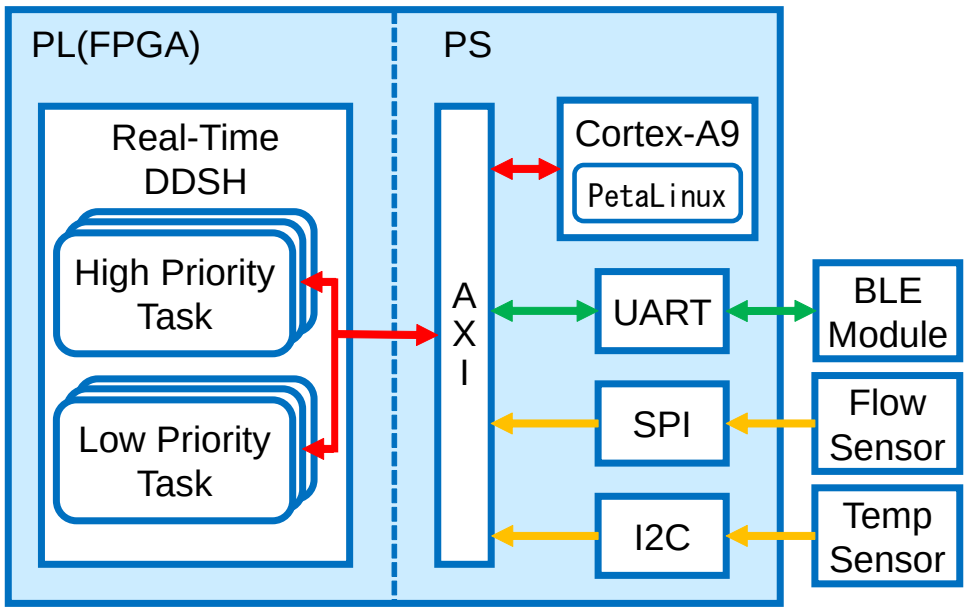


図 2.10　リアルタイムデータ駆動型センサノードの構成

2.7.2　データ駆動型センサハブ (DDSH)

EDF-DDP を DDSH として活用するためには，センサデータを処理可能なタスクセットを想定する必要がある．よって，本研究では，センサハブ機能をタスクセットとして想定し，流量・圧力・温度等のセンサデータを一定時間計測した値から統計量を抽出する．

統計量には複数の算出方法を用意し，統計量の重要度に応じて EDF スケジューリングを行う．例えば，高優先度のタスクは確実にデータを送信する必要があるため，平均値や分散値など比較的簡単な統計量の算出方法を用い，図 2.10 の High Priority Task として DDSH で処理される．低優先度のタスクは確実にデータを送信する必要はないが，送信時により有用なデータとして扱えるように，Short-term Fourier transform など比較的複雑な統計量の算出方法を用い，図 2.10 の Low Priority Task として DDSH で処理される．

2.8　結言

本章では，EDF-DDP の構成とスケジューリング制御について述べ，EDF-DDP をセンサハブ機能を実現する DDSH として活用するリアルタイムデータ駆動型センサノードの構成について述べた．

また，EDF-DDP のリアルタイム IoT システムにおける実用性を検討するために，対象となるアプリケーションとして，センシングを行い，データ分析するシステムを設定した．

次章では，本章で述べた EDF-DDP をデータ駆動型センサノードの DDSH として活用するための課題について述べ，課題に対する方針と提案について述べる．

第 3 章

DDSH におけるリアルタイム処理方式

3.1 緒言

本章では，前章で述べた構成を踏まえて，スケジューリング回路における多重度観測の方針について述べる．また，DDSH が取り扱うタスクとプログラムの定義について述べ，多重度観測の方針を踏まえた提案を述べる．そして，その提案を実現するためのマクロモジュール化アルゴリズム，観測点と同期点によるプログラム変形ついて検討し，キューイング機構と負荷観測機構の構成について述べる．

3.2 スケジューリング回路における多重度観測の方針

EDF スケジューリング回路では，タスク数が DDSH の持つ多重処理能力を超える場合，超過分のタスクのパケットをキューに保管し，EDF スケジューリングにより絶対デッドライン時刻の短いパケットを高優先度タスクのパケットとして扱い，優先的に処理が行われる．

その際，DDSH で処理されているタスク数を多重度として LM で観測されるが，LM が観測できるのは DDSH 内を周回しているパケットのみである．よって，パケット数とタスク数に相関がある単項演算と定数演算は，パケット数を観測することでタスク数を推定し，多重度を算出することが可能である．

しかし，二項演算や COPY の処理では，MM での発火とパケットの複製によってパケット数が増減する．そのため，二項演算やパケットの複製を行うプログラムを含むタスクでは，パケット数とタスク数に相関はなく，パケット数を観測することによる多重度の判定ができない．

LM の観測における課題の解決方針として，以下の 2 つの処理方式の要件が候補として挙げられる．

- ハードウェアスケジューラを二項演算やパケットの複製にも対応できるような構成に作り変える
- 既存のハードウェアスケジューラの構成に合わせてプログラムを変形する

ハードウェアスケジューラの構成を作り変える場合，実現すれば細粒度なスケジュールが可能であるが，ハードウェアスケジューリング回路は複雑になることが懸念される．場合によってはスケジューリング回路だけでなく DDSH の構成も変更しなければならない可能性もあり，既存の回路を他のスケジューリングアルゴリズムに応用する拡張性も損なわれる可能性がある．

既存のハードウェアスケジューラの構成に合わせてプログラムを変形する場合，着目するのはプログラムであるため，既存のスケジューリング回路の構成をそのまま活用可能である．しかし，既存のハードウェアスケジューラに合わせる関係上，処理できるプログラムの種類に制限が設けられる可能性があり，後述するパケット数とタスク数に相関を持たせるようにプログラムをマクロモジュール化する変形方法では，ハードウェアスケジューラの構成を作り変える場合よりも粗粒度なスケジュールとなりスケジューリング性能が低下する恐れがある．

以上のことから，本研究では，既存のハードウェアスケジューラは先行研究によって性能・回路規模ともに有用性が示されていることと粗粒度なスケジュールによる性能の低下も，コンテキストスイッチが頻繁に発生するソフトウェアスケジューラよりもリアルタイム性は高水準を維持できる可能性があることから，プログラム変形に着目して検

討を行う．

そのため，本研究では，LM で観測するパケットの粒度を粗くする観測方式を提案する．具体的には，パケット毎に観測していた従来の LM の処理方式を，一定間隔毎のパケットを観測点と定め，観測点のパケットが通過する際に観測する処理方式に変更する．そうすることで，既存のスケジューリング回路の構成をほとんど変更せずに二項演算などの演算が含まれた汎用タスクの多重度を観測でき，先行研究 [2] の回路規模を維持したままリアルタイムデータ駆動型センサノードへ組み込むことが可能となる．

その際，観測の間隔をマクロにしたことで，スケジューリング性能は低下することが予想される．そのため，観測方式の変更によるスケジューリング性能の低下が対象となるアプリケーションの実行にどの程度影響を及ぼすのかを検討する必要がある．

3.3　タスクとプログラムの定義

本研究で扱うタスクは，DDSH 上で動作するプログラムの単位を表しており，優先度クラスによってハードリアルタイムタスク (HR)，ソフトリアルタイムタスク (SR)，時間制約のないタスク (N) に分類される．

HR はデッドラインまでに終了しなかった場合にシステムに致命的な影響を及ぼすタスクであり，必ず時間制約を守る必要がある．SR はデッドラインまでに終了しなかった場合でも即座にシステムへ影響を及ぼすことはないが，終了時間によって処理結果の価値が徐々に落ちていく．そのため，時間制約を可能な限り完了する必要がある．時間制約のないタスクはデッドラインがなく，時間制約を気にする必要はない．

また，プログラムは DDSH に実行させる命令の一連の流れを記載したものであり，表 3.1 にあるパケットが保持する情報を基に命令が処理される．本研究で取り扱うプログラムでは，単項演算・定数演算・二項演算・条件分岐が含まれているプログラムを対象とする．

表 3.1　パケットが保持する情報の詳細

フィールド名	名称	情報	ビット数
pri	priority	タスクの優先度	2bit
CNT	count flag	パケットの増減の識別	2bit
dt	deadline time	絶対デッドライン時刻	20bit
del_ctrl_sig	delete control signal	キューの削除フラグ	1bit
input_flag	input flag	キューイングフラグ	2bit
sel	select	入力/巡回パケットの識別	1bit
col	color	パケットの識別	3bit
gen	generation	パケットの世代	5bit
dest	destination	パケットの宛先	7bit
BR	branch flag	出力/巡回パケットの識別	1bit
LR	left or right	パケットの左右の識別	1bit
CP	copy flag	コピーフラグ	1bit
opc	operation code	命令コード	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

3.4　多重実行タスクの観測方式

本研究で提案する観測方式を 2 次方程式の解の公式に適用した様子を図 3.1 に示す．図 3.1 のように，一定範囲内の演算ノードの集合をモジュールとして扱い，タスク内の観測点を特定パケットに限定する．そうすることにより，マクロモジュール化した際，LM からは直線的なプログラムに見え，従来のパケット数からタスク数を判断する処理を適

用できる．

モジュールの出力から複数のパケットが観測される際は，LM で観測するパケットを観測点として設定し，他パケットを同期点として MM で観測点と待ち合わせを行う演算命令に PS で書き換え，MM で一時待機させる．

観測点となるパケットが LM で観測された後，観測点となるパケットの複製を行い，各同期点と観測点の複製パケットが発火し，同期点のパケットは実行を再開する．以上のことから，従来の回路の構成をほとんど変更することなく図 3.1 のような二項演算などの命令が含まれた汎用タスクの多重度を観測することが可能になる．

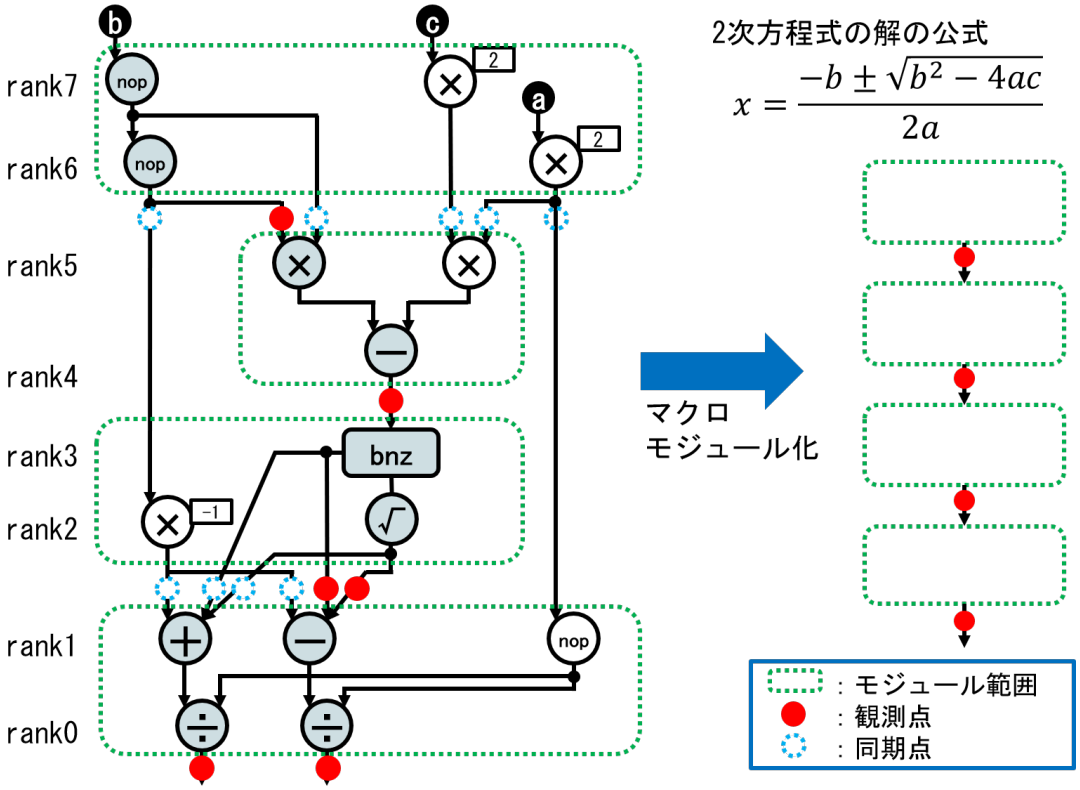


図 3.1　プログラムのマクロモジュール化

3.5　タスクプログラムの変形法

3.5.1　マクロモジュール化のアルゴリズム

タスク数を観測し，状況に応じて停止させる際は，プログラム中で多くのノードとデータ依存関係がある経路上のパケットを観測することでタスク全体を停止しやすくなる．

そのことから，マクロモジュール化を行うにあたり，クリティカルパスを中心としたデータ依存関係を持つノードを優先的に対象としてモジュールを作成することで，粒度が粗くなることによるスケジューリング性能の低下を緩和することが可能である．

よって，クリティカルパスを中心にモジュール設定を行うアルゴリズムを定式化し，Algorithm1 に示す．

Algorithm1 中の $rank$ は，ノードが実行する時の時間を表している．また，図 3.1 のように，条件分岐の出力を観測点に設定する場合，経路が True と False にの 2 通り存在するため，観測点も 2 か所設ける必要がある．このとき，1 つのモジュールに対して 2 つの観測点が存在するが，出力パケットは片方の経路しか通らないため，LM での観測も 1 か所のみとなり，他演算と同様に，タスクの実行停止/再開が実現できる．

Algorithm 1 $M \times M$ モジュール化アルゴリズム

1: 他ノードとデータ依存関係が最も多いクリティカルパス CP を 1 つ選択
2: 対象ノード $T_N \leftarrow CP$ に含まれる $rank0$ のノード
3: **while** ∃ 未処理ノード **do**
4:　**if** $T_N \in CP$ **then**
5:　　T_N から CP のノード M 個づつを含むモジュールを作成
6:　　**if** CP が条件分岐上の経路である **then**
7:　　　同 $rank$ の条件分岐の経路となるノードをモジュールに組み入れる.
8:　　**end if**
9:　**else**
10:　　T_N を含むモジュールを作成
11:　**end if**
12:　**for** 各モジュール **do**
13:　　同 $rank$ のノードをモジュールに組み入れる※
14:　**end for**
15:　Update T_N：未処理ノードから最小 rank ノードを選択※
16: **end while**

※ノードから CP へのデータ依存関係を優先

実際に，図 3.1 の 2 次方程式の解の公式を解くプログラムに Algorithm1 を適用させた手順を下記に示す．また，今回は $M = 2$ と仮定する．

ステップ 1　他ノードとデータ依存関係が最も多いクリティカルパスを 1 つ選択する．選択した様子を図 3.2 に示す．

ステップ 2　$rank0$ から順にクリティカルパス上のノード 2 個ずつを含むモジュールを作成する．この時，モジュールに含んだノードが条件分岐上の経路である場合，同 $rank$ にあるノードをモジュールに組み入れる．モジュールを作成し，$rank0$ と

$rank1$ の条件分岐上の経路のノードをモジュールに組み入れた様子を図 3.3 に示す．

ステップ 3　各モジュールに対し，同 $rank$ のノードを 2×2 の範囲を超えないようにモジュールに組み入れる．この時，組み入れるノードはクリティカルパスへのデータ依存関係があるものを優先する．ノードをモジュールに組み入れる様子を図 3.4 に示す．

上記より，2 次方程式の解の公式を解くプログラムはマクロモジュール化される．もしステップ 3 の終了時点でマクロモジュール化されていないノードが存在している場合，マクロモジュール化されいないノードの中で最小 $rank$ のノードを対象ノードとして更新し，対象ノードを中心とした最大 2×2 のモジュールを作成していく．

その後，図 3.5 のように，モジュールからの出力でクリティカルパス経路上のパケットを観測点として設定し，その他のモジュールからの出力パケットを同期点として設定する．条件分岐上の経路からの出力がある場合は，True/False どちらの経路にも観測点を設定して確実に観測できるようにする．こうすることで，LM で観測するパケットの特定が可能である．

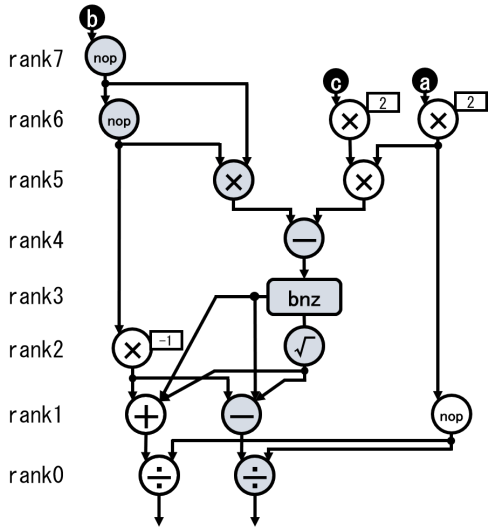


図 3.2　ステップ 1：クリティカルパスの選択

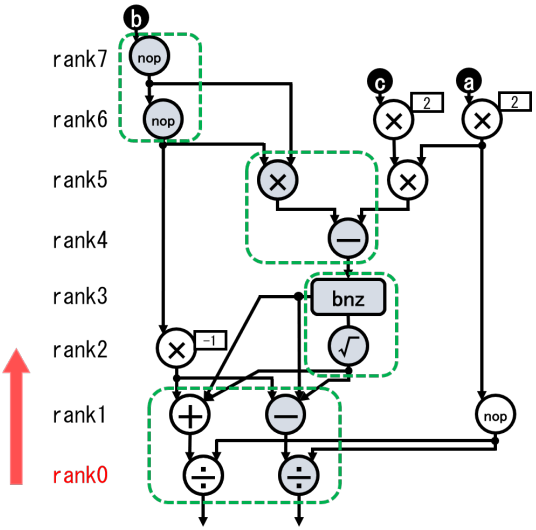


図 3.3　ステップ 2：モジュールの作成

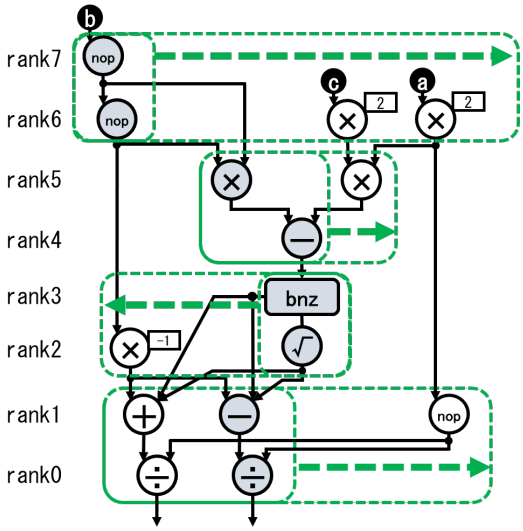


図 3.4　ステップ 3：ノードの追加

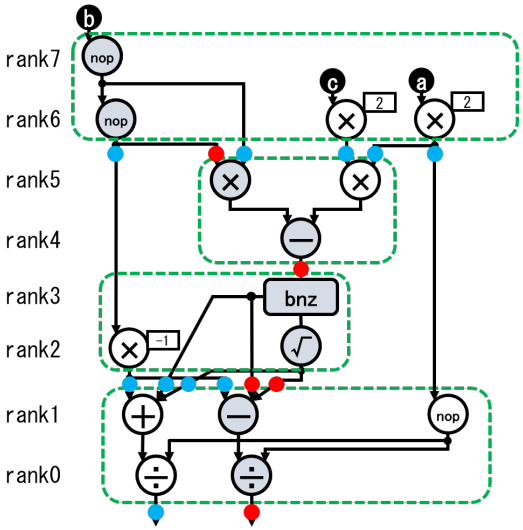


図 3.5　観測点と同期点の設定

3.5.2　観測点と同期点のプログラム変形

観測点と同期点の動作を実際のプログラムで行うためにプログラムの変形を行う．具体的には，観測点と同期点を実現するためにパケット複製命令と同期命令をプログラムに追加する．各動作の詳細を下記に示す．

- パケット複製命令 (copy)

観測点として設定されたパケットが LM を通過し観測された後，同期点の数だけパケットの複製を行う．この時，ALU では何も処理を行わず，COPY でパケットの複製のみ行う．複製したパケットは MM で待ち合わせをしている同期点として設定されたパケットと発火する．

- 同期命令 (sync)

  MM で待ち合わせを行い，観測点として設定されたパケットと発火する．この時，同期点として設定されたパケットは synq の右パケット，観測点として設定されたパケットは sync の左パケットとなるようにプログラムを構成する．

  発火後は，ALU で右パケットのデータを保持して DMEM へ出力される．

パケット複製命令と同期命令をプログラムに追加した様子を図 3.6 に示す．図 3.6 は，図 3.5 のプログラムを一部抜粋したものである．モジュールからパケットが出力されると，同期点は sync により MM で待ち合わせを行い，観測点は LM で観測される．LM での観測が終わると copy によってパケットが複製され，同期点として設定されたパケットと発火する．

これにより，パケット複製と同期命令により同期点の実行待機/再開を実現可能になるが，プログラムに同期命令を追加することがオーバーヘッドとなり，既存の EDF 回路よりスケジューリング性能は低下する．さらに，パケットの複製による DDSH 内を巡回するパケット数の増加と同期命令による MM の待ち合わせ領域の占有により，DDSH の処理性能の低下も予想される．

しかし，プログラムを並行して実行するため SW スケジューラよりはコンテキストスイッチの発生回数，オーバーヘッドの長さともに改善され，性能の向上が予想される．

また，観測点と同期点の実現方法には，同期命令の他にロード・ストア命令を用いる方法もある．ロード・ストア命令を用いた場合，MM ではなく DMEM のメモリ領域を使用するため，同期命令を用いるよりも DDSH の処理性能の低下を低減することが見込まれる．しかし，プログラムにパケットの複製命令とロード命令，ストア命令の 3 命令

を追加する必要があるため，同期命令を用いるよりもオーバーヘッドは大きくなってしまう．

本研究では，稼働時間を短くすることで消費電力量の削減を図っているため，オーバーヘッドが小さいパケットの複製命令と同期命令の追加によって観測点と同期点を実現している．しかし，同時に処理するタスクが多くなるほどパケットの複製による MM の占有は無視できなくなるため，パケットの複製命令とロード・ストア命令による観測点と同期点の実現についても精査する必要がある．

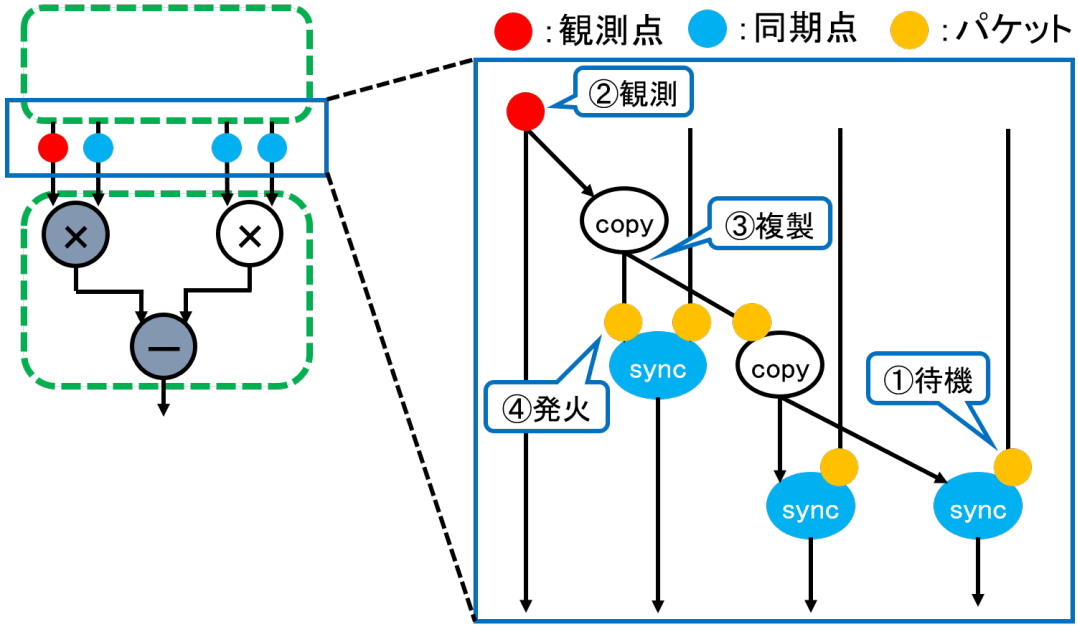


図 3.6　同期点と観測点の動作

3.6　ハードウェアスケジューラ回路

マクロモジュール化アルゴリズムに基づいた観測点と同期点のプログラム変形に伴い，EDF-DDP のキューイング機構と負荷観測機構の構成を修正する必要がある．修正後の回路がどのように動作するかの詳細を下記に示す．

3.6.1　キューイング機構

キューイング機構の回路構成を図 3.7 に示す．DT から受け取ったパケットは各 Queue (HR_PQ，SR_PQ，N_Q) と DL に送られる．

HR_PQ は，ハードリアルタイムタスクが格納され，必ず時間制約を守る必要があるため，MUX で最優先に選択される．

SR_PQ は，ソフトリアルタイムタスクが格納され，必ずしも時間制約を守る必要はないものの，時間経過とともにタスクの価値が下がってしまうため MUX ではできる限り優先して選択される．

N_Q は，時間制約のないタスクが格納され，DDSH の多重処理能力に余裕がある時に MUX で選択される．

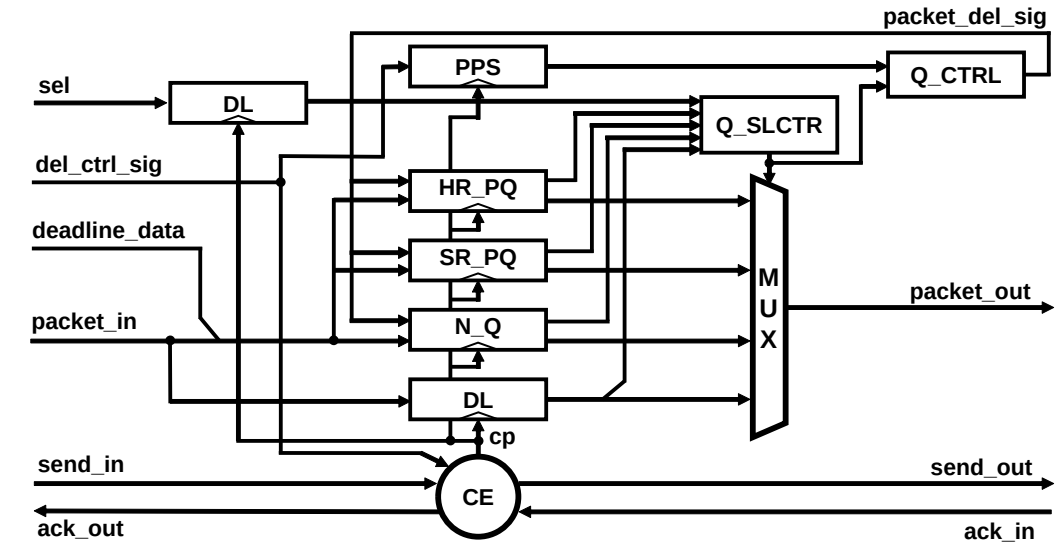
各 Queue は，DT から受け取ったパケットが観測点に設定されているパケットである場合に優先度に応じた Queue へ格納する．マクロモジュール化されたパケットや同期点として設定されたパケットは DL に格納される．

PQ は図 3.8 のような構成になっており，DDSH の多重処理能力に余裕がある場合は，ENTRY0 にのみパケットが格納され，DL と同じ役割を果たすが，DDSH の多重処理能力を超える場合，ENTRY1 以降にもパケットが格納されていき，一時保管される．

この時，PQ は EDF に基づいたスケジューリングを行い，絶対デッドライン時刻が最も短いパケットが PQ から出力される．

Q_SLCTR では，各 Queue 内に保存されているパケットの数とパケットの種類を基に，どの Queue のパケットを次ステージへ出力するかの制御信号を生成する．MUX では，Q_SLCTR の制御信号を基にパケットを選択して出力を行う．

パケットが PQ から出力された後，そのパケットの情報はキューに保存されたままであるため，キューの消去フラグである del_ctrl_sig を用いてキュー内の出力済みパケットを消去する．



PPS : Previous Packet State
Q_CTRL : Queue Controller
Q_SLCTR : Queue Selector
CE : C Element with Packet Eliminator
HR_PQ : Priority Queue for Hard Real-Time Task
SR_PQ : Priority Queue for Soft Real-Time Task
N_Q : Queue for Normal Task

図 3.7　Q の回路構成

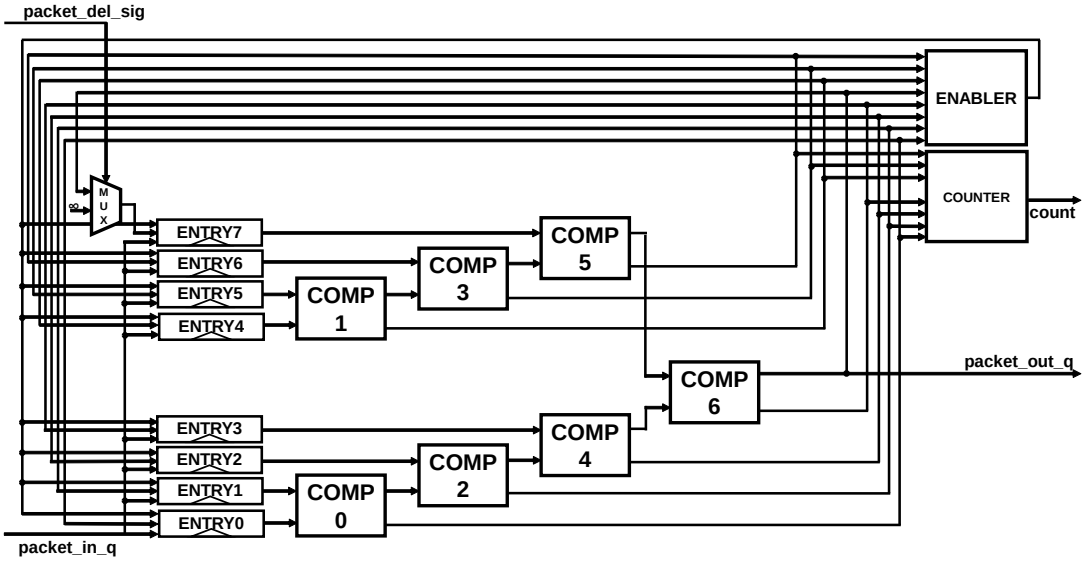


図 3.8　PQ の回路構成

3.6.2 負荷観測機構

DDSH 内を周回する観測点に設定されたパケット数を観測しタスク数を推定する．観測点となるパケットの判別には EDF-DDP でパケット数の増減を識別する情報である count flag (CNT) を表 3.2 のように変更し利用する．

表 3.2 CNT の詳細

CNT (2bit)	パケットの種類 (変更前)	パケットの種類 (変更後)
00	-	-
01	パケット数増加	観測点 (DDSH 内を周回)
10	パケット数減少	観測点 (外部へ出力)
11	パケット数変化なし	タスク数変化なし

観測したパケットの種類が観測点 (DDSH 内を周回) である場合，新たなタスクが DDSH 内で処理されていると判断しタスク数を 1 増加する．観測したパケットの種類が観測点 (外部へ出力) である場合は，タスクの実行が終了したと判断しタスク数を 1 減少する．

また，タスク数が DDSH の持つ多重処理能力を超える場合は，キューの消去フラグである del_ctrl_sig を生成し，キュー内に超過分のパケットを保存しておくように制御する．

3.7 結言

本章では，DDSH が取り扱うタスクとプログラムの定義について述べ，多重度観測の提案としてマクロモジュール化アルゴリズムと観測点と同期点によるプログラム変形について述べた．また，キューイング機構と負荷観測機構の構成と動作について述べた．

本研究の提案方式によって二項演算や分岐命令を含むプログラムのタスクも処理可能になると予想されるが，その一方でマクロな観測によるスケジューリング性能の低下，

観測点と同期点のためにパケットの複製と同期命令の追加による DDSH 自体の処理性能の低下が懸念される．また，回路構成も EDF-DDP から一部修正しているため，回路規模を EDF-DDP と比較して，どのくらい増加しているかを確認する必要がある．

次章では，提案方式の回路規模の評価と性能評価を行い，提案方式が有用であるかの確認を行う．

第 4 章

評価

4.1 緒言

本章では，前章で提案した方式を取り入れた DDSH を設計し，回路規模の評価と性能評価を行うことで，提案方式が有用であるかの検討を行う．

まず，リアルタイムデータ駆動型センサノードの評価環境について述べ，PS と PL の動作・仕様について述べる．そして，DDSH の回路規模とスケジューリング性能をそれぞれ評価した結果について述べ，その結果から提案方式が有用であるかを判断する．

4.2 評価環境

本研究では，Xilinx 社製 Zynq7010 の評価ボードである Digilent 社製 ZyboZ7-10 にリアルタイムデータ駆動型センサノードを構築した．リアルタイムデータ駆動型センサノードの構築に必要な環境を下記に示す．

- センサノード用ハードウェア
  - ZyboZ7-10 の Zynq7010
    - PS：PetaLinux，インタフェース (UART，I2C，AXI)
    - PL：DDSH
  - BLE，温度センサ
- 開発用ソフトウェア
  - Vivado (Version：2022)

  - Vitis (Version：2022)
  - Petalinuxtools (Version：2022)
- 使用言語
  - Verilog-HDL
  - C 言語

4.3　PS の動作

PS では，センシングとデータの通信を行う．Vivado で図 4.1 のように，BLE とセンサのインタフェース回路を設定する．インタフェース回路の制御は組み込み用 OS である PetaLinux で行われており，図 4.2 のように，温度センサからデータを取得し，BLE を介して端末に温度データを送信できることを確認した．

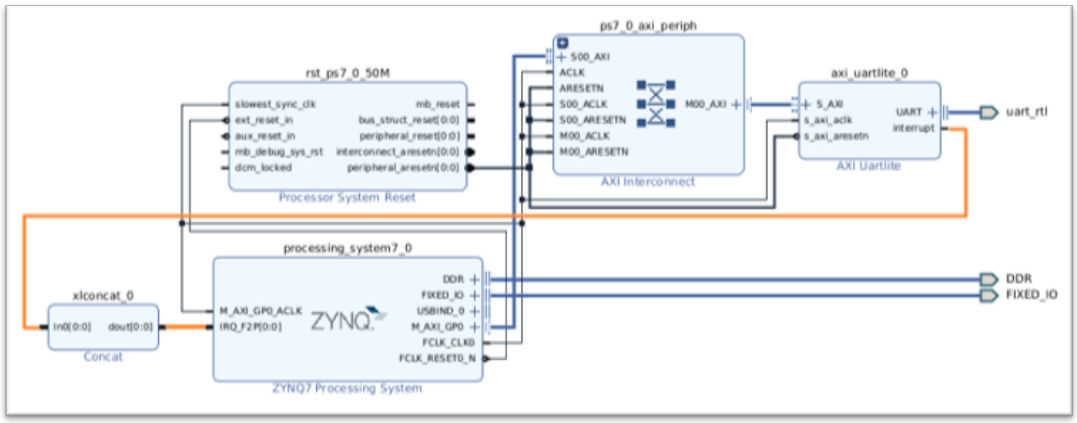


図 4.1　PS のブロックデザイン

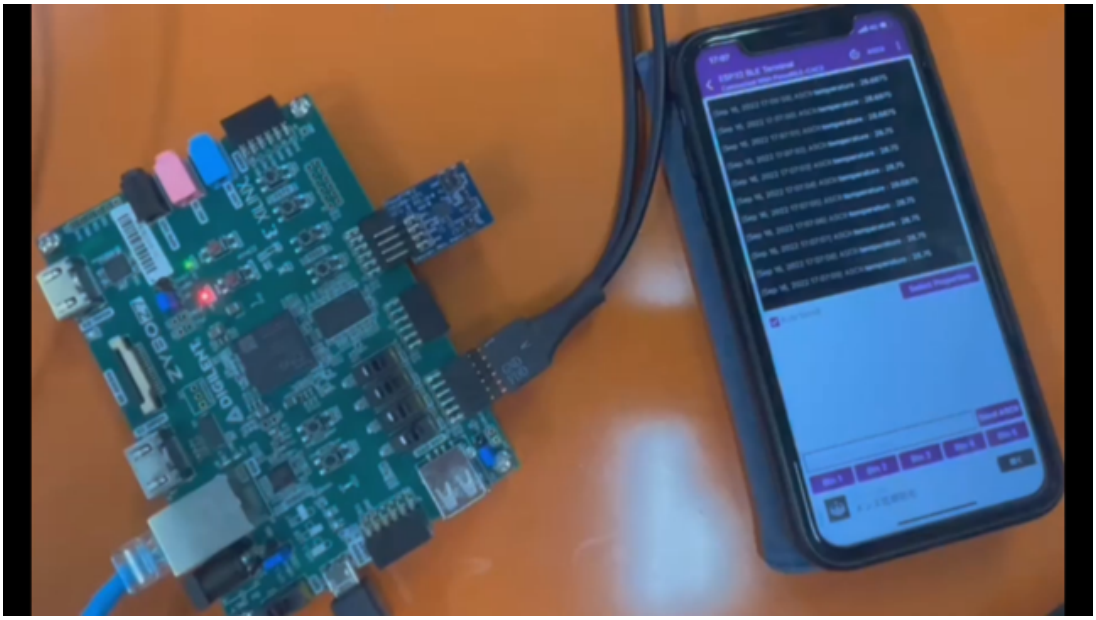


図 4.2　PS と端末の通信

4.4　DDSH の仕様

DDSH は，対象とするアプリケーションのインターフェースによってアーキテクチャが異なり，それに伴ってパケットサイズ，メモリ容量，エントリ容量も異なる．そのため，本研究で定める提案 EDF-DDP の設計仕様を示す．提案 EDF-DDP で用いるパケットのパケット長，メモリの容量，MM と Q で用いるエントリの容量をそれぞれ表 4.1，表 4.2，表 4.3 に示す．

表 4.1 入力パケットの設計仕様

	パケット長
DF	42bit
M (from DF)	67bit
M (from LM)	47bit
DT	68bit
Q	68bit
CST	42bit
MM	62bit
ALU	61bit
DMEM	38bit
PS	38bit
LM(B)	46bit

表 4.2 メモリの設計仕様

	容量
タスクメモリ	25bit × 8words
絶対デッドライン時刻メモリ	20bit × 8words
定数メモリ	20bit × 64words
内部メモリ	24bit × 64words
データメモリ	16bit × 1024words
PS メモリ	17bit × 128words

表 4.3　エントリの設計仕様

	容量
MM	19bit × 64entries
Q	66bit × 8entries × 3queues

4.5　評価用プログラムの仕様

本研究では，センシングを行い，データ分析するシステムを対象アプリケーションとして設定する．センサデータのスケジューリングは，ハードリアルタイム性を要求されることはないが，値が激しく変化する圧力などは DDSH 内のパケット流量が著しく増加し，スケジューリングの失敗を招く恐れがある．

よって，評価用のプログラムとして，図 4.3，図 4.4 のような 1 入力に対し複数の出力があり，負荷が漸増するプログラムと複数の入力に対し 1 つの出力があり，負荷が漸減するプログラムの 2 種類を設定し，DDSH 内のパケット流量の増減に伴うスケジューリング性能・DDSH の処理性能を評価する．

また, DDSH でスケジューリングができるように, 本研究の提案であるマクロモジュール化を行い，観測点と同期点を設定したプログラムをそれぞれ図 4.5 と図 4.6 に示す．

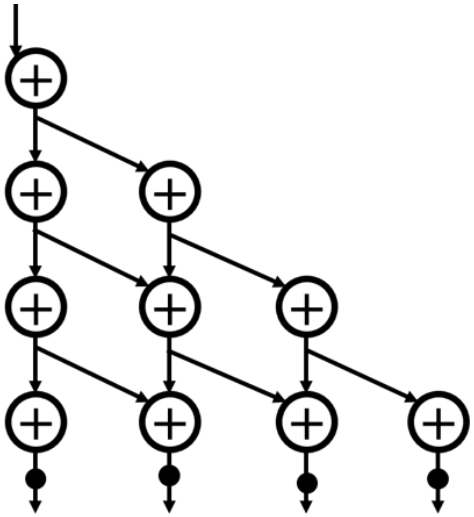


図 4.3　負荷漸増プログラム

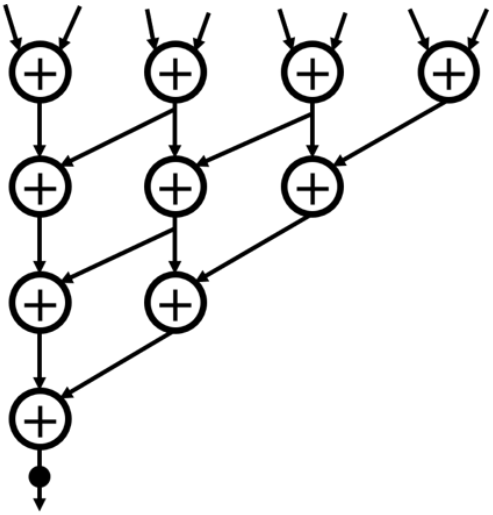


図 4.4　負荷漸減プログラム

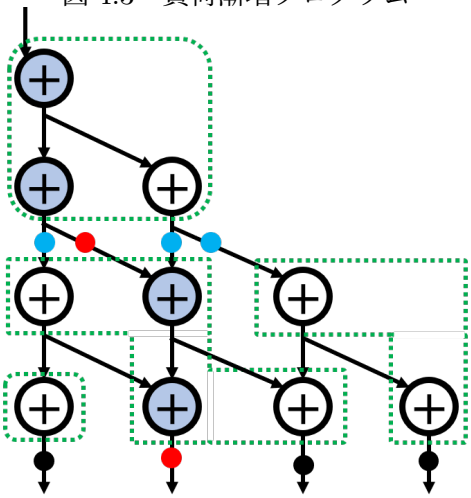


図 4.5　負荷漸増プログラム
のマクロモジュール化

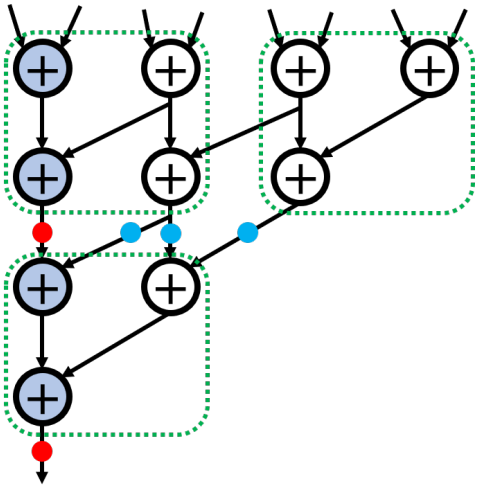


図 4.6　負荷漸減プログラム
のマクロモジュール化

4.6　DDSH のスケジューリング性能評価

DDSH 上で図 4.3，図 4.4 のプログラムを以下の組み合わせで実行した．

- 負荷漸増プログラム (4 タスク)
- 負荷漸減プログラム (4 タスク)
- 負荷漸増プログラム (2 タスク)・負荷漸減プログラム (2 タスク)

実行した結果，どの組み合わせでも正常に動作することが確認できた．図 4.3，図 4.4 は，パケット複製と二項演算を用いた典型的なタスクに属していることから，本研究の提案によりタスクをリアルタイム処理する見通しが得られたと判断できる．

4.7　DDSH の回路規模評価

DDSH と EDF-DDP の回路規模を比較し，提案によって DDSH が EDF-DDP の回路構成をほとんど変更することなく実装できているかを確認する．回路規模は，比較する要素として Slice 数と Register 数の 2 つを確認する．DDSH と EDF-DDP の回路規模を表 4.4 に示す．

表 4.4　メモリの設計仕様

	DDSH	EDF-DDP	増減率
Slice	1375 個	1303 個	0.05%減
Register	3477 個	3496 個	0.01%増

表 4.4 より，Slice 数は 0.05%減少し，Register 数は 0.01%増加することが確認できた．

DDSH は観測点と同期点の処理を追加したため，Register 数が増加したのは想定通りである反面，Slice 数の減少は意図した結果にならなかった．

このことは Vivado の最適化による回路の自動配置・配線が要因であると考えられる．現状，設計ツールの自動最適化のアルゴリズムを解析することは難しく，より厳密に比較するためには，ユーザによる手動の回路の配置・配線を行う必要がある．

しかし，Slice 数・Register 数の増減率はどちらも 0.1%未満であり，自動配置・配線による回路規模への影響があったとしても，DDSH と EDF-DDP の回路規模はほとんど同じであると判断できる．

以上のことから，本研究の提案により既提案ハードウェアである EDF-DDP を活用可能であることが確認できた．

4.8　結言

本章では，提案方式を基に設計した DDSH がスケジューリング性能・回路規模の観点から有用であるかの検討を行った．その結果，性能面では典型的なタスクを実行できることを確認でき，回路規模の面では，既提案のハードウェアである EDF-DDP をほとんど変更せずに実装できることを確認した．

以上のことから，既提案ハードウェアを活用して実用的なタスクのリアルタイム処理を実現する見通しが得られたと判断できる．

次章では，本論文で述べた内容をまとめるとともに，提案方式の今後の課題を述べる．

第 5 章

結論

5.1 本論文のまとめ

近年，IoT デバイスは様々な分野で利用されており，その中でも，スマート家電や IoT 化された電子機器が増加するコンシューマ，スマート工場やスマートシティが拡大する産業用途，コネクテッドカーの普及により IoT 化の進展が見込まれている自動車の分野での IoT デバイス数の高成長が予想されている．これらの分野のシステムでは，リアルタイム性を要求することが多く，複数のタスクを制御するためのスケジューリングが必要である．

リアルタイムスケジューリングのアルゴリズムとして，動的優先度方式が挙げられる．代表的なアルゴリズムとして EDF と LST が挙げられ，EDF は絶対デッドライン時刻が近いタスクに高い優先度を割り当てるアルゴリズムであり，LST は余裕時間が短いタスクに高い優先度を割り当てるアルゴリズムである．EDF と LST は，シングルコアにおいて，リアルタイムシステムで最適なスケジューリングであると考えられるが，マルチコアでは，EDF は実行時間を考慮しないためデッドラインミスを予想できないという問題がある．LST は実行時間を考慮するためデッドラインミスを予想できるが，余裕時間の近いタスクが複数存在する場合，それぞれのタスク優先度が短い期間で変化するためスラッシングが発生し，スケジューリングオーバーヘッドが大きくなる可能性がある．

オーバーヘッドが大きくなる問題を解決するために，スケジューリングを搭載するプロセッサに着目する．DDP は多重処理が可能であり，DDP の多重処理能力に余力がある場合はオーバーヘッドが発生することなく複数タスクの実行が可能である．

先行研究では，DDP と動的優先度方式に着目し，マルチコア化に向けて，DDP シングルコアに動的優先度方式のスケジューリング回路を搭載する方法が提案されている．また，EDF スケジューリング回路を搭載した DDP では回路規模の最適化も行われている．

しかし，既提案のスケジューリング回路はどれも単項演算か定数演算のみで構成されたプログラムのタスクを対象としており，リアルタイムシステムでスケジューリング回路を搭載した DDP を実用的に利用するためには，二項演算を含む汎用性のあるプログラムのタスクを対象として処理できる方法を見出す必要がある．

以上のことから，本研究では，センシングを用いたシステムを対象として，DDSH を用いたリアルタイムデータ駆動型センサノードの構築を目指す．そのために，汎用的なプログラムのタスクでもリアルタイム処理可能な方式を提案した．

具体的には，マクロモジュール化アルゴリズムと観測点と同期点によるプログラム変形を用いたマクロな観測を行うことで，二項演算を含むプログラムのタスクの多重度観測が実現可能になると予想した．

実際に，Xilinx 社製 ynq7010 の評価ボードである Degilent 社製 ZyboZ7-10 にリアルタイムデータ駆動型センサノードを構築し，DDSH の回路規模とスケジューリング性能を評価した結果，既提案ハードウェアである EDF-DDP よりも Slice 数は 0.05%減少し，Register 数は 0.01%増加することが確認できた．また，パケット複製と二項演算で構成された典型的なプログラムである負荷漸増プログラムと負荷漸減プログラムを様々な組み合わせで 4 タスク同時実行した結果，全て正常に動作することが確認できた．

これらのことから，本研究で提案した多重度観測の方式によって，既提案ハードウェアを活用して実用的なタスクのリアルタイム処理を実現する見通しが得られた．

5.2　今後の課題

本研究の今後の課題を以下に示す．

- マクロモジュール化アルゴリズムにおける条件分岐とロード・ストア命令

本研究では単項演算・定数演算・二項演算・条件分岐が含まれているプログラムを対象としてマクロモジュール化アルゴリズムを定義した．しかし，条件分岐の種類は分岐後しばらくしたら True/False の経路が合流するものを対象としており，経路によって独立した処理を行うものに関してはマクロモジュール化アルゴリズムが適用できるのか定かではない．また，今後はロード・ストア命令を含むプログラムについても考慮する必要があり，本研究で提案したマクロモジュール化アルゴリズムが全ての汎用プログラムを網羅しているかを検討する必要がある．

- マクロモジュール化のサイズの決定方法

  本研究では, マクロモジュール化のアルゴリズムをプログラムに適用する際, $M \times M$ モジュールのサイズを $M = 2$ と仮定して適用している．しかし，プログラムの規模や要求されるスケジューリングの精度は様々であるため，アプリケーション毎にモジュールのサイズを適切に設定できるアルゴリズムが必要である．

- 観測点と同期点の実現方法

  観測点と同期点の実現方法は，パケットの複製命令と同期命令をプログラムに追加することと，パケットの複製命令とロード・ストア命令をプログラムに追加することの 2 通りが挙げられる．各方法の長所と短所を下記に示す．

  本研究では，稼働時間を短くすることで消費電力量の削減を図っているため，オーバーヘッドが小さいパケットの複製命令と同期命令をプログラムに追加する方法を提案したが，どちらの方法が適しているかは改めて精査する必要がある．

  また，パケットの複製命令と同期命令をプログラムに追加する方法では，条件分岐上の経路に観測点が設定されている場合，True/False によって 2 通りの経路の観測点が存在している．このとき，同期点と待ち合わせをする際は，True/False どちらの場合でも発火できる必要がある．そのことから，条件分岐の種類によっては同期点を実現できない可能性がある．

  – パケットの複製命令と同期命令の追加

    2 命令の追加で実現可能なため，ロード・ストア命令を追加するよりもオーバー

ヘッドは小さい．しかし，同時に実行するタスクが多いほどパケットの複製による MM の占有率は高くなり，DDSH の性能が著しく低下する恐れがある．

– パケットの複製命令とロード・ストア命令の追加

同期点として設定されたパケットを DMEM のメモリ領域に格納するため，MM がパケットの複製によって占有されることを防ぎ，DDSH の性能の低下を低減できる．しかし，パケットの複製命令、ロード命令、ストア命令の 3 命令を追加する必要があるため，パケットの複製命令と同期命令の追加よりもオーバーヘッドは大きくなる．

- スケジューラビリティの評価

本研究では，DDSH のスケジューリング性能の評価を行ったが，厳密にはスケジューラビリティも評価する必要がある．そのために，評価に適したベンチマーク・タスクセットについて検討し，スケジューラビリティの評価手順を確立する必要がある．

- DDSH のパケット処理能力の評価

マクロな観測により，観測の粗粒度によってはパケットを停止してもしばらく動作する可能性がある．その際，スケジューラは正しく動作できていたとしても，DDSH のパケット処理能力が限界を超える可能性がある．よって，DDSH の多重処理能力は DDSH 本来の性能よりも低く見積もる必要がある．よって，どの程度見積もりを立てることで DDSH の性能を最大限活かせるかを検討する必要がある．

- ソフトウェアスケジューラとの性能比較

マクロな観測と観測点・同期点の追加により，本来のハードウェアスケジューラよりも性能は低下する．しかし，ソフトウェアスケジューラよりも性能が向上していることは評価できれば，一定水準のスケジューリング性能は保証することが可能である．よって，ソフトウェアスケジューラとの性能比較を行い，ハードウェアで処理することでどの程度性能が向上するのかを検討する必要がある．

- リアルタイムデータ駆動型センサノードの性能評価

本研究では，PS の動作確認と PL の DDSH の回路規模・スケジューリング性能の

評価を行った．その際，同期回路と非同期回路間のタイミング同期を行うインタフェース回路を実装していないため，PS と PL 間のインタフェース回路を検討し，リアルタイムデータ駆動型センサノードの実時間にをけるリアルタイム処理性能を確認する必要がある．また，性能評価を行う際は，評価対象の回路が最適化されていないと有用な結果得ることができないため，静的タイミング解析 [11] を行うなど回路の最適化を検討し，評価に適した回路を設計する必要がある．

- 消費電力量の評価

  本研究の提案方式を取り入れることで，リアルタイムデータ駆動型センサノードの目標として掲げている消費電力量の削減を実現できているのか評価する必要がある．

- ハードリアルタイム処理への応用

  マクロなスケジューリングを行う程，観測点と同期点の数は増えて性能は低下するため，自動運転技術などの精密なリアルタイム処理が要求されるシステムへの適用が現段階では難しいのではないかと予想される．そのため，性能低下の要因となっている同期点を MM で待ち合わせることを止め，新たな一時待機の方法を検討してハードリアルタイム性を保証する必要がある．

- マルチコア化に向けたアルゴリズムへの拡張

  本研究では，シングルコアを想定して検討を行ったが，ビッグデータなどの大規模なデータをエッジ端末として処理する場合，マルチコア化が必要になると予想される．そのため，マルチプロセッサシステムに最適なリアルタイムスケジューリングであると言われている Fluid Scheduling[12] へのスケジューリング回路の拡張を検討する必要がある．Fluid Scheduling は，タスクの実行と停止を繰り返すことで流動的なスケジューリングを実現しているため，タスク切り替えによるオーバーヘッドが発生してしまうが，データ駆動型プロセッサであれば，コンテキストスイッチの発生を抑制することが可能であるため，性能向上につながる可能性がある．

謝辞

本研究に際して，丁寧かつ熱心なご指導を受け賜りました岩田誠教授に心から深く感謝申し上げます．貴重なお時間を割いて相談に乗り，熱心なご指導を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．

研究以外にも，学問における知識のご教授，就職活動における客観的な自己への評価など，大変お世話になりました．岩田誠教授の研究室に配属が決まってから今日にいたるまで，研究室での日々は私の価値観を決定づける程の重要なものとなりました．ここに感謝の意を表します．

本研究の論文の副査をお引き受けくださり，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，並びに福本昌弘教授に心より感謝申し上げます．限られた時間の中で的確なご指導の程を賜り，本研究において成すべきことが明確になりました．ここに感謝の意を表します．

研究室の先輩としてご支援，ご協力いただいた，Tamnuwat Valeeprakhon 氏，張震氏，和田悠伸氏，楠田健太氏，汐見興明氏，長野寛司氏，井上聡氏に心より感謝申し上げます．

研究室の同期としてご支援，ご協力いただいた，尾ノ井嶺卓氏，筧拓也氏，古田雄大氏に心より感謝申し上げます．

研究室の後輩としてご支援，ご協力いただいた，髙橋龍一氏，仁野槙人氏，渡邊竜也氏，植元陸氏，坂口白磨氏，松坂拓海氏，計屋和希氏，市ノ木一希氏，石橋璃貴氏，椎葉啓介氏，伊藤雅敏氏，岡村健勝氏，山下拓海氏に心より感謝申し上げます．

最後になりましたが，日頃よりご支援いただいた関係者の皆様に，心より御礼申し上げます．

参考文献

[1] K. Fukuda, H. Shibuta, and M. Iwata, “Priority-Based Hardware Scheduler for Self-Timed Data-Driven Processor,” Proceedings of the 2017 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA’17), pp. 245—251, July 2017.

[2] 岡野秀平, “データ駆動型プロセッサの EDF スケジューリング回路の最適化の検討”, 高知工科大学学士学位論文, 2021.

[3] 総務省, “令和 3 年度 情報通信白書 IoT デバイスの急速な普及,” https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r03/html/nd105220.html, 2022 年 2 月 1 日参照.

[4] K. Suito, R. Ueda, K. Fujii, T. Koga, H. Matsutani, N. Yamasaki, “The Dependable Responsive Multithreaded Processor for Distributed Real-Time System,” IEEE Micro, Vol. 32, No. 6, pp. 52—61, Dec. 2017.

[5] Qing Li, Csroline Yao, “リアルタイム組込み OS 基礎講座,” 株式会社 翔泳社, 東京, 2006.

[6] Giorgio C. Buttazzo, “Hard Real-Time Computing Systems Predicatable Scheduling Algorithms and Applications,” Springer Science & Business Media, NY, 2011.

[7] K. Ramamritham, j.A. Stankovic, and P.-F. Shiah, “Efficient scheduling algorithms for real-time multiprocessor systems,” IEEE TPDS, Vol. 1, No. 2, pp. 184—194, Apr. 1990.

[8] V. Salmani, M. Naghibzadeh, and A.H. Taherinia, “Performance Evaluation of Deadlinebased and Laxity-based Scheduling Algorithms in Real-time Multiprocessor Environments,” The 6th WSEAS International Conference on Systems Theory and Scientific Computation(ISTASC’06), pp. 121–126, Aug. 2006.

[9] H. Terada, S. Miyata, and M. Iwata, “DDMP’s: Self-Timed Super-Pipelined DataDriven Multimedia Processors,” Proceedings of the IEEE, Vol. 87, NO. 2, pp. 282—296, Feb. 1999.

[10] Y. Wada, K. Fukuda, and M. Iwata, “Least Slack Time Hardware Scheduler Based on Self-Timed Data-Driven Processor,” Proceedings of the 2018 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA’18), pp. 249—255, July 2018.

[11] G. Gimenez, A. Cherkaoui, G. Cogniard and L. Fesquet, ”Static Timing Analysis of Asynchronous Bundled-Data Circuits,” 2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), Vienna, Austria, 2018, pp. 110-118, doi: 10.1109/ASYNC.2018.00036.

[12] A. Santo and N. Yamasaki, “Real-Time Execution based on Fluid Scheduling by using IPC Control Scheme,” 2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW) , Matsue, Japan, 2021, pp. 459-463, doi : 10.1109/CANDARW53999.2021.00085.