令和元年度

修士学位論文

LST スケジューラを搭載した 実時間データ駆動プロセッサの FPGA 実現とその評価

FPGA Implementation and Evaluation of Real-Time Data-Driven Processor with Least-Slack-Time Hardware Scheduler

1225131　　和田 悠伸

指導教員　　岩田 誠

2020 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

情報学コース

要 旨

LST スケジューラを搭載した
実時間データ駆動プロセッサの
FPGA 実現とその評価

和田 悠伸

近年の IoT(Internet of Things) デバイスは様々な分野で普及し，それに伴って高機能化・高性能化の需要が高まっている．特にプラント制御や車載等のミッションクリティカル用途ではマルチコア上でのリアルタイム性も要求される．

リアルタイム処理におけるタスクの優先度に基づく動的スケジューリング方式には，デッドラインベースの EDF(Earliest Deadline First) と余裕時間ベースの LST(Least Slack Time) スケジューリングが代表的である．前者は，マルチコア上でのデッドラインミスを予測できない欠点がある．一方後者は，デッドラインまでの余裕時間をタスク実行時間から算出するため，マルチコア上でもデッドラインミスを起こさずスケジューリングが可能である．

本研究では，各種センサ等から到来する様々なデータストリームの多重処理を省電力で実現できるデータ駆動型プロセッサ DDP(Data-Driven Processor) に着目し，そのマルチコア化を前提としたハードウェアスケジューラ回路の構成法を検討する．複数の実時間タスクを多重に優先処理するために，全タスクの最小余裕時間 LST スケジューリングを基準にしたスケジューラ機構を提案し，それを搭載した DDP コアを FPGA (Field-Programmable Gate Array) 上に実装した．評価した結果，システム稼働率を 80%以内で運用すれば実用的に活用できることが確認できた．

キーワード　least slack time，セルフタイム型パイプライン，データ駆動型プロセッサ (DDP)，field-programmable gate array (FPGA)

Abstract

FPGA Implementation and Evaluation of Real-Time Data-Driven Processor with Least-Slack-Time Hardware Scheduler

Yushin Wada

Recently internet of things (IoT) devices are used in various fields, and they are required to operate with higher functionality and performance. Especially, in mission critical systems, it is necessary to perform real-time processing on multicore system.

As for dynamic scheduling algorithms for the real time processing, earliest deadline first (EDF) and least slack time (LST) scheduling are typical ones. Since EDF has no predictability of the deadline miss, it cannot be used for multi-/many-cores. LST algorithm predicts the deadline miss even if there is overload condition in a certain core of them.

In this research, we focuse the self-timed on data-driven processor(DDP) that can realize multiple processing of various data streams coming from sensors with low power consumption. In this paper, scheduler circuit based on LST scheduling that prioritizes multiple real-time tasks is proposed. We implemented DDP core with LST scheduler on an FPGA (Field-Programmable Gate Array). The result confirmed that the system could be used practically if the system operating rate was kept under 80%.

key words least slack time, self-timed pipeline, data-driven processor (DDP), field-programmable gate array (FPGA)

目次

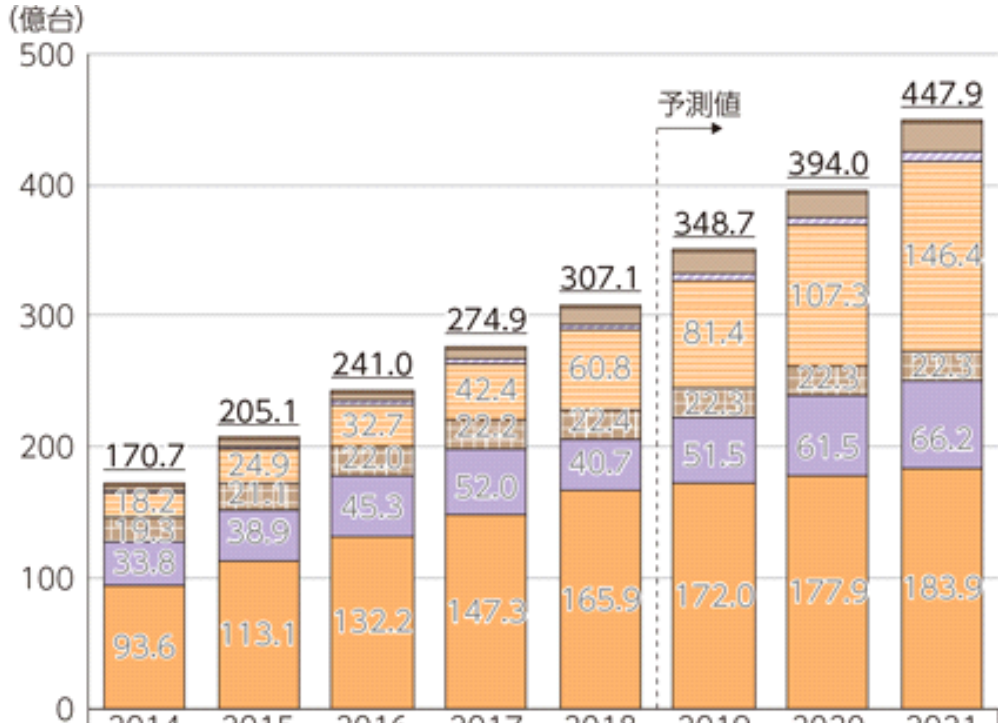
図目次

表目次

第 1 章

序論

近年の IoT(Internet of Things) デバイスは，図 1.1 に示すように，年々増加しており，2021 年には 447 億個にまで増加すると予測されている．



	2014	2015	2016	2017	2018	2019	2020	2021
	170.7	205.1	241.0	274.9	307.1	348.7	394.0	447.9
軍事・宇宙・航空	0.03	0.04	0.04	0.05	0.03	0.04	0.05	0
自動車	3.8	4.7	6.0	7.5	13.5	16.6	19.1	22.0
医療	2.0	2.3	2.8	3.4	3.9	4.8	5.9	7.1
産業用途	18.2	24.9	32.7	42.4	60.8	81.4	107.3	146.4
コンピュータ	19.3	21.1	22.0	22.2	22.4	22.3	22.3	22.3
コンシューマ	33.8	38.9	45.3	52.0	40.7	51.5	61.5	66.2
通信	93.6	113.1	132.2	147.3	165.9	172.0	177.9	183.9

図 1.1　世界の IoT デバイス数の推移及び予測 (文献 [1] より引用)

特にコネクテッドカーの普及による「自動車・輸送機器」やデジタルヘルスケアの市場が拡大している「医療」，スマート工場やスマートシティが拡大する「産業用途」分野の高成長が予測されている．その中で IoT デバイスは高機能化・高性能化に伴いシステムの性能要求が高まっている．これまで CPU のクロック周波数を高めることで処理能力を向上させていたが，消費電力や発熱量が増加する問題が発生している．こうした問題を解決する技術として CPU 内のコアを複数実装することで各コアの処理負荷を緩和させるマルチコア化が進んでいる．また組込みシステムの多くはリアルタイム性を求められる．リアルタイムシステムでは入力を受けてから結果を出力する際に出力内容の正しさだけでなく時間制約 (デッドライン時刻) を守る必要がある [4]．時間制約はその厳密性から，ハードリアルタイムタスクとソフトリアルタイムタスクに分類される．ハードリアルタイムはデッドラインミスを起こした場合，システムに致命的な影響を与えるタスクである．ソフトリアルタイムタスクはデッドラインミスを起こした場合，システムの性能が低下する可能性があるタスクである．これらのタスクの時間制約を守るために，タスクに優先度を設定し，優先度に基づいたタスクスケジューリングが必要となる．これにより組込みシステムの多くはリアルタイム処理とマルチコアの両立が求められている．

リアルタイム処理を実現するするには，アプリケーションに含まれるすべてのタスクの時間制約を保証するために，各タスクの優先度を考慮してタスクの実行を割り当てるかをスケジューリングする必要がある．

リアルタイムスケジューリングのアルゴリズムには，ROTS で用いられるラウンドロビン方式や静的優先度方式，動的優先度方式がある．ラウンドロビン方式は実行する時間単位を決め，順番に実行される方式である．時間単位を短くすればするほど多くのタスクが均一に実行されるがタスクの切り替えによるオーバーヘッドが増加する問題がある．静的優先度方式では，各タスクの優先度はタスク実行前にあらかじめ割り当てられ，実行中に優先度が変化することはない．静的優先度方式は周期の短いタスクから順に高い優先度を割り当てる RM(Rate Monotonic) が代表的である．RM はあらかじめ優先度を決めるため前提条件として周期タスクでなければならない．一般的に，IoT デバイスにおけるリアルタイムタスク

は，周期的，または非周期的に実行される．そのため静的優先度方式では，タスクの優先度は実行時に更新されないため，非周期タスクを処理できない．そのため，動的優先度方式を採用する必要がある．ソフトウェアによるアプローチとして，リアルタイムオペレーティングシステム (RTOS) を用いることが一般的である．RTOS はスケジューリングや資源管理等，リアルタイムシステムに必要な機能が特化している．本来のタスク実行の稼働率を低下させないためにスケジューリング時に発生するオーバーヘッドを削減する必要がある．しかし，ソフトウェアによるスケジューリングでは計算上のオーバーヘッドが大きく，スケジューリングアルゴリズムを使用するため，平均稼働率が低下する問題ある．そのためソフトウェアを用いた動的優先度方式によるリアルタイム処理の実現は困難である．そこでスケジューラをハードウェアで実装し，スケジューリングを専用ハードウェアで行うことでソフトウェアのオーバーヘッドを削減し，動的優先度方式のスケジューリングが可能となる．

動的優先度方式では，各タスクの優先度はタスク実行中に動的に変化する．動的スケジューリング方式として Earliest Deadline First(EDF) スケジューリングと Least Slack Time(LST) スケジューリングが代表的である．EDF スケジューリングは絶対デッドライン時刻に基づいて優先度の設定を行う動的スケジューリングアルゴリズムである．デッドラインに近いタスクに高い優先度を与え，デッドラインに遠いタスクに低い優先度を与える．LST スケジューリングは Slack Time(余裕時間) に基づいて優先度の設定を行う動的スケジューリングアルゴリズムである [5]．余裕時間は絶対デッドライン時刻，実行時間の差分で定義される．余裕時間は時間経過と主に減少し，余裕時間が短いほど高い優先度を設定される．EDF，LST スケジューリングはシングルコアでのタスクスケジューリングにおいて最適なスケジューリングである [6]．

しかし，マルチコア上でタスクを実行する場合に EDF と LST を比較すると，EDF スケジューリングは実行時間を加味していないため各コアにタスクを割り当てる際，デッドラインミスを予測できない問題がある．一方，LST スケジューリングはデッドラインまでの余裕時間を実行時間から算出しているため，デッドラインミスを起こさずスケジューリングが可能である．しかし，LST スケジューリングはシングルコア・マルチコアのスケジューリング

において余裕時間が近いタスクが複数存在する場合，一方の余裕時間がもう一方の余裕時間より短くなり優先度が変化する．これを繰り返すことによってタスクを頻繁に切り替える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恐れがある [7].

上記の問題を解決するために，データ駆動計算モデルに着目する．データ駆動計算モデルは演算に必要なパケットがそろうことで実行される．データに依存関係がない演算は並列に実行できる．そのためデータ駆動計算モデルのハードウェア実装である DDP(Data-Driven Processor) は多重処理が可能であり，コンテキストスイッチのオーバーヘッドなしに複数タスクを多重に実行できる．

複数タスクの多重処理によりスラッシングを緩和できるデータ駆動型プロセッサ DDP に着目し，マルチコア化に向けて DDP シングルコアに搭載可能な LST ハードウェアスケジューラを提案されている [8][9]．しかし，提案されている DDP コアはシミュレーションでの性能評価のみで，実時間の性能評価は行われていない．LST スケジューラを搭載した DDP の実用化のためには現実的な環境で評価する必要がある．

実用化のための環境として負荷分散を行うエッジコンピューティングに用いられる IoT デバイス FPGA(Field-Programmable Gate Array) に着目した．FPGA はユーザのシステム用途に合わせて自由に変更が可能なため，様々な応用に適している．

本研究では，DDP のマルチコア化を前提とし，LST スケジューリングを基準にしたスケジューラ機構を搭載した DDP コアを FPGA 上に実装して回路規模と実時間性能を評価する．

これ以降の本論文において，第 2 章では動的スケジューリング方式をまとめ，DDP のアーキテクチャと DDP におけるスケジューリング制御を述べ，DDP に LST スケジューラを搭載する際に必要な機構を述べる．

第 3 章では，スケジューリング制御を搭載した DDP に加え，余裕時間計算回路とタスクキュー回路での余裕時間再計算回路の構成を述べる．また，余裕時間の計算に必要なタスクの実行時間の定義と余裕時間の更新に要する回路を述べ，回路規模の最小化するために保持する情報と回路内に記憶する情報の最適化を図る．

第 4 章では，LST スケジューラを搭載した DDP を FPGA 設計ツール Quartus Prime 18.0 を用いて実装し，回路評価とリアルタイム性能の性能評価を行った結果を考察する．

第 5 章では，本研究の評価をまとめ，今後の課題を述べ，本論文を総括する．

第 2 章

データ駆動型プロセッサにおけるリアルタイム処理

2.1 緒言

本章では，リアルタイムシステムにおいてスケジューリングされるタスクが持つパラメータとまとめ，動的優先度方式の代表的なスケジューリングアルゴリズムである EDF と LST の 2 つを比較する．その後，セルフタイム型パイプライン (STP:Self-Timed Pipeline) で構成された DDP のアーキテクチャ，先行研究で提案されている，DDP におけるスケジューリング制御を述べ，DDP に LST スケジューリング機構を搭載する DDP の方針を述べる．

2.2 タスクの定義

タスクは主に図 2.1 に示すパラメータを持つ．

タスクは実行要求時刻に起動すると，実行可能状態になる．その後，スケジューリングされ，実行される．各タスクは絶対デッドライン時刻までに実行を完了する必要がある．絶対デッドライン時刻は実行要求時刻と相対デッドライン時間の和で求められる．各タスクは絶対デッドライン時刻を超えて実行された場合，デッドラインミスとなる．また絶対デッドライン時刻までにタスク実行の開始を遅らせることが可能な最大遅延時間を余裕時間と呼ぶ．各タスクは余裕時間がゼロになるまでタスクの実行開始を遅らせることができ，実行要求時

2.2 タスクの定義

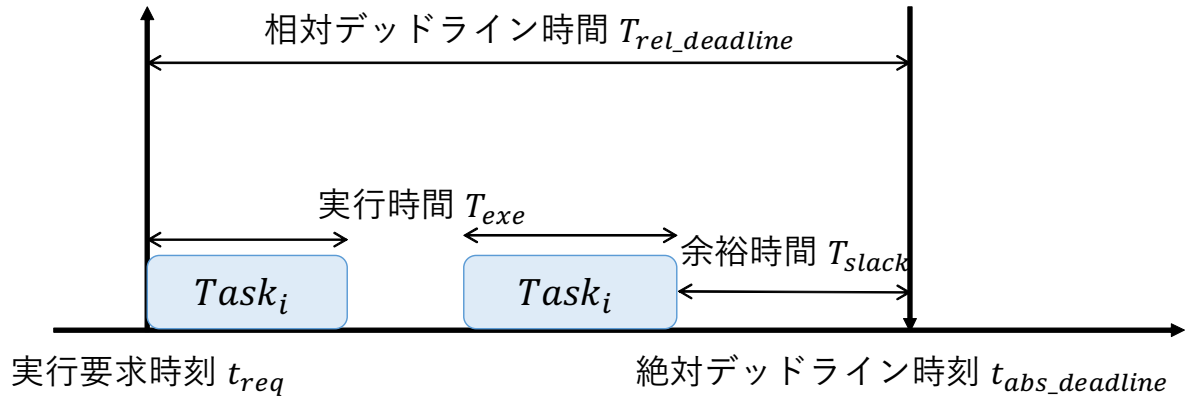


図 2.1　タスクの各パラメータ

刻における初期余裕時間は

$$T_{slack} = (t_{abs\_deadline} - t_{req}) - T_{exe} \tag{2.1}$$

ある時刻 t における余裕時間は

$$T_{slack} = (t_{abs\_deadline} - t) - T_{remain\_exe} \tag{2.2}$$

で求められる．T_{remain_exe} は，時刻 t における残り実行時間である．

リアルタイム処理を実現するためには，各タスクがデッドラインミスを起こさないように，タスクの実行をスケジューリングする必要がある．リアルタイム処理性能 (スケジューリング性能) には以下の指標がある．

- システム稼働率

  システムに含まれるコアの稼働率である．

- スケジューリング成功率

  要求されたタスクセット/タスクのうち，デッドラインミスを起こすことなく実行を完了したタスクセット/タスクの割合である．

- 実行時間

  実行要求時刻から，実行完了までに実際にかかる時間である．

- タスク実行時間のジッタ

  最小実行時間と最大実行時間の差である．

2.3　動的スケジューリング方式の比較

代表的な動的スケジューリング方式として EDF スケジューリングと LST スケジューリングが提案されている [2]．EDF スケジューリングとはデッドライン時刻の短いタスクに高い優先度を付与するスケジューリング方式である．LST スケジューリングとは余裕時間の短いタスクに高い優先度を付与するスケジューリング方式である．余裕時間は図 2.1 で示したように絶対デッドライン時刻，実行時間の差分で定義される．時間が経過すると実行されていないタスクの余裕時間が減少していきタスク間の優先順位が変化する．

理論的には EDF スケジューリング，LST スケジューリングともにシングルコア上でのプロセッサ使用率が 100%まで機能するが，マルチコア上でのスケジューリングにおいて EDF スケジューリングは実行時間を加味しないため，デッドラインミスを予測できない問題がある．例えば，表 2.1 のタスクセットの場合デッドラインミスが発生する．EDF スケジューリングの場合，各コアにデッドラインが近いタスクを割り当て，その後実行時間が長いタスクを割り当てるため図 2.2 のようにタスク T1 と T2 がスケジューリングされ，T3 の実行時にデッドラインミスが発生する．一方，LST スケジューリングでは余裕時間の算出に実行時間を加味しているため，図 2.2 のように余裕時間の短い Task3 をコアに割り当てた後 Task1 と Task2 を別のコアに割り当てることでデッドラインミスを起こさずタスクスケジューリングが可能となる．

しかし，LST スケジューリングはシングルコア・マルチコアどちらにおいてもスケジューリングを行う際，余裕時間が近いタスクが複数存在する場合，図 2.3 のように複数タスクの実行が交互に切り替わるようにスケジューリングしてしまう．このように複数タスクのタスクスイッチが頻繁に行われる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増大する問題がある [7][10]．

そこで上記の問題を解決するためにタスクスイッチによるオーバーヘッドなしに，多重処理可能であるデータ駆動型プロセッサ DDP 着目したスケジューリング制御について述べる．

表 2.1　パケットの詳細情報

	実行時間	デッドライン時刻
Task1(T1)	2	4
Task2(T2)	2	4
Task3(T3)	7	8

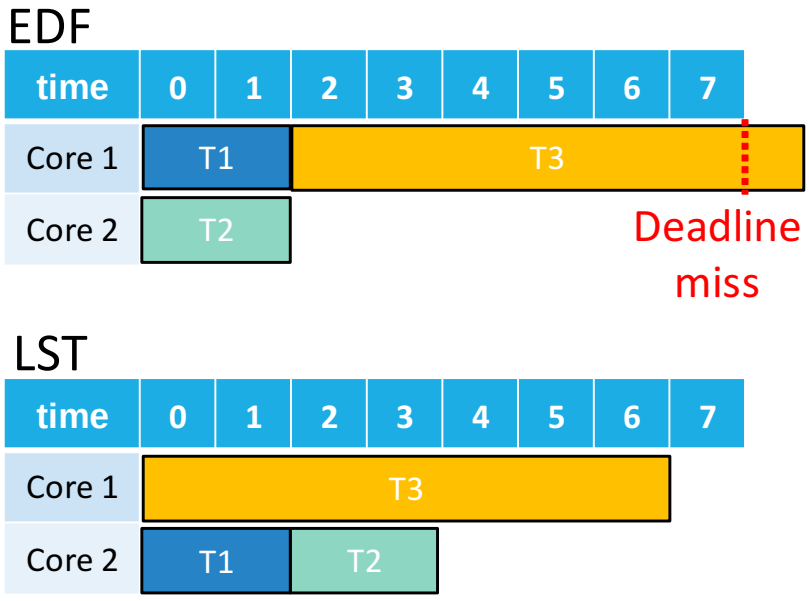


図 2.2　マルチコア上のスケジューリング

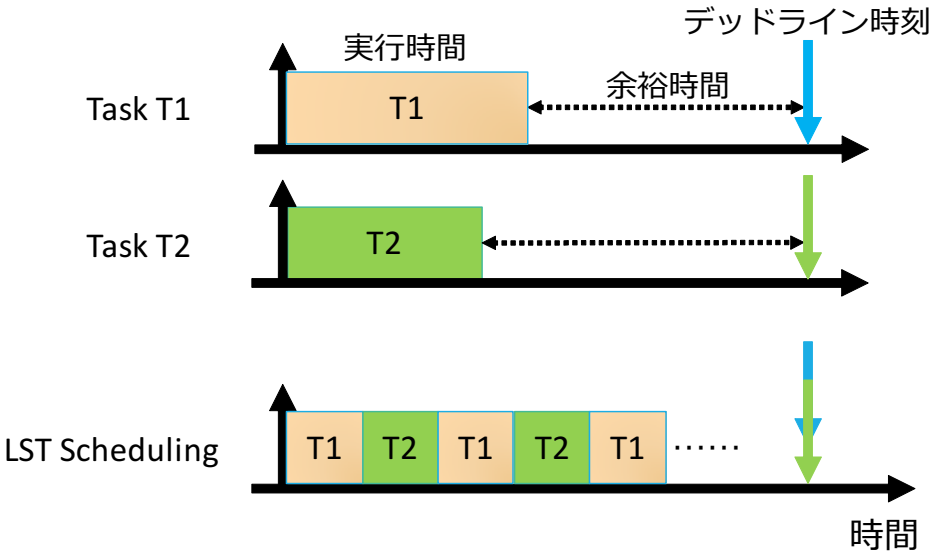


図 2.3　LST スケジューリングの例

2.4　データ駆動型プロセッサアーキテクチャ

本研究で対象とする DDP はセルフタイム型パイプラインで構成されている．STP は図 2.4 のように複数の転送制御回路 (Coincidence flip-flop:C 素子) でパイプラインステージ内にデータが到着した時に，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 信号) とデータ転送許可信号 (Ack 信号) でやり取りすることでデータを処理する．有効なデータを持たないパイプラインステージは dynamic 電力を消費しないため，データ転送時にのみ電力を消費する省電力機能を有している．また，タスクが多重に実行された場合パケット転送要求信号を送信するがデータ転送許可信号で許可されるまでデータラッチ解放信号が送られない．これにより，STP は多重実行時における負荷変動を自律的に緩衝特長を有している．

STP で構成された DDP はパケットレベルでデータ駆動計算モデルの動作を忠実に実現できる．DDP の構成を図 2.5 に示す．DDP を構成する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を周回してきたパケットの合流を調停し，CST に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使用する定数が格納されている．定数命令を保持するパケットである場合は，パケットが持つ情報と対応する定数を定数メモリから読み出し，パケットに付加する．定数命令を保持するパケットでない場合は，そのまま MM へ出力される．
- パケット待ち合わせ機構 (Matching Memory Content Addressable Memory：MM-CAM)

  CST からパケットを受け取り，対応するパケットと待ち合わせをするために，一時的にレジスタに保持され，MMRAM にデータを格納する．対応するパケットが到着すると，対応する 2 つのパケットの情報からデータを MMRAM から読み出す．

- パケットデータ保持機構 (Matching Memory Random Access Memory： MMRAM)
  C 素子の CP 信号と同期して，対となるパケットと結合する情報の書き込みと読み出しを行う．書き込み信号が有効な場合は，指定アドレスに書き込みを行う．読み出しは指定アドレスから読み出す．
- 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もとに，演算処理を行う．演算結果をパケットに書き込み，DMEM へ出力する．
- データメモリ機構 (Data Memory：DMEM)
  ALU からパケットが到着し，ロード命令，ストア命令が実行される場合，データメモリに対して読み込み，書き込みを行う．読みだしたデータはパケットに付加し，PS へ出力する．
- パケット複製機構 (Copy： COPY) パケットが保持している CP の値をもとにパケットを複製するか判断し，複製が必要な場合はパケットを複製する．
- 命令フェッチ機構 (Program Storage：PS)
  PS ではメモリに次の宛先ノード番号と実行する命令コードが格納されており，パケットが持つ情報を書き換る．その後，B にパケットを出力する
- 分岐機構 (Branch Unit：B)
  パケットが保持する情報から，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，転送する．

DDP は外部からパケットが入力されると，DDP 内部で周回するように実行される．その後分岐機構で外部に出力されるか，そのまま DDP 内部を周回して命令を実行するか判断する．上記の機構で構成された DDP に必要な基本パケット情報を表 2.2 に示す．これらの情報をもとに各機構で処理する．

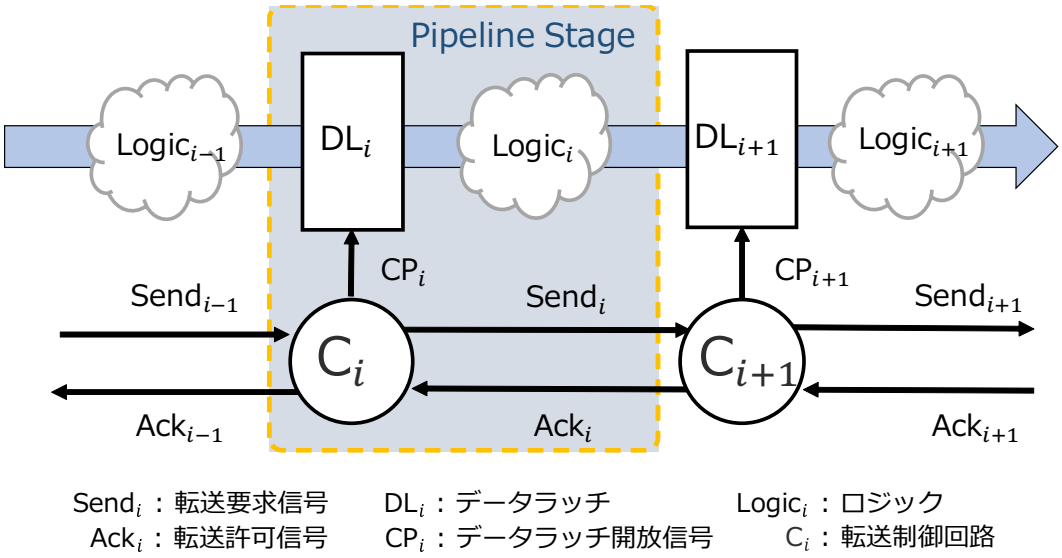


図 2.4　STP の構成

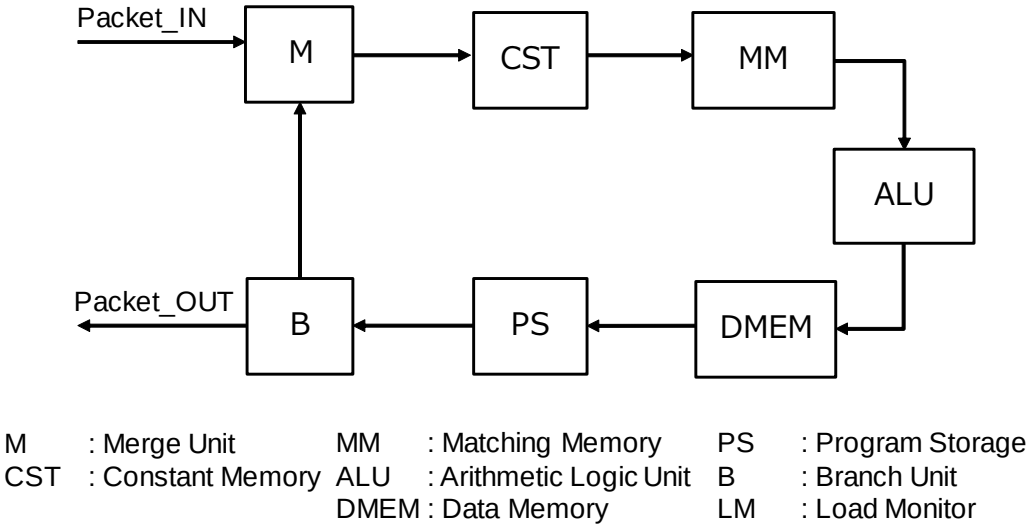


図 2.5　DDP のステージ構成

2.5　DDP におけるスケジューリング制御

DDP は多重処理が可能なため処理負荷の限界を超えない限り，スケジューリングは原理的には不要である．しかしその限界を超えるタスク要求が発生する場合はスケジューリングが必要となる．本来，DDP はスケジューリング機構が搭載されていない．DDP においてスケジューリング制御を行うには DDP 内の処理負荷を検知する機構で実現できる．しか

表 2.2　パケットの詳細情報

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	7bit
dest	destination	パケットの宛先	8bit
LR	left or rignt	パケットの左右の識別	1bit
CP	copy flag	コピーの有無	1bit
OPC	operation code	命令コード	4bit
C	carry flag	桁上りの有無	1bit
Z	zero flag	ゼロフラグ	1bit
data	data	演算データ	16bit

し，LST スケジューラを搭載するにあたって STP 回路による DDP の多重処理能力，負荷変動緩衝能力，省電力性能の特長を損なわず実現する必要がる．スケジューラの実装において集中型スケジューラと分散型スケジューラの方式がある．集中型の場合 DDP の各機構からタスクの状態を収集，分析しスケジューリングする．しかし，各機構からスケジューラに情報をフィードバックする必要あるため，各機構は常に稼働しなければならない．集中型スケジューラでは STP 回路による省電力性能が損なわれる．そのため，集中型スケジューラよりも分散型スケジューラで情報を管理するほうが望ましいと考えられる．また，STP 回路による勘定パイプライン構成に組み込み可能な分散型スケジューラで実装する．

DDP はデータ駆動原理に基づいており，タスク内の命令によって DDP 内でのパケット量が変化する．先行研究 [9] では単項演算または即値演算を縦続接続させた直線状プログラムを対象としていた．そのため，1 タスクに対してパケットは 1 つで実行され，パケット量の変化状況を処理負荷で観測することで DDP 内の処理負荷がわかる．しかし，単項演算または即値演算を縦続接続させた直線状プログラムのみでは実用的なタスクでないため，より二項演算やコピー命令を含む並列処理プログラムのような実用的なタスクの処理構造を対象

にする必要がある．1 タスクに対して複数パケットで実行される場合，パケット量とタスク数は一致しないため，処理タスク数を観測することで DDP 内の処理負荷を観測することが可能である．

また，DDP では対応するパケットが到着するまでは Matching Memory で到着を待つためタスクの実行を中断する場合には一部のパケットを停止させることでタスク全体の実行を中断できる．そのため，多重処理可能タスク数を超える場合優先度の低いタスクに属するパケットをタスクキュー機構によってキューイングすることでタスクの停止を行う．

2.5.1 負荷観測機構 (Load Monitor:LM)

処理タスクを観測することでスケジューリングを行うか判断する．処理タスクは DDP 内に入力されたタスク数によって増減する．DDP の処理負荷と多重処理可能タスク数を比較してタスクキュー機構の処理が変わる．

実行するタスクのプログラムよってパケット量が異なる．具体的には単項演算または即値演算を縦列接続した直線状プログラムと二項演算，Copy 命令による複数パケットを用いるプログラムである．これらのプログラムにおけるパケットとタスクの関係を以下に示す．

- 即値演算を縦続接続された直線状プログラム
  - 1 タスクに対して 1 パケット
  - パケットをタスクとして扱う
- 二項演算，Copy 命令による複数パケットを用いるプログラム
  - 1 タスクに対して複数パケット
  - パケット量が増減してもタスク数に影響しない

このため即値演算や単項演算のみの直線状プログラムは，パケット量がタスク数と同じためパケット量を見ることで処理負荷を観測し，複数パケットを用いるプログラムでは，タスク数で処理負荷を観測する．

- タスクが増加する要因
  - 外部から入力される
- タスクが減少する要因
  - 外部に出力される

2.5.2 タスクキュー機構 (Task Queue Unit:TQ)

多重処理可能タスク数を超える場合，パケットをタスクキュー機構にキューイングすることでパケットの停止を行う．直線状プログラムの場合，1 タスク対して 1 パケットのためそのタスクに属するパケットをキューイングすることでタスクを中断できる．複数パケットを用いるプログラムでは 1 タスクに対してパケットは複数存在するがタスクの実行を中断する場合には一部のパケットを停止させることでタスク全体の実行を中断できる．

これにより，外部から入力されたパケットがタスクキュー機構に到着し，多重処理可能タスク数を超える場合，一部のパケットをタスクキュー機構に退避させることで一時的なパケットの停止を行う．外部出力命令のパケットがタスクキュー機構に到着し，多重処理可能タスク数を超えない場合はそのパケットはタスクキュー機構を通過せず削除される．多重処理可能タスク数を超える場合 TQ に到着したパケットは処理負荷に関わらず一度キューイングされる．タスクキュー機構に退避しているパケットがある場合は退避しているパケットと到着したパケットから優先度の高いパケットを選択して出力する．

2.5.3 余裕時間情報の保持

DDP のスケジューリング制御は処理負荷を観測する機構と各タスクの優先度に基づいてタスクを中断・再開する機構を DDP 内に搭載することで実現されている [3][9]．図 2.5 に示す DDP の構成に加え，スケジューリング制御を搭載した DDP 構成を図 2.6 に示す．LST スケジューリングでスケジューリングを行うには，タスク実行要求時に余裕時間を求める．余裕時間の計算にはデッドライン時刻と実行時間を用いる．各タスクに属するパケットに対

2.5　DDP におけるスケジューリング制御

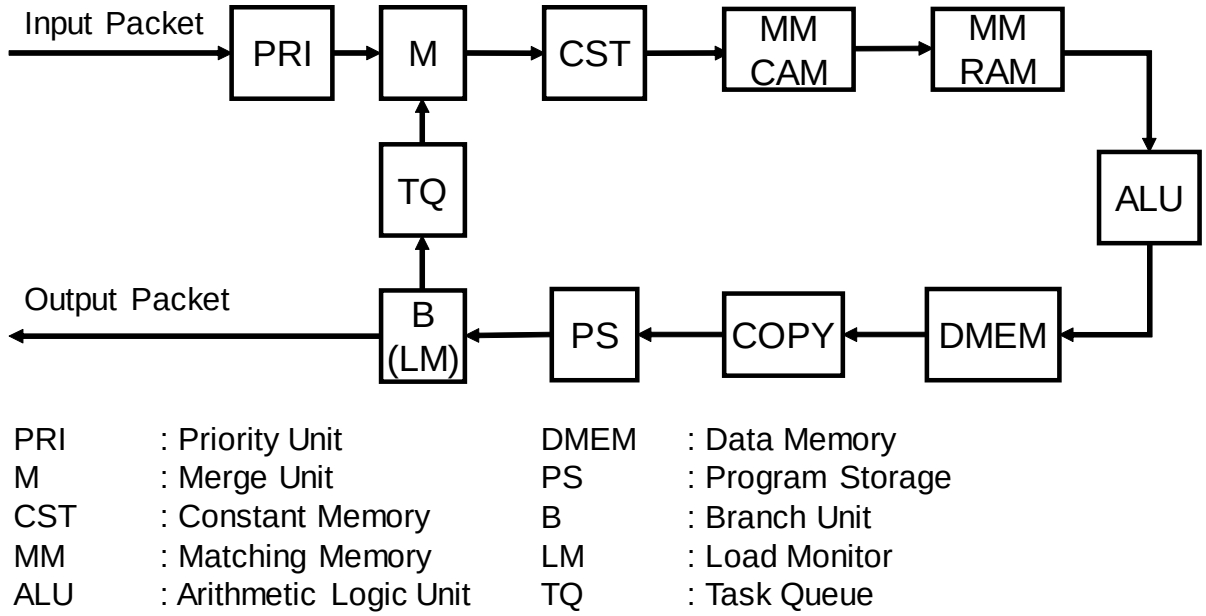


図 2.6　スケジューリング制御を搭載した DDP のステージ構成

応するデッドライン時刻と実行時間から余裕時間を計算する機構が必要である．DDP におけるタスク実行要求は DDP に入力したときであるため DDP の入力前に余裕時間を計算する．また，タスクが実行している間タスクキューにキューイングしているタスクの余裕時間は減少する．このため，DDP 内部に余裕時間を更新する機構が必要である．DDP 内のタスクキュー機構でキューイングしているタスクの余裕時間の更新を行うため，タスクキュー機構内部でタスクに属するパケットに余裕時間と更新に必要な情報を用いて余裕時間の更新を行う．

余裕時間の情報を保持する方式として以下の方式がある．

- パケット

  余裕時間の情報をパケットに付与した状態で DDP 内を周回する．

- メモリ

  余裕時間の情報をメモリに格納し，TQ 機構にキューイングする際にメモリから余裕時間を参照する．

先行研究 [9] は余裕時間の情報はパケットに付与してタスク実行している．しかし，これ

	ステップ1	ステップ2	ステップ3
入力／出力	1／1	1／1	複数／複数
命令	単項演算 または即値演算	＋ 二項演算	＋ 分岐命令
copy／巡回	なし／なし	あり／なし	あり／あり

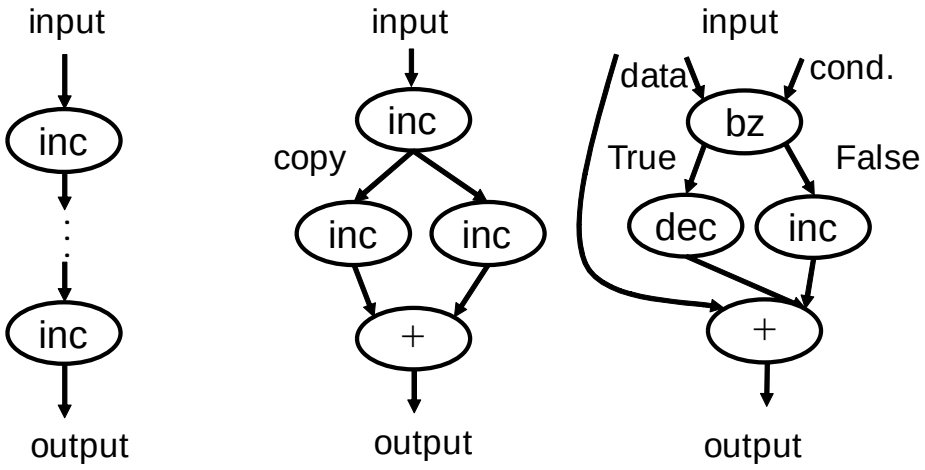


図 2.7　実用化に向けたタスク一般化

まで DDP は図 2.7 のステップ 1 を対象とした単項または即値演算のみを実行するタスクしか対応していない．実用化するにはステップ 2，ステップ 3 に対応した DDP にする必要がある．余裕時間情報をパケットに付与する方式では Copy 命令によるパケットの複製が行われた際，1 つのタスクに異なる 2 つの余裕時間情報を持つことになる．そこで本研究では，メモリに格納する方式を採用した．これにより，パケットが複製された場合でも 1 タスクに 1 つの余裕時間情報を扱うことが可能である．また，余裕時間情報は TQ 機構以外では使用されない．パケット方式の場合余裕時間情報を使用しない機構にも余裕時間情報を送る必要があったため，回路に無駄があった．メモリ方式を採用することでメモリと TQ 機構のみ余裕時間情報を扱うため無駄な回路の削減となる．

LST スケジューリングを適用するために上記の内容を踏まえ以下の機構を実装する．

- 初期優先度機構 (Priority Unit：Pri)

  外部からパケットが入力された時，余裕時間に必要な情報を用いて余裕時間を計算する

機構.

- 余裕時間更新機構 (Slack Time Update)

  DDP 内でパケットが中断・再開するタスクキュー機構に到着するたびに実行されていないタスクに属するパケットの余裕時間を更新する機構.

- 余裕時間メモリ (Slack Time Memory : STM)

  DDP に入力されたパケットのタスクの余裕時間を格納する．また，DDP 内で周回したパケットの場合メモリから余裕時間をパケットに付与し，TQ 機構にパケットを出力する.

上記の機構を追加することで LST スケジューラを搭載した DDP が実現できる．余裕時間更新機構はタスクキュー機構内で実装する.

2.6 結言

LST スケジューリングアルゴリズムと STP で構成された DDP のアーキテクチャ及びスケジューリング制御を述べ，LST スケジューリング機構を搭載するための必要な要素を述べた．DDP における LST スケジューリングは余裕時間の計算と再計算を行う機構を追加することで実現できる．また余裕時間情報をメモリに格納し必要な機構のみパケットに付与することで回路の増加を防ぐ.

本研究の DDP は図 2.1 のステップ 2 までのタスクに対応している．今後ステップ 3 のタスクに対応するためには，複数の入出力の場合でもタスクに属するパケットが全て入出力した場合に処理負荷の増減を行う機構に変更する必要がある.

次章では本章で述べた要素を踏まえた上で LST スケジューリングを搭載した DDP の構成と余裕時間計算とタスクキュー機構の構成について述べる.

第 3 章

LST スケジューラを搭載した DDP

3.1 緒言

本章では前章で述べた方針を踏まえ，実行時間の定義と初期優先度回路，タスクキュー回路，スラックタイムメモリの構成を述べる．タスクキュー回路内の余裕時間の更新方法と Queue の制御方法について述べる．

3.2 全体構成

LST スケジューラを搭載した DDP の全体構成を図 3.1 に DDP のパケットが保持する情報を表 3.1 示す．

3.3 実行時間の定義

DDP での LST スケジューリングではタスクの実行が要求されると余裕時間の計算を行う．余裕時間の計算には相対デッドライン時間及び実行時間を定義する必要がある．DDP におけるタスクの実行時間は各ステージの C 素子によってパケットが転送されるまでにかかる時間となる．ステージ間でのパケットの転送の動きを図 3.2 に示す．パケットが混雑していない場合，実行時間は $T_f \times$ ステージ数 となる．タスクの 1 命令の実行時間は DDP 内を 1 周した時間となる．T_f は図 3.2 に示すように C 素子から出力される CP 信号から次のステージの C 素子から出力される CP 信号までの時間となる．パケットが混雑している場合，実行時間は $(T_f + T_r) \times$ ステージ数 となる．本研究ではパケットが混雑しないように

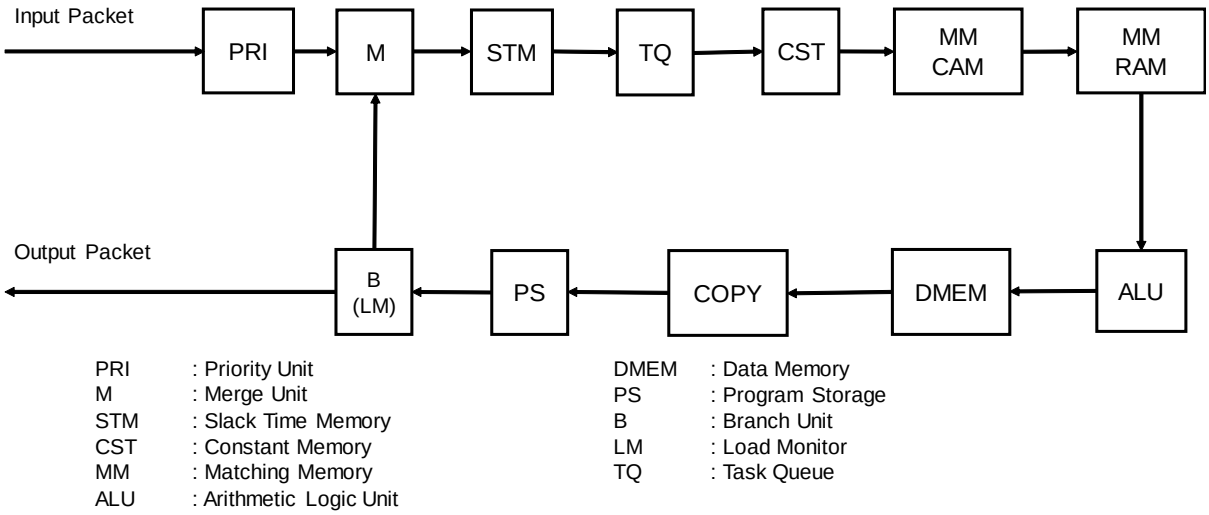


図 3.1　LST スケジュラーを搭載した DDP の構成

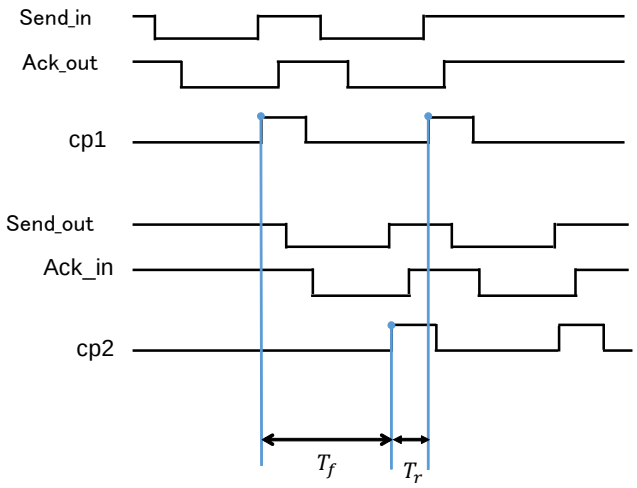


図 3.2　ステージ間でパケットが転送されるまでにかかる時間

DDP 多重度を 2 タスクに設定したため，実行時間は $T_f \times$ ステージ数 として考慮する．

3.4　タスク優先度クラスの定義

本研究での優先度クラスを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) に分類する．ハードリアルタイムタスクはデッドライン

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
eqt	enqueue time	TQ にキューイングした時刻	20bit
st	slack time	余裕時間	20bit
lap	lap	入力, 周回パケットの識別	1bit
del	delete	TQ の削除フラグ	1bit
in flg	input flag	TQ の入力フラグ	1bit
pri	priority	タスク優先度クラス	2bit
color	Color	パケットの識別情報	5bit
gen	Generation	パケットの世代 (順番)	5bit
dest	Destination	パケットの宛先	13bit
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	32bit

までに必ず処理を完了しなければならないタスクで，デッドラインを超えると処理結果の価値はなくなる．ソフトリアルタイムタスクはデッドラインまでに処理を可能な限り完了する必要があり，デッドラインを超えると処理結果の価値は時間経過とともに減少する．時間制約のないタスクはデッドラインの無いタスクである．優先度はハードリアルタイムタスクが最も優先度が高く時間制約のないタスクが最も優先度が低い．

各タスクに優先度クラスを設定するために，優先度クラス (priority:pri) を用いる．優先度クラスは各タスクごとに設定し，優先度の高いタスクの実行が優先される．

3.5 Priority Unit

初期優先度機構 PRI の詳細構成を図 3.4 に示す．タスク実行が要求され，図 3.3 の入力パケットが余裕時間計算回路に入力される．パケットが入力されると，パケットが保持するパケットの識別情報 color をアドレス値として TASK MEMORY から優先度クラス pri と相対デッドライン時間 (Relative Deadline:rd)，実行時間 (Execution Time:et) を出力する．C 素子からデータラッチ開放信号 CP 信号が送られると TASK MEMORY から出力した相対デッドライン時間と実行時間をもとに SLACK TIME CALCULATOR で余裕時間 (Slack Time:st) を算出する．計算式は $st = rd - et$ となる．算出された余裕時間 st を MERGE で優先度クラス pri と余裕時間 st を入力パケットに保持して次のステージに出力する．また一度タスクキュー機構 TQ にキューイングするため，入力フラグ in flg，削除フラグ del, 周回フラグア lap を付与する出力パケットのフォーマットを図 3.5 に示す．

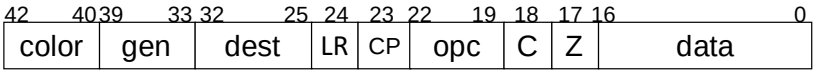


図 3.3　入力パケットフォーマット

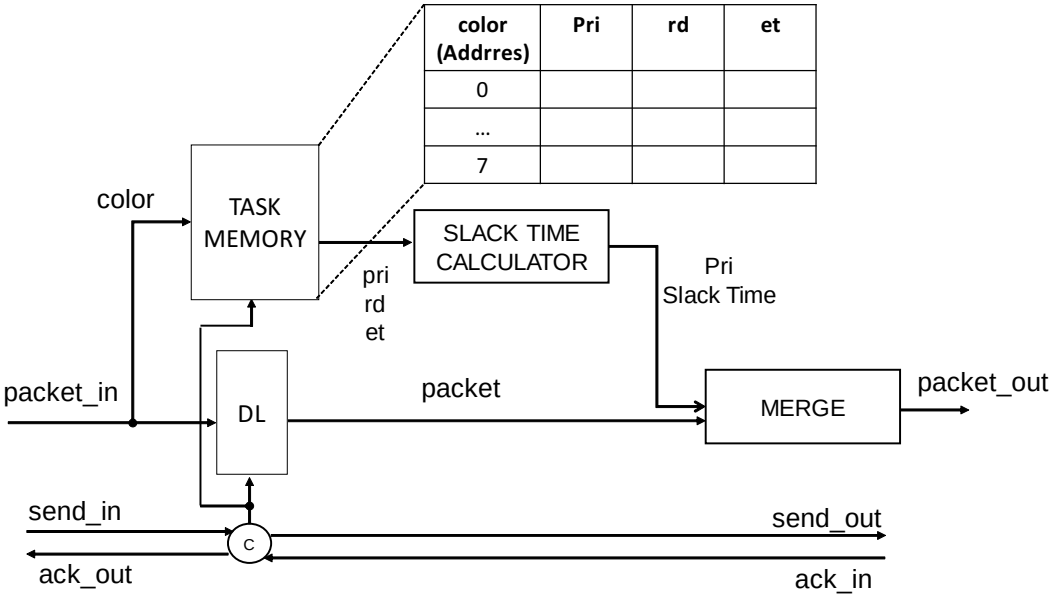


図 3.4　PRI 機構の構成

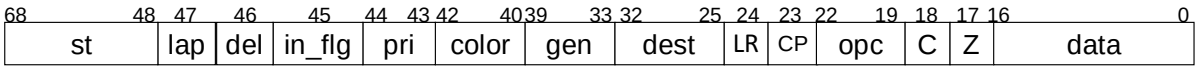


図 3.5 PRI 機構のパケットフォーマット

3.6 Sack Time Memory，Task Queue 回路

STM 及び TQ 回路の構成を図 3.6 に示す．タスクキュー回路は主に Queue 及び削除付き転送制御回路 (CE 素子) から構成される．DDP の処理負荷を超える場合，パケットを Queue に退避することで一時的なタスクの停止を実現する．Queue は優先度クラスを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) に分類し，それぞれに対応した Queue(HR_PQ，SR_PQ，N_Q) を用意した．

STM にパケットが到着した場合 DDP に入力されたパケットか周回したパケットによって処理が異なる．

- 入力パケット

  入力したパケットの場合，Slack Time Memory に PRI 機構で計算された余裕時間を格納する．その後余裕時間を付与したまま TQ 回路に出力される．
- 周回パケット

  周回したパケットの場合，識別情報 (color,generation) をもとに自タスクの余裕時間を読み出しパケットに付与する．その後 TQ 回路に出力される．

また，入力，周回パケット問わずパケットが STM に到着した際，TQ 回路からキューイングされたパケットの余裕時間をフィードバック情報 (color,generation,slack time) としてフィードバックする．

タスクキュー回路では，入力パケットが到着するとパケットが保持している優先度クラス pri から対応した Queue(HR_PQ，SR_PQ，N_Q) に振り分けられる．Priority Queue の構成を図 3.7 に示す．入力パケットは対応した Priority Queue 内に入力され，ST UPDATE で Priority Queue に到着した現時刻 time をパケットに保持し，ENTRY にキューイング

される．ENABLER によって複数の ENTRY に同じパケットがキューイングされないようにパケットがキューイングされていない ENTRY1 つに書き込み許可を与え，それ以外の ENTRY は書き込み許可を与えないよう制御される．各 ENTRY にキューイングされているパケットは余裕時間の一番短いパケットがソートされ出力され，DDP 内を巡回する．ソートは 4 つの ENTRY ごとに分けて挿入ソートでソートする．ソートされたパケットは各 ENTRY にフィードバックされ，図 3.7 の ENTRY6 または ENTRY3 に最も余裕時間の短いパケットがキューイングされる．lap 情報から入力パケットと判断された場合，一度処理負荷観測機構を通過する必要があるため，余裕時間を無視して最優先度でソートされる．

余裕時間更新機構 ST UPDATE ではキューイングされている各パケットの余裕時間の更新を行う．入力パケットとキューイングされているパケットの処理方法を以下に示す．

- 入力パケット
  1. TIMER から入力パケットが TQ 回路に到着した時刻 (現時刻) を受け取る．
  2. TQ 回路に到着した時刻 (現時刻) をパケットに保持する．
  3. ENTRY にキューイングする．
- キューイングしているパケット
  1. TIMER から入力パケットが TQ 回路に到着した時刻 (現時刻) を受け取る．
  2. 現時刻をキューイングしている各パケットの TQ 回路に到着した時刻で差し引いてキューイングしていた待ち時間を算出する．
  3. 各パケットの余裕時間を待ち時間で差し引いて余裕時間を更新する
  4. TQ 回路に到着した時刻を現時刻に更新して各パケットに保持する．
  5. ENTRY にキューイングする．

これにより余裕時間の更新に必要なデータを TQ 回路に到着した時刻と余裕時間で計算することでパケットに付与するデータ量を抑えることができる．

また TQ 回路は Queue を制御する複数のコントローラで構成されている．

- CE_CTRL

負荷観測機構からの情報をもとにパケットをキューイングするかを制御する．キューイングする場合は，CE 素子に削除制御信号を送ることで次のステージにパケットが出力されないようにする．

- Q_SEL

  各 Priority Queue から出力されるパケットの中で最も優先度が高いクラスのパケットを出力するように Q_MUX を制御する．

- Q_CTRL

  各 Priority Queue から出力されるパケットが次のステージに出力されたかどうかを観測するコントローラである．パケットが次のステージに出力された場合，そのパケットを Priority Queue 内の ENTRY から削除する必要がある．Priority Queue にパケットの削除信号 paket_del_sig を送り，削除対象のパケットがキューイングされている Queue の出力されたパケットがキューイングされている ENTRY7 を初期値に上書きすることでパケットを削除する．

次に Queue の制御方法を表 3.2 に示す．N_M は DDP 内の処理負荷，N_{th} は多重処理可能タスク数の限界を表す．外部から入力されたパケットが TQ 回路に到着し，DDP の処理負荷の限界を越える場合，パケットを Queue に退避 (Queueing) する．外部に出力されるパケットが処理負荷の限界を超えない場合は，そのパケットは TQ 回路を通過させずに削除する．その他の命令で各条件の場合，すでにパケットが Queue にキューイングしている場合はキューイングしているパケットと TQ 回路に到着したパケットで余裕時間の短いパケットを出力する．キューイングしていない場合はそのまま TQ 回路を通過する．

3.7 結言

本章では LST スケジューラを搭載した DDP の全体構成と実行時間の定義を述べた．また初期優先度機構，Slack Time Memory，TQ 回路の詳細な構成及び Queue の制御方法を述べた．これらの機構によりコピー命令によるパケットの複製が発生した場合でも同じ余裕

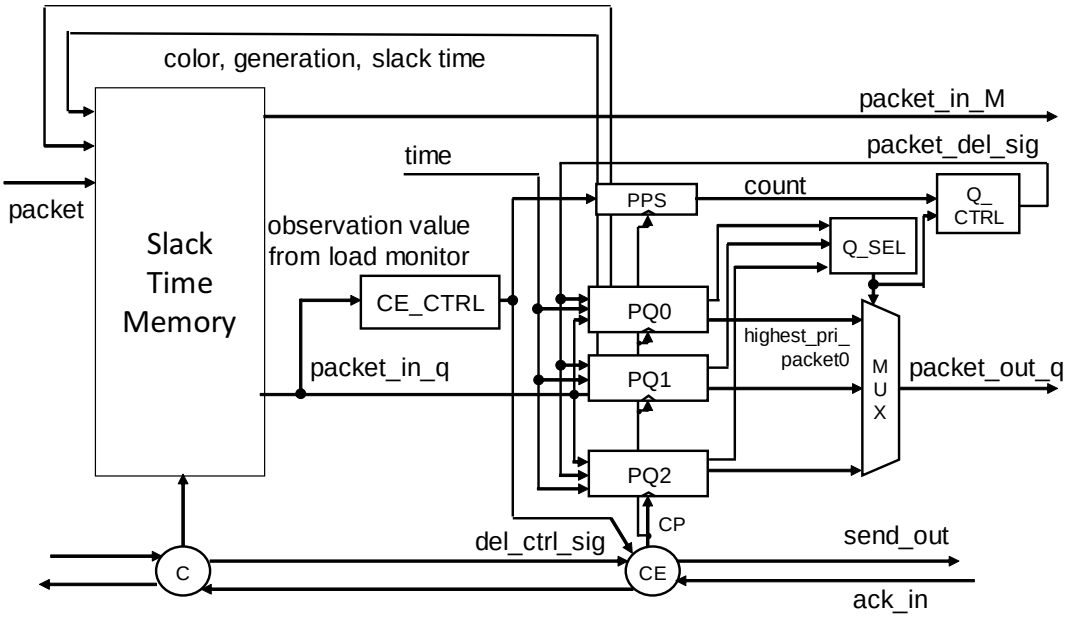


図 3.6 TQ 回路構成

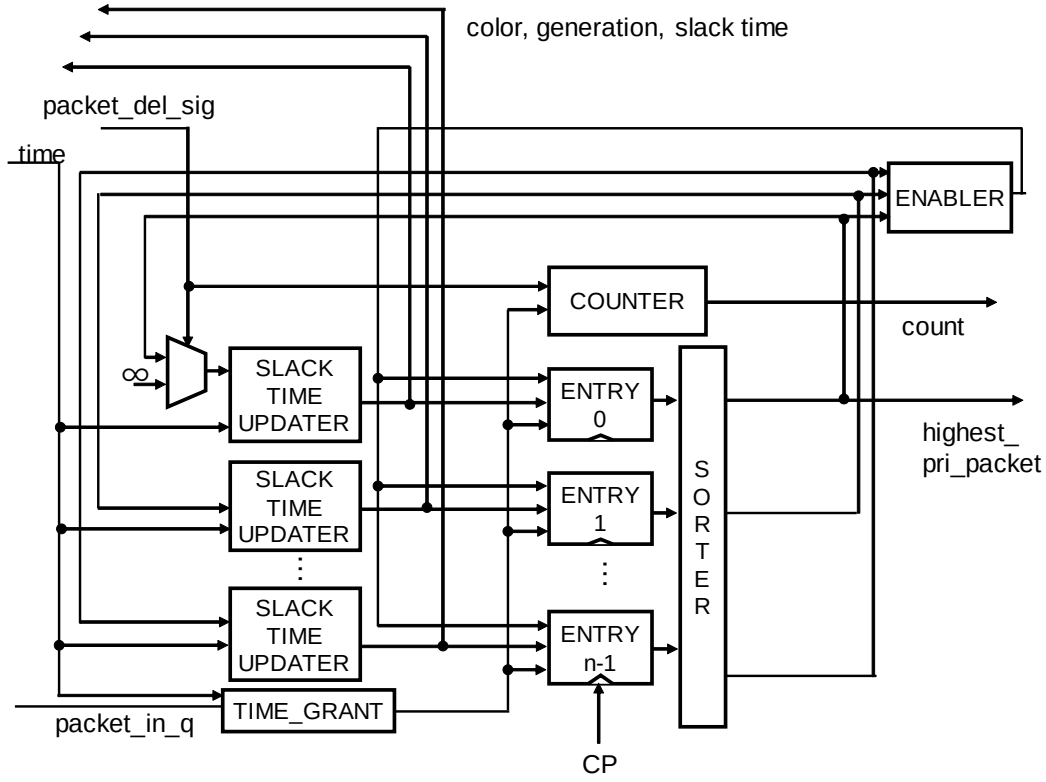


図 3.7 Priority Queue の構成

表 3.2　Queue の制御方法

	Queue の動作		
	input	output	others
$N_M < N_{th}$	通過	削除	通過
$N_M = N_{th}$	通過	通過	通過
$N_M > N_{th}$	Queueing	スケジューリング	スケジューリング

時間情報を扱うことができる．次章では，LST スケジューラを搭載した DDP の回路規模とリアルタイム性能を評価し，結果を述べる．

第 4 章

評価

4.1 緒言

本章では，第 2 章，第 3 章で述べた LST スケジューラを搭載した DDP を設計し FPGA 上に回路実装した．本研究では，FPGA チップは Intel 社 MAX10-50A を用いた．実装した DDP の回路規模及び，リアルタイム性能を評価し，結果を述べる．回路規模評価にはオリジナルの DDP と比較する．リアルタイム性能評価では周期タスクを対象としたタスクセットを作成し DDP コア稼働率とスケジューリング成功率を評価した．

4.2 評価環境

LST スケジューリングを搭載した DDP を設計し，FPGA 上に回路実装した．本研究で用いた FPGA チップは Intel 社 MAX10-50A である．また，設計には Intel 社 FPGA 用設計ツール Quartus Prime 18.0 を用いた．実装した DDP の基本仕様を表 4.1 に示す．

本研究では回路の最適化は Quartus によるコンパイル設定にてパフォーマンスと消費電力を均整に設定した．DDP の転送制御回路 C 素子間にデータラッチのセットアップホールド違反を防ぐために，遅延素子を各ステージに 48 個接続した．LST スケジューラを搭載した DDP の TQ 機構は構造上他ステージよりセットアップ時間が膨大になると予想されるため TQ の C 素子間のみ遅延素子を 112 個接続した．

表 4.1　FPGA 実装した DDP の基本仕様の比較

	Original	Proposed
パケットフォーマット		pri : 2bit
	col : 3bit	
	gen : 8bit	
	dest : 7bit	
	LR : 1bit	
	CP : 4bit	
	C : 1bit	
	Z : 1bit	
	data : 16bit	
		slack time : 20bit
メモリ		PRI : 42bit×32 words
	CST : 20bit× 64words	
	MMRAM : 24bit× 64entry	
	DMEM : 16bit × 1024words	
	PS : 16bit × 128words	
Queue の容量		8 × 3 words
Queue のパケットサイズ		88bit

4.2.1　評価タスクセット

実装した DDP のリアルタイム性能評価のために，評価タスクセットを作成した．本評価では周期的に実行するタスクを対象とする．よって，相対デッドライン時間を 1 周期として，周期的にタスクが起動する．以降相対デッドライン時間を周期と呼ぶ．また，全タスクは時刻 0 で実行要求され，予め決められた周期毎に実行要求される．DDP の入力ポートは 1 つであるため，同時刻に要求されたタスクは 100ns ずつ遅延させて順次入力した．

本研究では以下のタスクをそれぞれ評価する．

- 即値演算を N 個縦続接続したタスク
- LPF を実行する 9 段 FIR タスク

即値演算のみを行うプログラムと並列処理プログラムを用いてそれぞれ評価する．しかし，パケットの宛先を決める dest 情報が 7bit のため，縦続接続では即値演算 128 個しか接続できない．同様に LPF を実行する FIR タスクも段数に制限がある．そこで，即値演算を縦続接続したタスクを図 4.1 のように変更する．パケットの data にタスクの実行時間に相当するデータを与えて，sub 命令で data から 1 ずつ減らしていく．bzl 命令で data が 0 の場合に出力し，0 でない場合は sub 命令に巡回する．これにより即値演算を縦続接続したタスクに相当するプログラムにすることが可能となり，即値演算を縦続接続したタスクのリアルタイム性能評価は図 4.1 のプログラムを実行して評価する．パケットの data に与える実行時間に相当するデータを以下の式に示す．

$$\text{データ} = \frac{\text{タスクの実行時間}}{DDP2 \text{ 周分の周回時間}} \tag{4.1}$$

FIR 型 LPF ではフィルタの段数に応じてパケット命令も増加する．

以下に FIR 型 LPF の式 4.2 とプログラムの図 4.2 を示す．

$$Y_n = a_0 * X_n + a_1 * X_{n-1} + ... + a_8 * X_{n-8} \tag{4.2}$$

実装した DDP で LPF を実行する場合，dest 情報が 7bit のため 9 段のフィルタしか実装できない．そのため評価するにあたって，タスクの実行時間分の LPF を実行するために，実行完了したパケットを再度 DDP に入力し LPF を実行する．これにより，LPF を実行する N 段の FIR フィルタタスクと同様の処理を行うことが可能となる．これをタスクの実行時間分行うため再度 DDP に入力して実行する回数は以下の式**??**となる．

$$\text{再実行回数} = \frac{\text{タスクの実行時間}}{LPF \text{ を実行する 9 段 } FIR \text{ フィルタの実行時間}} \tag{4.3}$$

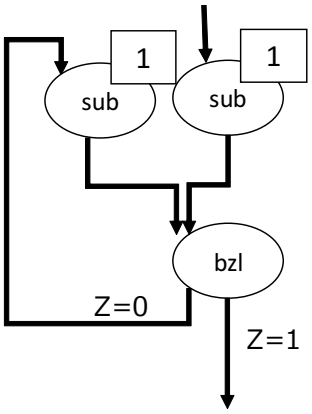


図 4.1　即値演算を縦続接続したタスクのデータフローグラフ

4.2.2　タスクセット内のパラメータ

タスクセットは実行要求時刻，実行時間，周期のパラメータを持つ．各タスクの稼働率と周期は以下の範囲からランダムに決定した．

- 周期

  1ms, 2ms, 4ms, 5ms, 10ms, 20ms
- タスク稼働率

  $1\% \sim 80\%$
- 実行時間

  周期 $\times$ タスク稼働率

上記の範囲でそれぞれタスクセットを作成した．生成したタスクセットの例を図 4.3 に示す．20ms はすべてのタスクの周期の最小公倍数であるため，20ms 内の実行の様子を観測すれば全タスクの組み合わせを網羅することができる．

4.2.3　タスクセットの生成方法

DDP コアで多重処理可能なタスク数を P，タスクセットに含まれるタスクの合計稼働率を $U_{taskset\_util}$ とおくと，システム稼働率 U_{sys_util} は，$U_{taskset\_util}$ / P で定義される．

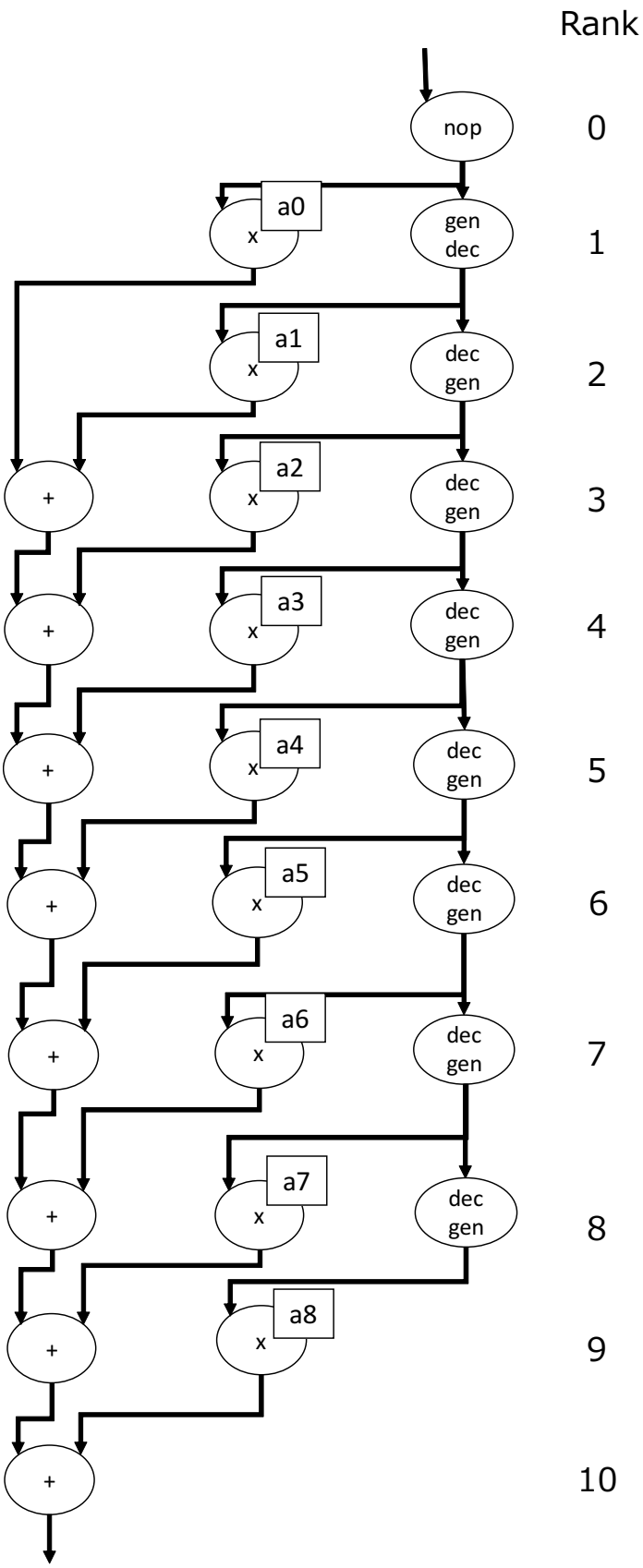


図 4.2　FIR 型 LPF のデータフローグラフ

タスク	実行時間 [ms]	周期 [ms]	余裕時間 [ms]
0	9	20	11
1	0.78	1	0.22
2	0.4	2	1.6
3	0.68	4	3.32

図 4.3　評価タスクセット例

生成するタスクセットのシステム稼働率に応じて，$U_{taskset\_util} \ / \ P \leq U_{sys\_util}$ である限り，新しいタスクをタスクセットに追加し，追加したタスクの稼働率を $U_{taskset\_util}$ に加算する．最後に生成するタスクのみ，$U_{taskset\_util} \ / \ P = U_{sys\_util}$ となるように，タスクの稼働率を調整した．

4.2.4　評価回路

FPGA 実装した評価回路を図 4.4，図 4.5 に示す．回路規模評価では DDP コアのみの規模を評価する．以下に各回路の動作を説明する．

- TASK INVOCATION

  各タスクが実行要求されるとタスクの識別情報やデータを保持する TASK MEMORY に Send 信号とメモリアドレスを出力する．PACKET GENERATOR が処理を行っている場合は Ack 信号で処理が終わるまで出力しない．
- TASK MEMORY

  TASK INVOCATION から出力されたアドレス値をもとにタスクの識別情報とデータを出力する．即値演算を縦続接続したタスクの場合，データは式 4.1 の実行時間に相当するデータを格納しておく．
- PACKET GENERATOR

TASK MEMORY から出力された識別情報とデータを用いて，DDP のパケットフォーマットに合わせたパケットを生成する．

- MERGE UNIT

  LPF を実行する FIR フィルタタスクを対象とした際，出力されたパケットは再度 DDP に入力される．実行要求タスクとタイミングが合わせないように MERGE UNIT で調停する．また，再度入力する場合，世代情報 generation を更新して入力する．

- BRANCH UNIT

  出力されたパケットが到着した際，外部に出力するか再度 DDP に入力するかを確認する．タスクの実行時間分の実行回数を行っていた場合，外部に出力する．それ以外は再度 DDP に入力するために MERGE UNIT に出力する．

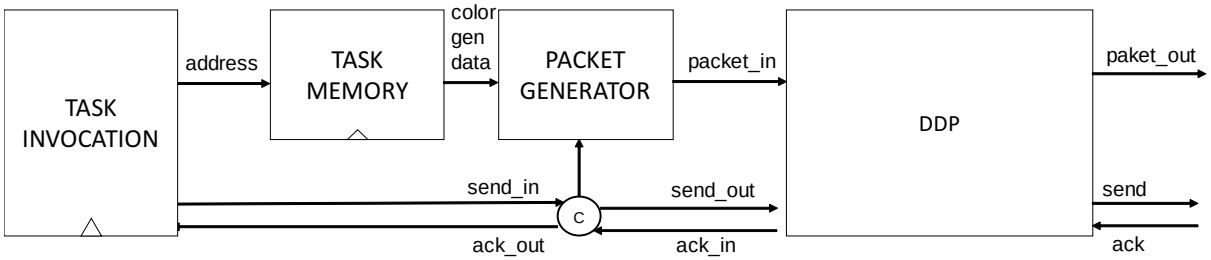


図 4.4　即値演算を縦続接続したタスクの評価回路

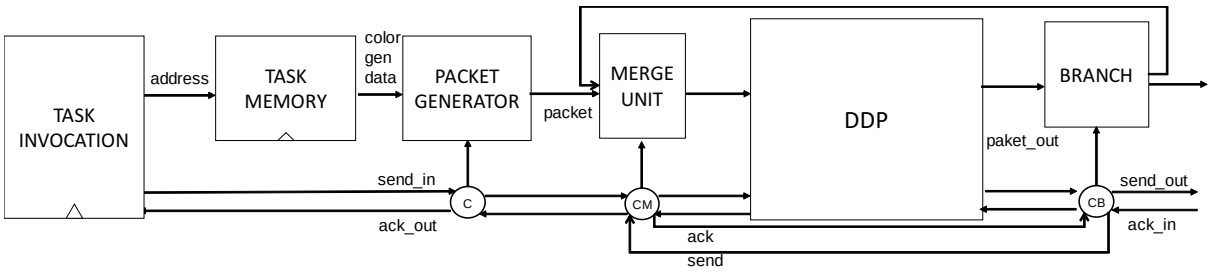


図 4.5　LPF を実行するタスクの評価回路

表 4.2　回路規模比較

	Original	Proposed
LE 数 [elements]	3474	24433
Register 数 [registers]	1755	4341
Memory 量 [bit]	20864	22464
周回時間 [ns]	460	760

4.3　回路規模評価

オリジナルの DDP の回路規模との比較を表 4.2 に示す．LST スケジューラを搭載した DDP の LE 数は従来の DDP の約 7 倍の回路規模となった．PQ 回路では LST スケジューリングのために，余裕時間 20bit とキューイング時刻 20bit をパケットに付与した状態でタスクの優先度ソートおよびキューイングを行っているため，回路が増大した．また，その PQ 回路を 3 クラス分搭載したため，より増加した．よって，今後，STM に FPGA の内蔵メモリを用いることや PQ 回路の最適化が必要である．

4.4　リアルタイム性能評価

FPGA 上でのコア稼働率とスケジューリング成功率を測定した．測定には，組み込みオシロスコープ回路である SignalTap II を用いた．コア稼働率は DDP にタスクが入力された時刻から，処理タスク数 1 で実行した時間と，処理タスク数 2 で実行した時間をそれぞれ計測する．最大周期まで DDP が実行した時間をコアの稼働率として評価した．スケジューリング成功率は式 4.4 から計算する．

$$\text{成功率} = \frac{\text{全タスクがスケジューリングに成功したタスクセット数}}{\text{総タスクセット数}} \tag{4.4}$$

リアルタイム性能評価の条件は以下の通りである．

- DDP コアの多重処理可能タスク数：2
- 評価タスク数：20 セット

- システム稼働率：80%, 90%, 95%

DDP コアの稼働率は，多重処理可能タスク数が 2 であることから，式 4.5 で計算する．

稼働率 = 処理タスク数 1 で実行中の稼働率 + 処理タスク数 2 で実行中の稼働率 (4.5)

4.4.1　即値演算を縦続接続したタスクでの性能評価

即値演算を縦続接続したタスクを生成した評価タスクセットで実行したコアの稼働率を図 4.6 に示す．システム稼働率が 80%，90%，95%と増加するにつれて多重処理数が自律的に変化している．データく駆動型プロセッサの柔軟な多重処理能力が活かされていることが確認できる．一方で，システム稼働率が 90%以上になると，デッドラインミスが発生するタスクセットもあり，表 4.3 に示すようにスケジューリング成功率 100%を維持できなかった．これは，LST アルゴリズムの限界でもあるが，システム稼働率を 80%以内で運用すれば実用的に活用できることを示唆している．一般的な車載 MCU(Micro Controller Unit) ではシステム稼働率 50%以内で運用されている．表 4.3 の結果からシステム稼働率の上限を 80%に向上することが可能となった．

また Python で作成したアーキテクチャシミュレータによるスケジューリング成功率を 4.4 に示す．アーキテクチャシミュレータは LST スケジューラを搭載した DDP の動作を模擬したプログラムである．コアの性能を示すためのパラメータとして，DDP コアの多重処理可能タスク数と DDP ないのタスクの周回時間を設定することができる．多重処理可能タスク数を超えた場合は周回時間が経過するごとに，タスクの中から多重度を超えない範囲で優先度の高いタスクを選択し，実行する．アーキテクチャシミュレータでは実行時間はパケットが混雑していない場合の周回時間でシミュレーションする．また，1 タスクに 1 パケットを前提としているためアーキテクチャシミュレータは即値演算を縦続接続したタスクのみを対象としたシミュレータである．アーキテクチャシミュレータでの性能評価と実時間 FGPA での評価を比較した結果，ほぼ同じ結果が得られた．

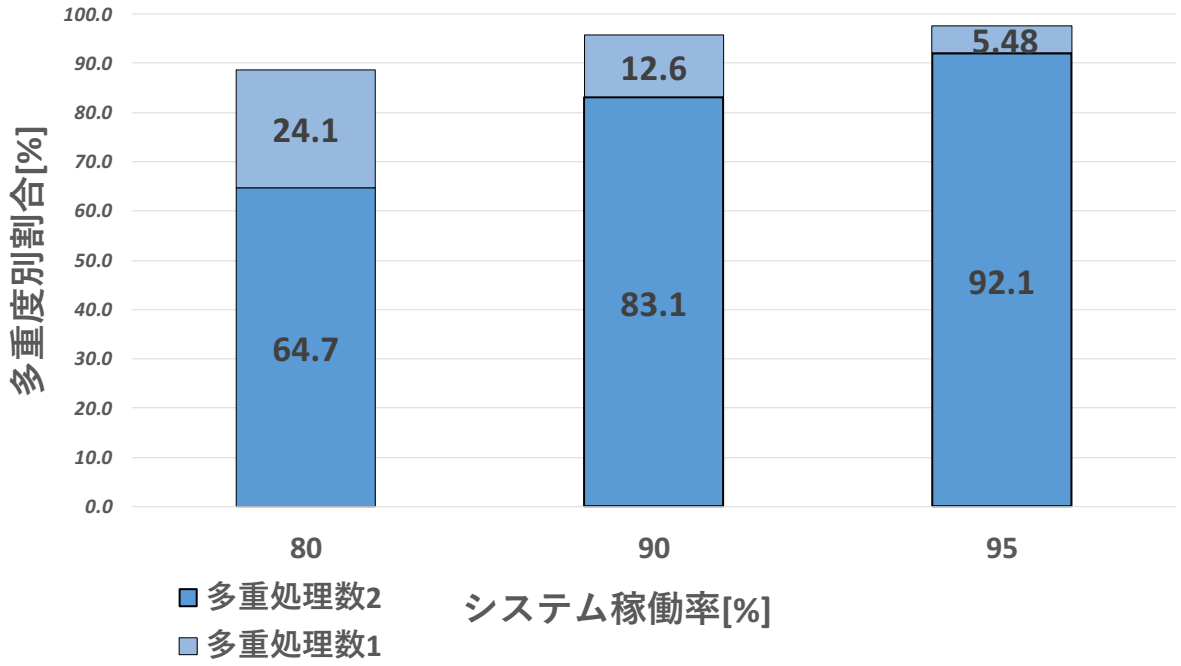


図 4.6　即値演算を縦続接続したタスクの多重度別コア稼働率

表 4.3　提案 DDP のスケジューリング成功率 [%]

平均システム稼働率 [%]	80	90	95
成功率 [%]	100	95	85

4.4.2　FIR 型 LPF タスクでの性能評価

FIR 型 LPF タスクでの性能評価を行ったが，実装した DDP では評価タスクセットを実行することができなかった．本研究で用いた図 4.2 の FIR 型 LPF はコピー命令によって Rank 2 の段階で 4 パケット複製される．STP 回路によって実装した DDP はステージ数に応じて多重処理できる．本研究で実装した DDP は 11 ステージで構成されている．LPF タスクを 4.2.1 節で述べたように 3 タスク実行した場合 DDP のステージ数に対して周回するパケットは最大 12 パケットとなる．そのため，STP 回路による多重処理能力を超えてしまい，デッドロックが発生してしまい実行することができなかった．また，一部パケットをキューイングしても，独立した処理を行うパケットはタスクが停止しているにもかかわらず

表 4.4　シミュレーションによるスケジューリング成功率 [%]

平均システム稼働率 [%]	80	90	95
成功率 [%]	100	98	89

実行される．そのパケットがさらにコピー命令を行うとさらにパケットが複製される．その場合でもデッドロックが発生してしまう問題がる．今回評価タスクのパケット数が実装した DDP のステージ数より超えたためデッドロックが発生した．そこで DDP の各機構の処理を細分化し，デッドロックが発生しないステージ数にすることで解決できるのではないかと考える．しかし，タスク数が増加するとともにステージ数も細分化しなければならないため根本的な解決ではない．

並列処理プログラムを実行するための対処案を示す．

- キューイング回路の改良

  パケットをキューイングした際，実装した DDP では一部のパケットを停止しても即値演算を行うパケットは二項演算を行う命令まで実行される．そのためタスクを一時停止する際，そのタスクに属するすべてのパケットをキューイング回路でキューイングする機構にする．これにより，独立した演算がタスクを停止しているにもかかわらず実行されず，コピー命令によるパケットの複製は行われない．

- プログラムの変形

  FIR 型 LPF タスクを変形したデータフローグラフを図 4.7 に示す．このデータフローグラフでは Rank 4，Rank 9 に二項演算のパケットと decgen 命令の 2 つのパケットが到着するまで Maching Memory ステージで待機させる nop 命令を用意する．二項演算を行うパケットがキューイング回路でキューイングされた場合，decgen 命令を行うパケットは nop 命令で待機される．これにより，decgen 命令のパケットが Rank 12 まで実行されず，タスクの停止が行える．

4.5　結言

本章では LST スケジューラを搭載した DDP の回路規模とリアルタイム性能を評価した．システム稼働率に応じて，多重処理数が自律的に変化しており，データ駆動型プロセッサの柔軟な多重処理能力が活かされていることが確認でき，システム稼働率を 80%以内で運用すれば実用的に活用できる．次章では，本論文で述べた内容をまとめるとともに，提案回路の今後の課題について述べる．

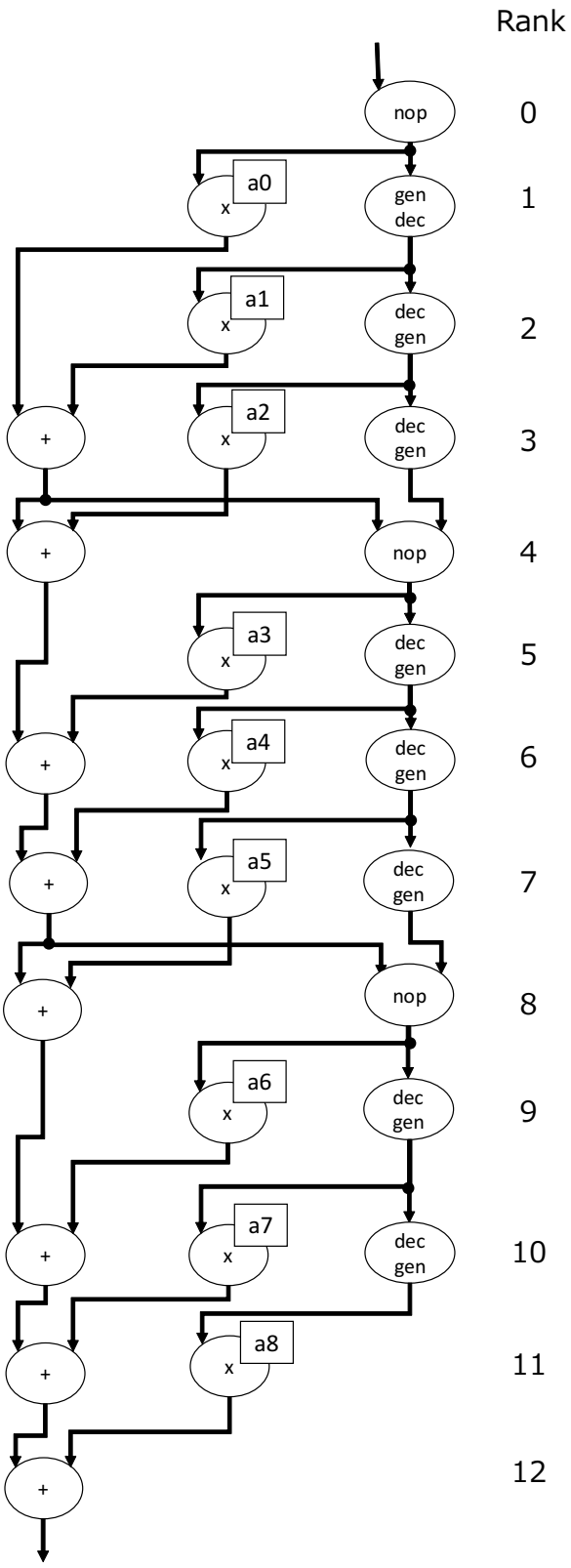


図 4.7 改良案の FIR 型 LPF のデータフローグラフ

第 5 章

結論

近年の IoT(Internet of Things) デバイスは年々増加しており，2021 年には 447 億個にまで増加すると予測されている．

自動車・輸送機器や医療，産業用途の分野の高成長が予測されている．その中で IoT デバイスは高機能化・高性能化に伴いシステムの性能要求が高まっている．これまで CPU のクロック周波数を高めることで処理能力を向上させていたが，消費電力や発熱量が増加する問題が発生している．こうした問題を解決する技術として CPU 内のコアを複数実装することで各コアの処理負荷を緩和させるマルチコア化が進んでいる．また組込みシステムの多くはリアルタイム性を求められる．リアルタイムシステムでは入力を受けてから結果を出力する際に出力内容の正しさだけでなく時間制約 (デッドライン時刻) を守る必要があるそのためタスクに優先度を設定し，優先度に基づいたタスクスケジューリングが必要となる．これにより組込みシステムの多くはリアルタイム処理とマルチコアの両立が求められている．

動的スケジューリング方式として EDF スケジューリングと LST スケジューリングが代表的である．EDF，LST スケジューリングはシングルコアでのタスクスケジューリングにおいて最適なスケジューリングである．しかし，マルチコア上でタスクを実行する場合に EDF と LST を比較すると，EDF スケジューリングは実行時間を加味していないため各コアにタスクを割り当てる際，デッドラインミスを予測できない問題がある．一方，LST スケジューリングはデッドラインまでの余裕時間を実行時間から算出しているため，デッドラインミスを起こさずスケジューリングが可能である．しかし，LST スケジューリングはシングルコア・マルチコアのスケジューリングにおいて余裕時間が近いタスクが複数存在する場合，一方の余裕時間がもう一方の余裕時間より短くなり優先度が変化する．これを繰り返す

ことによってタスクを頻繁に切り替える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恐れがある．また各種センサ等から到達する様々なデータストリームの多重処理を低消費電力で処理しなければならない．

そこで，演算に必要なパケットがそろうことで実行され，データに依存関係がないデータ駆動計算モデルで動作するデータ駆動計算モデルのハードウェア実装である DDP は多重処理が可能であり，コンテキストスイッチのオーバーヘッドなしに複数タスクを多重に実行できる．複数タスクの多重処理によりスラッシングを緩和できるデータ駆動型プロセッサ DDP のマルチコア化に向けて DDP シングルコアに搭載可能な LST ハードウェアスケジューラを提案されているが，提案されている DDP コアはシミュレーションでの性能評価のみで，実時間の性能評価は行われていない．LST スケジューラを搭載した DDP の実用化のためには現実的な環境で評価する必要がある．

負荷分散を行うエッジコンピューティングに用いられる IoT デバイスには様々な種類があるが，その中でも FPGA はユーザのシステム用途に合わせて自由に変更が可能なため，様々な応用に適している．

本研究では DDP のマルチコア化を前提とし，LST スケジューリングを基準にしたスケジューラ機構を搭載した DDP コアを FPGA 上に実装して回路規模と実時間性能を評価した．

本章において第 2 章はまず，リアルタイムシステムにおけるタスクのパラメータをまとめた．そして動的優先度方式である EDF，LST を比較し，デッドラインミス予測性の観点から LST が有効であることを述べた．また DDP におけるスケジューリング制御と LST スケジューラを搭載する際の余裕時間の計算及び更新と余裕時間情報の扱いを述べた．

第 3 章では，第 2 章で述べた方針をもとに設計した．LST スケジューラを搭載した DDP の全体構成，初期優先度機構 PRI，Slack Time Memory 及び TQ 回路について述べた．

第 4 章では第 3 章で設計した LST スケジューラを搭載した DDP を FPGA 上に実装し回路規模とリアルタイム性能の評価について述べた．実装には Intel 社 MAX10-50A，Intel 社 FPGA 用設計ツール Quartus Prime 18.0 を用いて設計し，回路規模は 7 倍の回路規模と

なった．リアルタイム性能評価を行った結果，一般的な車載 MCU のシステム稼働率 50%に対してシステム稼働率の上限を 80%に向上することができた．しかし，並列処理プログラムはパケット量に対して DDP の多重処理能力を超えたため実行することができなかった．

以下に本研究の今後の課題を示す．

- DDP コアのマルチコアの検討

  本研究では DDP シングルコアでの実時間評価のため今後 DDP コアを用いたマルチコアシステムの検討が必要である．

- LST スケジューラを搭載した DDP の最適化

  本研究の FPGA 実装では設計ツールの最適化機能のみを用いた．今後は DDP の構造的特徴を加味した更なる回路最適化が望まれる．

- DDP コア内スケジューラへの Fluid scheduling の適用

  提案回路は，システム稼働率 90%においてスケジューリング成功率が 100%に達していない．よって，LST スケジューリングでは理論的には最適な手法でないことが判ったため，マルチコアにおける最適なスケジューリングアルゴリズムである，Pfair[11][**?**] や LLREF[12][13] などの Fluid scheduling アルゴリズムの検討が必要である．

  これらのアルゴリズムは，スケジューリングのオーバヘッドが大きいため，実用的ではない．しかし，DDP は複数のタスクを多重に処理可能であり，タスク切り替え時にコンテキストスイッチが発生しないため，DDP コア内のスケジューラに Fluid scheduling アルゴリズムを適用することで，性能向上につながる可能性がある．

- TQ 回路の見直し

  オリジナルの DDP との回路規模評価では 4.5 倍回路が増大した．スケジューラ機構による回路規模の増加が原因であるため，TQ 回路のレジスタをブロック RAM に変更する等回路の最適化が必要である．

- パケットの情報量の見直し

  本研究で実装した DDP のパケット情報量では実行できるタスク数や長時間の運用に対

応できる情報量を保持していない．実用的な運用を想定した場合の識別情報やデータ，命令の bit 数を検討する必要がある．

- より実用的なタスクの適応

  評価では直線状プログラムは実行できたが，二項演算，コピー命令を含む並列処理プログラムの実行ができなかった．そのため，今後複数の並列処理プログラムタスクを実行できる DDP の検討と複数入出力，条件分岐命令を行うタスクに対応する必要がある．

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って熱心な御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．研究以外のこと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．

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，ならびに鵜川始陽准教授に心より感謝申し上げます．

研究室の後輩として，日頃ご支援，ご協力いただいた，修士 1 年の楠田健太氏，汐見興明氏，長野寛司氏，学部 4 年の井上聡氏，西岡周真氏，学部 3 年の岡野秀平氏，古田雄大氏，尾ノ井嶺卓氏，筧拓也氏，小谷拳聖氏，高見結衣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きました関係者の皆様に，心より御礼申し上げます．

参考文献

[1] 総務省, “令和元年度 情報通信白書 第 2 節 デジタル経済を支える ICT の動向, ” https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r01/pdf/n1200000.pdf, 2020 年 2 月 1 日参照.

[2] H. Terada, S. Miyata, and M. Iwata, “DDMP's: Self-Timed Super-Pipelined Data-Driven Multimedia Processors,” Proceedings of the IEEE, Vol.87, No.2, pp.282-296, Feb. 1999.

[3] 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.

[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. 2012.

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

[6] A.K. Mok, “Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment,” Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1983

[7] V. Salmani, M. Naghibzadeh, A.H.Taherinia, “Performance Evaluation of Deadline-based and Laxity-based Scheduling Algorithms in Real-time Multiprocessor Environments,” The 6th WSEAS International Conference on Systems Theory and

Scientific Computation (ISTASC'06), August 18-20, 2006.

[8] Kazuma Fukuda, Yushin Wada, and Makoto Iwata, "Decentralized Hardware Scheduler for Self-Timed Data-Driven Multiprocessor," Proceedings of the 2018 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'18), pp.256-261, July 2018.

[9] Yushin Wada, Kazuma Fukuda, and Makoto 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.

[10] S.-H. Oh, and S.-M. Yang, "A Modified LeastLaxity-First Scheduling Algorithm for RealTime Tasks," Proc. 5th International Conference on Real-Time Computing Systems and Applications, Hiroshima, Japan, 1998

[11] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, pp. 345-354, May 1993.

[12] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen, "An Optimal Real-Time Scheduling Algorithm for Multiprocessors," IEEE International Real-Time Systems Symposium, Dec. 2006.

[13] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen, "Synchronization for an optimal real-time scheduling algorithm on multiprocessors," IEEE International Symposium on Industrial Embedded Systems, pp. 9-16, July 2007.