令和 6 年度

修士学位論文

# セルフタイム回路による<br>データ駆動型プロセッサ向き<br>入出力 I/F 回路の構成法

## A Study on Adjustable I/O Interface Circuit for Self-Timed Data-Driven Processors

1275097　　植元　陸

指導教員　　岩田　誠

2025 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

## セルフタイム回路によるデータ駆動型プロセッサ向き入出力 I/F 回路の構成法

植元 陸

近年の IoT エッジ機器には，より柔軟な機能性，高性能化と省電力動作が求められている．セルフタイム回路により構成されたデータ駆動型プロセッサ DDP(Data-Driven Processor) はデータの多重並列処理により高性能化を実現しつつ，データ到着時にのみ動作することで省電力動作が可能であり，柔軟な機能性を提供可能な FPGA(Field Programmable Gate Array) 上に実装することで IoT エッジ機器のコアとして有望である．

非同期回路である DDP は，従来の同期回路によるアプリケーションプロセッサ AP(Application Processor) と連携することが実用的であり，タイミング同期，FIFO を用いたバッファリング，データおよびプロトコル変換機能を持つ I/F 回路 (DDP I/F) が必要となる [2]．また，これら各機能の既存手法を活用する場合，FIFO の種類に合わせたプロトコル変換回路をデータ転送時間が調整できるように設計する必要がある．よって，n-flop synchronizer を適切な位置に組み込むことで，必要な機能性および信頼性を実現しつつ，スループットや信頼性を独立して調整できる回路構成法を検討した．

提案する非同期 FIFO ベースの回路構成法を使用し，典型的な設計仕様に基づいて DDP I/F 回路を設計し，AMD 社 FPGA チップである Zynq-7010 上に実装して評価した．また，設計仕様を変更した場合の性能についても評価した結果，同期 FIFO ベースの回路構成法と比較して少なくとも 3.7 倍以上のスループットと，より高いリソースあたりのスループットを達成し，スループットや信頼性を独立して調整できることも確認した．

**キーワード**　IoT (Internet of Things), データ駆動型プロセッサ, 入出力 I/F 回路

# Abstract

## A Study on Adjustable I/O Interface Circuit for Self-Timed Data-Driven Processors

Riku UEMOTO

With the wide spread of Internet of Things (IoT), more flexible functionality, higher performance, and lower power consumption of IoT edge devices are demanded. A self-timed data-driven processor (DDP) achieves high performance because of its parallel processing capability and low power operation because it operates only when data arrives. The self-timed DDP implemented on Field Programmable Gate Array (FPGA) is promising as the core of IoT edge devices since FPGA can provide flexible reconfigurability.

The self-timed DDP could become practical for coordinating with a clock-synchronous application processor (AP). In this case, the AP-DDP coordinated system requires an I/O interface circuit for self-timed DDP (DDP I/F) with timing synchronization, buffering function using FIFO and protocol conversion functions. However, it is necessary to design a protocol conversion circuit that matches the FIFO type to adjust the data transfer time. In this paper, we designed a protocol conversion circuit that matches an asynchronous FIFO. In addition, we investigated a circuit configuration method by which the system can achieve the required functionality and reliability and adjust its required throughput and reliability independently by incorporating an n-flop synchronizer at the appropriate position.

We designed a DDP I/F circuit based on typical design specifications using the

proposed asynchronous FIFO-based circuit configuration method and then implemented it on AMD's Zynq-7010 FPGA chip, and evaluated it. The performance of the DDP I/F circuit with changing design specifications was also evaluated. The results showed that the proposed asynchronous FIFO-based DDP I/F circuit achieved at least 3.7 times higher throughput than the synchronous FIFO-based one. We also have improved the throughput per resource higher than the synchronous FIFO-based circuit configuration method and identified that the throughput and reliability could be independently tuned.

***key words*** Internet of Things, data-driven processor, I/O interface circuit

# 目次

# 図目次

図目次

# 表目次

# 第 1 章

# 序論

IoT(Internet of Things) 技術の普及と発展により，工場や農業，自動車などの様々な分野で IoT 技術が活用されており，IoT エッジ機器には各応用分野に特化したアーキテクチャの採用による，柔軟な機能性が求められる．加えて，現代の IoT エッジ機器は，複数のセンサからデータを取得し，アクチュエータを制御するためにそれらのデータを分析したり，他の機器やクラウド上のサーバと通信するため，様々な種類のデータを多重に処理する必要がある．また，エッジ機器上でクラウドに集約するデータの前処理を行うことで，クラウドの処理負荷分散，およびネットワークの負荷軽減を実現するエッジコンピューティングにより，エッジ機器における処理データ量が増加しているため，高性能化が求められる．さらに，エッジ機器の設置環境によってはバッテリーを用いて電源を供給する必要があり，バッテリー交換の頻度を抑え，コストを削減するために省電力動作が求められる．

セルフタイム回路により構成されたデータ駆動型プロセッサ DDP(Data-Driven Processor)[1] は高性能かつ省電力な動作が期待できるアーキテクチャであり，柔軟な機能性を提供可能な FPGA(Field Programmable Gate Array) 上に実装することで，IoT エッジ機器のコアとして有望である．DDP は様々な種類のタスクを多重に処理できるため，エッジ機器として高性能化を実現できる．また，セルフタイム回路を用いて構成することでデータの到着時のみ動作させることが可能となり，処理するデータがない時間帯の待機時電力を削減できるため，省電力動作が期待できる．FPGA は回路情報を柔軟に書き換え可能なデバイスであり，応用分野に特化した命令セットアーキテクチャを採用することが出来る．

セルフタイム回路により構成した DDP は非同期回路であり，従来の同期回路によるアプリケーションプロセッサ AP(Application Processor) 上で動作する既存のソフトウェアを再

利用することで開発コストを削減できるため，AP と連携することが実用的である．本研究では，このシステムを AP-DDP 連携システムと呼ぶ．AP-DDP 連携システムの実現のために，AP-DDP 間において動作タイミングを同期し，DDP の処理能力に応じて入出力を行うための，データおよび転送制御プロトコル変換回路を有する I/F 回路 (DDP I/F) が必要となる．AP-DDP 間のような同期-非同期回路間のデータ転送では，同期回路上の DFF(D Flip-Flop) に非同期信号が入力されることで，Setup/Hold 時間制約に違反して出力信号が不安定になるメタステーブル状態が発生する可能性があり，回路の誤動作につながる．そのため，非同期信号を同期化する機能が必要である．また，AP 側のデータ転送レートと DDP の処理能力は，平均的には同じになるように設計されるが，複数のデータをまとめて転送する場合，一時的に変動する可能性がある．この変動に対応するため，FIFO を用いたバッファリング機能が必要である．さらに，転送要求信号 send，転送許可信号 ack を用いた 2-phase，または 4-phase ハンドシェイクによりデータ転送を制御するセルフタイム回路のプロトコルに変換する機能や，AP または DDP が扱う形式に合わせたデータ変換機能も必要である．これらの機能の実現に加え，FPGA の柔軟性を活かして，様々な設計仕様の変更に対して適応可能な DDP I/F 回路の構成法を確立することで，AP-DDP 連携システムをより汎用的に活用することができる．設計仕様の変更としては，スループットや信頼性等の設計目標，応用領域に特化したアーキテクチャ，あるいは，FPGA チップのグレードなどの変更が考えられる．

非同期信号の同期化手法や同期-非同期回路間の I/F 回路は各所で研究されており，n-flop synchronizer[3][4]，非同期 FIFO[5]，pausible clock[6][7], click-element を用いた同期-非同期回路間 I/F[8] などが挙げられる．これらの手法は，動作タイミングの同期，バッファリング，データおよびプロトコル変換といった機能を持っているが，これらの全ての機能を持つ I/F 回路については研究されていない．よって，本研究では，これら全ての機能を持ち，様々な設計仕様の変更に対して適応可能な DDP I/F 回路の構成法を提案する．

本稿の第 2 章では，DDP やセルフタイム回路，AP-DDP 連携システム，タイミング同期について述べ，DDP I/F 回路に必要な機能性，および信頼性について説明する．また，既

存の非同期信号の同期化手法や同期-非同期回路間の I/F 回路の特徴について述べるとともに，DDP I/F 回路の構成法を検討する上での課題について説明し，提案手法の方針について述べる．

第 3 章では，第 2 章で述べた課題を解決可能な DDP I/F 回路の基本構成 [2] について説明し，各モジュールの動作と設計目標を満たすための回路の構成方法について述べる．

第 4 章では，提案した構成法を用いて典型的な設計仕様で DDP I/F 回路を構成し，AMD 社の FPGA チップである Zynq-7010 を用いた Digilent 社の評価ボード Zybo Z7-10 に実装して，信頼性，スループット，リソース使用量について評価する．また，スループットや信頼性等の設計目標，データやメモリサイズなどのスケール，FPGA チップのグレードを変更した場合の性能について評価し，提案した DDP I/F 回路の構成法が様々な設計仕様の変更に対して適応可能であることを示す．

第 5 章では，本研究で提案する DDP I/F 回路の構成法についてのまとめと，今後の課題について述べる．

# 第 2 章

# 機能性および信頼性

## 2.1 緒言

本研究では，従来の同期回路によるアプリケーションプロセッサ AP(Application Processor) と，セルフタイム回路で構成された非同期回路であるデータ駆動型プロセッサ DDP(Data-Driven Processor) を連携するために必要な DDP 向き入出力 I/F 回路 (DDP I/F) の基本回路を FPGA 向きに設計し，アプリケーションや目標性能に合わせた回路構成法の確立を目標としている．そこで，本章では，DDP アーキテクチャや DDP I/F 回路に求められる機能性，信頼性について述べる．また，同期-非同期回路間のデータ転送に必要となる，非同期信号の同期化に関する既存手法について説明し，DDP I/F 回路の基本回路の要件や設計方針について論じる．

## 2.2 DDP アーキテクチャ

本研究で扱う DDP は図 2.1 に示すような環状パイプラインで構成されており，データ駆動の原理により動作するプロセッサである．DDP に入力されたデータは演算実行に必要なデータが揃うまで待ち合わせ，プログラムを読み出して演算を実行する．演算結果は DDP 内部を周回し，待ち合わせと演算実行を繰り返すことでデータを処理する．データの処理が終了すると，外部へデータを出力する．このような仕組みにより，異なるタスクを多重に処理することができるため，センサやその他のデバイスから送信されたデータなどの複数種類のデータを処理する必要がある IoT エッジ機器として高性能な動作が期待できる．また，後

述するセルフタイム回路を用いてパイプラインを構成することでデータの到着時のみ動作するため，省電力な動作も期待できるプロセッサである．

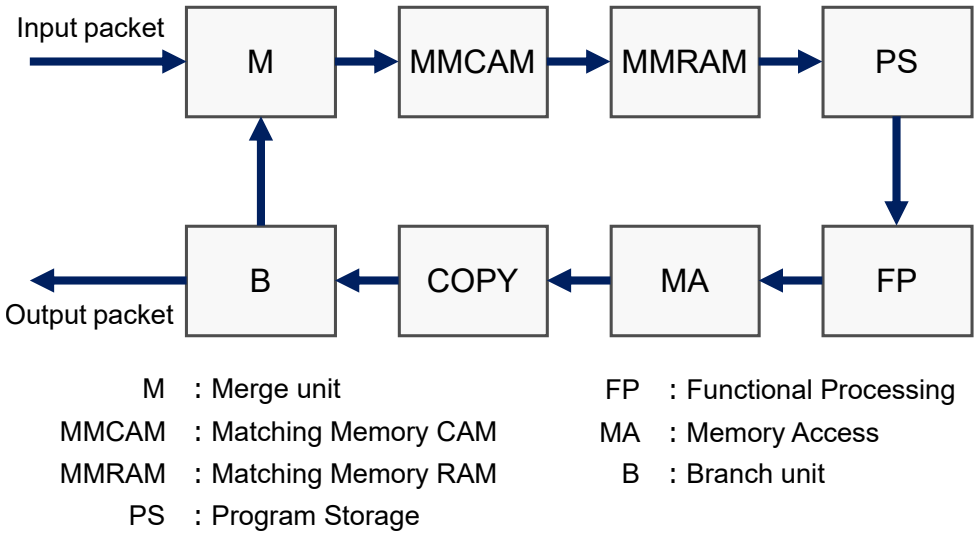


図 2.1　データ駆動型プロセッサ (DDP) の基本構成

DDP では，処理するデータに対してヘッダを付加したパケット形式でデータを処理する．DDP 入出力パケットフォーマット例を図 2.2 に示す．

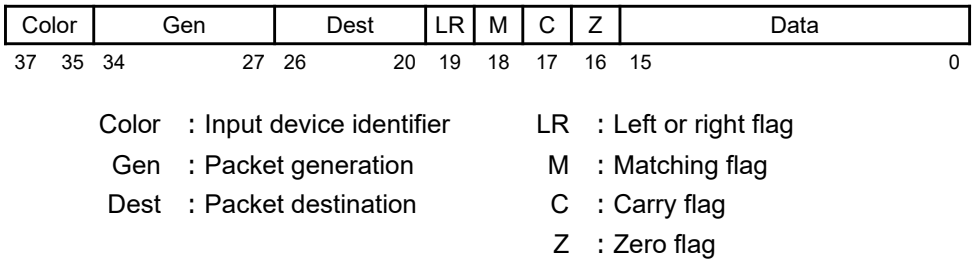


図 2.2　DDP 入出力パケットフォーマット例

ヘッダには各ステージにおけるデータの処理方法を示す情報が格納されており，各ステージで必要に応じて利用される．図 2.2 における Color は入力デバイスの識別に使われ，Gen はパケットの世代，Dest はパケットの宛先を示す．また，LR は左パケットであるか右パケットであるか，M は待ち合わせが必要であるかを示し，C は直前の演算で発生した桁上げ，Z は直前の演算結果がゼロであったことを示す．なお，実行する演算や B ステージにお

ける分岐など，その他必要な情報は PS ステージで読み出されるプログラムに含まれており，各ステージごとに入力されるパケットフォーマットは異なる．

また，本研究で対象とする DDP における各パイプラインステージの機能を以下に示す．

- 合流調停機構 M (Merge unit)

  DDP 外部からの入力パケットと B ステージから転送される DDP 内部周回パケットの合流を調停し，データを 1 つずつ受け入れ MMCAM ステージへ出力する．
- 待ち合わせ機構 MMCAM (Matching Memory CAM)

  パケットヘッダに含まれる Color，Gen，Dest，LR，M の情報を用いて，パケットの待ち合わせを制御する．待ち合わせを行う場合，MMCAM ステージで CAM(Content Addressable Memory) を用いて待ち合わせ情報を保持し，MMRAM ステージで対応するデータ部と各フラグを保持する．演算実行に必要なパケットが揃った場合，または定数演算を実行する場合には，待ち合わせを行わずにパケットがそのまま MMRAM ステージへ出力される．
- データ保持 & 定数読み出し機構 MMRAM (Matching Memory RAM)

  待ち合わせを行う場合にパケットのデータ部および各フラグを内部の RAM(Random Access Memory) に保持し，パイプライン上のパケットを削除する．演算に必要なパケットが揃った場合，対となるパケットのデータ部および各フラグを RAM から読み出し，2 つのパケットのデータ部を結合して PS ステージへ出力する．また，定数演算を実行する場合には，パケットヘッダの Dest をアドレスとして内部の定数メモリから定数データを読み出し，パケットと定数データを結合して PS ステージへ出力する．
- 命令フェッチ機構 PS (Program Storage)

  内部のプログラムが格納されたメモリに対して，パケットヘッダの Dest をアドレスとしてアクセスすることで，実行する演算コードや命令実行に必要な情報を含む新たなヘッダ情報を読み出し，パケットのヘッダ情報を書き換えて FP ステージへ出力する．
- 演算機構 FP (Functional Processing)

パケットヘッダに含まれる演算コードをもとに，内部の ALU(Arithmetic Logic Unit) を用いて演算を実行し，MA ステージへ出力する．

- メモリアクセス機構 MA (Memory Access)

  ロード/ストア命令が実行された場合に，内部に保持しているデータメモリに対して読み書きを行う．ロード命令の場合，パケットのデータ部をロードしたデータに書き換えた後，COPY ステージへ出力する．その他の命令の場合，FP ステージから出力されたパケットをそのまま COPY ステージへ出力する．

- 複製機構 COPY

  パケットヘッダの情報をもとに，必要に応じてパケットの複製を行い，順番に B ステージへ出力する．

- 分流制御機構 B (Branch unit)

  パケットヘッダの情報をもとに，COPY ステージから出力されたパケットを外部へ出力，または内部を周回するように制御する．

## 2.3 セルフタイム回路

本研究で対象とする DDP は，セルフタイム回路を用いたパイプライン STP(Self-Timed Pipeline) により構成された非同期回路で実現されている．セルフタイム回路は，データの到着をトリガとしてローカルクロック信号を生成することでデータの転送を制御する回路であり，グローバルクロック信号を用いず各パイプラインステージをデータ到着時のみ動作させることができる．STP の基本構成を図 2.3 に示す．

図 2.3 では，一致記憶フリップフロップ (Coincidence Flip-Flop) や C 素子 (C-element) と呼ばれるセルフタイム回路を用いている．この STP のタイミングチャートを図 2.4 に示す．i 段目のパイプラインステージ $\mathrm{stage_i}$ にデータを転送する場合，まず $\mathrm{stage_{i-1}}$ における転送制御回路である C 素子 $\mathrm{C_{i-1}}$ が転送要求信号 $\mathrm{send_{i-1}}$ をアサートすることで，$\mathrm{stage_i}$ における C 素子 $\mathrm{C_i}$ が転送許可信号 $\mathrm{ack_{i-1}}$ をアサートする．その後，$\mathrm{C_{i-1}}$ が $\mathrm{send_{i-1}}$ をネ

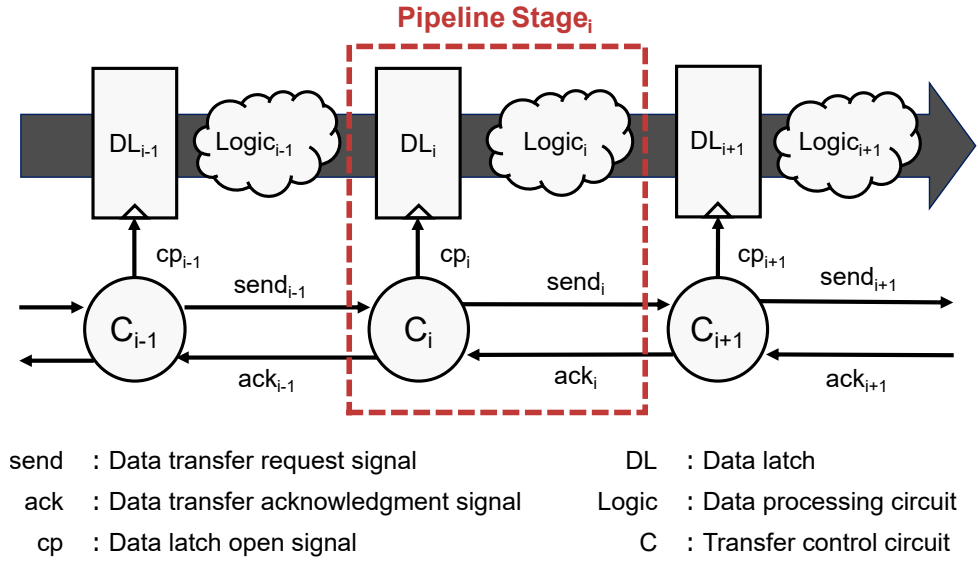


図 2.3　セルフタイム回路を用いたパイプライン (STP) の基本構成

ゲートすることで，stage$_i$ のデータラッチ DL$_i$ に対するローカルクロック信号 cp$_i$ が生成され，stage$_{i-1}$ から stage$_i$ に対してデータが転送される．また，C$_i$ が転送要求信号 send$_i$ をアサートすることで，同様にして stage$_i$ から stage$_{i+1}$ にデータが転送される．さらに，C$_i$ は ack$_{i-1}$ をネゲートすることで，stage$_{i-1}$ と stage$_i$ の間で再びデータ転送が可能となる．このように，STP では隣接する C 素子間のハンドシェイクによりデータ転送を制御するため，省電力動作を実現することができる．ここでは，転送要求信号 send，転送許可信号 ack の 4-phase ハンドシェイクによりデータ転送を制御する C 素子を用いて説明したが，2-phase ハンドシェイクにより転送を制御する click-element と呼ばれるセルフタイム回路も存在する．本研究では，どちらのセルフタイム回路で構成された DDP に対しても対応可能な I/F 回路の構成法を確立するために，4-phase または 2-phase ハンドシェイクのプロトコルを変換する機能が求められる．

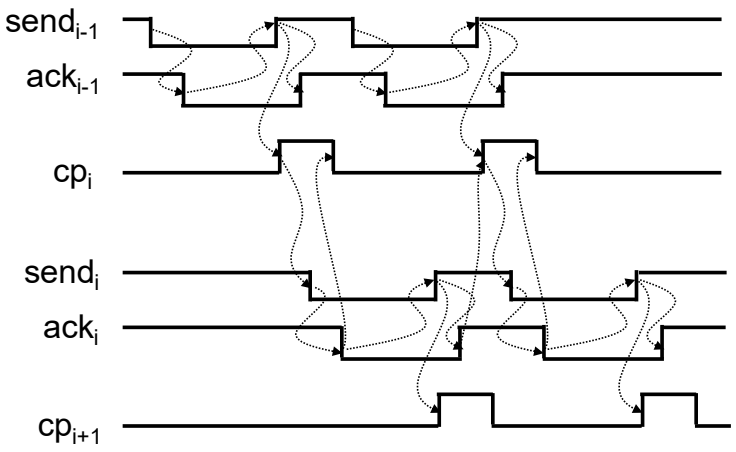


図 2.4　STP のタイミングチャート

## 2.4　AP-DDP 連携システム

本研究は，従来の同期回路によるアプリケーションプロセッサ AP(Application Processor) とセルフタイム回路により構成した非同期回路である DDP をアクセラレータとして連携した IoT システムへの応用が想定される．この AP-DDP 連携システムは図 2.5 に示すように構成され，センサデータの取得やアクチュエータの操作，他のデバイスやクラウド上のサーバとの通信を AP 上で行い，データの処理を DDP 上で行うことを想定している．そのため，AP-DDP 間でデータを転送する I/F 回路が必要となる．

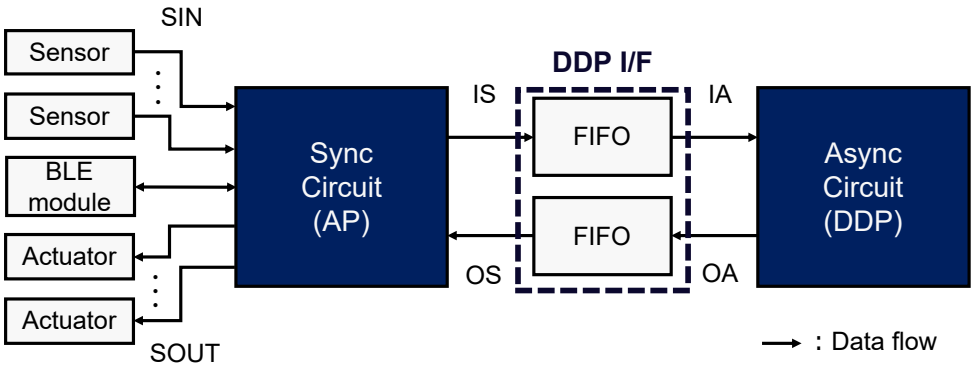


図 2.5　AP-DDP 連携システム

図 2.5 において，AP-I/F 間，また I/F-DDP 間の平均的な入出力レートをそれぞれ，$R_{IS}$，

$R_{OS}$，$R_{IA}$，$R_{OA}$ と表したとき，$R_{IS} = R_{IA}$，および $R_{OS} = R_{OA}$ となる必要がある．しかし，AP 側の転送レートと DDP 側の処理能力は異なる可能性があり，特に AP-I/F 間では DMA(Direct Memory Access) や割り込み処理を用いて，複数データをまとめて転送することも想定される．これらのことから，AP-DDP 間の平均的な入出力レートは同じであっても一時的な入出力レートは変動してしまう可能性があるため，一時的にデータを格納するバッファが必要となる．本研究では，I/F 回路内部に FIFO(First-In First-Out) を組み込むことにより，この変動に対応する．

## 2.5　タイミング同期

図 2.5 に示すように，AP-DDP 連携システムでは同期-非同期回路間でのデータ転送が行われる．一般的に，同期回路側の DFF(D flip-flop) に非同期信号が入力されると，Setup/Hold 時間制約に違反して出力信号が不安定になるメタステーブル状態が発生する可能性があり，回路の誤動作につながる．したがって，本研究においても，DDP から出力された非同期信号を AP 側で利用するために同期化する必要がある．

非同期信号の同期化手法や I/F 回路は各所で研究されており，n-flop synchronizer[3][4]，非同期 FIFO[5]，pausible clock[6][7], click-element を用いた同期-非同期回路間 I/F[8] などが挙げられる．

### 2.5.1　n-flop synchronizer

n-flop synchronizer は，図 2.6 のように構成され，DFF を n 段カスケード接続することで構成される同期化回路である．このような構成で，1 段目 DFF($DFF_1$) で入力信号を同期化し，仮にメタステーブル状態が発生して $DFF_1$ の出力信号 $q_1$ の状態が不安定になったとしても，2 段目 DFF($DFF_2$) から n 段目 DFF($DFF_n$) で，それぞれの出力信号 $q_2$ から $q_n$ に伝搬されるまでの間に信号を安定させることで，同期回路側での Setup/Hold 時間制約違反の発生を防ぐことができる．一般的に，メタステーブル状態の継続時間は確率的に決ま

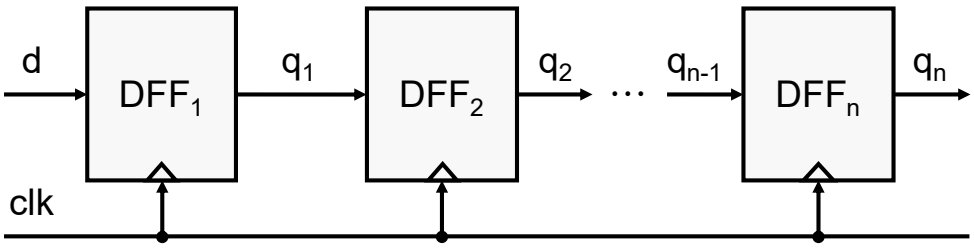


図 2.6　n-flop synchronizer

り，継続時間が長いほど発生確率が低くなることが知られている．DFF の段数 n を増加させると，メタステーブルを解消するための時間をより長く確保できるため，回路の誤動作確率を抑えることができる．しかし，それに伴い遅延時間が増加するため，適切な段数に設定する必要がある．

### 2.5.2　非同期 FIFO

非同期 FIFO は，図 2.7 のように構成される．一般的な同期 FIFO はデータの読み書きが同一のクロック信号に同期して行われるのに対し，非同期 FIFO はデータの読み書きが異なるタイミングで実行可能な FIFO である．FIFO からのデータの読み出しは読み出しクロック信号 rclk に同期して行われ，読み出し許可信号 rden がアサートされているときに読み出しデータ rdata が読み出される．また，FIFO へのデータの書き込みは書き込みクロック信号 wclk に同期して行われ，書き込み許可信号 wden がアサートされているときに書き込みデータ wdata が書き込まれる．その他の基本的な動作は，同期 FIFO と同様であり，データの読み書きが行われた際に FIFO の読み出しポインタ raddr および書き込みポインタ waddr の値を更新し，これらの値を比較することで，FIFO が empty または full 状態であるか否かを判定する．

図 2.7 において，empty および full フラグを生成する Flag Gen は図 2.8 のように構成される．非同期 FIFO の場合，データ読み書きのタイミングが異なるため，読み出しポインタ raddr および書き込みポインタ waddr はそれぞれ互いに異なるクロック信号で動作する非

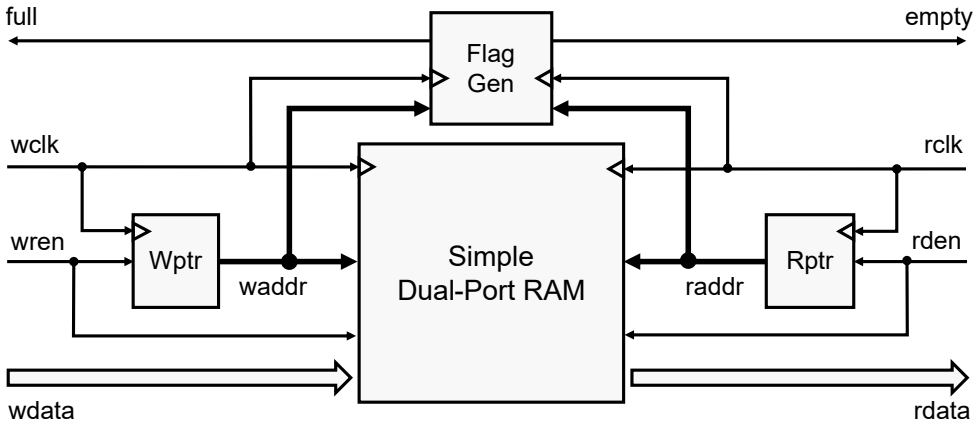


図 2.7　一般的な非同期 FIFO の構成

同期信号である．したがって，FIFO 内部で empty および full フラグを生成する際に，それぞれ waddr, raddr を同期化する必要がある．図 2.8 に示すように，Flag Gen はモジュール Sync を用いて waddr と raddr を同期化する．一般的に，各ポインタの同期化には，前述した n-flop synchronizer が用いられるため，図 2.8 の Sync は n-flop synchronizer で構成される．

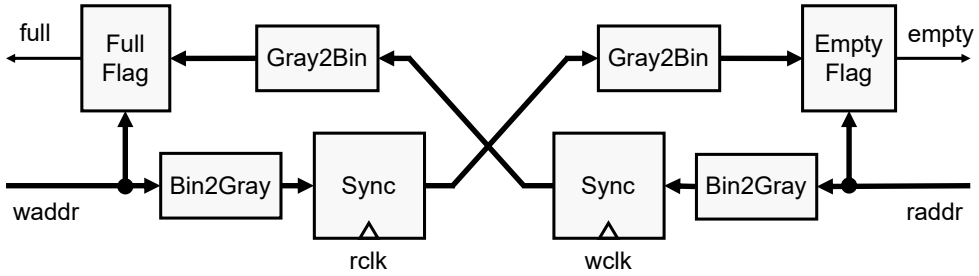


図 2.8　一般的な非同期 FIFO における Flag Gen の構成

同期化する読み出しポインタ raddr，および書き込みポインタ waddr は複数ビットの信号であるため，各ビットに対して同期化する必要がある．しかし，FIFO は 1 データの読み書きで各ポインタ値が 1 ずつ増加するという特徴を持つ．そこで，一般的に各ポインタを同期化する際には，隣接する値の間でビットが 1 ビットだけ変化するという特徴を持つグレイコードに一時的に変換する．図 2.8 では，Bin2Gray を用いてバイナリコードをグレイコー

ドに変換し，Gray2Bin を用いてグレイコードをバイナリコードに変換する．これにより，1 データの読み書きごとに発生する非同期信号を 1 ビットに抑えられるため，信頼性を向上させることができる．

これらのことから，非同期 FIFO はデータの読み書きを異なるタイミングで実行可能であるが，同期 FIFO よりもリソース使用量が増加し，回路規模が大きくなるという特徴がある．

### 2.5.3　pausible clock

pausible clock は，一時的にクロック信号を停止することにより同期化する手法であり，一般的に図 2.9 のように構成される．

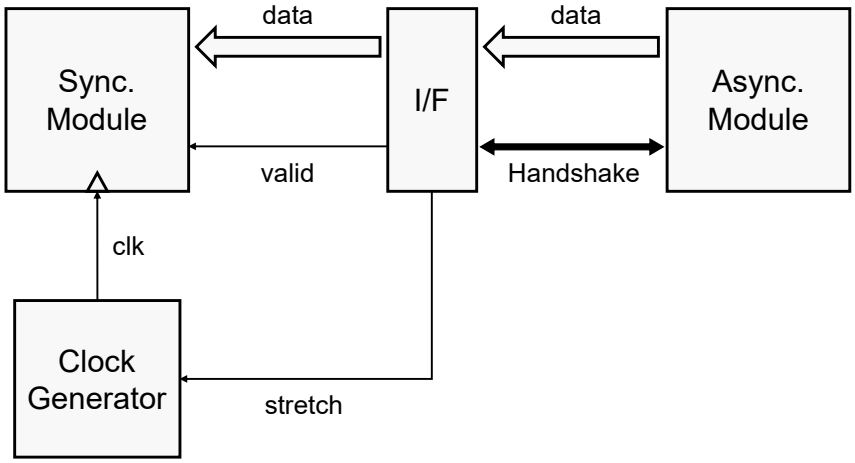


図 2.9　pausible clock を用いた同期化の仕組み

本手法では，非同期信号が同期回路側に入力される際に，クロック信号を発生させる Clock Generator に対してクロック停止要求を出力することにより，Setup/Hold 時間制約違反によるメタステーブル状態の発生を確実に防ぐことができる．したがって，非同期信号の入力による回路の誤動作確率を 0 にできる．

また，本手法を FIFO 内部に適用した pausible bisynchronous FIFO[9] という手法も提案されており，誤動作確率が 0 かつ，低遅延なデータ転送が可能である．

しかし，クロックを一時停止した際に同期回路側全体の動作が停止するため，同期回路側の処理が一時的に遅延するという問題や，Clock Generator が必要になるといった問題がある．特に本研究における AP-DDP 連携システムでは，AP-I/F 間のデータ転送はバスを通じて行われるため，同期回路側のクロック信号は AP 側から提供されるバスクロックとなる．その場合，一時停止させることはできないため，本研究にはこの手法を適用することができない．

### 2.5.4　click-element を用いた同期-非同期回路間 I/F

click-element を用いた同期-非同期回路間 I/F[8] は，前述した 2-phase ハンドシェイクによりデータ転送を制御するセルフタイム回路である click-element と，DFF を 2 段カスケード接続した 2-flop synchronizer を活用した I/F 回路である．非同期回路側の転送制御は click-element により行い，同期回路側の転送制御は 2-flop synchronizer により同期化した信号を用いて行うことにより，低遅延で高速にデータを転送する．

## 2.6　DDP 向き入出力 I/F 回路の要件

本章で述べた内容をまとめると，DDP 向き入出力 I/F 回路 (DDP I/F) には，動作タイミングの同期，4-phase および 2-phase ハンドシェイクのデータ転送制御プロトコルの変換，データのバッファリングという 3 つの主機能が必要となる．また，対象となる IoT アプリケーションや目標性能に対して柔軟に対応可能な構成法が求められる．しかし，各同期化手法や I/F 回路が対応可能な機能については，第 2.5 節に述べた通りであり，3 つの主機能を持つ同期化手法や I/F 回路は存在しない．特に，上記の既存手法を活用する場合，FIFO の種類に合わせたプロトコル変換回路をデータ転送時間が調整できるように設計する必要がある．また，n-flop synchronizer を適切な位置に組み込むことで，必要な機能性および信頼性を実現しつつ，スループットや信頼性を独立して調整できる回路構成法が必要となる．

第 2.4 節で述べたように，本研究ではデータを一時的にバッファリングするために，FIFO

を活用する．より小さい回路規模で高い性能を得るためには FIFO に合わせた転送制御プロトコルの変換回路が要求されるため，本研究では click-element を用いた同期-非同期回路間 I/F は活用しない．また，第 2.5 節で述べたように，タイミング同期手法の pausible clock は本研究に適用できないため，非同期 FIFO 内部で n-flop synchronizer を組み込むか，同期 FIFO の外部で n-flop synchronizer を用いて同期化する必要がある．したがって，本研究において前述した要件を満たす DDP I/F 回路は，新たに設計するプロトコル変換回路と非同期 FIFO を組み合わせた手法 (Async-FIFO I/F) と，新たに設計する n-flop synchronizer を組み込んだプロトコル変換回路と同期 FIFO を組み合わせた手法 (Sync-FIFO I/F) の 2 つに大別することができる．どちらの手法も，n-flop synchronizer の DFF 段数により信頼性を調整可能である．Async-FIFO I/F は，ハンドシェイクによる DDP 入出力の際に n-flop synchronizer による遅延の影響をほぼ受けないため，Sync-FIFO I/F よりも高速に動作可能である．リソース使用量については Async-FIFO I/F の方が多いが，その差は AP や DDP と比較するとかなり小さいため，一般的な IoT アプリケーションにおいても，リソース使用量の増加量やそれに伴う消費電力の増加量も許容範囲であると考えられる．これらのことから，本研究では非同期 FIFO を用いた Async-FIFO I/F の基本回路を設計し，対象アプリケーションや目標性能に合わせた構成法を提案する．また，転送制御プロトコルの変換回路は，2-phase，および 4-phase ハンドシェイクのどちらにも対応可能となるように回路を設計する．

## 2.7 結言

本章では，DDP アーキテクチャの仕組みや，DDP I/F 回路に求められる機能性，信頼性の詳細について述べた．また，非同期信号の同期化に関する既存手法や，同期-非同期回路間データ転送を実現するための既存の I/F 回路について説明し，DDP I/F 回路の基本回路の要件や設計方針について述べた．

本研究では，設計した回路の詳細な FPGA 配置手法について検討していないが，DDP

I/F 回路も一部は非同期回路であり，回路の配置結果ごとに異なる遅延により，性能が変化する可能性がある．そのため，データ転送時間が調整できるように設計するだけでなく，今後はより小さな回路規模で高い性能を引き出せる回路の FPGA 配置方法も検討する必要がある．

# 第 3 章

# DDP 向き入出力 I/F 回路の構成法

## 3.1 緒言

本章では，第 2 章で述べた機能性と信頼性を実現しつつ，データ転送時間が調整可能なプロトコル変換回路を FIFO の種類に合わせて設計し，スループットや信頼性を独立して調整できる DDP I/F 回路の基本構成，および各モジュールの動作について説明する．また，スループットおよび信頼性の設計目標などの設計仕様変更に対応するための DDP I/F 回路の構成法について述べる．

## 3.2 DDP I/F 回路の基本構成

本研究で提案する DDP I/F 回路は図 3.1 のような基本構成とした．このような構成にすることで，入出力ごとに第 2 章で述べた機能性，および信頼性を実現するための既存手法を活用することができる．DDP に対するデータ入力は Input FIFO，Packet Gen，Input C-element の 3 つのモジュールからなる Input I/F で行われ，データ出力は Output FIFO，Data Extract，Output C-element の 3 つのモジュールからなる Output I/F で行われる．

Input FIFO および Output FIFO は入出力データを一時的に保持するとともに，AP-DDP 間の動作タイミングを同期する．Packet Gen および Data Extract は，パケットの生成とデータの抽出を行い，Input C-element および Output C-element によって，データの入出力を制御する．

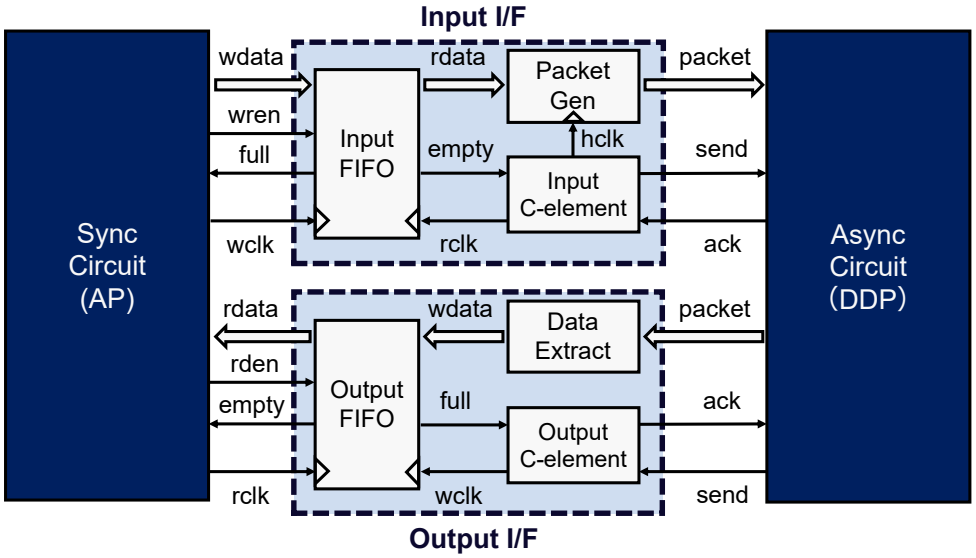


図 3.1　DDP I/F 回路の基本構成

## 3.3　Input FIFO および Output FIFO

第 2 章で述べたように，DDP I/F 回路にはデータを一時的に保持する FIFO が必要である．Input FIFO および Output FIFO は，それぞれ入出力データを保持するための非同期 FIFO であり，AP-DDP それぞれのタイミングでデータを読み書き可能である．

Input FIFO は図 3.2 のように構成されており，基本構成は一般的な非同期 FIFO と同様で，読み出しアドレス rclk と書き込みアドレス wclk をもとに Flag Gen 内部で empty，および full フラグを生成する際にタイミングを同期する．

Input FIFO 内部の Flag Gen は図 3.3 のように構成され，非同期信号である読み出しアドレス raddr を AP 側に入力し full フラグを生成する際に，n-flop synchronizer を複数ビット分持つモジュール Sync により書き込みクロック wclk に同期させる．一般的な非同期 FIFO と同様に，Bin2Gray，および Gray2Bin を用いて一時的にグレイコードに変換することで，1 データ入力あたりの非同期信号の発生を 1 ビットに抑え，信頼性を高めている．ただし，同期回路である AP 側から出力された信号を非同期回路である DDP 側へ入力する際にはタイミングの同期が必要ないため，書き込みアドレス waddr は n-flop synchronizer を

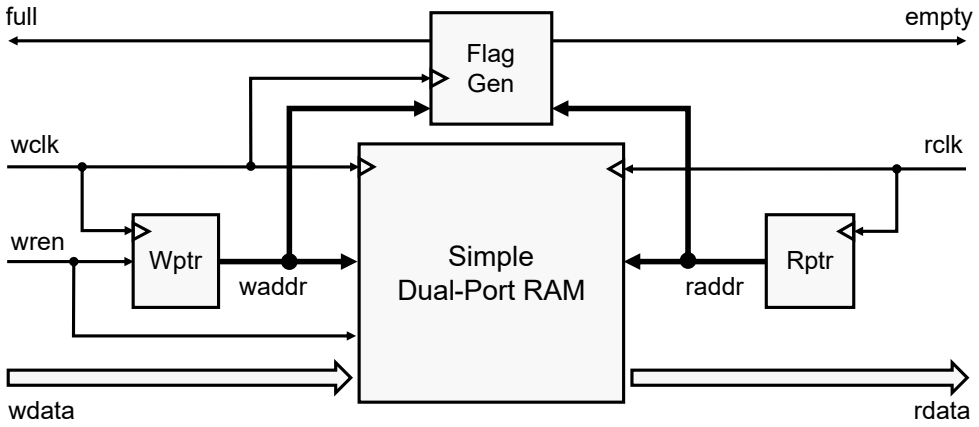


図 3.2 Input FIFO の構成

通過せずそのまま empty フラグの生成に利用される．同様にして，Output FIFO は書き込みアドレス waddr を AP 側に入力し empty フラグを生成する際に，Bin2Gray，Gray2Bin，および n-flop synchronizer を用いて読み出しクロック rclk に同期させる．

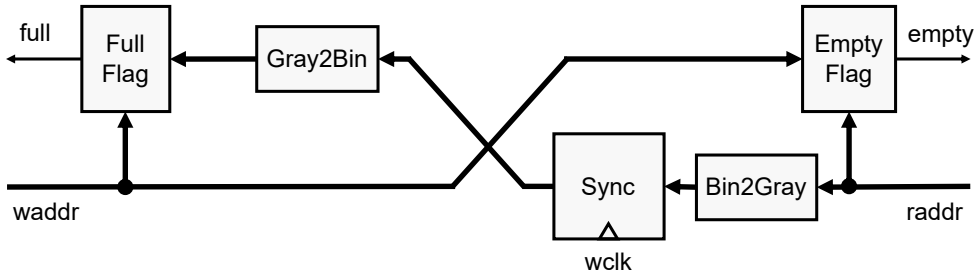


図 3.3 Input FIFO における Flag Gen

Input FIFO の Flag Gen において，empty フラグは複数ビットの信号である waddr と raddr の比較によって生成されるため，それらの値が変化した際にグリッチが発生する可能性がある．empty フラグは waddr と raddr の値が一致したときのみアサートされるため，empty 状態から各アドレスが変化した場合，必ず not empty の状態になる．したがって，empty フラグがアサートされているときにネゲートされてしまうグリッチ (empty $1 \to 0 \to 1$) は発生せず，empty フラグがネゲートされているときにアサートされてしまうグリッチ (empty $0 \to 1 \to 0$) のみ発生し得る．そのため，Input C-element においてこの

グリッチに対する対策が必要となる．Output FIFO の場合，full フラグの生成において同様にグリッチ (full $0 \rightarrow 1 \rightarrow 0$) が発生し得るため，Output C-element において対策が必要となる．

Input FIFO および Output FIFO に格納されるデータは，DDP で処理するデータに，利用する DDP パケットヘッダが格納されたヘッダメモリのアドレスを付加したものである．

これらのモジュールは，バッファリング機能に加え，n-flop synchronizer により動作タイミングを同期する機能も実現する．また，n-flop synchronizer により発生する遅延はスループットに影響を与えないため，DFF 段数 n の増加により信頼性を独立して向上させることが出来る．

## 3.4 Packet Gen および Data Extract

DDP では，処理に必要なヘッダを付加したパケット形式でデータを扱う必要があるため，図 3.4 のように構成される Packet Gen において入力パケットを生成する．まず，1 つの信号線を複数に分割するモジュール Slice によって，Input FIFO の読み出しデータ rdata を，ヘッダメモリのアドレス addr と DDP で処理するデータ data に分割する．そして，複数の信号線を 1 つに連結するモジュール Concat によって，addr をもとにヘッダメモリから読み出したヘッダ情報を data と連結することで，DDP パケットを生成する．ヘッダメモリは，Input C-element が生成するヘッダクロック hclk に同期して読み出しを行う．

DDP から出力されたパケットのヘッダ情報は AP 側にとって不要なものも含まれているため，Data Extract によりパケット中の不要な情報を取り除き，必要なヘッダ情報およびデータを抽出する．これらのモジュールにより，データの変換機能を実現する．

## 3.5 Input C-element

本研究で対象とする DDP は，セルフタイム回路によってデータを転送するため，AP-DDP 間における転送制御プロトコルの変換も必要となる．Input C-element は，FIFO-

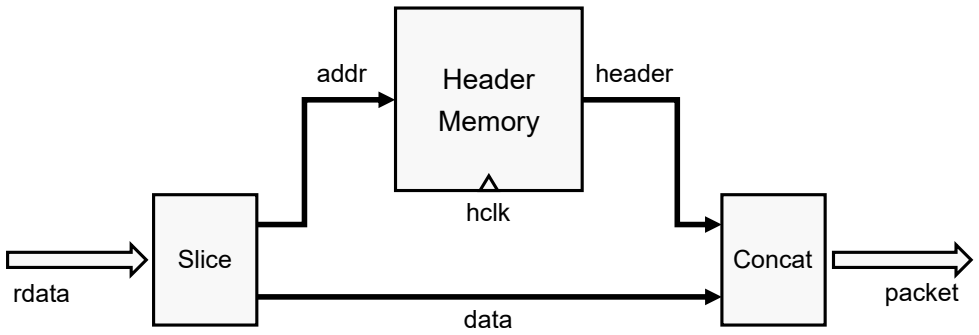


図 3.4 Packet Gen

DDP 間の転送制御プロトコルを変換し，DDP に対してデータを入力するためのモジュールである．第 2 章で述べたように，セルフタイム回路には 2-phase ハンドシェイクによるものと 4-phase ハンドシェイクによるものがある．本研究では，2-phase 用，および 4-phase 用の Input C-element を入出力信号を揃えて設計することで，DDP 側の仕様に合わせて選択できるようにした．

### 3.5.1 2-phase ハンドシェイクに対応した Input C-element

2-phase ハンドシェイクに対応した Input C-element を図 3.5 に示し，その動作を表すタイミングチャートを図 3.6 に示す．

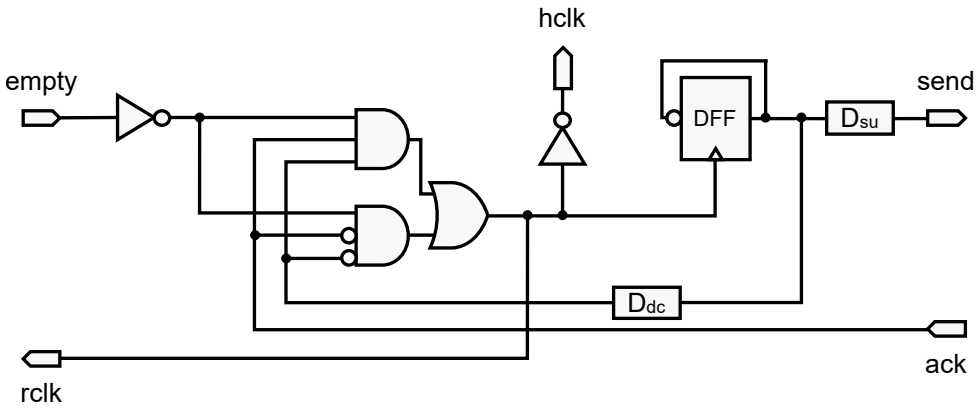


図 3.5 Input C-element (2-phase)

まず，マスターリセット信号 mr により回路を初期化後，Input FIFO の empty 信号がネゲートされていれば DDP への入力動作を開始し，Input FIFO に対する読み出しクロック rclk，およびヘッダメモリの読み出しクロック hclk をアサートしつつ，DDP を構成するセルフタイム回路との間で send，ack によるハンドシェイクを行う．

rclk のアサートにより Input FIFO から読み出されたデータは，Packet Gen を通過してヘッダメモリにアクセスし，ヘッダ情報を読み出して DDP パケットとなる．このとき，rclk がアサートされてから読み出されたデータがヘッダメモリに到着して安定するまでは，hclk がアサートされてはならない．一方で，この回路は 2-phase ハンドシェイクにより転送を制御することから，send および ack が変化するたびにデータ転送が行われるため，それまでに hclk をアサートし，ヘッダ情報を読み出して生成したパケットが後段のステージに到着する必要がある．これらのタイミング制約を満たすために，遅延素子 $D_{dc}$ の遅延量を調節し，hclk のデューティ比を調整する必要がある．$D_{su}$ はローカルクロック信号の Setup 時間制約に関するタイミング保証用の遅延素子であり，この遅延量を調整してスループットを増減できる．

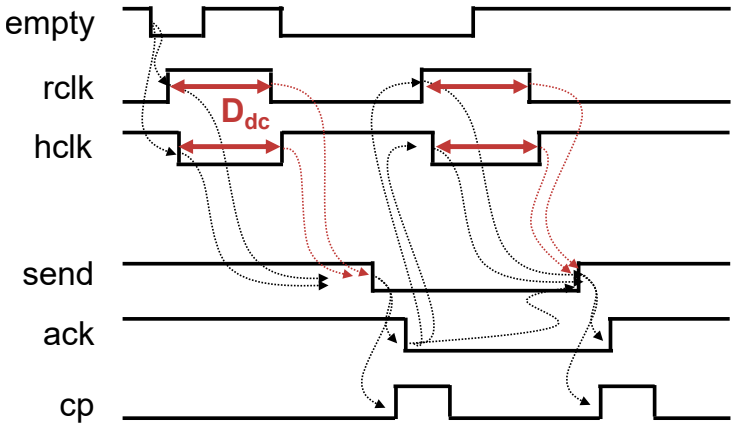


図 3.6 Input C-element (2-phase) のタイミングチャート

### 3.5.2 4-phase ハンドシェイクに対応した Input C-element

4-phase ハンドシェイクに対応した Input C-element を図 3.7 に示し，その動作を表すタイミングチャートを図 3.8 に示す.

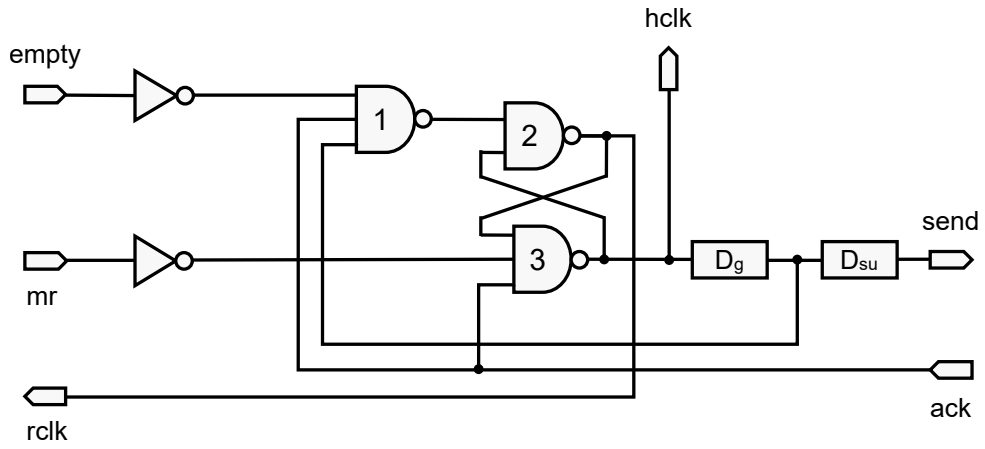


図 3.7 Input C-element (4-phase)

2-phase ハンドシェイクに対応した Input C-element と同様に，mr により回路を初期化後，Input FIFO の empty 信号がネゲートされていれば DDP への入力動作を開始し，Input FIFO に対する読み出しクロック rclk，およびヘッダメモリの読み出しクロック hclk をアサートしつつ，DDP を構成するセルフタイム回路との間で send，ack によるハンドシェイクを行う．この回路は 4-phase ハンドシェイクにより転送を制御するため，send および ack がアサート後，ネゲートされることでデータの転送が完了する.

ハンドシェイクの際に，NAND1 においてグリッチが発生する可能性がある．これを防ぐための遅延素子 $D_g$ を用いて遅延を挿入する．また，図 3.7 に示す回路構成では，内部の NAND2 および NAND3 が構成する SR-latch がグリッチを吸収することにより，3.2 で述べた empty のグリッチ (empty $0 \rightarrow 1 \rightarrow 0$) による回路の誤動作が発生しないように対応している．$D_{su}$ はローカルクロック信号の Setup 時間制約に関するタイミング保証用の遅延素子であり，この遅延量を調整してスループットを増減できる.

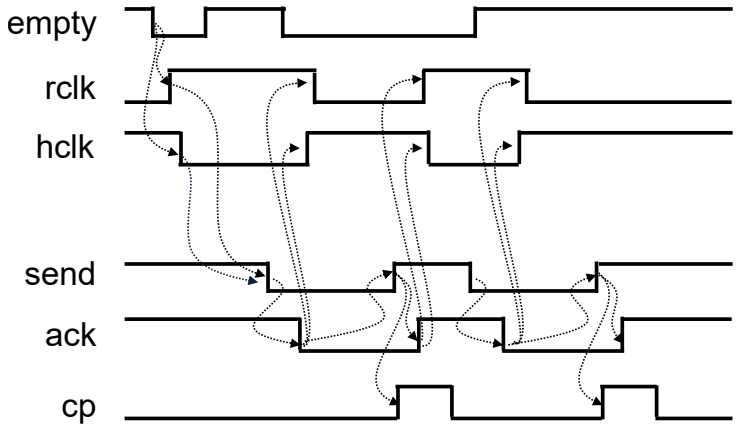


図 3.8 Input C-element (4-phase) のタイミングチャート

## 3.6 Output C-element

前節と同様に，Output C-element は，FIFO-DDP 間の転送制御プロトコルを変換し，DDP からデータを出力するためのモジュールである．本研究では，Output C-element についても 2-phase 用，および 4-phase 用の回路を入出力信号を揃えて設計した．

### 3.6.1 2-phase ハンドシェイクに対応した Output C-element

2-phase ハンドシェイクに対応した Output C-element を図 3.9 に示し，その動作を表すタイミングチャートを図 3.10 に示す．

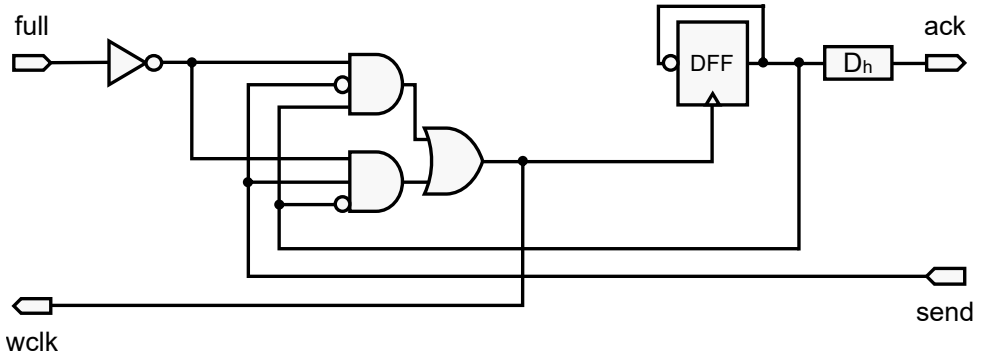


図 3.9 Output C-element (2-phase)

まず，mr により回路を初期化後，Output FIFO の full 信号がネゲートされている状態で，send がアサートされたときに DDP からの出力動作を開始し，Output FIFO に対する書き込みクロック wclk をアサートしつつ，DDP を構成するセルフタイム回路との間で send，ack によるハンドシェイクを行う．$D_h$ はローカルクロック信号の Hold 時間制約に関するタイミング保証用の遅延素子であり，この遅延量を調整してスループットを増減できる．

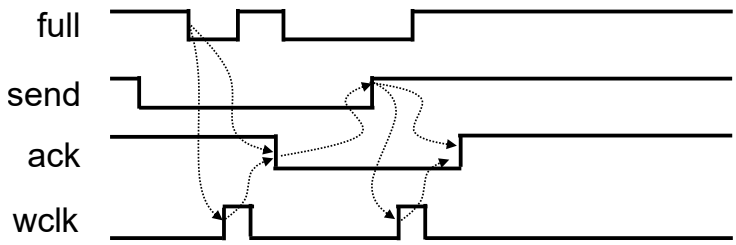


図 3.10 Output C-element (2-phase) のタイミングチャート

### 3.6.2 4-phase ハンドシェイクに対応した Output C-element

4-phase ハンドシェイクに対応した Output C-element を図 3.11 に示し，その動作を表すタイミングチャートを図 3.12 に示す.

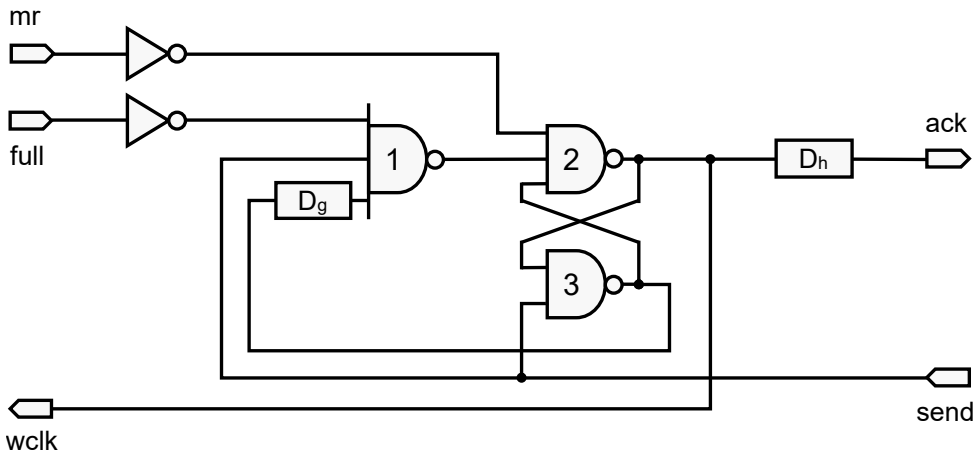


図 3.11 Output C-element (4-phase)

2-phase ハンドシェイクに対応した Output C-element と同様に，mr により回路を初期

化後，Output FIFO の full 信号がネゲートされている状態で，send がアサートされたときに DDP からの出力動作を開始し，Output FIFO に対する書き込みクロック wclk をアサートしつつ，DDP を構成するセルフタイム回路との間で send，ack によるハンドシェイクを行う．この回路は 4-phase ハンドシェイクにより転送を制御するため，send および ack がアサート後，ネゲートされることでデータの転送が完了する．

4-phase に対応した Input C-element と同様に，ハンドシェイクの際に，NAND1 においてグリッチが発生する可能性がある．これを防ぐための遅延素子 $D_g$ を用いて遅延を挿入する．$D_{su}$ はローカルクロック信号の Hold 時間制約に関するタイミング保証用の遅延素子であり，この遅延量を調整してスループットを増減できる．

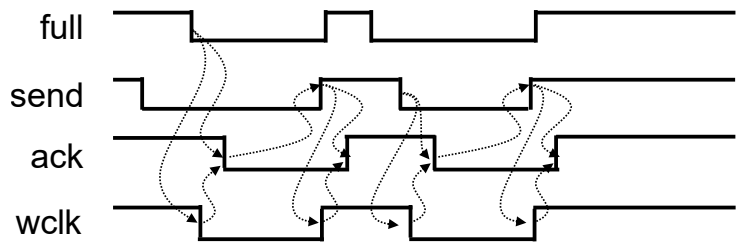


図 3.12　Output C-element (4-phase) のタイミングチャート

## 3.7　結言

本章では，機能性と信頼性を実現しつつ，各研究課題を解決するための，非同期 FIFO を用いた DDP I/F 回路の基本構成，および各モジュールの動作について述べた．Input C-element および Output C-element については，2-phase ハンドシェイクと 4-phase ハンドシェイクに対応した回路を設計し，それぞれの動作について説明した．また，スループットおよび信頼性の設計目標などの設計仕様変更に対応するための DDP I/F 回路の構成法について述べ，スループットおよび信頼性を独立して変更可能であることを示した．

本研究では，2-phase ハンドシェイクによりデータ転送を制御するセルフタイム回路である Click-element をもとにして，2-phase に対応した Input/Output C-element を設計

した．しかし，提案した基本回路では，FIFO からの入力である empty/full 信号のグリッチに対応できていないため，今後よりグリッチに対して頑健な 2-phase 用 Input/Output C-element の設計が必要である．

# 第 4 章

# 評価

## 4.1 緒言

本章では，非同期 FIFO を用いた提案 DDP I/F 回路構成法に加え，同期 FIFO を用いた DDP I/F 回路を設計して，AMD 社の FPGA チップに実装し，信頼性，スループット，リソース使用量について評価する．まず，IoT アプリケーションの典型的な設計仕様に対して評価を行い，その後，設計仕様の変更に対する評価を行う．

## 4.2 評価方法

本研究では，非同期 FIFO を用いた提案 DDP I/F 回路 (Async-FIFO I/F) に加え，同期 FIFO による DDP I/F 回路 (Sync-FIFO I/F) を設計して，AMD 社 FPGA チップである Zynq-7010(廉価版)，および ZU3EG(高性能版) に同社の EDA ツール Vivado (ML edition 2022.1) を用いて実装し，信頼性，スループット，リソース使用量について比較評価した．なお，Sync-FIFO I/F の構成については付録 A に示す．

非同期信号を同期化する n-flop synchronizer を FPGA 上に実装する際には，1 段目の DFF で発生したメタステーブル状態の信号を後段の DFF に伝搬させないために，各 DFF を可能な限り近くに配置し，配線遅延を最短にすることが重要である．本研究では，AMD 社の FPGA チップ上に Vivado を用いて実装する際，n-flop synchronizer を構成する各 DFF に async_reg 属性を付与することで，各 DFF 間の配線を最短にする．また，Input FIFO および Output FIFO を構成する Simple Dual-Port RAM は，AMD 社が提供する

IP(Intellectual Property) を使用し，FPGA 上の回路資源である Block RAM を利用する．Input C-element，および Output C-element の遅延素子は，LUT(Lookup Table) をカスケード接続することで実装する．

n-flop Synchronizer の信頼性の指標には，メタステーブル状態が発生し回路が誤動作するまでの平均時間である MTBF(Mean Time Between Failure) が使用され，式 (4.1) により求められる [10].

$$MTBF = \frac{e^{S/\tau}}{T_W \times F_C \times F_D} \tag{4.1}$$

ここで，$F_C$ はクロック周波数，$F_D$ はデータレート，$S$ はメタステーブル解消のために確保した時間，$T_W$ は setup/hold 時間，$\tau$ はメタステーブル解消に関連した時間を表している．$T_W$，および $\tau$ はデバイス固有の定数である．このように求められる MTBF と $F_C$ を用いると，式 (4.2) のように誤動作確率 $P_{mal}$ を求めることができる．本研究では，提案した構成法により構成した回路の信頼性を，n-flop synchronizer の誤動作確率により評価する．算出した値は，百万分率 ppm(parts per million) で表す.

$$P_{mal} = \frac{1}{MTBF \times F_C} \times 10^6 = \frac{T_W \times F_D}{e^{S/\tau}} \times 10^6 \tag{4.2}$$

式 (4.1)，および式 (4.2) に含まれるデバイス固有定数 $\tau$ は，事前実験により評価する．過去の研究 [11] で述べられているテスト回路を用いて，30 秒間のメタステーブル発生回数を実測し，測定時間 30 秒を発生回数で割ることにより，MTBF を算出する．実測時の $F_C$ や $F_D$，$S$，$T_W$，および算出した MTBF を用いると，式 (4.1) より $\tau$ を求めることができる．これを 5 回試行したときの $\tau$ の平均値を算出した．その結果，$F_C = 50MHz$, $F_D = 30MHz$, $S = 681.7ps$ のとき，$T_W = 322.0ps$ である DFF において，$\tau = 41.6ps$ であった．よって，本研究では，$\tau = 41.6ps$ を用いて各 I/F 回路の誤動作確率を算出する．

スループットについては，各 I/F 回路の最大パケット転送速度を実遅延シミュレーションにより評価し，1 秒間あたりの転送可能なパケット数を算出する．リソース使用量については，LUT，Register，Slice，および Block RAM の使用量を用いて評価する．

## 4.3　典型的な設計仕様における評価

まず，表 4.1 に示すような，IoT アプリケーションの典型的な設計仕様において，提案 Async-FIFO I/F および Sync-FIFO I/F で比較評価した．また，2-phase，および 4-phase ハンドシェイクの両方の回路について評価した．

表 4.1　典型的な設計仕様

| | |
|---|---|
| DDP 入出力パケット | 38 bits (16-bit data 含む) |
| Input/Output FIFO | 23bits × 32words |
| ヘッダメモリ | 22bits × 128words |
| 遅延素子の遅延量 (2-phase) | 12 LUTs (約 6 ns) |
| 遅延素子の遅延量 (4-phase) | 8 LUTs (約 4 ns) |
| 同期回路上のクロック周波数 | 50 MHz |

DDP 入出力パケットは，IoT 向けのアーキテクチャとして第 2 章で述べたパケットフォーマットを採用し，ヘッダ部を 22 bit，データ部を 16 bit とする 38 bit で構成した．Input/Output FIFO は，IoT 機器に取り付けたセンサのデータサンプリングレートが一般的な 1 kHz 程度であってもデータが溢れないように，どちらも 32 words のデータを格納可能にした．また，ヘッダメモリは実行する DDP プログラムに応じて変更可能となるように，十分なサイズとして 128 words のデータを格納可能にした．また，Input/Output C-element，DDP を構成する C 素子，および Click-element に含まれる遅延素子は，ローカルクロックの適切なタイミングを保証するために十分な遅延量として，2-phase 用で 12 LUTs (約 6 ns)，4-phase 用で 8 LUTs (約 6 ns) に設定した．ただし，同様にローカルクロックの適切なタイミングを保証するために，2-phase 用の Input C-element に含まれる遅延素子 $D_{dc}$ は，その半分の 6 LUTs (約 3 ns) に設定した．同期回路上のクロック周波数は，一般的な IoT 用途を想定し，50 MHz とした．

### 4.3.1　信頼性

前節で述べた誤動作確率を求める式 (4.2) を用いて n-flop synchronizer の DFF が 2 段である 2-flop，および 3-flop の信頼性を評価した．このとき，どちらも $\tau = 41.6ps$，$F_D = 100MHz$ であり，2-flop，3-flop における $S$ はそれぞれ 18107.4 ps，37340.6 ps，$T_W$ の平均はそれぞれ 278.5 ps，276.0 ps であった．平均的なデータレート $F_D$ は，DDP を含めたボトルネックとなるパイプラインステージのデータ転送時間に依存して決まる．本研究では，最悪のケースの評価として，$F_D = 100MHz$ とした．Async-FIFO I/F と Sync-FIFO I/F はどちらも，1 データ転送時に n-flop synchronizer を通過する非同期信号は 1 ビットのみであるため，信頼性は同等である．評価の結果，DFF 段数が 2 段である 2-flop は $2.6 \times 10^{-185}$ ppm，3-flop は $4.1 \times 10^{-386}$ ppm であった．一般的な IoT アプリケーションに要求される 10 ppm を目標とした場合，クロック周波数 50 MHz の条件下では，2-flop で十分な信頼性を達成できた．

### 4.3.2　スループットおよびリソース使用量

2-flop synchronizer を組み込んだ Async-FIFO I/F と Sync-FIFO I/F のスループット，およびリソース使用量を比較した．その結果，表 4.2 に示す結果が得られた．2-phase の場合，Input I/F および Output I/F の両方で，Async-FIFO I/F の方が Sync-FIFO I/F よりも Slice 数が 2 倍程度に増加するが，スループットがそれぞれ 4.1 倍，3.7 倍に向上し，Slice 数あたりのスループット (Throughput/Slice) は，どちらも Async-FIFO I/F の方が高く優れていることが示された．また，4-phase においても，Input I/F および Output I/F の両方で同様の結果を確認し，スループットはそれぞれ 3.8 倍，4.4 倍を達成した．Slice 数あたりのスループット (Throughput/Slice) に関しても，どちらも Async-FIFO I/F の方が高く優れていることが示された．

表 4.2　スループットおよびリソース使用量の比較 (Zynq-7010)

| | 2-phase | | | | 4-phase | | | |
|---|---|---|---|---|---|---|---|---|
| | Input I/F | | Output I/F | | Input I/F | | Output I/F | |
| FIFO type | Async | Sync | Async | Sync | Async | Sync | Async | Sync |
| Throughput[M pps] | 51.4 | 12.5 | 46.8 | 12.5 | 31.6 | 8.3 | 36.7 | 8.3 |
| LUT | 38 | 16 | 37 | 19 | 36 | 16 | 33 | 19 |
| Register | 25 | 16 | 25 | 16 | 24 | 15 | 24 | 15 |
| Slice | 16 | 9 | 16 | 8 | 14 | 8 | 15 | 8 |
| Block RAM | 1 | 1 | 0.5 | 0.5 | 1 | 1 | 0.5 | 0.5 |
| Throughput/Slice | 3.2 | 1.4 | 2.9 | 1.6 | 2.3 | 1.0 | 2.4 | 1.0 |

## 4.4　設計仕様の変更に対する評価

次に，前節の典型的な設計仕様を基準として，スループットおよび信頼性の設計目標，応用領域に特化したアーキテクチャ，FPGA チップのグレードを変更した場合の性能について評価した．

### 4.4.1　スループットおよび信頼性の設計目標の変更

前節のスループットおよび信頼性を 100 %として，それらの設計目標の変更要求に対して適応可能な性能について評価した．図 4.4.1 は，スループットの設計目標変更に対して適応可能な性能を示したグラフであり，スループット目標を x 軸，そのスループット目標が与えられたときに実現可能なスループットを y1 軸，信頼性を y2 軸としている．図 4.4.1 の上段に示すグラフは 2-phase，下段に示すグラフは 4-phase の結果を表しており，左側に示すグラフは Input I/F，右側に示すグラフは Output I/F の結果を表している．この結果より，2-phase および 4-phase のどちらにおいても，Input I/F，Output I/F の両 I/F とも

に 80 %から 120 %の範囲でスループット変更に適応可能である．加えて，Sync-FIFO I/F はスループットに対して信頼性が反比例して変化するのに対し，Async-FIFO I/F は信頼性が一定である．

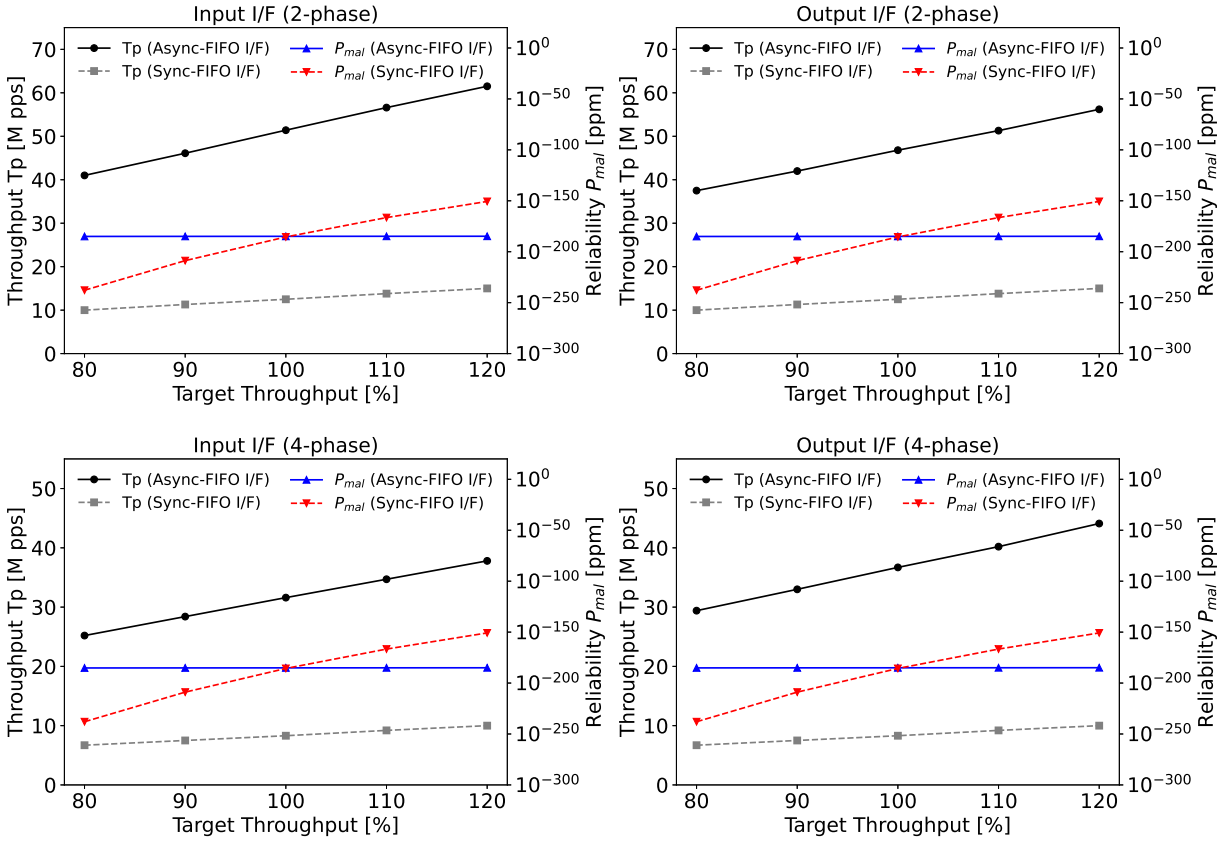


図 4.1　スループット目標変更に対して適応可能な性能 (Zynq-7010)

また，図 4.4.1 は，同様に信頼性目標を変更した際に適応可能な性能について示したグラフである．この結果より，信頼性目標を変更した場合，2-phase および 4-phase のどちらにおいても，両 I/F ともに 25 %から 400 %の範囲で信頼性変更に適応可能である．加えて，スループット目標を変更した場合と同様に，Sync-FIFO I/F は信頼性が向上するとスループットが低下する一方で，Async-FIFO I/F はスループットが一定であることが確認できる．なお，図 4.4.1 では表示できるスケールの都合上，信頼性を示す誤動作確率 $P_{mal}$ を $10^{100}$ 倍して表示している．

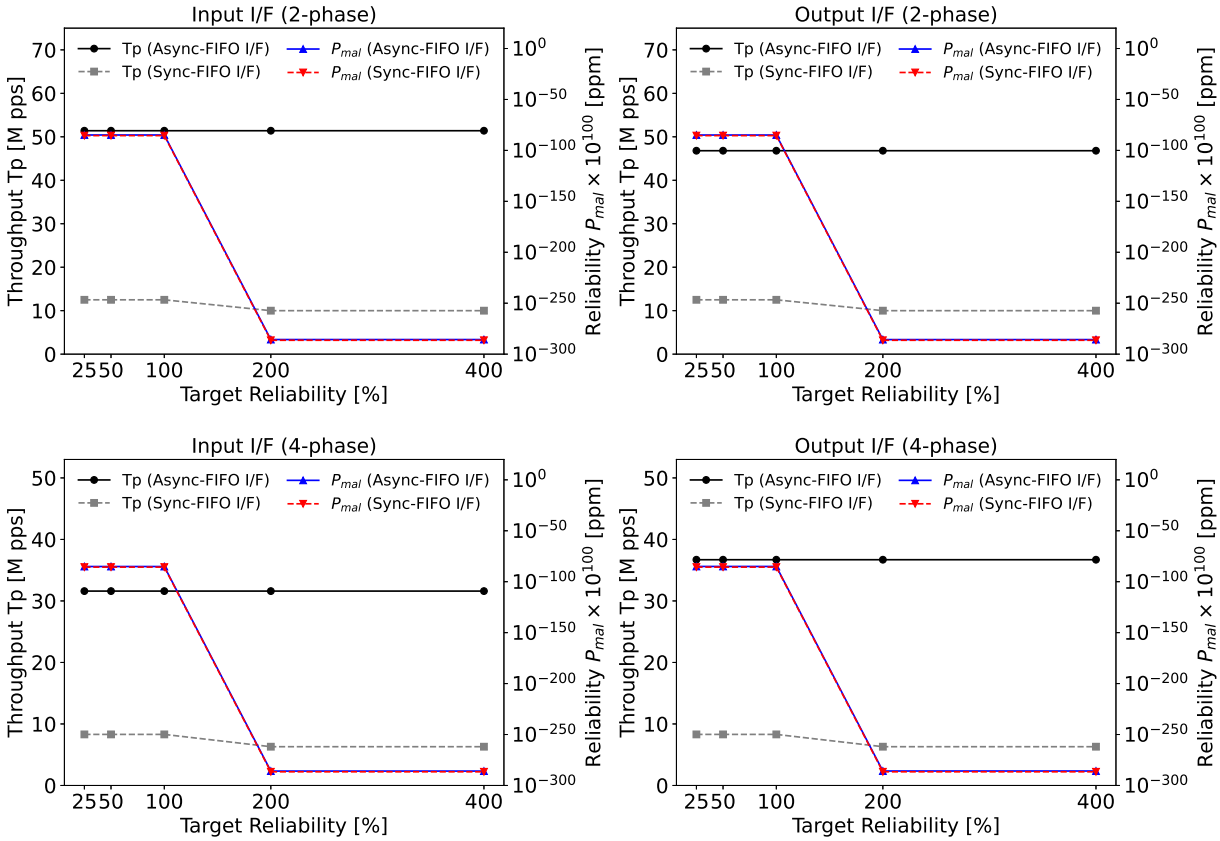


図 4.2　信頼性目標変更に対して適応可能な性能 (Zynq-7010)

### 4.4.2　応用領域に特化したアーキテクチャへの変更

同様に，データやメモリサイズに関するスケールを変更した時の性能について評価した．本研究では，DDP 入出力パケットのデータサイズ，Input/Output FIFO サイズ，ヘッダメモリサイズをスケール変更の対象としており，表 4.1 に示す典型的な設計仕様を 100 %として，スケールを 25 %から 400 %に変更する．これにより設定した，各スケールごとのデータやメモリのサイズを表 4.3 に示す．

図 4.4.2 は，これらのスケールを x 軸とし，そのスケールで構成した際に得られるスループットを y1 軸，信頼性を示す誤動作確率を y2 軸に設定したときの評価結果を示している．この結果より，2-phase および 4-phase のどちらにおいても，両 I/F ともに 25 %から 400 %のスケール変更に適応可能である．また，Async-FIFO I/F のスループットはスケール変更時に多少の変動があるものの，25 %から 400 %の範囲では Sync-FIFO I/F と比較

表 4.3　スケールごとのデータサイズおよびメモリサイズ

| | 25 % | 50 % | 100 % | 200 % | 400 % |
|---|---|---|---|---|---|
| DDP 入出力パケット データサイズ [bits] | 4 | 8 | 16 | 32 | 64 |
| Input/Output FIFO サイズ [words] | 8 | 16 | 32 | 64 | 128 |
| ヘッダメモリサイズ [words] | 32 | 64 | 128 | 256 | 512 |

して少なくとも 3 倍以上のスループットが安定して得られることを確認した．Async-FIFO I/F では，FPGA 上への配置・配線結果の変動により，ハンドシェイクにかかる時間が変動することでスループットが変化したと考えられる．スループットの変動に対して，スケール変更が与える影響については，より詳細な評価が必要となる．

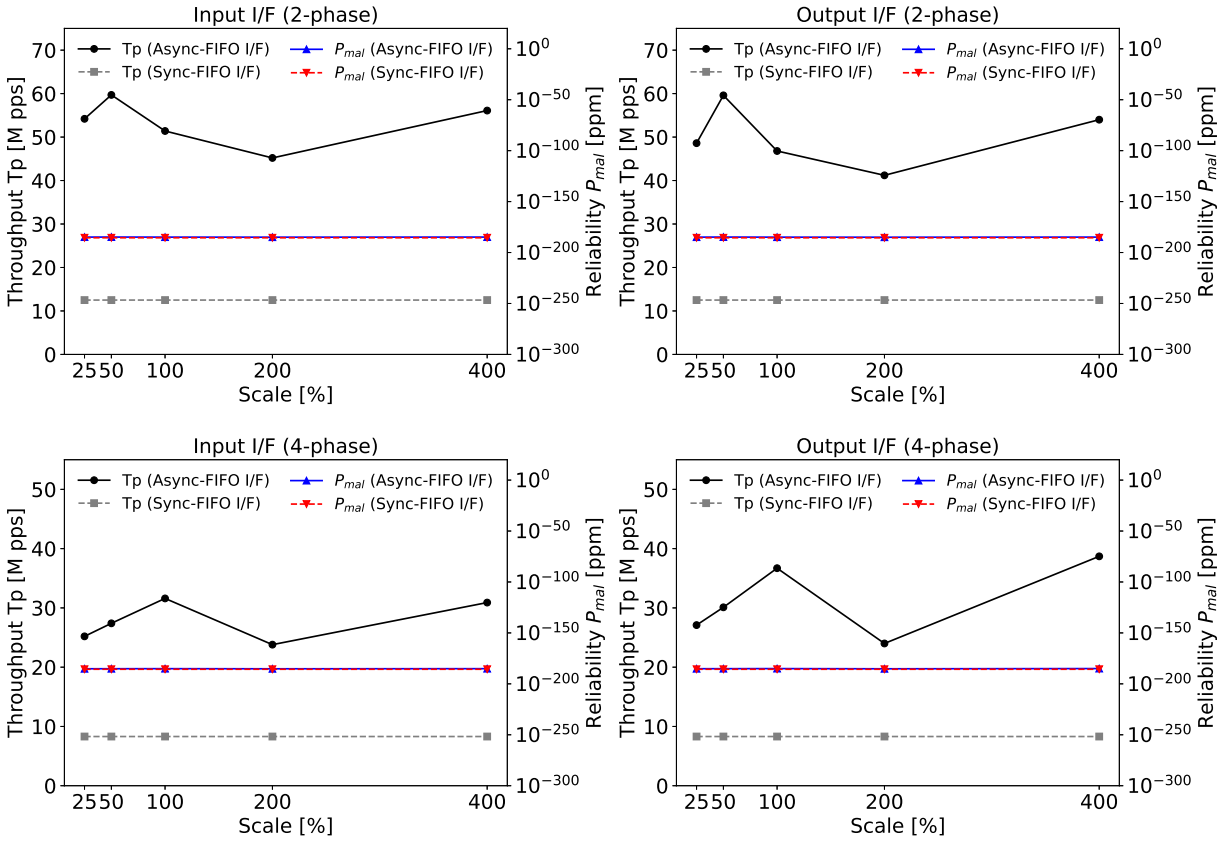


図 4.3　スケール変更に対して適応可能な性能 (Zynq-7010)

### 4.4.3　FPGA チップのグレードの変更

Zynq-7010 より高性能な FPGA チップである Zynq Ultrascale+ デバイスである ZU3EG を用いて，同様に評価を行った．

#### 典型的な設計仕様における評価

まず，表 4.1 に示す典型的な設計仕様で提案 Async-FIFO I/F および Sync-FIFO I/F を ZU3EG 上に実装して比較評価する．ただし，デバイスの高性能化に伴い，LUT 等の回路素子の内部遅延や配線遅延の減少し，ローカルクロックの適切なタイミングを保証されない可能性があるため，2-phase 用の遅延素子は 24 LUTs，4-phase 用の遅延素子は 16 LUTs に設定した．Zynq-7010 上への実装時と同様に，Input C-element に含まれる遅延素子 $D_{dc}$ は，その半分の 12 LUTs に設定した．また，デバイスの高性能化に伴って同期回路上のクロック周波数も向上させることを想定し，100 MHz に設定した．

信頼性については，Zynq Ultrascale+ デバイスに対して回路を実装後，Vivado が提供している report_synchronizer_MTBF コマンドを実行することで，実装した synchronzier の MTBF レポートを取得して活用する．クロック周波数の向上と同様に，データレート $F_D$ も最悪のケースを想定して 200 MHz に設定して評価を行った．その結果，信頼性を示す誤動作確率は，2-flop で $3.0 \times 10^{-153}$ ppm，3-flop で $4.2 \times 10^{-130}$ となり，ZU3EG 上への実装ではクロック周波数 100 MHz の条件下でも 2-flop で十分な信頼性を達成できた．

信頼性評価結果を踏まえて，Async-FIFO I/F，および Sync-FIFO I/F で用いられる n-flop synchronizer として，2-flop synchronizer を採用し，スループットおよびリソース使用量について評価した結果を表 4.4 に示す．なお，Zynq Ultrascale+ デバイスでは Slice の代わりに，隣接する 2 つの Slice を 1 つとして扱う CLB (Configurable Logic Block) を用いるため，表 4.4 でも Slice 使用量の代わりに CLB 使用量を示している．この結果より，Zynq-7010 と同様の評価結果を確認し，特に CLB 数あたりのスループット (Throughput/CLB) という観点で，提案 Async-FIFO I/F の方が，Sync-FIFO I/F より高く優れていることが

示された．また，Zynq-7010 上への実装時より優れたスループットが得られ，FPGA チップのグレード変更にも適応可能であることが示された．

表 4.4　スループットおよびリソース使用量の比較 (ZU3EG)

| | 2-phase | | | | 4-phase | | | |
|---|---|---|---|---|---|---|---|---|
| | Input I/F | | Output I/F | | Input I/F | | Output I/F | |
| FIFO type | Async | Sync | Async | Sync | Async | Sync | Async | Sync |
| Throughput[M pps] | 133.2 | 25.0 | 123.0 | 25.0 | 66.2 | 16.7 | 108.6 | 16.7 |
| LUT | 51 | 16 | 42 | 19 | 35 | 15 | 35 | 18 |
| Register | 25 | 16 | 25 | 16 | 24 | 15 | 24 | 15 |
| CLB | 11 | 6 | 8 | 5 | 7 | 5 | 8 | 4 |
| Block RAM | 1 | 1 | 0.5 | 0.5 | 1 | 1 | 0.5 | 0.5 |
| Throughput/CLB | 12.1 | 4.2 | 15.4 | 5.0 | 9.5 | 3.3 | 13.6 | 4.2 |

**スループットおよび信頼性の設計目標変更に対する評価**

次に，スループットおよび信頼性の設計目標を 100 %として，それらの設計目標の変更要求に対して適応可能な性能について評価した．図 4.4.3 は，スループットの設計目標変更に対して適応可能な性能を示したグラフであり，スループット目標を x 軸，そのスループット目標が与えられたときに実現可能なスループットを y1 軸，信頼性を y2 軸としている．この結果から，2-phase および 4-phase のどちらにおいても，両 I/F ともに 80 %から 120 %の範囲でスループット変更に適応可能であることが示された．また，Zynq-7010 上への実装時と同様に，Sync-FIFO I/F はスループットに対して信頼性が反比例して変化するのに対し，Async-FIFO I/F は信頼性が一定であることが確認できる．

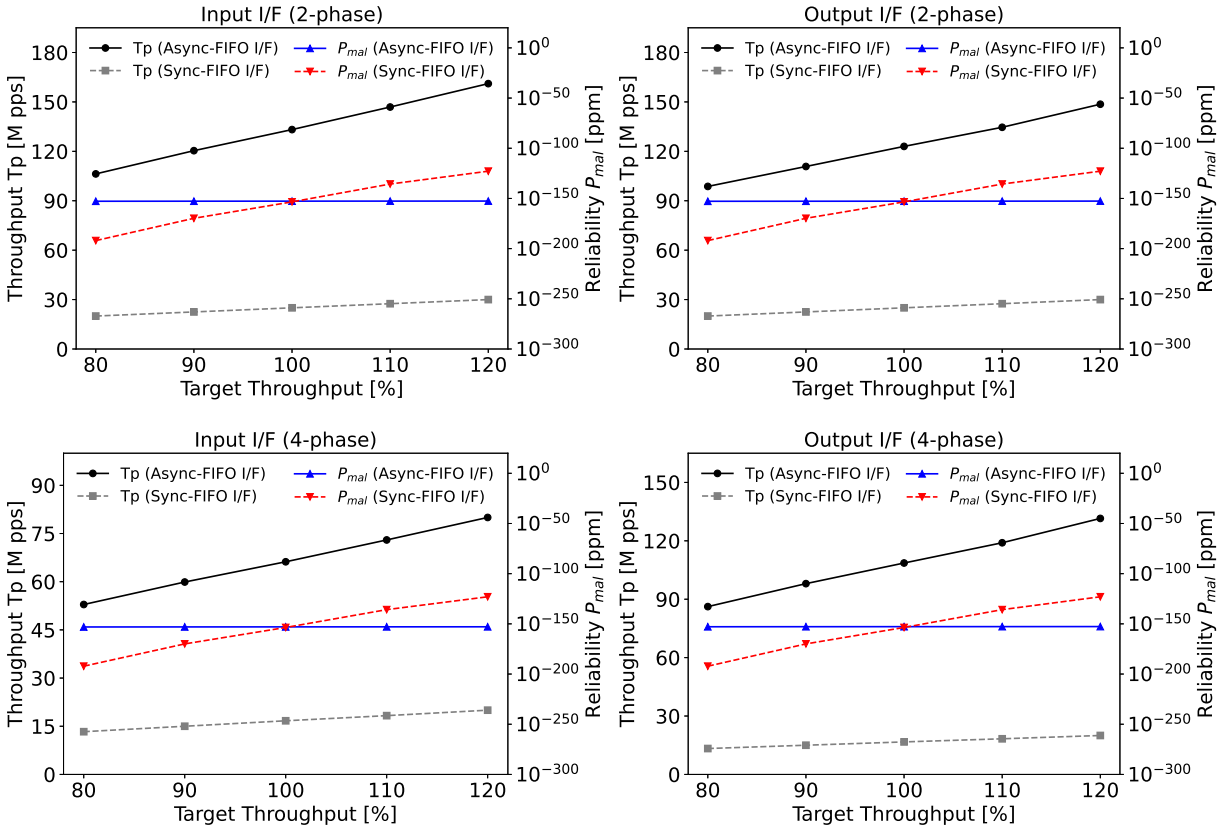


図 4.4　スループット目標変更に対して適応可能な性能 (ZU3EG)

また，図 4.4.3 は，信頼性目標を変更した際に適応可能な性能について示したグラフである．この結果より，信頼性目標を変更した場合，2-phase および 4-phase のどちらにおいても，両 I/F ともに 25 %から 400 %の範囲で信頼性変更に適応可能であることが示された．加えて，スループット目標を変更した場合と同様に，Sync-FIFO I/F は信頼性が向上するとスループットが低下する一方で，Async-FIFO I/F はスループットが一定であることが確認できる．また，これらの結果から，Zynq-7010 上への実装時より高いスループットが得られ，FPGA チップのグレードを変更しても同様に，ある程度の設計目標の変更に対して適応可能であることが示された．

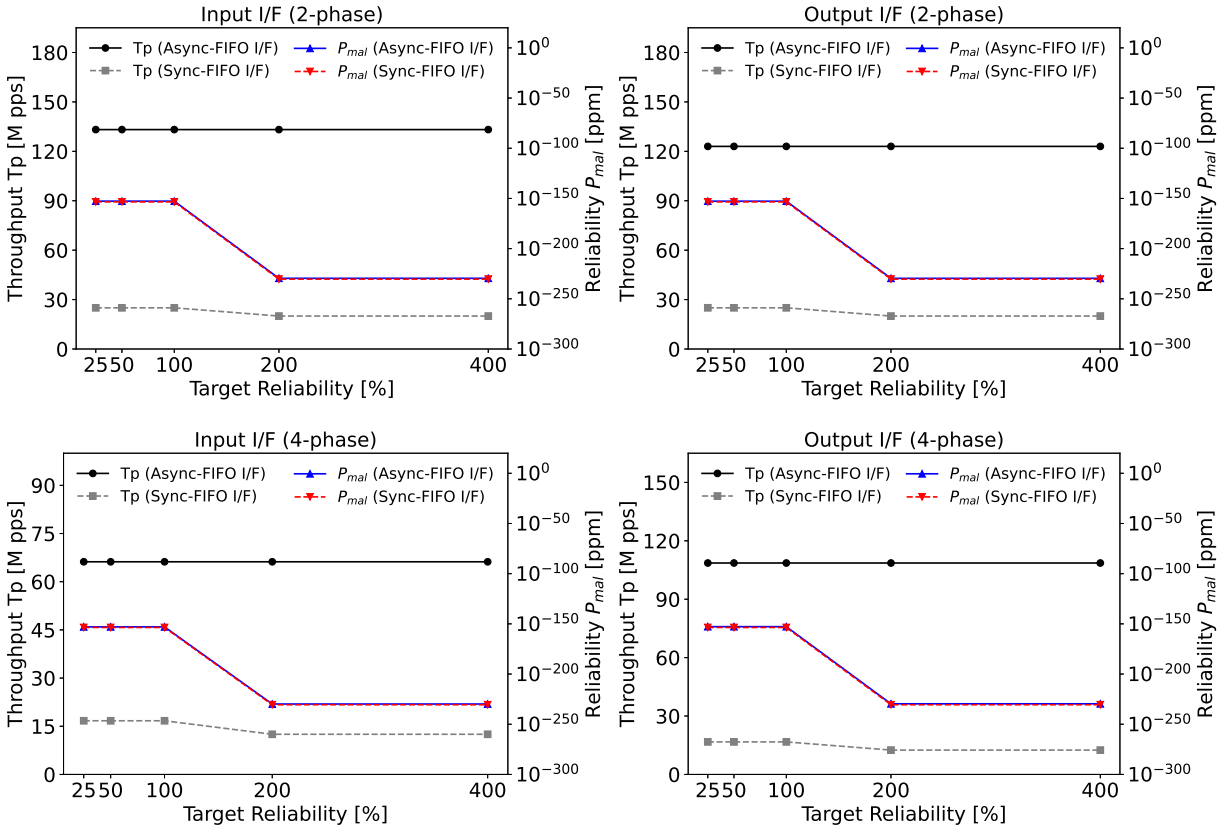


図 4.5　信頼性目標変更に対して適応可能な性能 (ZU3EG)

**応用領域に特化したアーキテクチャへの変更に対する評価**

最後に，データやメモリサイズに関するスケールを変更した時の性能について評価した．表 4.3 に示すスケールを x 軸とし，そのスケールで構成した際に得られるスループットを y1 軸，信頼性を示す誤動作確率を y2 軸に設定したときの評価結果を図 4.4.3 に示す．この結果より，2-phase および 4-phase のどちらにおいても，両 I/F ともに 25 %から 400 %のスケール変更に適応可能である．また，Async-FIFO I/F のスループットはスケール変更時に多少の変動があるものの，25 %から 400 %の範囲では Sync-FIFO I/F と比較して少なくとも 3 倍以上のスループットが安定して得られることを確認した．これらの評価結果から，FPGA チップのグレードを変更しても，ある程度のスケール変更に対して適応可能であることが示された．また，ZU3EG 上への実装により，Zynq-7010 よりも高い性能が得られ，より高いスケーラビリティを有することが確認できた．

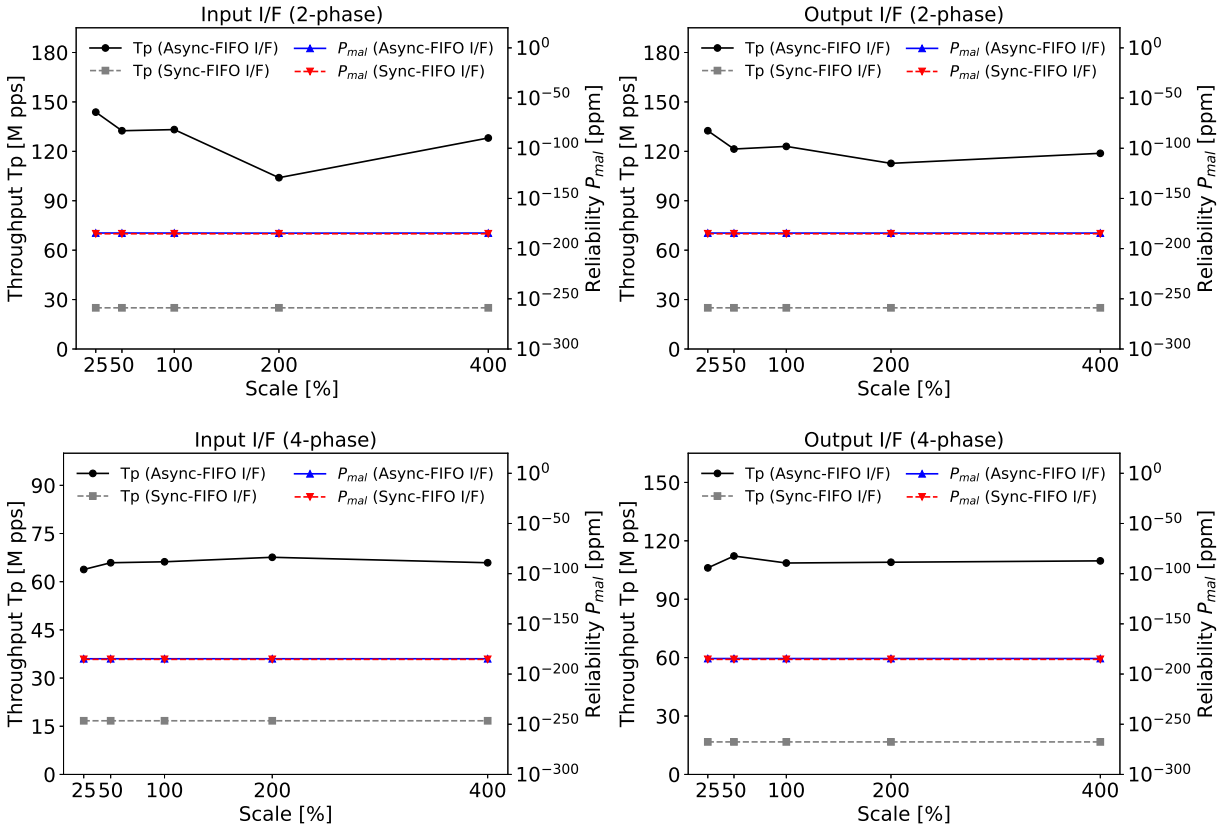


図 4.6　スケール変更に対して適応可能な性能 (ZU3EG)

## 4.5　結言

本章では，非同期 FIFO を用いた提案 DDP I/F 回路構成法に加え，同期 FIFO を用いて DDP I/F 回路を設計して，AMD 社の FPGA チップに実装し，信頼性，スループット，リソース使用量について評価した．まず，IoT アプリケーションの典型的な設計仕様に対して評価を行い，リソースあたりのスループットという観点で Async-FIFO I/F の方が優れていることを示した．その後，設計仕様の変更に対する評価を行い，Sync-FIFO I/F はスループットに対して信頼性が反比例して変化するのに対し，Async-FIFO I/F は信頼性が一定のままスループットを向上させることができることを示し，より優れたスケーラビリティを有することも確認した．

本研究では，スループットの設計目標変更は 80 %から 120 %，信頼性の設計目標変更，およびスケール変更は 25 %から 400 %の範囲についてのみ評価した．今後，これらの範囲

外に対しても評価を行い，提案 DDP I/F 回路構成法の汎用性を確認する必要がある．また，実用化を考えると，AP-DDP を連携したシステム全体の評価，および AP のみで実現したシステムの評価を行い，性能や消費電力に関する評価も行う必要がある．

# 第 5 章

# 結論

IoT 技術の普及と発展により，工場や農業，自動車などの様々な分野で IoT 技術が活用されており，IoT エッジ機器には各応用分野に特化したアーキテクチャの採用による，柔軟な機能性，高性能化，省電力動作が要求される．セルフタイム回路により構成されたデータ駆動型プロセッサ DDP はデータの多重並列処理により高性能化を実現しつつ，データの到着時にのみ動作することで省電力な動作が期待でき，柔軟な機能性を提供可能な FPGA 上に実装することで，IoT エッジ機器のコアとして有望である．

非同期回路である DDP は，従来の同期回路によるアプリケーションプロセッサ AP 上で動作する既存のソフトウェアを再利用することで開発コストを削減できるため，AP と連携することが実用的である．この場合，AP-DDP 間において動作タイミングを同期し，DDP の処理能力に応じて入出力を行うための，データおよび転送制御プロトコル変換回路を有する DDP I/F 回路が必要となる [2]．また，これらの機能の実現に加え，FPGA の柔軟性を活かして，様々な設計仕様の変更に対して適応可能な DDP I/F 回路の構成法を確立することで，AP-DDP 連携システムをより汎用的に活用することができる．設計仕様の変更としては，スループットや信頼性等の設計目標，応用領域に特化したアーキテクチャ，あるいは，FPGA チップのグレードなどの変更が考えられる．

非同期信号の同期化手法や同期-非同期回路間の I/F 回路は各所で研究されており，n-flop synchronizer[3][4]，非同期 FIFO[5]，pausible clock[6], click-element を用いた同期-非同期回路間 I/F[8] などが挙げられる．これらの手法は，動作タイミングの同期，バッファリング，データおよびプロトコル変換といった機能を持っているが，これらの全ての機能を持つ I/F 回路については研究されていない．特に，上記の既存手法を活用する場合，FIFO の種類に

合わせたプロトコル変換回路をデータ転送時間が調整できるように設計する必要がある．また，n-flop synchronizer を適切な位置に組み込むことで，必要な機能性および信頼性を実現しつつ，スループットや信頼性を独立して調整できる回路構成法が必要となるため，これを検討した．

提案する構成法により典型的な設計仕様に基づいて構成した DDP I/F 回路 (Async-FIFO I/F) は，非同期 FIFO を用いて構成され，AMD 社 FPGA チップである Zynq-7010 に実装して評価した結果，同期 FIFO を用いて構成した場合 (Sync-FIFO I/F) と比較して少なくとも 3.7 倍以上のスループットを達成でき，リソースあたりのスループットの観点でも高く優れていることが示された．また，設計仕様を変更した場合の性能についても評価した結果，Sync-FIFO I/F はスループットが信頼性に反比例して変化するのに対し，提案 Async-FIFO I/F はスループットと信頼性を独立して調整でき，より優れたスケーラビリティを有することが確認できた．

以下に，本研究の今後の課題を示す．

- 各プロトコル変換回路の FPGA 配置方法の検討

  本研究におけるスケール変更時の評価結果から，FPGA 実装時の回路の配置結果による配線遅延の違いがスループットに影響を与えていると考えられる．そのため，今後，提案 DDP I/F 回路の基本回路，特に Input/Output C-element の FPGA 配置方法を検討し，可能な限り小さな回路規模で高性能な DDP I/F 回路を実現する方法を検討する必要がある．
- 2-phase 用 Input/Output C-element における empty/full のグリッチへの対応

  本研究において提案した 4-phase 用 Input/Output C-element は，Input/Output FIFO が出力する empty/full フラグのグリッチに対応したが，2-phase 用 Input/Output C-element は対応できていない．誤動作につながる可能性があるため，実用的にはグリッチの影響を受けない構成を検討する必要がある．
- 消費電力に関する評価

本研究では未着手であったが，実用の際には消費電力も考慮する必要がある．一般的には，リソース使用量や回路規模と消費電力の間には相関があるため，リソース使用量を用いて推定することも可能である．本研究の場合，相対的には Async-FIFO I/F のリソース使用量が Sync-FIFO I/F の 2 倍程度になったが，その増加量を実際の DDP のリソース使用量と比較すると，1%程度である．そのため，消費電力はほとんど変わらず，一般的な IoT アプリケーションを想定しても，その差は許容範囲であると考えられる．また，今後は，実用を想定した IoT エッジデバイス全体の消費電力を評価する必要がある．AP のみでアプリケーションを実現した場合と，AP-DDP 連携システムで実現した場合を考えると，その性能や消費電力はアプリケーションごとに使い分けることが望ましいと考えられる．特に，AP-DDP を連携する場合，AP-DDP 間のデータ転送時間は AP のみで実現する場合と比較してオーバヘッドになる．また，DDP は非同期タイミングでデータが到着する場合に導入するメリットが増大するため，DDP 上で動作させるプログラムが比較的大きなものである場合で特に有用であると考えられる．

- 実用的なプログラムを実行する DDP と連携させたテスト

  本研究では AP-DDP との連携を想定しているが，シミュレーションによる評価では，実際にプログラムを実行する DDP と連携させたテストが未着手である．今後は，実用的なプログラムを実行する DDP と連携させ，動作確認，および本研究における評価結果の検証を行う必要がある．

今後，以上の課題を解決することにより，AP-DDP 連携システムの実用化が可能となる．本研究における提案 DDP I/F 回路構成法は，DDP 以外のセルフタイム回路を用いた非同期回路にも適応可能であると思われる．タイミング同期，バッファリング，データおよびプロトコル変換といった機能は，基本的にはどのような非同期回路にも共通で必要になることが多く，性能の設計目標やスケールを調整可能であるため，利用する回路に合わせた I/F 回路を構成できると考えられる．

リソースあたりのスループットという観点では Async-FIFO I/F の方が優れていたが，

Sync-FIFO I/F は同期回路であり，EDA ツールを用いて自動でタイミング検証を行うことが出来るため，Async-FIFO I/F より設計が容易であるというメリットが存在する．このことから，バッチ処理により非同期回路の動作テストを行う場合など，よりよい性能が要求されない場面では活用が期待できると考えられる．このように，今後は性能や消費電力以外の観点も踏まえた総合的な評価を行うことで，DDP に限らず，AP と非同期回路を連携させる際の I/F 回路構成法として適応範囲を広げ，非同期回路の活用の基礎となり得ると考えられる．

# 謝辞

本研究を進めるにあたり，自らの貴重なお時間を割いていただき，様々なご指導をいただきました岩田誠教授には深く感謝申し上げます．研究内容の相談だけでなく，発表練習や論文執筆の際にもたくさんのアドバイスをいただいたことで，無事本研究を進めることができました．今後も，本研究を通して学んだことを活かしつつ，日々精進していきたいと思います．

ご多忙の中，本研究の副査をお引き受け下さり，また，様々な疑問点を指摘していただいた福本昌弘教授，および松崎公紀教授に心より感謝申し上げます．

張震氏，Tamnuwat Valeeprakhon 氏には研究室の先輩として，研究で行き詰まったときに相談に乗っていただきました．非常に居心地良く，研究に取り組みやすい環境でした．心より感謝申し上げます．

坂口白磨氏，松坂拓海氏には研究室の同輩として，切磋琢磨しながらも，励まし合いながら研究に取り組めたことに大変感謝しています．研究の中で分からない点を聞き合い，知識を共有できたため，効率よく研究を進めることができました．これからも，良き仲間として互いに頑張りましょう．

市ノ木一希氏，伊藤雅俊氏，岡村健勝氏，山下拓巳氏には研究室の後輩として，そして同じ修士過程の学生として日頃からご支援ご協力いただきました．心より感謝申し上げます．

奥平舜理氏，大崎絢斗氏，片岡拓心氏，門屋陽丈氏，山崎浩正氏，苅田勇人氏には研究室の後輩として，日頃からご支援ご協力いただきました．心より感謝申し上げます．

両親には，研究に集中できる環境を提供していただき，無事に本研究をやり遂げることができました．心より感謝申し上げます．今後は社会人として社会に貢献できるよう，より一層成長できるよう励みたいと思いますので，これからもよろしくお願いします．

多大なる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] 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, 1999.

[2] R. Uemoto and M. Iwata, “Reliable I/F Circuit for Embedded Self-Timed Data-Driven Processors,” The 22nd International Conference on Embedded Systems, Cyber-physical Systems, and Applications (ESCS’24), 2024.

[3] S. Beer and R. Ginosar, “Eleven Ways to Boost Your Synchronizer,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 6, pp. 1040–1049, 2015.

[4] R. Ginosar, “Fourteen Ways to Fool Your Synchronizer,” Ninth International Symposium on Asynchronous Circuits and Systems, pp. 89–96, 2003.

[5] Z. Hao, L. Liu, and B. Tian, “The Principle and Applications of Asynchronous FIFO,” IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA), pp. 277–279, 2023.

[6] D. L. Oliveira, T. Curtinhas, L. A. Faria, and L. Romano, “A Novel Asynchronous Interface with Pausible Clock for Partitioned Synchronous Modules,” IEEE 6th Latin American Symposium on Circuits & Systems (LASCAS), pp. 1–4, 2015.

[7] D. L. Oliveira, T. Curtinhas, V. L. V. Torres and L. Romano, “Design of Asynchronous Wrappers for High-Concurrency Multi-Point GALS Systems,” 2018 IEEE ANDESCON, Santiago de Cali, Colombia, pp. 1-6, 2018.

[8] S. Semba and H. Saito, “A Study on the Design of Interface Circuits Between Synchronous-Asynchronous Modules Using Click Elements,” SASIMI, pp. 139–144, 2022.

[9] B. Keller, M. Fojtik and B. Khailany, “A Pausible Bisynchronous FIFO for GALS Systems,” 2015 21st IEEE International Symposium on Asynchronous Circuits and Systems, Mountain View, CA, USA, pp. 1-8, 2015.

[10] C. Dike and E. Burton, “Miller and Noise Effects in a Synchronizing Flip-Flop,” IEEE Journal of Solid-State Circuits, Vol. 34, No. 6, pp. 849–855, 1999.

[11] S. Beer, R. Ginosar, M. Priel, R. Dobkin and, A. Kolodny, “The Devolution of Synchronizers,” IEEE Symposium on Asynchronous Circuits and Systems, pp. 94–103, 2010.

# 付録 A

# Sync-FIFO I/F

## A.1　Sync-FIFO I/F の基本構成

Sync-FIFO I/F の基本構成を図 A.1 に示す．基本的な構成は DDP I/F 回路と同様であるが，Sync-FIFO I/F は同期 FIFO ベースの DDP I/F 回路であり，同期 FIFO で構成する Input/Output FIFO に接続する信号は同期化された信号となる必要があるため，図 A.1 のように各モジュールには AP 側から DDP 側に送られる同期回路上のクロック信号が入力されている．また，それに伴い，不要となるヘッダメモリの読み出しクロック信号 hclk を削除し，Input/Output C-element から出力される Input/Output FIFO の読み書きクロック信号 rclk/wclk の代わりに，それぞれ読み書き許可信号 rden/wren を追加している．

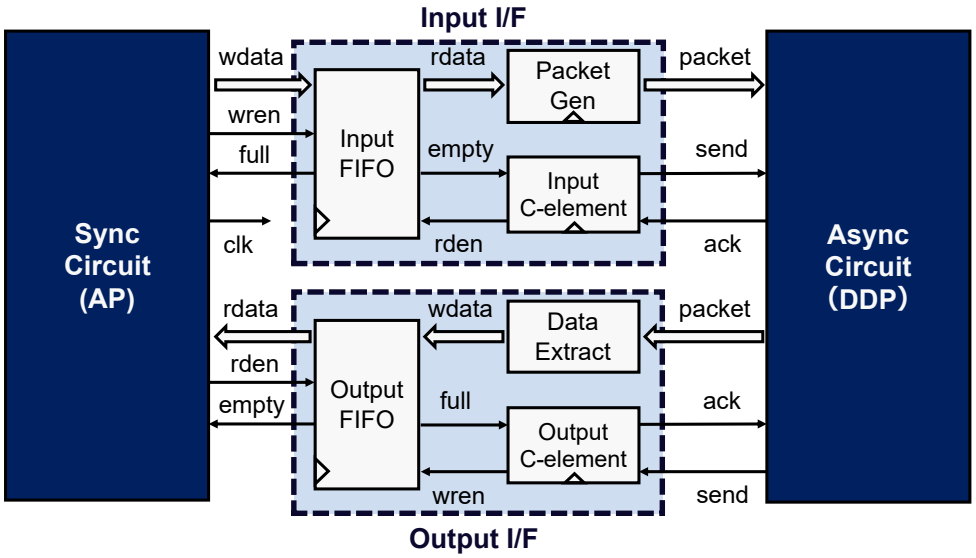


図 A.1　Sync-FIFO I/F の基本構成

## A.2　Sync-FIFO I/F を構成する Input C-element

Sync-FIFO I/F を構成する Input C-element についても，2-phase および 4-phase のどちらにも対応可能とするために，それぞれの回路を設計した．2-phase，4-phase 用の回路の構成を，それぞれ図 A.2，A.3 に示す．Sync-FIFO I/F では，動作タイミングを同期するための n-flop synchronzier を Input C-element 内に組み込み，DDP から出力された転送許可信号 ack を同期化し，その信号を用いて Input FIFO の読み出し許可信号 rden を生成する．

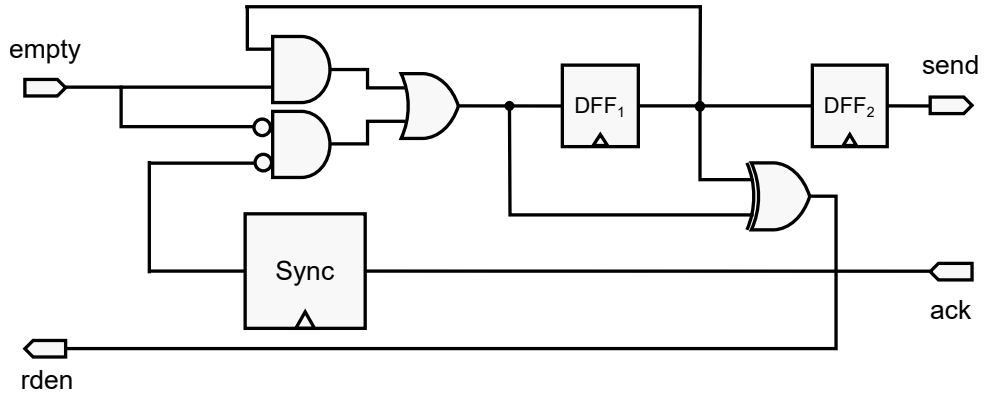


図 A.2　Sync-FIFO I/F における Input C-element (2-phase)

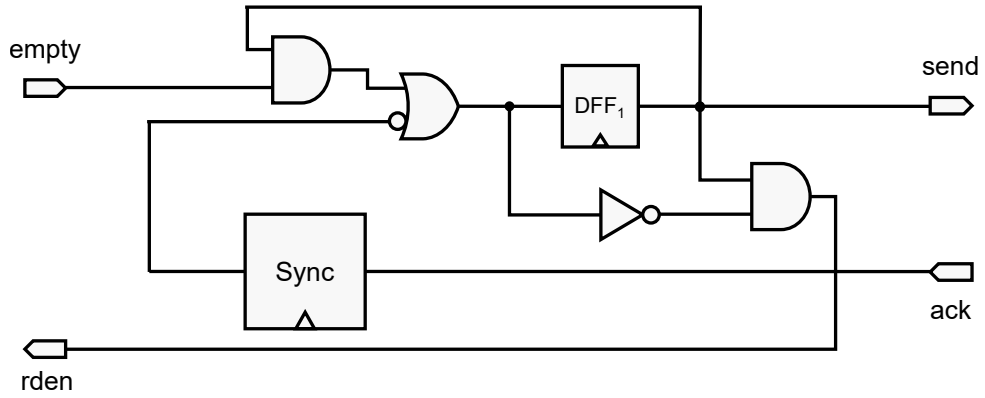


図 A.3　Sync-FIFO I/F における Input C-element (4-phase)

## A.3　Sync-FIFO I/F を構成する Output C-element

Sync-FIFO I/F を構成する Output C-element についても，2-phase および 4-phase のどちらにも対応可能とするために，それぞれの回路を設計した．2-phase，4-phase 用の回路の構成を，それぞれ図 A.4，A.5 に示す．Input C-element と同様に，Sync-FIFO I/F では動作タイミングを同期するための n-flop synchronzier を Output C-element 内に組み込むことで DDP から出力された転送要求信号 send を同期化し，その信号を用いて Output FIFO の書き込み許可信号 wren を生成する．

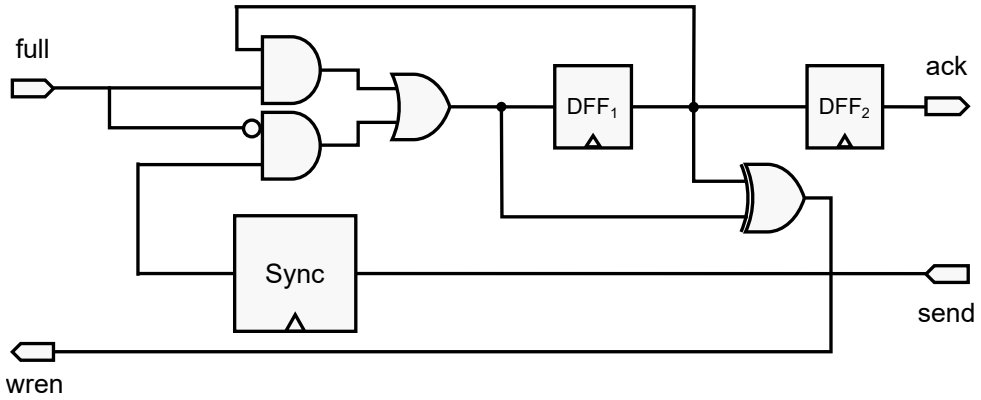


図 A.4　Sync-FIFO I/F における Output C-element (2-phase)

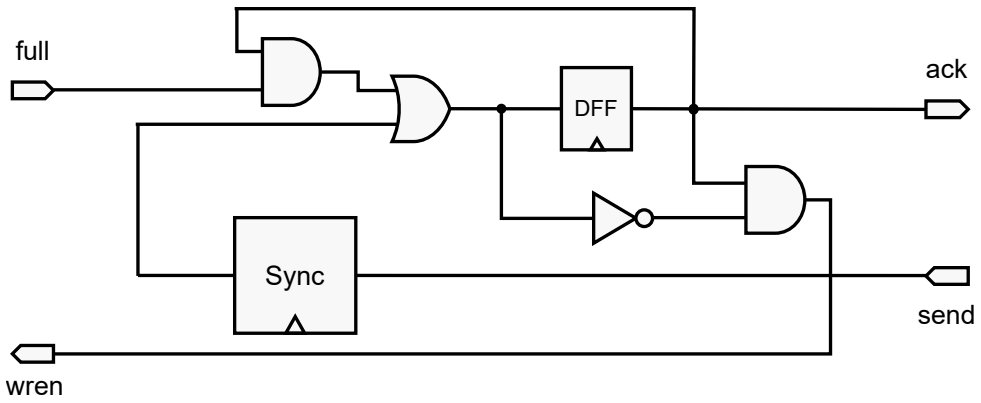


図 A.5　Sync-FIFO I/F における Output C-element (4-phase)