束データ方式を利用した
非同期回路に関する研究

石川 健一郎
Ken-ichiro Ishikawa
開放環境科学専攻
School of Science for Open and Environmental Systems

2006 年度

論文要旨

回路設計法の主流となっているクロック信号に同期して処理を行う同期式に対し，状態の遷移により処理を行う非同期式が近年，注目を集めている． 非同期式には同期式と比較し多くの利点がある．本論文では

- 回路の各機能ブロックが同期することなく動作する点
- それぞれの処理に応じた実行時間で処理を行うことが可能である点

に注目し，同期式とほぼ同様の回路を使用する束データ方式による非同期設計法をベースとした研究を行った． 本論文では以下の 2 つの研究について述べた．

(a) アービトレーション処理とデータ転送処理を分離し、処理を完全非同期で行うスイッチを開発した。このスイッチは独自に開発した特殊な回路を用いることにより、非同期分散アービトレーションによる調停を実現した。ROHM 社の 0.6μm プロセスを用いチップの作製を行った。シミュレーションにより評価においては全ポートを使用することにより 32Gbit/s の転送性能を示した。実チップと評価用ボードを製作し、条件付きながら実際に動作することを確認した。

(b) 束データ方式による非同期回路において、機能ブロックごとに処理に応じて実行時間を変化させることにより高速な演算を実現する方式として Speculative Completion が提唱されている。この方式を同期回路に応用し、サイクル毎にパイプラインの各ステージ（パイプライン構造ではない場合は 1 ステージのパイプライン構造と見なす）における実行時間の最も長いものにあわせて処理信号を送ることにより、高速な処理を実現する同期 Speculative Completion を提唱した。CPU コアに適用し、配線遅延付きシミュレーションにより評価を行ったところ、同期 Speculative Completion により CPU コアの制御が可能であることが実証された。処理速度の評価では比較用に制作された非同期式 CPU の 1.74 倍、Speculative Completion を採用した非同期式 CPU の 1.26 倍、同期式 CPU の 1.55 倍の性能を示した。

前者の研究は特定の回路を非同期式で制御することにより、より高速な処理が実現可能なことを示した。非同期式回路は回路構成が複雑なため動作のオーバーヘッドが同期式と比較し大きい場合が多いが、適した回路の場合、高速な動作を実現可能である。

後者の研究は同期式回路に非同期式回路の手法を導入することにより、より高速な処理が実現可能であることを示した。同期式回路はクリティカルパスの遅延により同期動作する全ての回路の処理時間が決定する。そのため、平均的な処理の遅延と比較し、クリティカルパスの遅延が極端に長い場合同期 Speculative Completion は有効に働く。

現在、非同期式回路設計法の応用分野は限られているが、全ての回路が同期する必要がある事により性能が低下する、設計が困難になるなどの問題がある場合、クリティカルパスにより処理時間が極端に長くなっている場合などの解決策として今後使われていくことが期待される。

Abstract

Recently, an asynchronous design using a state transition has received an attention as a solution of recent problems on synchronous designs. In several advantages of asynchronous signal processing, this study specially focuses on the following.

- Each function block can work without synchronized with a clock.
- The processing can be done with its own delay time.

Here, the bundled data method is adopted, since it can make the best use of the above two advantages using similar circuits as synchronous design.

In this thesis, two major research products are described.

(a) A pioneering asynchronous switch was designed, which processes all signals as an asynchronous manner, and its arbitration process and data transfer process are perfectly separated. A circuit simulation for this switch was made and it proved its 32-Gbps transfer rate. Actual switch chip and a printed circuit board for evaluating it were also implemented and confirmed the expected operation of this switch.

(b) Speculative completion, which attains high-throughput calculation by changing the processing time appropriately according to each function block, was proposed by Nowick. The latter half part of thesis is for applying it to synchronous circuits to propose a new method called synchronous speculative completion. This method transfers a processing signal synchronized with the current dominant delay time of the pipeline stage in order to enable further high-throughput calculation. The evaluation of this method is done by using a simple CPU core. This evaluation proved that the throughput using this method can be extended to be 1.65 times faster than the original one. This synchronous speculative completion method in particular can improve the throughput when applied to a process that is activated only in limited conditions and extends the latency of its critical-pass.

These studies proved its efficiency by the actual simulation and implementations. Now, application area of the asynchronous design is limited. However, the proposals in this thesis can be also applied to new architectures such as dynamically reconfigurable processors.

目 次

図 目 次

表 目 次

第1章　はじめに

現在考えられているデジタルICの動作原理は大きく分けて2種類ある．クロック信号と呼ばれる周期的な信号に合わせて動作する同期式と，状態の遷移によって動作する非同期式である．現在，主流となっている動作原理は同期式であり，ほぼ全てのチップで採用されている．同期式回路の利点としては構造が単純であるため設計が容易，動作が高速等が挙げられる．対して，非同期式回路は構造が複雑であるため設計が難しく，一般に動作速度は同期式と比較し遅い．そのため，非同期チップの開発環境を整備し，それを販売する企業が現われなかった．この開発環境の整備の遅れのため，研究対象として試験的にチップが作られる事はあったが，市販されるチップには採用されなかった．

しかし，近年になって，同期式の限界が露呈してきた．同期式ではクロック信号が全ての回路に同時に伝達することを前提に設計する．しかし，今日のプロセス技術の進歩による配線遅延の増加，回路規模の増大により，同期式の動作前提を守ることが困難になりつつある．また，回路規模の増大と共にモジュール設計が注目を集めているが，クロック信号が同時に伝達する前提を守りながら，クロック信号をモジュールに分配するためには，設計を変更する度にクロックツリーの再検討を行わなくてはならない．

これら同期式の問題点の解決策として，非同期式が注目を集めている．非同期式では一般に全ての制御はRequest信号とAcknowledge信号の組み合わせにより行われ，クロック信号による制御を行わない．そのため，配線遅延の増加，回路規模の増大によるクロック信号伝達のばらつきは問題にならない．また，モジュール設計では，モジュール間の通信線，Request信号線及びAcknowledge信号線を接続することにより，モジュール間の連結が可能である．このため，同期式のようにクロック信号の分配のために回路のクロックツリーの再検討を行う必要はない．

非同期式回路においてはクロック信号に関する問題は解決するが、制御回路が複雑になる問題がある．しかし，プロセスの改良により回路のコストが下がっているためこの問題は克服されつつある．また，モジュール設計において同期式により結合した場合と比較し，性能が下がる，全体の性能の予想が難しいなどの欠点があるが，性能の低下はプロセスの改良により素子遅延が減少しているため，改善されつつあり，性能予想の問題はEDAツールの発達により克服可能である．このように，非同期式設計を採用することにより，同期式が直面しようとしている問題を解決できる．

非同期式の利点としては上述の配線遅延の増加，回路規模の増大によるクロックスキューの問題の回避，モジュール設計の容易性の他に次のようなものがあげられる．

- 高性能

  同期式では全てのパイプラインのステージ(この論文ではパイプライン構造を採用していないチップは1ステージのパイプライン構造を採用していると見なす)の中で最も長いクリティカルパスによって全体の1サイクルの実行時間が決定する．しか

し，非同期式ではパイプラインの各ステージは各ステージの処理に必要な時間のみ実行に必要とする．また，2 線方式 (後述) を採用した非同期回路の設計方法には各ステージのクリティカルパスとなる処理の実行時間ではなく，各サイクルの実行時に実際に必要になる時間で動作する設計法が存在する．この特徴のため同期式よりも高性能になるケースがある．また，より設計が簡単な束データ方式 (後述) を採用した非同期回路においてもそのステージで実際に実行に必要になる時間に近い動作速度を実現する設計法がある．

- 高信頼性

  非同期式の設計法には素子の遅延を仮定しない設計法がある．この設計法により設計された回路は，同一の論理設計で製造プロセス，温度，電圧などの環境が異なる場合においても，動作することが可能である．

- 省消費電力

  非同期式で設計された回路は，Request 信号が伝達時のみ動作する．このため，同期式におけるクロックゲーティング技術と同じ効果が得られる．また，大きく電力を消費するクロック信号が存在しない．これらの特徴のため，特にアイドル状態時のダイナミック電力の消費が非常に少なくなる．また，同期式においては実行結果に影響を与えない遷移による電力消費が存在する．これに対し，非同期式の設計法には無駄な遷移が存在しない設計法があり，電力の消費をさらに抑える．

- 低 EMI(ElectroMagnetic Interference)

  同期式においては一定の周期を持つクロック信号にあわせて動作するため，電流は一定の周期で一斉に流れ出す．このため，特定の周波数の強い電磁波が放出され，電磁波障害の原因になる．だが，非同期式では各ステージがそれぞれのタイミングで動作するため，広い周波数にわたる弱い電磁波が放出される．このため，電磁波障害が起きる可能性は低い．

非同期式の研究の流れは 1989 年の完全非同期 CPU の開発 [MBL+89a] [MBL+89b] を契機に全ての回路を非同期で実現する完全非同期回路の研究に傾いた．完全非同期回路の研究は 1990 年代に最盛期を迎え，CPU 等多くの回路が完全非同期の回路で実現された．だが，スイッチに関する研究はスイッチ機能を持った演算回路である FleetZERO [CLJ+01] を除くと研究例はほとんどなかった．

これを踏まえ，本研究では開発が進められていない完全非同期スイッチを研究対象とした．同期式スイッチは各機能ブロックが同期して動作するため，データ転送と比較し重い処理であるアービトレーション処理により全体の処理速度が制限される．そのため，非同期式の各機能ブロックがそれぞれの処理を行う時間のみで動作する特徴を活かす事により大幅な性能改善が期待できる．同期式と異なり，開発環境が整っていないため多くの機能を持ったチップの開発は困難と判断し，単純な構造により実現可能な高スループットスイッチを目指した．

スイッチを開発するにあたり束データ方式，2 線方式と 2 種類ある非同期式回路設計法から束データ方式を選択した．束データ方式は先にあげた特徴のうち高信頼性，及び，無

駄な遷移が存在しないという特徴は持っていない．だが，同期式とほぼ同じ回路を使うことができ，小面積で高速な回路を作ることが可能である．2 線方式は先に挙げた全ての特徴を持っているが，専用の設計法による専用の回路が必要であり，束データ方式と比較した場合，面積，動作速度の面で不利である．また，2 線方式と比較し束データ方式はデータを表現する際に必要な信号線が少ないため同じ本数の信号線を使用する場合にはより多くのデータを転送することが可能である．

本研究では，束データ方式のデータを完全非同期でスイッチングし，高速な連続転送が可能なスイッチを提案する．

束データ方式のスイッチはアービトレーションに独自の回路を用い，転送には独自のプロトコルを用いることにより，連続転送時に非常に高い性能を持つように設計されている．HSPICE シミュレーションにより評価した後，実チップを作成し，専用ボード上で実機での評価を行った．

また，2000 年頃から従来の同期式回路を完全非同期式回路に置き換える研究が徐々に下火になり，代わって，非同期式の手法を同期式に取り入れ両者の特徴を持った回路の研究が盛んになってきた．

非同期式の手法，回路を同期式に取り入れ，高速な処理を目指す研究としては次の研究が代表例として上げられる．浮動小数点除算が扱う値に応じて大きく実行時間が変化することに着目し浮動小数点除算のみ非同期回路を導入した研究 [KOTG99]，CISC 型プロセッサのデコード回路に応用した研究 [RSG$^+$99][RSP$^+$00] である．また，グローバルクロック信号に合わせて全体の回路が動作しないことを利用した研究としては GALS(Globally Asynchronous Locally Synchronous) [Cha84] [SAM$^+$04] [RM03] があげられる．GALS は回路規模の増大，配線遅延の増加及びモジュール同士を接続する配線が長くなることによりモジュール間の配線がクリティカルパスになることを防ぐ技術であり，同期式で作られたモジュールを非同期式の回路で接続する．

この流れを受け，非同期式の，各サイクルにおいて実行する処理に応じて最適な実行時間での実行が可能である点に着目し，この特徴を同期式に応用する研究を行った．実行する処理に応じて最適な実行時間での実行が可能特徴は 2 線式の非同期回路の特徴であるが，同期式とほぼ同じ回路によって実装される束データ方式においても Speculative Completion と呼ばれる技術を用いることにより実現可能である．同期式では稀に実行される処理により 1 サイクルの実行時間が決定されている場合が多く，実行される処理に合わせて 1 サイクルの実行時間を変更可能にすることにより処理速度の改善が期待できる．

以上を踏まえ，同期 Speculative Completion を提案する．同期 Speculative Completion は Speculative Completion を改良し，同期式に適用したものである．同期式 Speculative Completion を導入することにより，時間が必要な処理を行う場合には次の処理を始めるためのクロック信号の送信を遅らせ，短い時間で終わる処理を行う場合には次の処理を始めるためのクロック信号を早く送信し効率の良い実行を目指す．

本論文の構成は次のようになっている．第 2 章では非同期式特有の素子，モデルなどについて述べる．第 3 章では非同期スイッチに関連のある過去の研究について述べ，第 4 章では非同期スイッチについて記述する．そして，第 5 章で同期 Speculative Completion に関連のある過去の研究について述べ，第 6 章では同期 Speculative Completion について記述し，第 7 章で全体をまとめる．

第2章　非同期式

この章では非同期式の歴史について述べた後，非同期特有の素子である C-element について記述する．その後，非同期式回路を設計する際に重要であるディレイモデルについて述べ，最後に束データ方式，2 線方式について述べる．

2.1　非同期式の歴史

非同期式の歴史は古い．今日のようなコンピュータが発明される前，デジタル回路といえば電話交換機の制御回路をさしていた時代には同期式という概念はなく，非同期式の回路のみが存在した．

1940 年代に電子計算機の歴史は同期式電子計算機によって始まるが，1950 年代には非同期式の先駆的な研究が Muller，Huffman らによって始められている．Muller は非同期式においてディレイモデル (素子，及び配線の遅延をどのように見積もるか）に関する研究を行い，特にディレイモデルの一種である Speed Independent モデルの理論的な研究で成果を上げた [MB59]．Huffman は非同期回路の動作解析を容易にするため回路が安定するまでは 1 つの入力しか変化しない単一入力変化基本モード (single-input-change fundamental mode) の仮定を提唱した [Huf64]．また，同時期には基本モードに対して，入力の変化に対する回路の反応が終わり安定状態になる前に入力が変化することを許す input-output モードに関する研究が行われている [McC62].

Muller は 1950 年代，1960 年代にかけてイリノイ大学において，Speed Independent モデルを用いたコンピュータ ILLIAC，ILLIAC II の設計を行った [Bre65]．この後，メインフレーム MU-5，Atlas において非同期回路が採用された．

60 年代後半にはセントルイスのワシントン大学で非同期式のマイクロモジュールが開発された [Cla67]．このマイクロモジュールを接続することにより，様々な特殊演算が可能であった．1969 年に Unger は今日一般的に使われている，データに Request 信号を畳み込む 2 線方式 (後述) を世に示した [Ung69]．また，1984 年には CVSL（Cascade Voltage Switch Logic） [HGDT84] が発表され 2 線方式の効率的な実現方式が明かになった．1987 年には STG（Signal Transition Graph）と呼ばれる非同期式回路設計時の解析に有用な表現方式が発表された [Chu86] [Chu87].

1989 年には非常に簡単なものだが，完全に非同期で動作するマイクロプロセッサをカリフォルニア工科大学の研究グループが開発した [MBL$^+$89a] [MBL$^+$89b]．これを先駆けとし，Manchester 大学の Amulet シリーズ (AMLUET1 [FDG$^+$94b] [Fur95] [WDF$^+$97], AMULET2e [FDG$^+$96] [FGT$^+$97] [FGR$^+$99]，AMULET3 [FGG98] [FEG00] [GBB$^+$00]), 東京大学の TITAC シリーズ [NUK$^+$94] [NTK$^+$97] [TKI$^+$97] など，様々な非同期マイクロプロセッサが開発された．Philips は非同期回路の低消費電力である特徴に着目し，非同期

80C51 プロセッサを搭載した完全非同期ポケットベルを開発し，製品化した [GBvB+98] [Gag98]．また，INTEL は RAPPID プロジェクトにおいて x86 命令セットのデコーダを開発した [RSG+99] [RSP+00]．このデコーダは当時の同期式のものと比較し 3 倍の速度，半分の消費電力を達成した．

2000 年代に入ると同期式回路の劇的な性能向上により，完全非同期回路により同期式と比較し優れた特徴を持つ回路を設計する研究は徐々に下火になり，代わって非同期式回路の手法を同期式回路に組み入れることにより，優れた特徴を持つ回路を設計する研究が盛んとなって来た．代表的なものとして，GALS(Globally Asynchronous Locally Synchronous) があげられる．GALS では同期式のモジュールを非同期式で接続する．各モジュールを同期式で動作させることにより高速な処理を実現し，各モジュールを非同期式で接続することによりグローバル配線がクリティカルパスになり全体の動作速度が落ちることを防ぐ．GALS は 1984 年の [Cha84] の論文から研究が始まったが 2000 年以降，IP によるモジュール開発が注目を集め始めるとともに脚光を浴びるようになった [YD96] [MVF00] [SM00] [MTMR02]．

1990 年代後半から開発環境を整備する研究が行われるようになった．CAD に関する研究の先駆けとしては Manchester 大学の Balsa があげられる [BE00] [Bar00] [EB02]．Balsa は非同期式の代表的なデータ表現方式である束データ方式，2 線式双方の回路を出力可能なハードウエア記述言語である．今日では，より一般的なハードウエア記述言語である Verilog-HDL 形式で出力可能なユタ大学の ACK [JBGK00]，東京大学の AINOS [OIN02] 等非同期チップ向けハードウエア記述言語が多く開発されている．

2.2　C-element

Muller の C-element は非同期特有の演算素子である．2 入力 C-element の場合，伝統的な回路は図 2.1，インバータ素子を使ったものとしては図 2.2 が挙げられる．図 2.1 の C-element はスタティック回路であり消費電力が少ない利点があるが，回路構造が複雑であり，必要なトランジスタ数が多いなどの欠点がある．図 2.2 の C-element はインバータ素子のループを用いたダイナミック回路であるため，消費電力が多い，ノイズに弱いなどの欠点があるが回路構造が単純で必要なトランジスタ数が少ないなどの利点がある．C-element は入力が全て 0 の場合 0，全て 1 の場合 1 を出力し，その他の場合は以前の出力を維持する．C-element は非同期回路において制御回路，演算回路双方で広く使用される．

2.3　ディレイモデル

非同期式ではディレイモデル，つまり，遅延をどの様に仮定するかが非常に大きな問題となる．同期式では通常，各素子，及び配線で発生する遅延の想定する使用環境下における最大値を求め，その値を使用し設計する．だが，非同期式では遅延を一定条件下の最大値から仮定しない方式等様々な方式が存在する．これは同期式では外部からの周期的な信号によって動作するため，全てのステージの最大遅延が目標とする遅延と比較し短い必要があるが，非同期式では内部の信号の遷移が伝わることにより動作するため，遷移が伝達すれば，それに必要な時間は問題にはならないためである．次の小節より代表的なディレイモデルを挙げる．

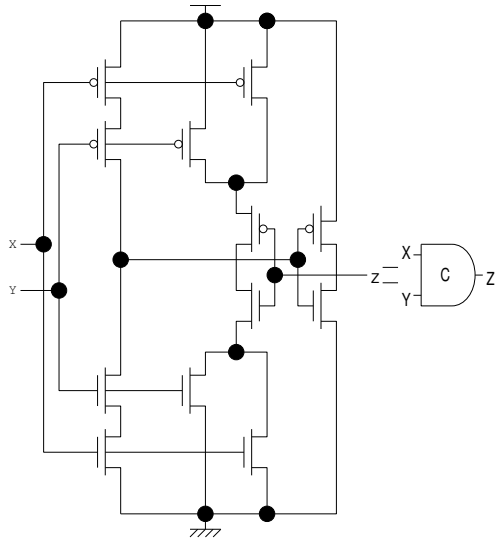

図 2.1: C-element（スタティック素子）

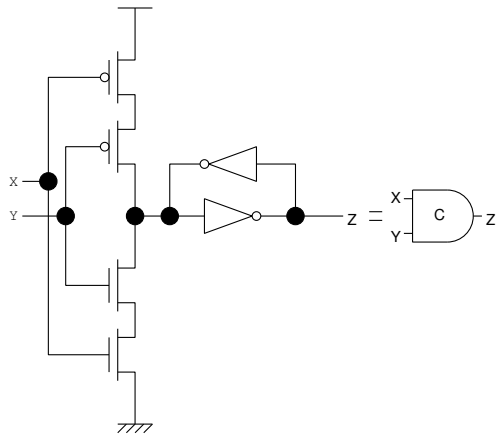

図 2.2: C-element（ダイナミック素子）

2.3.1　bounded delay model

同期式と同様に各素子の遅延の最大値を仮定するディレイモデルである．このモデルと単一入力変化基本モードを仮定した回路は非同期式黎明期に提唱され，Huffman 回路と呼ばれる [Huf64]．現在でも束データ方式（後述）で製作される非同期回路の一部はこのモデルを使用している．

2.3.2　SI(Speed Independent)

素子の遅延は有限と仮定するが上限を設けず，配線の遅延は 0 と見積もるディレイモデルである．初期に提唱され，ILLIAC，ILLIAC II のように実際にこのディレイモデルで作られたメインフレームも存在する．このディレイモデルが使用された時代は，素子遅延と比較して配線遅延が非常に小さかったため，SI モデルで作られたコンピュータが動作し

た．現在ではほとんど使われていない．

2.3.3　DI(Delay Insensitive)

素子の遅延，配線の遅延とも有限と仮定するが上限は設けないと見積もるディレイモデルである．このディレイモデルを採用して論理設計された回路は，温度，電圧，プロセス等の環境に依存せずに動作する．しかし，このディレイモデルと出力が安定するまで入力が変化しない仮定を採用して設計する場合，インバータ及び C-element のみが素子として使用可能であり，ほとんどの種類の回路が実現できない．これは以下のように証明される [Mar90].

(a) ある素子 g の入力の 1 つを入力 x とする．

(b) 他の入力の状態により入力 x の変化が素子 g の出力に関係ない場合 (以下場合 α と呼ぶ) が存在すると仮定する．

(c) DI モデルでは素子，配線ともに遅延を仮定しない．

(d) そのため，場合 α が存在する場合，入力 x の変化が伝達することなく素子 g の出力が変化し，それが設計者が意図した動作である必要がある

(e) つまり，場合 α が存在する場合，入力 x は素子 g の反応には影響を与えない．

(f) このことから入力 x は素子 g の入力ではない．

(g) (a) と (f) は矛盾するため (b) の仮定は正しくない．

(h) (a) から (g) より，素子 g に対する入力群を入力群 X とすると，常に次の順に反応が起きなければならない．

  (a) 入力群 X が全て変化する
  (b) 素子 g が演算結果を出力する．
  (c) 入力群 X が全て変化する
  (d) 素子 g が演算結果を出力する．

(i) このためには入力群 X がすべて変化するまで素子 g は反応してはならない．

(j) 複数の入力を持つ素子で入力すべてが変化するまで出力が変化しない素子は C-element のみである．

(k) (i)(j) より素子 g は C-element もしくはインバータ．

実際に動作する回路でありながら DI モデルを使用している限り動作を説明できない回路として図 2.3 のような回路がある．この回路は＜ [xi↑]; yo↑; [yi↑]; u↑; [u↑]; yo↓; [yi↓]; xo↑; [xi↓]; u↓; [u↓]; xo↓ ＞のように動作する様，意図して設計されたものである．[] 内は入力もしくは変化，[] 外はその結果，起きる変化をあらわしている．DI モデルで考えた

場合，[yi↑] の時，yi↑ の信号が線を通じて下の AND 素子に伝わる前に，C-element に伝わり u を立ち上げ，それが下の AND 素子に伝わり xo を立ち上げる場合を考える必要がある．このように DI モデルの仮定の元で実際に動作する回路を設計するのは極めて困難である．

2.3.4　QDI(Quasi-Delay Insensitive)

素子の遅延，配線の遅延とも仮定はせず，有限であると見積もるディレイモデルである [Mar90]．DI モデルとの違いはこの仮定に加え，同一の素子からの出力は同時に接続している素子に到達する (Isochronic Fork) としている点である．この仮定により DI モデルでは動作が保証されない図 2.3 の回路の動作が保証される．[yi↑] の時，QDI では C-element と下の AND には同時に信号が届くため，設計者の意図とは異なり xo↑ となる可能性を考えなくて良い．このように DI モデルとは異なり C-element 以外の複数入力素子の使用が許される．これは DI モデルにおいて C-element 以外の複数入力素子が許されない証明が成立するためには各素子，各配線の遅延が他の素子，配線とは無関係という条件が必要だが，QDI では配線の遅延に条件があるためである．QDI は普及しており，多くの非同期回路がこのディレイモデルを用いて設計されている．

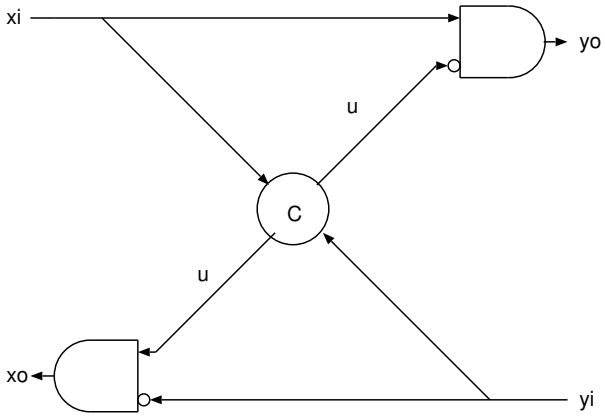

図 2.3: DI モデルでは動作しない回路の例

2.3.5　SDI(Scalable Delay Insensitive)

素子の遅延，配線の遅延の比は一定の割合に収まると見積もるディレイモデルである [NTK+99]．ある回路上の構成要素 C を考える．回路設計者が C の遅延を De と予想し，実際の遅延が Da だった場合，Scaling Ratio，R を R = Da/De とする．そして，実際の回路の構成回路 C1，C2 の Scaling Ratio が R1，R2 となるとき Scaling Variation，V を V=R1/R2 とする．このとき，V はプロセス技術などによって決まる値 K が決定されているとき $1/K \leqq V \leqq K$ に収まるとする．一般に同一のチップ内であれば CMOS の反応速度，配線遅延は一定の範囲に収まるためほとんどの条件下でこのディレイモデルを使用し設計された回路は動作する．QDI と比較し条件が緩いため高速かつ小面積な回路を製作可能である．東京大学の南谷研で盛んに研究が行われているが，その他の研究拠点では使用されていない．

2.4　束データ方式と 2 線方式

同期式ではある信号線が表しているデータはクロック信号に同期して決まる．だが，非同期式の場合クロック信号は存在しない．そのため，信号線上の情報の中で有効なデータを指定する必要がある．この方法は大きく分けて束データ方式と 2 線方式の 2 方式がある．

2.4.1　束データ方式

束データ方式では信号線のどの時点での値が有効であるかを表す Request 信号線を付け加えることによりデータを表現する．2 相式の場合，図 2.4 のようになる．Request 信号の立ち上り及び立ち下がりのデータが有効となり，データが読み込まれた後 Acknowledge 信号が返され，データが次の値を取る．4 相式の場合，図 2.5 のようになる．この例においては Request 信号が High の状態のデータが有効であるとしており，データを読み込んだ後に Acknowledge 信号が High になり，その後，Request 信号が Low になり，Acknowledge 信号が Low になり，データが次の値を取る．w は遅延回路により信号の遷移が遅延することを表す．

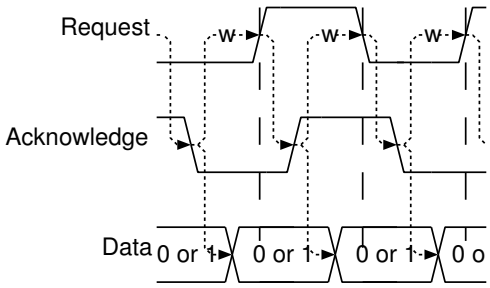

図 2.4: 束データ方式（2 相）

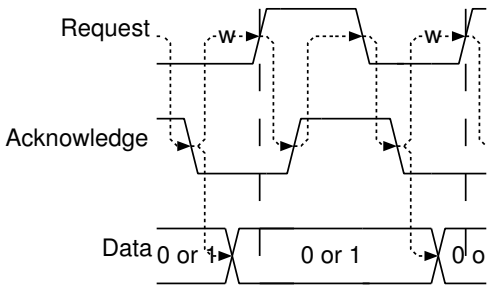

図 2.5: 束データ方式（4 相）

この束データ方式におけるパイプラインのステージは 2 相の場合，図 2.6 のようになる．図 2.6 の CP-Latch は図 2.8 のような構造をしており，C の入力と P の入力が同一のときは Data In からの入力が Data Out へ出力され，C の入力と P の入力が異なるときは Data In からの入力を遮断し，Data Out への出力を保持する．図 2.6 において，C-element の出力が変化する場合，同じステージの CP-latch の C への入力，P への入力は必ず異なるため，

CP-latch から Function Block へデータが出力されない．C-element1 の出力が変化する事により C-element1 に接続されている CP-latch は Function Block へデータを出力しない状態となり，C-element0 に接続されている CP-latch から Function Block へデータが出力される．C-element2 の出力が変化し，C-element2 に接続されている CP-latch の Fuction Block へのデータの出力が止まり，C-element0 からの Request 信号が DELAY によって遅延された後 C-element1 に入力される事により C-element1 の出力が変化し，接続されている CP-latch が Function Block へデータを出力する．また，C-element1 の出力が変化する事により C-element0 に接続されている CP-latch の Function Block への出力が止まる．

4 相の場合，図 2.7 のようになる．Function Block から出力されたデータは latch に入力され，その後，そのデータの Request 信号が latch を制御する C-element に入力される．C-element は Request 信号が届き，前方のステージの latch が次のステージの Function Block にデータを出力していない時，latch を制御し，Fuction Block へデータを出力する．C-element は同時に後方のステージの C-element へ信号を送り Request 信号の出力，及び後方のステージの latch から Fuction Block へのデータの入力を止める．

束データ方式では Request 信号はデータと同期する必要がある．そのため，Request 信号は常に信号線上のデータが安定した後にアサートされるように設計する必要がある．この条件を満たすため通常の束データ方式の回路の Request 信号線上には遅延素子がある．

束データ方式ではディレイモデルは bounded delay model もしくは SDI が用いられる．SDI を用いた場合，動作環境の変化に強いが遅延時間が長くなるため動作速度が遅い回路になる．各ステージの Request 信号に対する遅延が一定であることから明らかなように束データ方式では各ステージの実行時間は処理内容によらず一定である．

遅延素子のテストとして，(a) 長い配線，(b) インバータ素子，(c) 幅広いチャネルを持ったインバータ素子を用いたときの消費電力，遅延のばらつきのテストが行われている（図 2.9）[ION04]．0.35μ プロセスを用いて行われた研究では消費電力では表 2.1 に示す通り長いチャネルを持ったインバータを用いた場合が最も優れていることが示された．また，遅延のばらつき (K) ではインバータ素子を多く並べたものがもっとも優れていることが示された．束データ方式を用いた代表的なチップとしては CPU では AMULET1，AMULET2e，FPGA では PCA [KIN+01] などがあげられる．

	(a)	(b)	(c)
Energy(pJ)	1.46	0.669	0.249
K	1.18	1.09	1.22

表 2.1: 遅延線の評価

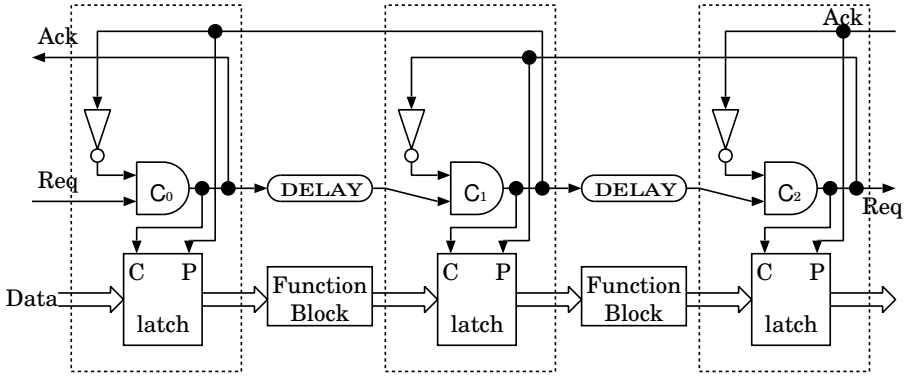

図 2.6: 束データ方式のパイプライン（2 相）

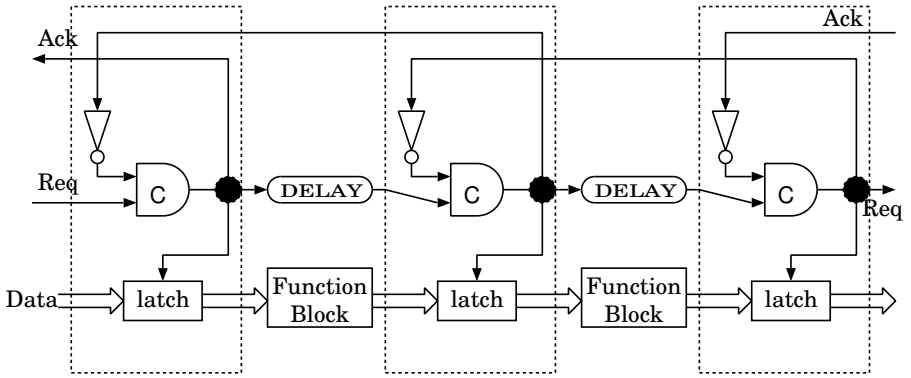

図 2.7: 束データ方式のパイプライン（4 相）

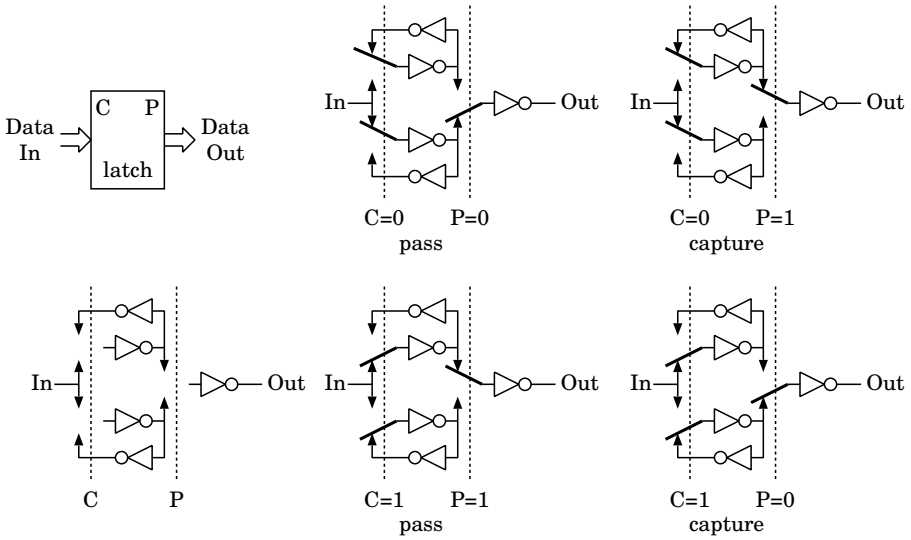

図 2.8: CP-latch

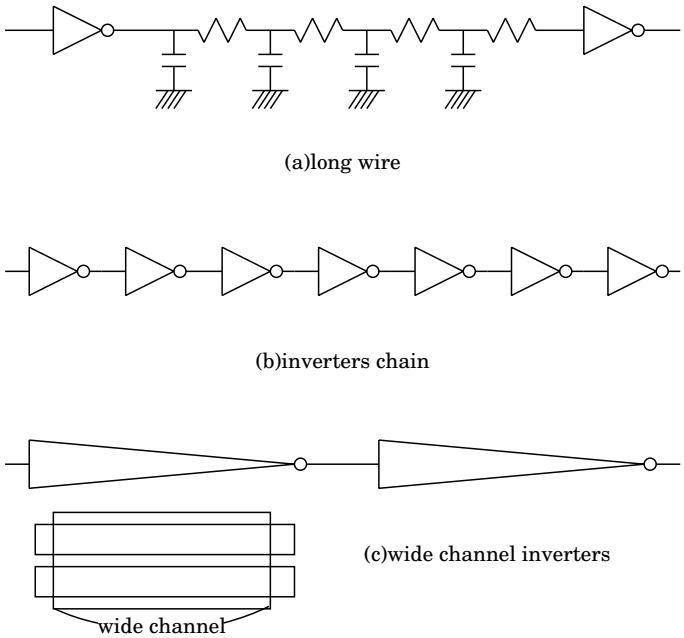
(a)long wire
(b)inverters chain
(c)wide channel inverters
wide channel

図 2.9: 回路モデル

2.4.2　2 線方式

2 線方式では 1bit のデータを表すために 2 本の信号線を使う．一方の信号線を 1 を表す信号線，もう一方の信号線を 0 を表す信号線とする．どちらもネゲートされている場合はデータが存在しない，0 を表す信号線がアサートされた場合は 0 のデータが存在する，1 を表す信号線がアサートされた場合は 1 のデータが存在すると判断する．0 を表す信号線，1 を表す信号線双方が同時にアサートされた場合エラーとなる．データ表現は 2 相式の場合は図 2.10 の (a)，4 相式の場合は図 2.10 の (b) のようになる．

2 線 4 相方式でのパイプラインのステージは図 2.11 のようになる．function block より出力されている線は 1 対の信号線を表す．ここでは各 function block の出力は 1bit とする．2 線方式では Request 信号はデータに畳み込まれている．そのため，1 つのデータを表す 2 本の線の一方がアサートされる事によりデータに対する演算が終わったことが表現される．これにより，演算回路の出力を監視することにより演算が終了したことを確実に検出できる．このため，2 線方式では実際に演算に必要な時間のみでパイプラインの各ステージを実行することが可能である．

図 2.11 は次のような挙動を示す．初期状態として，function block0 が処理を終了した直後とし，他の function block は処理を行っていない状態とする．そのため，

- function block0 は処理を終了した直後であるため入力対は一方が High，一方が Low，出力対も同様に一方が High，一方が Low
- Ack1，Ack2，Ack3 は Low，Ack1-inv，Ack2-inv，Ack3-inv は High

この場合次のような挙動を示す．

(a) Ack2-inv が High であるため function block0 の出力が function block1 へ入力される

(b) Ack1 が High になり，Ack1-inv が Low になる

(c) function block1 において処理が行われる

(d) function block1 の処理が終了し，出力対のいずれかが High になり，Data0-output もしくは Data1-output が High になる (function block1 の接続先 function block へ処理結果が入力される)

(e) function block1 の出力対のいずれかが High になったため，Ack2 が High に，Ack2-inv が Low になる

(f) 処理が進むとともに Data0-input，Data1-input が Low になる (function block0 の接続元 function block の出力がクリアされる)

(g) function block0 の入力対がいずれも Low になり，その結果，function block0 の出力対がいずれも Low になる

(h) Ack1 が Low になり Ack1-inv が High になる

(i) function block1 の入力対が Low になり，その結果，function block1 の出力対が Low になる

(j) Ack2 が Low になり，Ack2-inv が High になる

(k) 処理が進み Data0-input，Data1-input のいずれかが High になり，function block0 の入力対のいずれかが High になる

(l) function block0 において処理が行われる

(m) function block0 の出力対のいずれかが High になる

(n) 始めに戻る

2 線方式で作られた回路は 1 章であげた全ての非同期の利点を持つ．信号が一瞬でも変化すると演算終了を判定する回路で演算が終了したと判断されるため，グリッチの発生は許されない．通常，ディレイモデルは QDI 方式を用いて設計される．2 線方式を用いたチップは数多くあるが，代表的なチップとしては AMULET3i [FGG98] [GFC99] [FEG00] [GBB+00], miniMIPS [MLM+97] [MNPW01] などがあげられる．

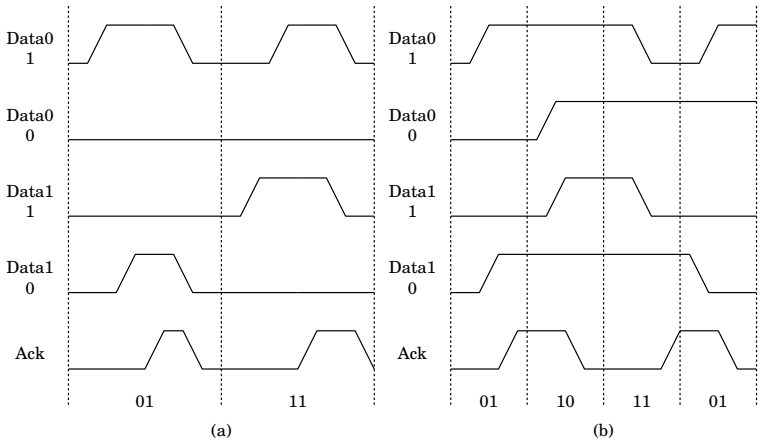

図 2.10: 2 線方式のデータ表現

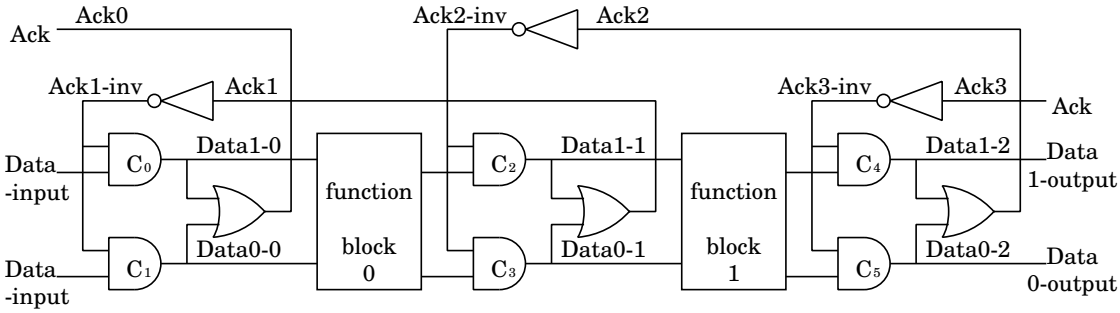

図 2.11: 2 線方式のパイプライン

2.4.3　本論文におけるデータ表現方式

2.4.1 節，2.4.2 節で述べたように非同期式におけるデータ転送方式は束データ方式と 2 線方式に分類される．本論文の研究では設計方針として 2 本のデータ線を使いデータを表現する 2 線方式は採用せず，少ない信号線でデータを表現可能であるため回路のコスト面で有利な束データ方式を採用した．データの確実な表現を目指すのであれば 2 線式を用いるべきであるが，2 線式によるコストの増大による表現可能なビット数の減少，回路規模の増加，内部処理回路における処理速度の低下は無視できない．本論文の研究では束データ方式によるジッタの問題などは無視できるレベルに留まると判断し，低コストで実現可能な束データ方式を採用した．

第3章　非同期スイッチに関する既存の研究

この章では非同期スイッチの既存の研究について述べる．ここでは非同期式スイッチの先行開発例である FLEETzero と FLEETzero で用いられる高速処理制御技術 GasP を紹介する．

3.1　非同期スイッチ

3.1.1　FLEETzero

FLEETzero[CLJ+01] は簡単な演算機能を持つ非同期スイッチである．新しい演算方式の実験，及び，非常に高速な非同期演算を可能とする GasP[SF01] の実チップに対する実装テストを目的としている．

従来の各機能ブロックを配線でつなぐ設計手法は配線遅延の増大により，性能的限界を迎える事が予想されている．FLEETzero はこの限界を超えるための，新しい設計手法を提案している．FLEETzero は図 3.1 に示す構造をしている．図に示されている Ship は演算回路であり、それぞれ表 3.1 に示す演算を行う．Ship0 は外部からデータを入力し，Ship1 は常に一定の値を出力する．Ship2，Ship6，Ship7 は FIFO として働き，Ship4 に値を入力した後，Ship3 に値を入力する事により加算が行われる．Ship-5 では入力された値の補数が求められる．左の Instruction Storage ではデータに対して行われる演算コードが保存されている．演算コードは Destination Horn，Source Funnel において，図に示されている演算回路 Ship の中から入出力を有効にする Ship を選択する．FLEETzero では Ship での演算を繰り返すことにより処理を行う．

非常に少ないトランジスタによる演算のみで処理の制御を可能にする GasP [SF01](関連技術で紹介) を採用することにより，高速な演算を可能にしている．スイッチ機能は演算の一種として用意されており，演算として Ship0 を選択することにより実行される．スイッチとしての性能は 8 入力 8 出力，データ幅 8bit となっており，一度に 1 入力を指定された出力先に出力することができる．0.35μm プロセスを用い，1.2Giga-Data-Items per second(1Data-Item は 8bit 幅) の性能を示した．

FLEETzero は非同期式で高速なスイッチが製作可能であることを明らかにするだけではなく，非同期式を用いた新しい演算手法が有効であることを示している．

Ship address	output	input
0	external input	external output
1	constant-A	data sink
2	FIFO-A(3stage)	FIFO-A(3stage)
3	adder-A	adder-A
4	constant-B	adder-A
5	complementor(3stage)	complementor(3stage)
6	FIFO-B(3stage)	FIFO-B(3stage)
7	FIFO-C(9stage)	FIFO-C(9stage)

表 3.1: FLEETzero command

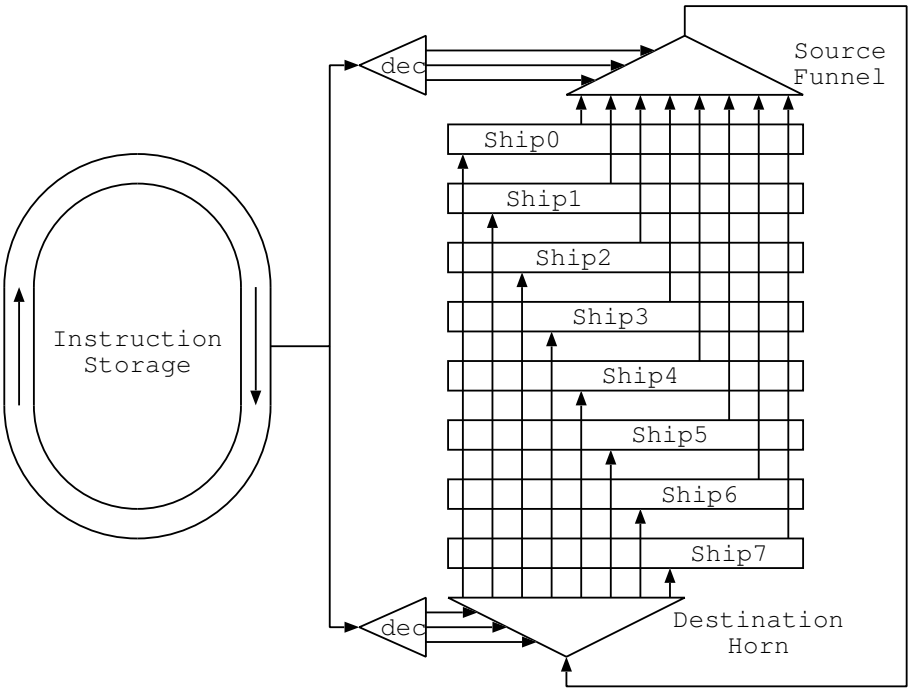

図 3.1: FLEETzero

3.1.2　関連技術：GasP

GasP [SF01] は束データ方式を用いた非同期式 FIFO の制御法の一種である．一般に非同期式においては Request 信号線と Acknowledge 信号線を使い制御を行うが GasP では Request 信号線のみを使用する．動作時に信号は少数のトランジスタのみ通過するため非常に高速に動作する特徴を持つ．

GasP 回路の一例として図 3.2 の回路があげられる．GasP では PATH（データの通り道）と PLACE（データを記憶する場所）が交互に並んでいる．この回路では以下のようにデータが進む．

(a) cc が短い High のパルスを p に送りデータを通過させる．

(b) インバータ c と N 型トランジスタ d が FULL（データが存在する）を意味する Low を次の PATH に出力する．

(c) P 型トランジスタ y が EMPTY（データが存在しない）を表す High を前の PATH に出力する．

(d) r，s のトランジスタによって遅延され，P 型トランジスタ t が回路を初期化する．

データを前進させる信号は abcd それぞれのトランジスタが反応しなければならないため 4 トランジスタディレイ，制御信号がパイプラインを逆行するためには xy のトランジスタが反応しなければならないため 2 トランジスタディレイ必要とする．GasP 方式は分岐する場合，データに依存する場合など様々な回路をわずかな変更で作り出すことができる．GasP 方式を利用したチップは実機が製作されており，動作することが確認されている．GasP 方式を利用したチップ FleetZERO [CLJ+01] では 1.2V から 4.8V の電圧で動作が確認された．

GasP は Molnar の asP*回路 [MJCL97] がベースとなっている．asP*回路をより高速に動作させるため，データがパイプラインを進むスピードを調整したものが GasP 回路である．一般にデータがパイプラインを伝搬していくスピードは遅く，制御信号がパイプラインを逆行していくスピードは速い．このため，GasP ではデータがパイプラインを前進する際のディレイを 4 ゲートディレイ，制御信号がパイプラインを逆行するスピードを 2 ゲートディレイとした．なお，データが前進するディレイ，制御信号が逆行するディレイ，共に 3 ゲートディレイとした dynamic asP*も考案されたがゲートが奇数の場合，信号が反転するため，回路が複数パターン必要になり実用性が低く，公表されなかった．

GasP 回路は利用可能な回路が限定されるが非同期式で極めて高速な動作を可能とする技術である．

3.2　まとめ

非同期スイッチ関連の研究として FLEETzero と関係する技術を取り上げた．関連研究として FLEETzero の名のみ上がることから明らかなようにこの分野の研究はほとんど行われていない．だが，スイッチは主要な構成回路であり，また，非同期式で設計することにより同期式とは異なった特長を持ったものを設計可能である．次章では非同期では同期

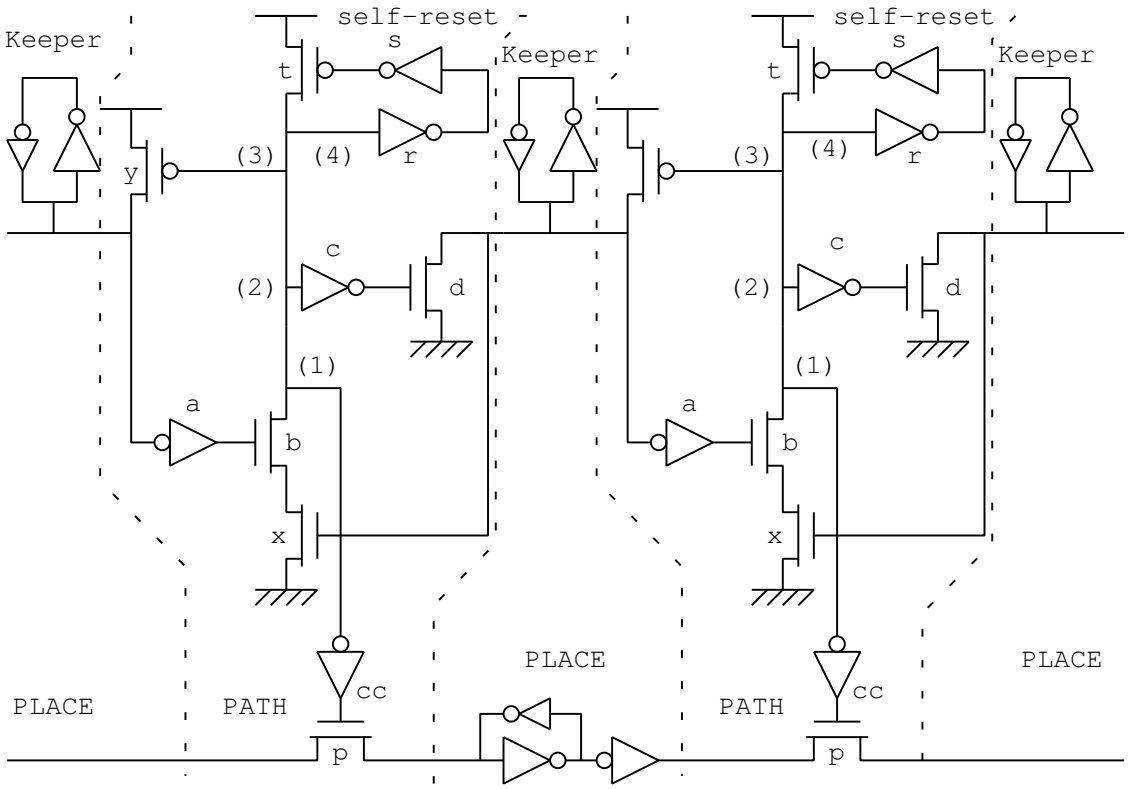

図 3.2: GasP

式とは異なった特徴を持った回路を作成することができることを踏まえた上で設計された非同期スイッチについて述べる.

第4章　非同期スイッチSASの設計と実装

4.1　概要

PC クラスタのノード間接続や高速 I/O 向けの SAN（System Area Network もしくは Server Area Network) 等，ネットワーク技術の進歩による高速化は近年著しく，そのリンク速度は 10Gbps に達している．

ネットワークにおけるスイッチの基本的な機能は，データを入力し，指定された出力先に出力可能であることを判定し，指定された出力先に出力することであるが，データの入出力処理と指定された出力先に出力可能であることを判定する処理は性格が異なる．そのため，スイッチが持つ機能によってはデータの入出力処理と出力先に出力可能であることを判定する処理では大きく処理時間が異なる場合がある．

このようなデータの入出力処理と出力先に出力可能であることを判定する処理の間に大きな処理時間の差が生じる場合においては，全ての回路で同じ処理時間を前提とする同期式で設計するのではなく，それぞれの回路がそれぞれの処理時間で処理を行う非同期式で設計する事により，より高速な処理を期待できる．

処理速度の遅い機能ブロックにより全体の処理速度が制限される回路に対する非同期式回路の応用の研究は盛んに行われているが，ネットワークの構成要素の一つであるスイッチの研究はほとんど行われていない．FLEETzero [CLJ+01] はスイッチ機能を持っているが，新しい計算機アーキテクチャの提案が主な製作動機であり，スイッチとして使われるために作られたものではない．

完全に非同期で動作するスイッチとして SAS（Simple Asynchronous Switch）を提案する．このスイッチは完全に非同期で動作し，アービトレーションを非同期で取ることができる．電気的に動作するクロスバースイッチの一種で最高 2GHz 相当の動作が可能である．4bit のデータ幅の 4 入力 4 出力のポートを持ち，それぞれの出力ポートで分散アービトレーションを取っているため，最高 32Gbit/s の処理能力がある．高速処理実現のため，ハンドシェイクは転送開始時と終了時のみ行いデータは WavePipeline 方式で転送するプロトコルを採用した．ハンドシェイクとデータ転送を極力分離した構造を持たせることにより，データ転送時は素子 5 個分の遅延のみでデータは転送される．

SAS はローム社の 0.6μm プロセスを使い，VDEC を通じて実機が製作された．HSPICE シミュレーションでは 1 つの信号線あたり 2Gbps の信号，1 ポートあたり 8Gbps の信号を転送し，全体で 32Gbps の転送能力があることが分かった．実機での動作も確認され，条件付きながら正しく動作することが確認された．

4.2　データの転送方法

スイッチを非同期式で設計することにより次の利点が期待される．

(a) データの授受を行うためにはハンドシェイクが必要であるが，非同期式を用いることによりクロック周波数によらないハンドシェイクが可能となる．

(b) 同期式では制御部とデータ転送部を同一クロックに同期させる必要があるが，非同期ではその必要がない．そのため，データ転送部の速度を制御部の速度によらずに設定可能である．

(b) の利点を生かすため，バンド幅を重視し，データの連続転送能力の向上を目指して設計した．そして，目標と関係ない部分はできるだけシンプルに設計した．そのため，第 2 章で述べたようにデータ表現方式として束データ方式を用いた．

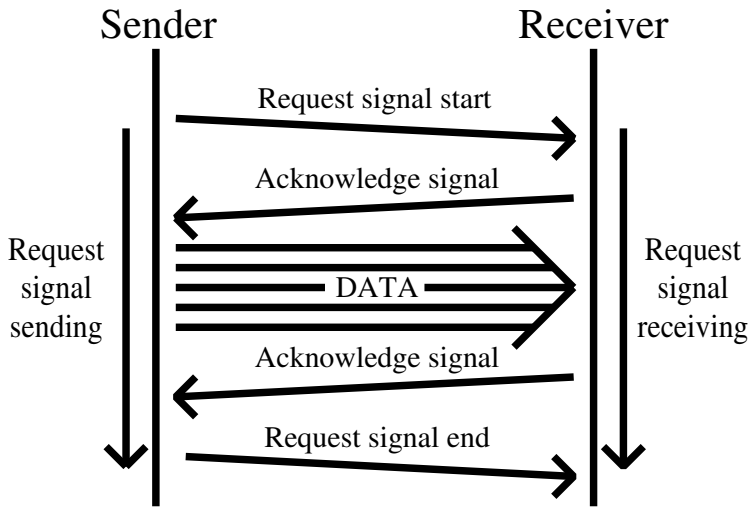

図 4.1: 転送時のハンドシェイクプロトコル

SAS ではハンドシェイクは転送開始・終了時のみとし，ブロック転送を行う際，データは Wave Pipeline 方式で送られることとした．図 4.1 に SAS の転送時のハンドシェイクプロトコルを図 4.2 にプロトコルのタイミングチャートを示す．通常，束データ方式においてデータの有効を表す信号を Request 信号と呼ぶが，データ転送時のハンドシェイクに用いる Request 信号と区別するため，この章では ready 信号と呼ぶ．

図 4.1，図 4.2 に示したプロトコルでは始めに Sender から Receiver に出力要求である Request 信号を送る．Request 信号は転送が終了するまで送り続けられる．スイッチにより，出力可能であると判定された場合，Receiver に Request 信号が転送され，Receiver は Acknowledge 信号を返す．Acknowledge 信号を受け取った Sender は Data の転送を開始する．ready 信号が High の時の Data 線上の信号が有効な値である．全てのデータを受け取った Receiver は Acknowledge 信号を返し，Acknowledge 信号を受け取った Sender は Request 信号の出力を止め，転送が終了する．

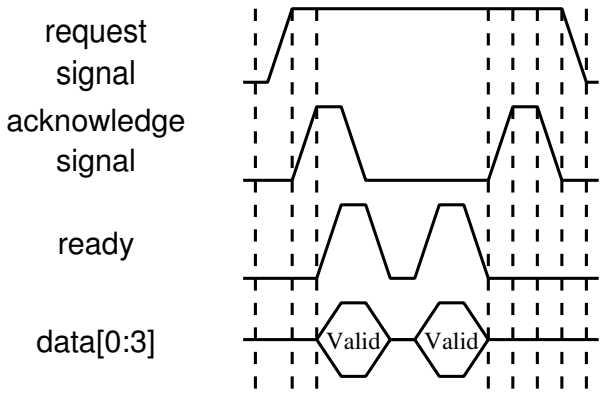

図 4.2: 転送時のタイミングチャート

一般のデータ転送方式とは異なりデータ送信通知信号（Request 信号）を，転送を行うための一連の信号のやりとりの一番始めから終わりまで出し続ける点が特徴である．これは Request 信号を送る際レイテンシが増加するのを防ぐためである．

束データ方式ではタイミングチャートで示したように ready 信号が High の時の値を有効な値と判断するため，ready 信号を含めた全てのデータ線において，同期がずれた信号線が存在した場合，転送しているデータとは異なったデータが有効な値として転送される恐れがある．このため，同じポートの各信号線におけるデータの遅延ができる限り同一になるように設計した．なお，実際に高速データ通信を行うためには直流成分を除去するためコード化が必要である．しかし，SAS では EDA ツールが使用できないため複雑な回路を設計することが困難であること，及び，高速な動作を実現するため可能な限り必要な処理を減らす設計方針に反することからコード化は Sender 及び Receiver で処理することとした．

データの受信終了時に Acknowledge 信号を転送先から転送元へ送信するが，データの転送量はデータの転送開始時にヘッダに付け加えることとする．

一般にネットワークで用いられるスイッチは複数個を縦列接続してネットワークを形成する必要がある．このため，SAS も制御信号を縦列接続されたスイッチに送るためのポートを設けた．

4.3　スイッチの設計

4.3.1　全体構成と信号線

4.3.1.1　スイッチの構成

前節での転送方式を実現するため，スイッチの各ポートを以下の 4 つのステージから構成する．

(a) 出線競合検出ステージ（head of line conflict detection）

(b) アービトレーションステージ（arbiter）

(c) 出力先決定ステージ (output port selection)

(d) 出力ステージ（output port）

4.3.1.2　各ポートの信号線

次に各ポートの入出力線を以下のように定める．信号線名は，-in，-out で，それぞれ対応する入力と，出力を示すようになっている．1 ポート当りのデータ幅は (m+1)bit，1 チップ当りのポート数は $2^{(n+1)}$ ポートとする．SAS では m=3，n=1 となっている．Request 線，Acknowledge 線，Data[0:m] 線，Ready 線，Conflict 線は縦列接続のための入出力を設ける．

- Request(-in,-out):ハンドシェイク要求を示し，信号のやりとりが終わるまでアサートされ続ける．
- Acknowledge(-in,-out)：Request 信号の応答信号であり，受け手が送り元に対してデータの受信の準備ができたこと，およびデータの受信を終えたことを伝える．
- Data[0:m](-in,-out),Ready(-in,-out): データ (Data[0:m]) は前述の通り束データ方式で送られ，Ready 信号は，このデータの有効を示す．
- Conflict(-in,-out):送り先に他のポートが出力しており，出力することが不可能な場合アサートされる．
- Addr[0:n]:(n+1)bit でデータの送り先ポート番号を伝える．スイッチはこの信号で指定したポートへデータを送る．

4.3.1.3　処理の流れ

各ステージと主な信号線は図 4.3 のような流れで処理される．

All Request，All Addr はそれぞれ全てのポートの Request 信号，Addr[0:n] 信号を表す．全てのポートの Request 信号と Addr[0:n] 信号が出線競合検出ステージに入力され，その結果と Request 信号を元にアービトレーションステージが実行される．その結果と Addr[0:n] 信号を元に出力先決定ステージにおいて，出力先となるポートの出力ステージに Ready 信号と Data[0:m] 信号が出力される．各ステージはポート毎に設けており，出力先が異なっていれば全てのポートが同時に信号を受け付けることができる．

4.3.2　各ステージの構造

本設計では，出力先決定ステージのみ Active-Low にし，他のステージは Active-High にすることでレイテンシを抑えている．

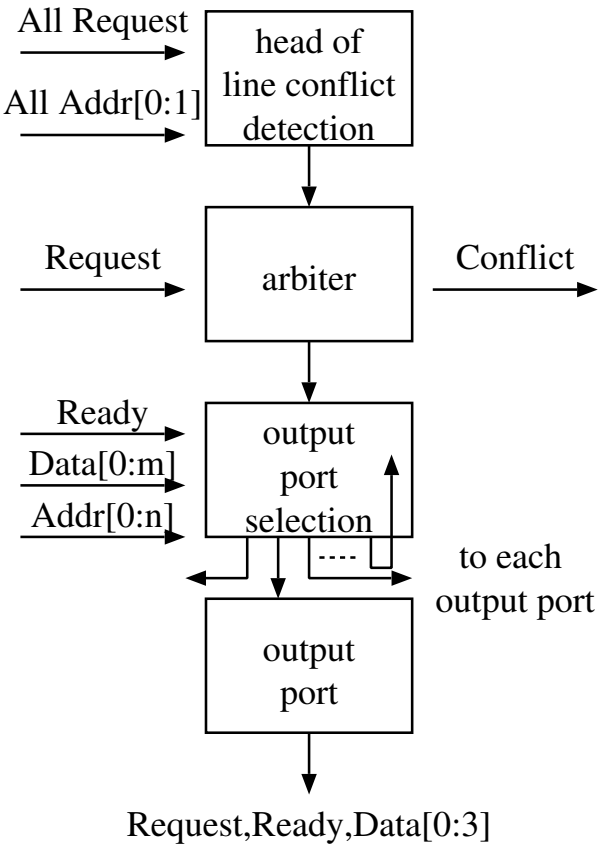

図 4.3: 制御用信号，データ信号の流れ

4.3.2.1　出線競合検出ステージ

出力の宛先に競合が発生していれば出力をアサート，発生していなければ出力をネゲートする．

各ポートの Request 信号，Addr[0:n] 信号を当該ポートの信号と XOR 素子で比較し，すべて一致するポートが 1 つでも存在した場合にはアサート，すべて一致するポートがない場合はネゲートする．

4.3.2.2　アービトレーションステージ

出線競合ステージからの信号と Request 信号を元にアービトレーションを行い，データと制御信号を出力可能なら出力をアサート，そうでなければ出力をネゲートする．

出線競合検出ステージからの信号が先にアサートされていれば，Request 信号がアサートされていても出力をネゲートする．そうでない場合は Request 信号がアサートされていれば出力をアサート，ネゲートされていれば出力をネゲートする．

このステージは，他のポートが Addr[0:n] で指定したポートに対して，先に出力している場合，Request 信号がアサートされた場合においても，ネゲート出力を維持しなくてはならない．

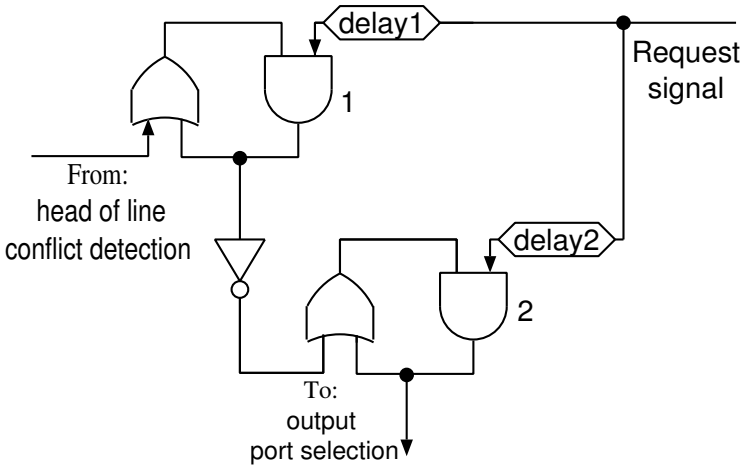

図 4.4: アービトレーションステージ

そのため，アービトレーションステージは図 4.4 に示す特殊な構造をしている．delay1,delay2 はそれぞれ，図 4.4 の 1，2 の AND 素子への，出線競合検出ステージを通ってきた信号に対して Request 信号の到着時刻を遅らせると共に，1,2 の AND 素子への信号の到着時刻もずらす (delay2 > delay1) 働きをする．これらの遅延素子により，アービトレーションステージが，メタステーブル状態に陥ることがないように保証している．

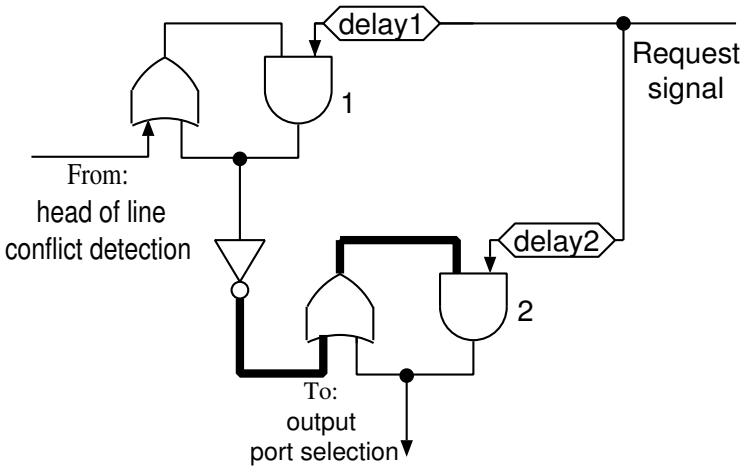

図 4.5: アービトレーションステージ (要求が存在しない場合)

図 4.4 の回路で出線競合検出ステージからの入力がネゲートされており，Request 線もネゲートされている場合，図 4.5 に示す状態となる．ここでは太い部分が High レベルを示す．この場合，要求が存在しないので，出力先決定ステージへの出力はネゲートされる．

図 4.5 の状態から，Request 線がアサートされると図 4.6 に示すように出力先決定ステージへの出力がアサートされる．

図 4.6 の状態で出線競合検出ステージからの入力がアサートされる，このため図 4.7 に示すように，他のポートが同じポートに出力を試みた場合，出力先決定ステージへの出力

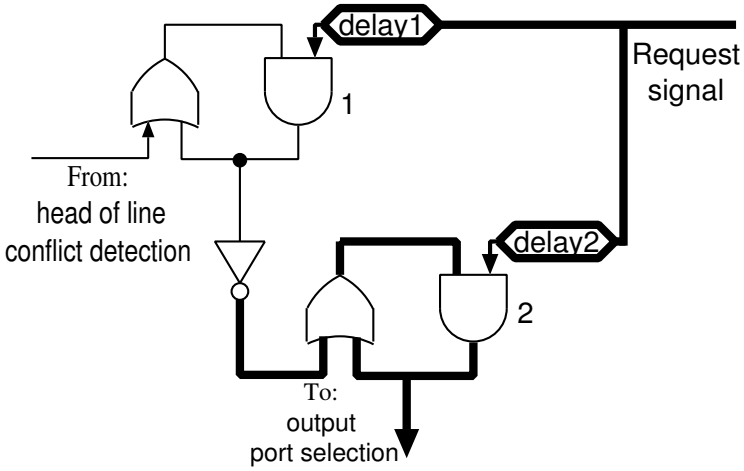

図 4.6: アービトレーションステージ (要求が受け付けられ出力を開始)

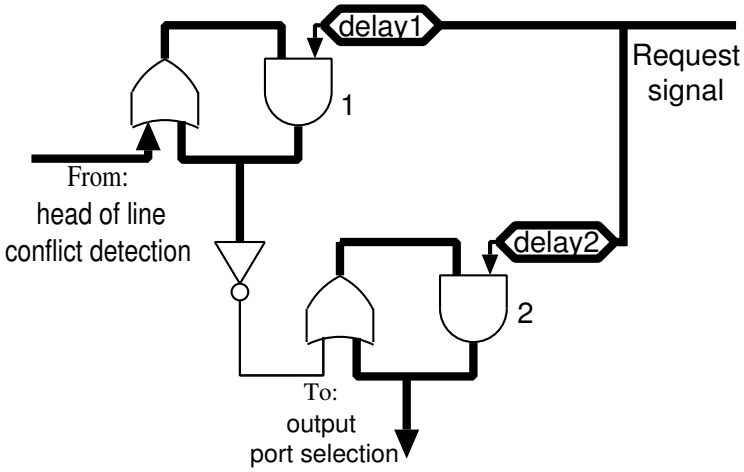

図 4.7: アービトレーションステージ (出力を維持)

はアサートされた状態を維持する．すなわち，出力中に他のポートが同じポートに出力を試みた場合においても，影響を受けない．

逆に，図 4.8 に示すように，他のポートが現在出力中のポートへの出力要求は拒否される．

4.3.2.3　出力先決定ステージ

出力先決定ステージは，3 入力 NAND と 4 つのインバータが 1 組となっている (図 4.9)．送信に必要な信号は Request 信号，Ready 信号，Data[0:m] 信号であるため，計 $m+3$ 本の信号線を必要とし，出力ポートは $2^{(n+1)}$ ポートあるため，$(m+3)\times 2^{(n+1)}$ 組設ける．また Acknowledge 信号，Conflict 信号を各信号の送り元の出力ステージへ送るため，NAND を 2 個 ×(各ポート向けに $2^{(n+1)}$ 個) 設ける．

これらに対応する出力ステージの要求は，データ線に有効なデータを載せると共に各信

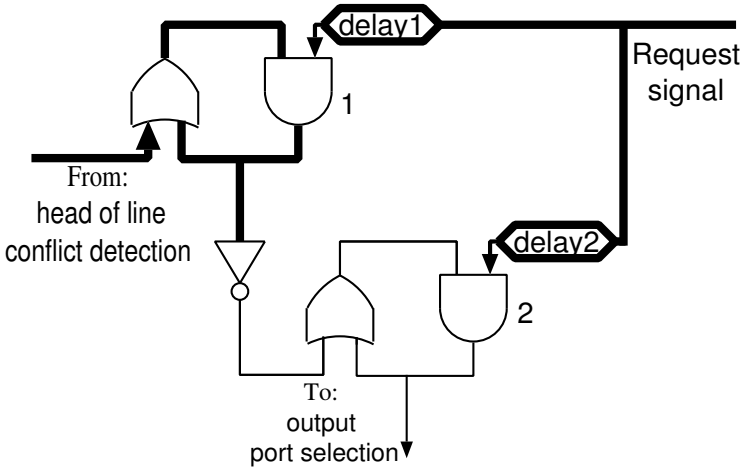

図 4.8: アービトレーションステージ (出力要求を拒否)

号線をアサートすることによって行なう.

要求を出すことのできる条件は，既にデータが有効になっていること，そのポートが出力を許されていること（アービトレーションステージの出力がアサートされていること），Addr[0:n] 線によって出力先として選択されていることの 3 点である.

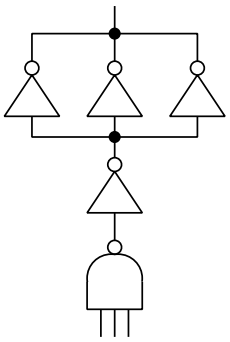

図 4.9: 出力先決定ステージの基本回路

図 4.9 のような構成になっているのは単一の NAND 素子のドライブ能力では出力ポートまでの回路をドライブすることができなかったためである．インバータによる増幅を行わない場合，動作速度が遅くなり，入力された信号の長さと出力する信号の長さが著しく異なる場合がある．一般にインバータを複数並列に繋げ，回路のドライブ能力を強化し高速な転送を行う方法は，ドライブする回路にインバータを配置し，1 つのインバータがドライブする配線の長さを短くする方法に比べ欠点が多い．しかし，本設計では各信号線が可能な限り同じ長さの配線を通ることを重視し，この設計を取った.

4.3.2.4　出力ステージ

出力ステージは各ポートの各出力先決定ステージからの信号を AND 素子でまとめている．この出力ステージへの入力は Active-Low(出力は Active-High) である．

本スイッチの設計上の特徴は，データ線を流れるデータ信号がほとんど素子を通過せずに出力される点にある．データ線が影響を受けるのは出力先決定ステージと出力ステージの 2 つだけであり，データ線のデータが通過するのは 3 入力 NAND 素子，NOT 素子 2 個と AND 素子が 2 個だけである．このため，データはこのスイッチを低レイテンシで通過することができる．

4.4　チップへの実装

本実装では VDEC の ROHM 社の 0.6μm プロセスを利用したため，チップのピン数の制約に応じて入出力ポート数とデータのビット幅を決定した．ポート数を 4 とし，1 ポートあたりのビット幅を 4 bit としたのは SAS では 1 ポート当たり信号線が 18 本必要であり，チップにおける利用可能なピン数は 87 本だからである．

チップは正方形ダイを用いるので各辺を一つのポートに割り当てることとした．

ここまでで述べた回路を製作するにあたり，セルライブラリは用いず，すべての回路素子のレイアウトを行ない，セルの配置配線も完全に人手によって行なった．すなわち，このスイッチはフルカスタムレイアウトで作成した．これは，以下の条件を満足する必要があるためである．

- 前節で述べたとおりデータはほとんど素子を通過せず出力されるため，配線遅延が支配的である．配線を強力にドライブするためには，現在の VDEC でサポートされているライブラリではドライブ能力が不足し，オリジナルのセルが必要となった．最もドライブ能力が必要とされる出力先決定ステージから出力ステージへ信号を伝搬させる際，ROHM 社の 0.6μm プロセス用の標準的なライブラリを使用した場合，500MHz 程度の信号の送信が可能である．しかし，今回特別に作った素子の場合，拡散容量が減少するよう考慮し，回路が出来る限り密集するように設計したため，同程度の面積で 4GHz の信号を送ることが可能である．この素子の使用により高い周波数の信号を波形を崩すことなく転送することが可能となった．

- 今回のスイッチは，データと Ready 線の遅延合わせを全てのポートに対して行なう必要がある．このため，各ポートのデータのレイテンシはポートに関わらず一定である事が要求される．データのレイテンシをポートごとに一定にするために本設計では点対称の回路を製作する手段を選択した．自動配置配線機能を用いると，このようなレイアウトにはならない．このため，配置，配線も人手で行なう必要があった．

Ready 信号と Data[0:3] 信号は，ずれる事が許されないため，極力同じレイアウトの素子，同じ長さの線を通る設計とした．

Cadence 社の LayoutPlus を用いて作成したフルカスタムレイアウトを図 4.10 に示す．前述のようにほぼ点対称のレイアウトとなっている．図中のように各ステージを配置した．出線競合検出ステージが中心近くにある．これは各ステージからの Addr[0:3] 信号をほぼ

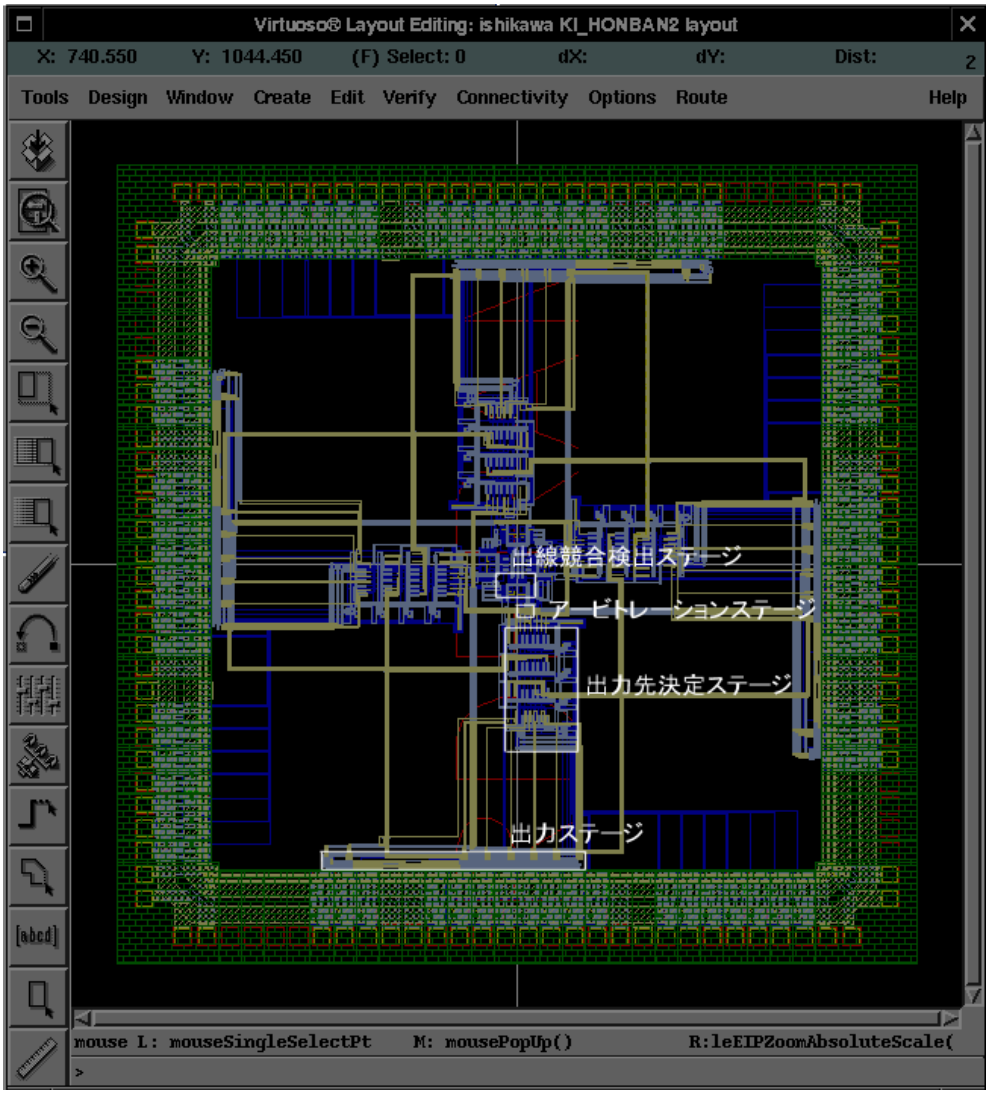

図 4.10: レイアウト全体図

同時に受け取ることを可能にするためである．離れた位置に配置した場合，配線遅延による信号のずれを計算し設計する必要がある．アービトレーションステージにおいてもタイミングのずれは問題になるが，必要な他のポートからの信号は全て出線競合検出ステージから 1 つにまとめられ出力されるたものを受け取る．これにより，タイミングが大きな問題にはならないため中心から少し離れた位置に配置している．出力先決定ステージから各ポートへ信号を出力するための信号線が出ている．前述のように同じポートへ出力される信号線はほぼ同じ長さになるように調整されている．出力ステージは出力ドライバの近くに配置されており，各ポートからの出力をまとめ出力ドライバに送っている．

青の領域及び白の領域が配線，緑の領域が N 型拡散層，ピンクの領域が P 型拡散層，ピンクがかった赤の線がゲート，黄色の領域が N － WELL を意味している．

ゲート数は NAND 換算で 1030 となった．

4.4.1　誤動作の可能性

上記 4 ステージのうち，出線競合検出ステージ，出力先決定ステージ，出力ステージは単純な組み合わせ回路であるが，アービトレーションステージはフィードバックを含むため，正しく動作させるために設計上の工夫がなされている．

図 4.4 のアービトレーションステージは，他の入力ポートの Request 信号が自ポートの Request 信号より若干遅く，同時に，もしくは若干早くアサートされた場合に，出線競合検出ステージからの信号と Request 信号との関係によりハザードを引き起こす可能性がある．これを防ぐために delay1，delay2 の 2 つの遅延素子を使用した．delay1，delay2 は以下のような条件を満たす必要がある．

(a) delay1 は出線競合検出ステージからの信号と delay1 を通った信号がほぼ同時に AND1 に到達するように設計する必要がある．

(b) 出線競合検出ステージからの信号が AND1 → inverter → OR2 の順に伝搬し，AND2 に到達するまでの時間（始めに通ることになる OR1 素子は含まない）を a，delay2 を通過した信号がその後，AND2 を通過し OR2 素子に到達するまでの時間を b とする．このとき，a-b>(delay2 の通過時間-delay1 の通過時間) が成り立つ必要がある．

(c) 出線競合検出ステージからの信号が OR1 → AND1 → inverter → OR2 の順に伝搬し，AND2 に到達するまでの時間を c とする．このとき，c>delay2 となる必要がある．

(a) の条件は同じ Request 信号が出線競合検出ステージを通った信号と，直接アービトレーションステージに到達した信号の間で到着時刻が大きくずれ，各信号の長さが変わってしまうのを防ぐためのものである．この条件は多少のずれが許される．

(b) の条件は後から他のポートが出力しようとした際に出力不能になるのを防ぐためにある．この条件が満たされない場合，出力する権利があるのにもかかわらず出力しないケースが出てくるため，スイッチの利用効率が悪くなる．

(c) の条件はアービトレーションを正しく行うためのものである．この条件が守られない場合複数のポートの入力が 1 つのポートに同時に出力可能となる．

これらの条件のうち，3 は必ず満たされなければならず，1，2 は満たされた方が効率の良いスイッチとなる．また，素子は温度などの環境によって動作速度が変わるのを考慮しなければならない．このため，想定している動作環境下で回路を正しく動作させる delay1，delay2 の長さをシミュレーションにより検証し，決定した．

また，Request 信号をアサートした状態で出力先ポートを変更した場合アービトレーションが正しく動作しないことが確認された．しかし，このスイッチのデータ転送では，出力中の他のポートへの出力先変更はプロトコル違反であるため，実用上の問題は生じない．

4.4.2　Conflict 信号

Conflict 線は同時に信号を返した場合，同時に再度 Request 信号が入力され衝突が続く恐れがあるため，意図的に各ポートで異なった遅延になる設計とした．

連続転送を前提としているため，転送時間は Conflict 信号の遅延と比較し十分長い．そのため，同期式の回路とは異なり，Conflict 信号の遅延が長いポートにおいても出力可能

になる可能性は低くなるが，必ず出力不可能になるわけではない．だが，フェアネスを考慮にいれた場合，現状の設計には問題がある．

ここで，さらにフェアネスを向上させる回路について述べる．フェアネスを向上させる回路の Conflict 信号の遅延回路を図 4.11 に示す．

SR-latch は Request 信号及び Conflict 信号によって制御される．初期値は High である．Conflict 信号が High の場合 SR-latch の出力 (Q) は High になり，Request 信号が High の場合 SR-latch の出力 (Q) は Low になる．

この回路により次のようにフェアネスが向上する．一度出力を行ったポート (Request 信号が High になったポート) は選択される Conflict 信号の遅延が長いものになるため，出力の優先順位が下がった状態になる．出力が行えなかった回路 (Conflict 信号が High になったポート) は Conflict 信号の遅延が短いものになるため，出力の優先順位が高い状態になる．前述のように連続転送を前提とするため，転送時間は Conflict 信号と比較し十分長い．そのため，出力可能な状態になるのは確率的に決定する．この特徴のため，本回路により完全に平等なフェアネスは保証されないが，転送速度に全く影響を与えずにフェアネスを高めることが可能である．

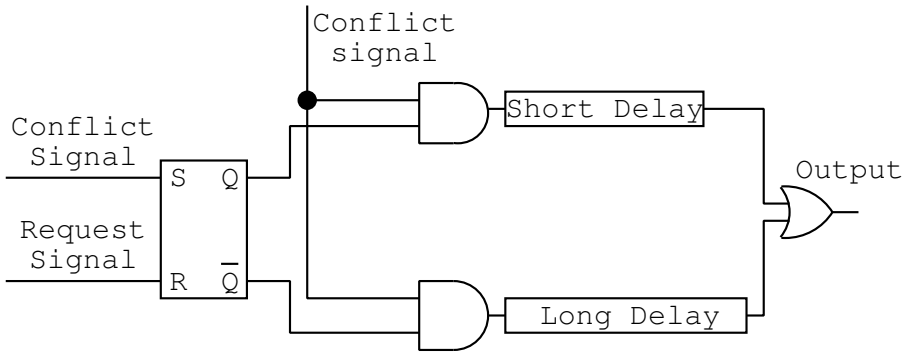

図 4.11: コンフリクト回路の遅延回路

4.5　評価

4.5.1　評価法

本論文でスイッチの評価を行うにあたり全ての信号は Return-to-Zero 方式で転送されるとした．Data 線で信号を送る際，直流成分を除去するため NRZI(Non Return to Zero Inverted) などのコーディングを行う必要があるが今回の評価では行っていない．

4.5.2　回路シミュレーションによる評価

4.5.2.1　評価環境

実機での評価に先立ち回路シミュレーションによる評価を示す．Cadence 社のツールを用いて評価を行った．LayoutPlus を用いてレイアウトされた回路から，PDRACULA を

用いて結線情報を抽出し，HSPICE を用いてシミュレーションを行った．入出力バッファ部分を含んだシミュレーションは行うことが不可能であったため，この部分をのぞいたシミュレーションを行い，必要とする場合，入出力バッファ部分の遅延を加算した．スペックから，入力バッファの遅延が最大 0.135ns，出力バッファの遅延が最大 1.748ns であるので入出力バッファを通過するために必要な時間は多少の余裕を見て 2ns と見積もった．

4.5.2.2　シミュレーションによる転送速度の評価

このチップが前提としているプロトコル (図 4.1) で一つ信号を送るのに必要な時間をシミュレーションしたところ図 4.12 のようになり，9.3ns 必要であることがわかった．図 4.12 中の 1 で Request 信号が送られ，2 でそれに対して Acknowledge 信号が返され，3 でデータが転送され，4 で Acknowledge 信号が返され，5 で Request 信号の送出が終わり，一つのデータの転送が終了している．このプロトコルは大量のデータを一度に送る場合に効率良く動作する設計となっているため，一つのデータを送る場合の転送速度はさほど速くない．また，入出力バッファの通過時間を考慮し同じプロトコルで転送すると図 4.13 のように 23.25ns 必要となり，データを単独で転送する際に入出力バッファの通過時間が大きく影響していることがわかる．

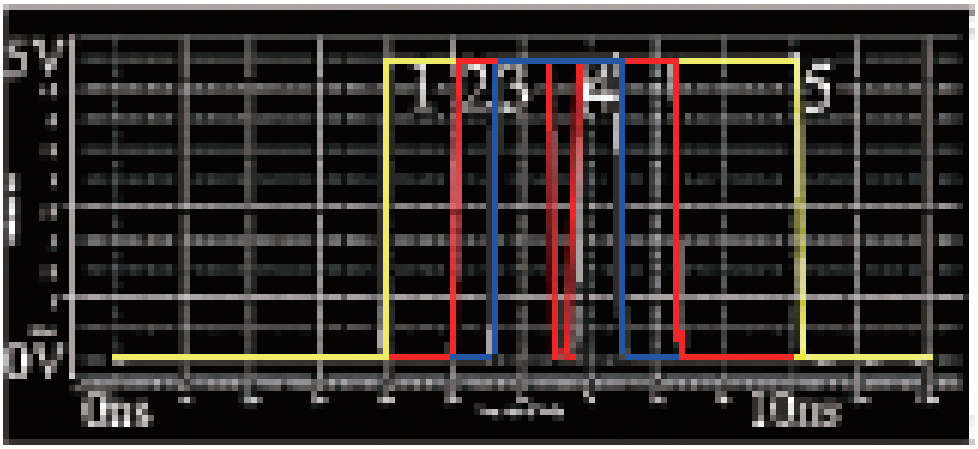

図 4.12: プロトコルを遵守し一つデータを送った場合 (入出力バッファ遅延考慮せず)

データを一つ送るたびに Acknowledge 信号を返す場合は図 4.14 に示すように一つの信号を送るのに 12.75ns 必要とする (入出力バッファの遅延を考慮)．つまり，データを一つ転送するごとに Acknowledge 信号を返す方式では，78Mbit/s の転送速度しか実現できない．

次にこのチップのプロトコルに沿った使い方である連続転送モードでの転送速度をシミュレーションした．入力として，方形波（実際には立ち上がり立ち下がりが 0.0001ns である台形波），より現実的な立ち上がり立ち下がりが 0.1ns の台形波，正弦波の 3 種類の波形を用いた．方形波の場合を図 4.15 に示す．ライン当りの転送速度は 2.00Gbit/s となった．また，台形波の場合を図 4.16 に示す．ライン当たりの転送速度は 1.82Gbit/s となった．正弦波の場合を図 4.17 に示す．ライン当りの転送速度は 1.85Gbit/s となった．すなわち，チップの限界性能としてはチップの持つ 4 ポート (各 4 ライン) をあわせて 32Gbit/s の

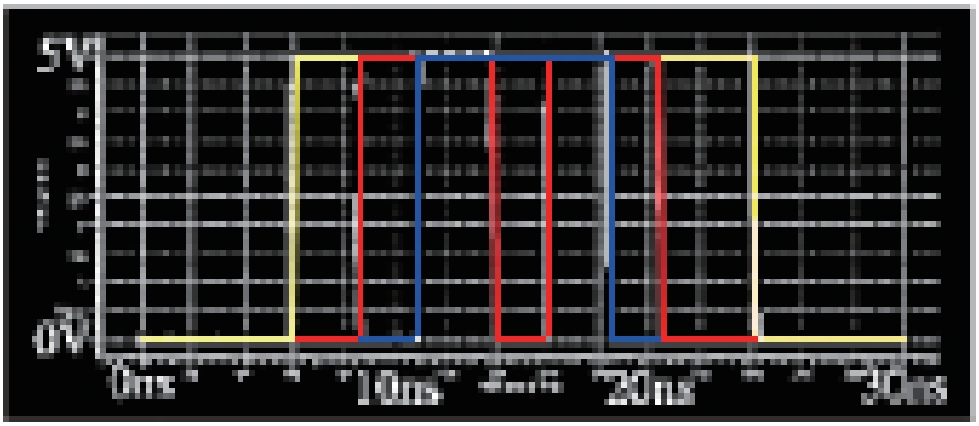

図 4.13: プロトコルを遵守し一つデータを送った場合 (入出力バッファ遅延考慮)

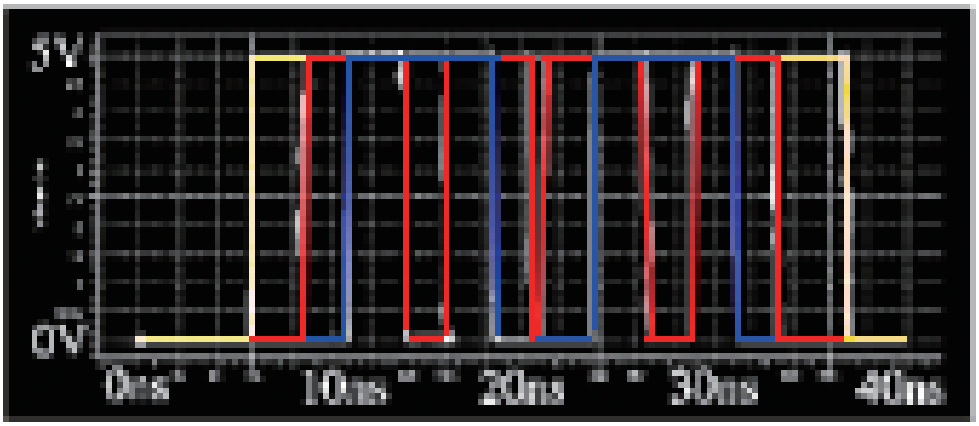

図 4.14: 一つデータを送るごとに Acknowledge 信号を返す場合

転送速度を持つ．また，今回，実機でのテストで使用する正弦波を入力とした場合はチップ全体として 29.6Gbit/s の転送速度を持つことがわかった．

いずれの場合においても同時に入力した Ready 信号と Data[0:3] 信号の出力波形はほぼ完全に一致することが確認された．

4.6　実チップによる評価

4.6.1　評価用ボード

実チップを用いて評価を行なうために図 4.18 に示すテスト基板を作成した．設計図は図 4.19 のようになっている．左側のポートは Request 線， Acknowledge 線， Conflict 線，Data 線の遅延を測る事が出来るよう，これらの信号の入力用スイッチには，SR ラッチによるチャタリング防止回路を備えている．また，50MHz，100MHz，200MHz のクロックジェネレータの出力がデータとして入力される設計とした．下のポートと右のポートは組

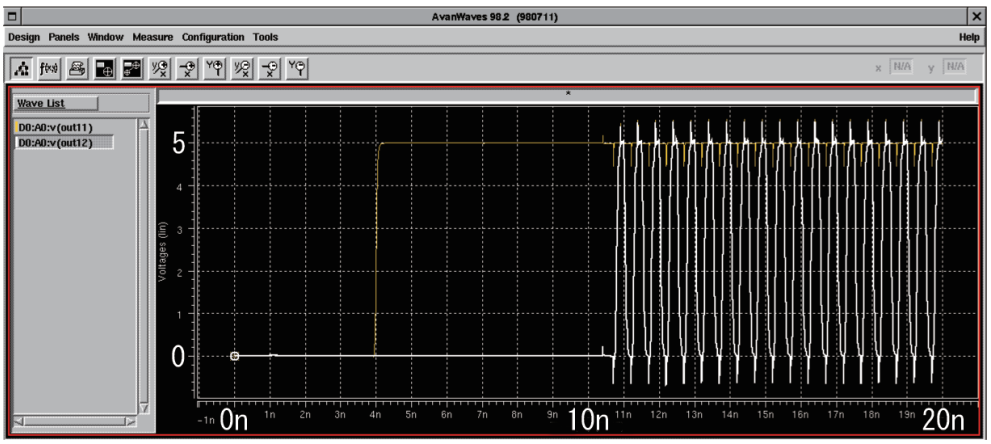

図 4.15: 方形波を連続転送した場合

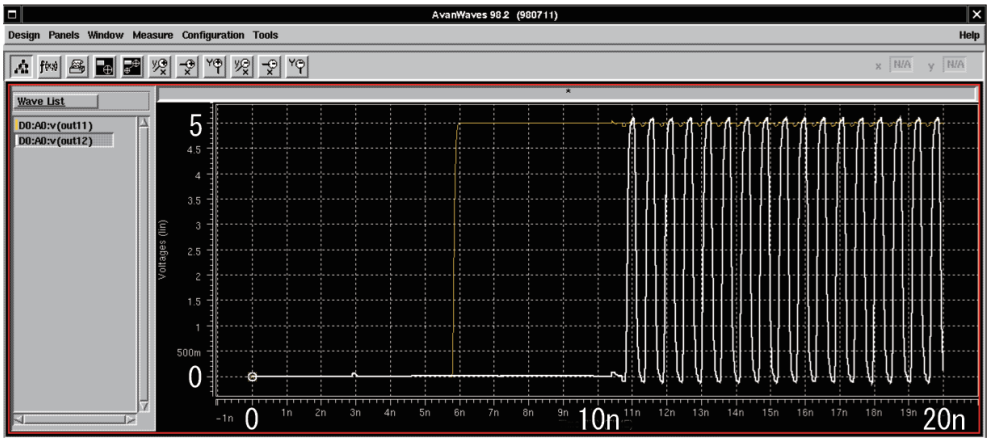

図 4.16: 台形波を連続転送した場合

になっており，同時に Request 信号が出された場合，両方の要求を却下し，Conflict 信号を正しく出力することを確認できるように，SR ラッチによるチャタリング防止回路を備えている．上のポートはチャタリング防止措置を行わない信号入力に対する挙動を見るため，各信号線，データ線を入力用スイッチに直結し，直接，アサート，ネゲートできる機能のみを持つ．すべてのポートは入力線への入力信号と，出力線からの出力信号を検出可能なポストを持つ．

4.6.2　実チップの動作

図 4.18 の基板を用いて実チップが正しく動作をするかテストを行った．図 4.19 に設計図を示す．本チップ試作は東京大学大規模集積システム設計教育研究センターを通し ROHM(株) および凸版印刷 (株) の協力で行われた．ほぼすべての機能が正しく動作したが，アービトレーションの動作で以下の問題があることが確認された．4 ポートのうち，3 ポートではアービトレーションを正しく行なうことが可能だが，図 4.18 で上方に位置する

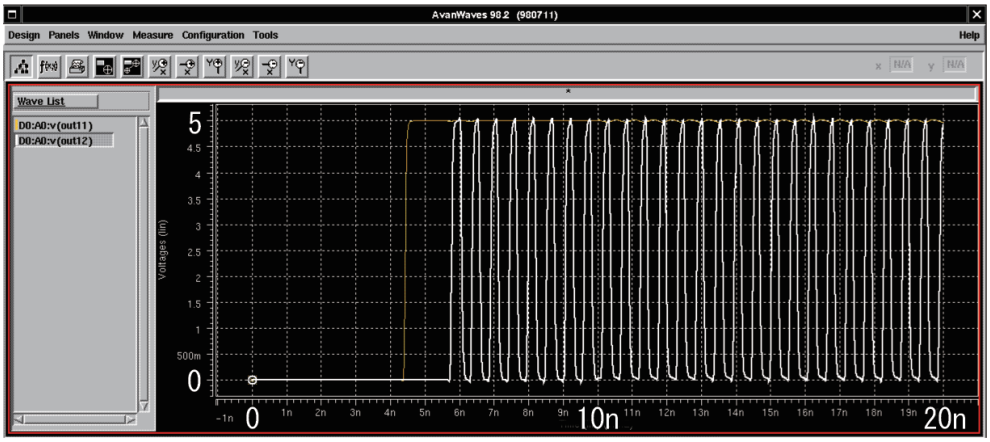

図 4.17: 正弦波を連続転送した場合

ポートが他のポートがすでに出力を行っているポートに出力を始めようとすると，出力が可能となる．この際，上方に位置するポートが割り込んで出力している間も，割り込まれたポートからの出力は正しく出力され続ける．つまり，上方に位置するポートの出力と，割り込まれたポートの出力の OR を取ったデータが出力される．チップの向きを変えた場合においても常に上方に位置するポートが割り込む事が出来る．他のポートはチャタリング防止機能を持っているのに対し，上方に位置するポートのみ基板側に SR ラッチによるチャタリング防止機能が備わっていないことから，チャタリングによりアービトレーション回路内にある配線がループしている回路がメタステーブル状態になり，正しく動作できなくなっている事が予想される．

Request 信号の制御回路だけを考えれば，基板側にチャタリング防止策を取っている以外はほぼ同一構造の左側に位置する回路は必ず正しく動作することから，スイッチのチャタリングを起こした入力が Request 信号として入力された場合には対応できないことがわかった．この問題はボード上にチャタリング防止回路を持つことにより解決可能な問題であると考えられる．なお，HSPICE シミュレーションによりチャタリングによるアービタのメタステーブル状態の再現を試みたが再現することはできなかった．

下方のポートおよび右方のポートを使い，同時に同じポートに出力要求をした場合の挙動をテストしたが，下方のポート，右方のポート，共に出力を拒否され，Conflict 信号が返って来るという設計通りの動作をした．また，その際，Conflict 線の遅延の微調整の効果を調べ，設計通り下方と右方の Conflict 信号の出力タイミングが必ずずれることを確認した．

4.6.3　遅延の測定

各線の遅延を測定した．結果は表 4.1 のようになった．実機の遅延の 10-12ns というのは 10ns より大きく，12ns 未満であることを意味する．

シミュレーションの値と実機での値との間には 4-5ns の開きがあるが，これは入力のファンアウトと出力用ドライバの周波数特性の問題であると考えられる．この問題に関しては

図 4.18: テスト基板

後述する．

4.6.4　実機での転送速度

実機での転送速度を測るために 50MHz，100MHz，200MHz のクロックジェネレータがデータ線に接続されている．それぞれこのチップの動作電圧である 5V の電圧でプルアップされている．シングルエンデッド転送では一般に 5V の振幅の場合 100MHz 程度の信号が転送限界であるが，限界性能を見るため 200MHz の信号を用意した．

クロックジェネレータの特性により，50MHz の入力波形は振幅が 6V 近く確保されたが，100MHz の入力波形は 4V，200MHz の入力波形は 3.5V の振幅となった．スイッチからの出力と比較した場合，50MHz，100MHz の場合の出力波形は 5V 程度の振幅を維持したが，200MHz において入力波形に対して振幅の減衰が見られ，入力波形の振幅が 3.5V であるのに対し出力波形の振幅は 2.5V 程度となった．この様子を，入力波形を図 4.20，図 4.21，図 4.22 に，それを入力とした場合の出力波形を図 4.23，図 4.24，図 4.25 に示す．

この評価により SAS はシングルエンデッド転送において期待される 100MHz の信号を 5V の振幅でドライブすることが可能であることが実証された．

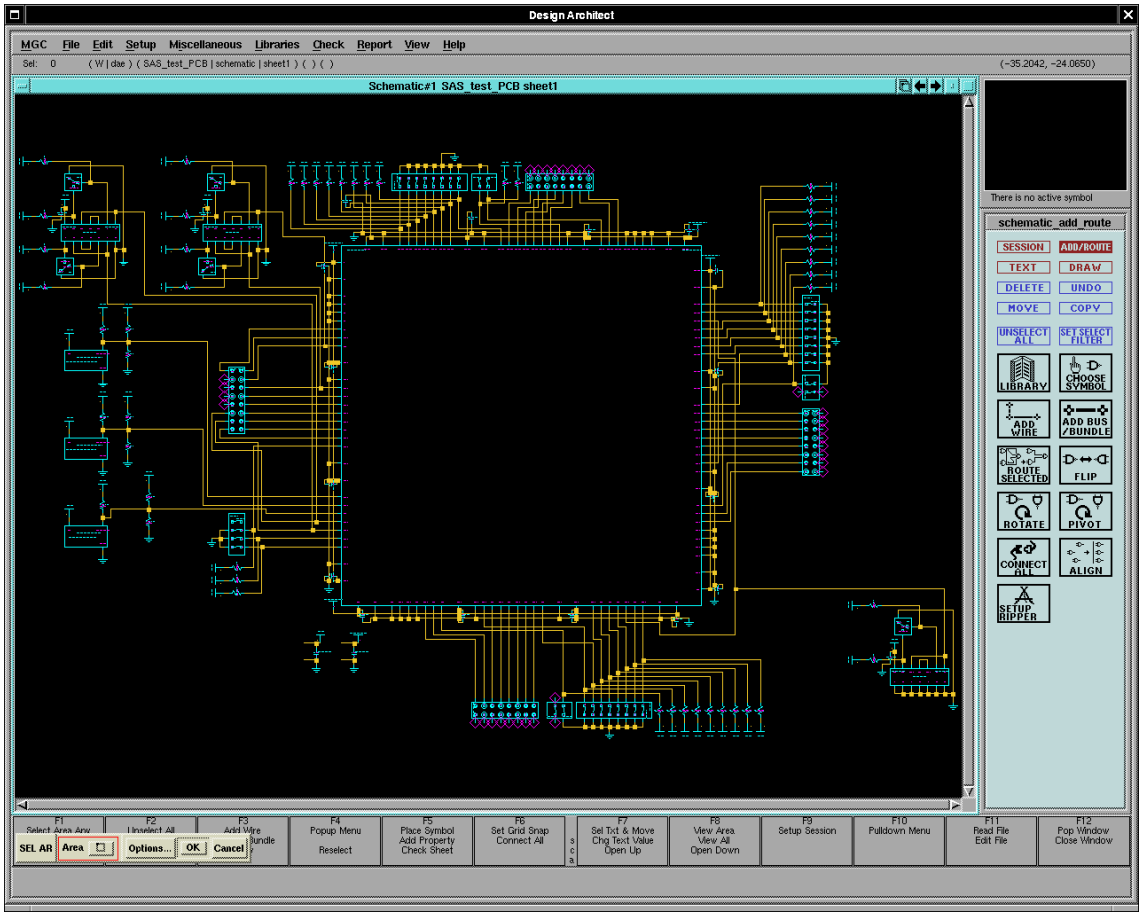

図 4.19: テスト基板の設計図

4.7　まとめ

完全に非同期で動く高速スイッチを開発した．このスイッチは 4 入力 4 出力で一つのポートにつき 4 ビットのデータ幅を持っている．0.6μm プロセスで製作されたがシミュレーションによる評価においては 1 ポートにつき 8Gbit/s，チップ全体では 32Gbit/s の性能を示した．実チップは，条件付きながら正しく動くことが確認されたが，転送方式の問題からチップ全体で 1.6Gbit/s から 3.2Gbit/s の性能となった．

今回は，5V のシングルエンデッド転送の限界により性能が制限されたが，より振幅の狭い転送方式を用いることにより更に高速な転送が可能になる可能性がある．また，ディファレンシャル転送などを導入することにより更に高速な転送を実現できる可能性がある．今回は面積上の制約及び素子の反応速度上の制約により単純なスイッチの製作に留まったが，非同期型 Wave Pipeline 技術は，より優れたプロセスの利用により，ネットワークプロセッサなど，ネットワーク上のデータに対してある程度複雑な演算を施す IC への利用が考えられる．また，ブロードキャスト機能やバーチャルサーキット等，高い機能を持つスイッチにも応用可能な要素技術である．

表 4.1: 信号の Delay

line	遅延 (入出力バッファ考慮)	実機の遅延
Request 線	3.0ns (5.0ns)	10-12ns
Acknowledge 線	1.5ns (3.5ns)	6-8ns
Ready 線	0.6ns (2.6ns)	6-8ns
Data 線	0.6ns (2.6ns)	6-8ns
Conflict 線	1.8ns (3.8ns)	6-8ns

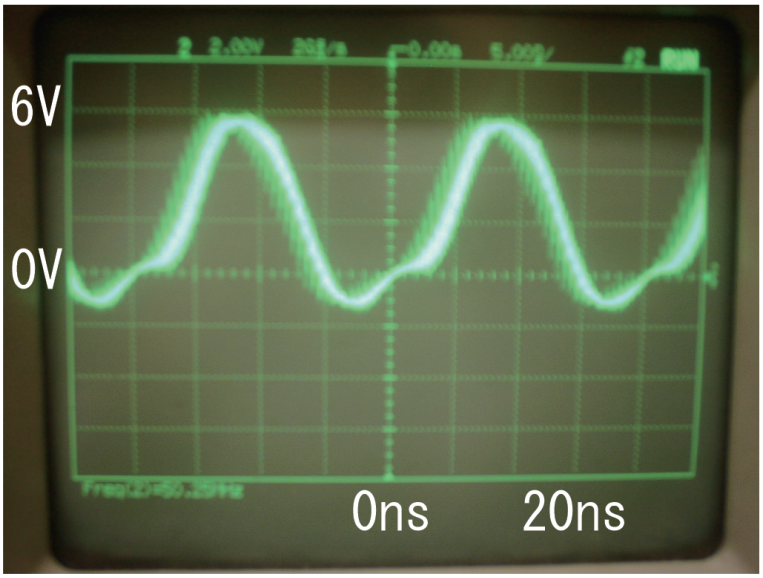

図 4.20: 入力波形 (50MHz)

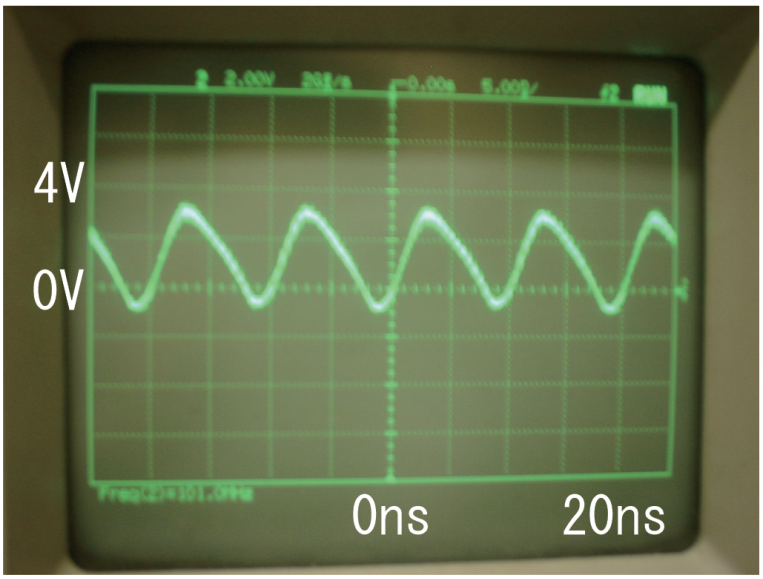

図 4.21: 入力波形 (100MHz)

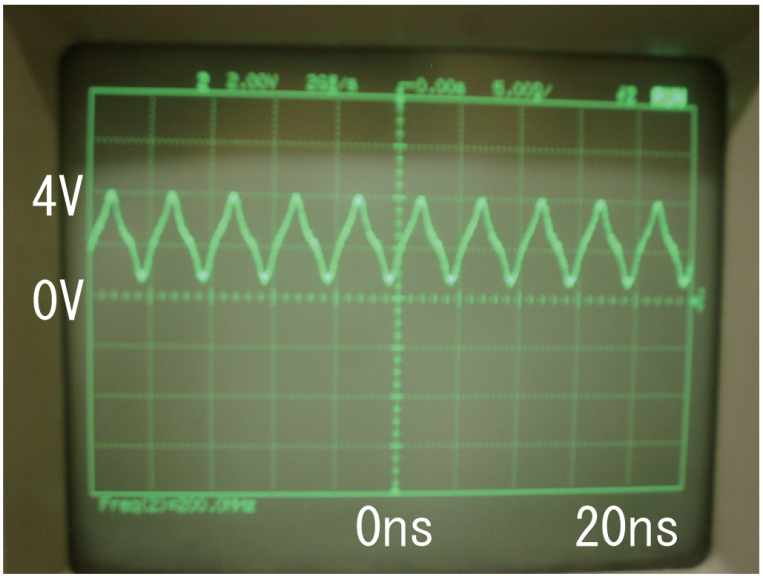

図 4.22: 入力波形 (200MHz)

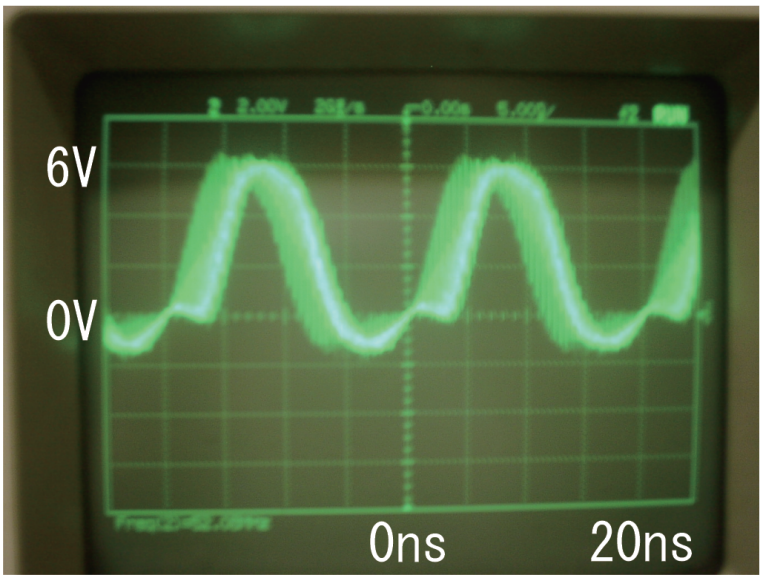

図 4.23: 出力波形 (50MHz)

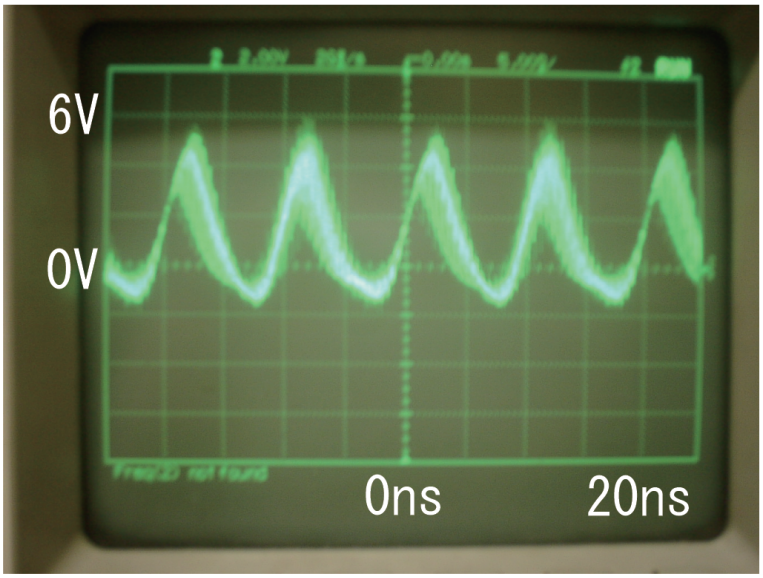

図 4.24: 出力波形 (100MHz)

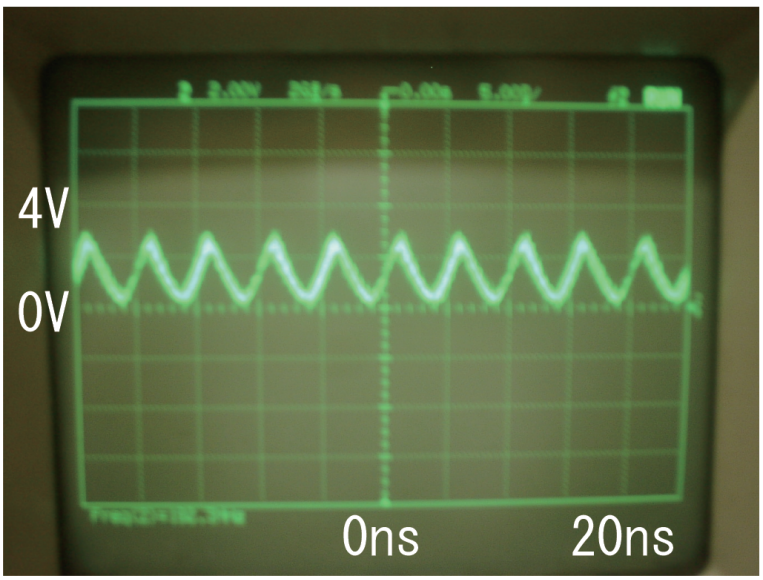

図 4.25: 出力波形 (200MHz)

第5章　非同期回路の同期回路に対する応用に関する既存の研究

1990 年頃全ての回路を非同期回路と考えた非同期研究者だったが，90 年代の同期式回路設計技術の急速な発展により，超省消費電力プロセッサなど非常に限定した場合を除いた場合，非同期式回路は同期式回路と比較し優れた特性を示さないことが分かった．

そのため，非同期式研究者の関心は回路全てを非同期式で設計するのではなく，特に優れた特性を示す回路のみ非同期式で設計する手法，非同期式の手法を同期式に導入することにより優れた特性を持つ回路を設計する手法などの研究に向けられるようになった．

このような流れの中で，本章では非同期回路の同期回路に対する応用に関する既存の研究について述べる．

まず，同期式と非同期式を組み合わせることによる高速化を目指す手法の代表例として，GALS(Globally Asynchronous Locally Synchronous) 及び，Intel の RAPPID について述べる．次に同期 Speculative Completion のベースとなる手法 Speculative Completion を紹介する．さらに同期 Speculative Completion による回路は回路全体を単一のタイミング線で動作させている束データ方式の非同期回路であると言える．本論文では CPU に同期 Speculative Completion を適応させ評価するため，同じ CPU の中で束データ方式を用いた非同期で動作するものの代表例と，そこで用いられた高速化手法マイクロパイプラインを紹介する．

5.1　非同期式を導入した同期式回路

5.1.1　Globally Asynchronous Locally Synchonous

GALS(Globally Asynchronous Locally Synchronous) は一つのチップ内を複数の機能ブロックに分け，各機能ブロックはそれに付属するリングオシレータにより同期式で動作させ，機能ブロック間の通信は非同期式で行う回路設計法である．この方式では機能ブロックは同期式で動作することにより，容易な実装，高速な動作を実現し，機能ブロック間の通信は非同期式で行うことにより，通信回路の配線遅延のため全体の処理速度が低下するのを防ぐ．

5.1.1.1　Asynchronous-Synchronous Interface

1984 年に Chapiro が発表した論文が GALS の始まりである [Cha84]．Chapiro が発表した論文では各機能ブロックがリングオシレータからの駆動信号により同期式で動作し，機能ブロック間の通信は非同期式で動作する．しかし，非同期回路と同期回路を接続する

記憶素子がメタステーブルの状態の時に，機能ブロックがデータを読み込む事を防ぐため，各機能ブロックがデータをやり取りする際にはリングオシレータからの駆動信号を止め，機能ブロックが処理を実行しない方式を採用していた．このため実行効率が非常に悪かった．

1986 年に Yun と Donohue によって発表された方式 [YD96] はリングオシレータに排他的制御回路を加え，入力したデータがメタステーブル状態になっている可能性がある場合はリングオシレータからの信号を遅らせることによりメタステーブル状態の信号が機能ブロックに読み込まれることを防ぐ．この方式では機能ブロック間でデータをやり取りする際に機能ブロックを止める必要はないため，効率の良い実行が可能となる．この方式を元に改良を加えた研究が盛んに行われた [MVF00][SM00][MTMR02].

ここでは代表的な例として Mutterbach が考案した GALS 方式による回路のブロック図を示す (図 5.1)．GALS モジュール内に処理回路である Locally Sychronous Island，入出力の処理を行う Port が入力用，出力用が 1 つずつ，及び，Locally Synchronous Island への駆動信号を出力する Local Clock Generator が存在する．Port は Req 信号及び Ack 信号を用い外部の非同期回路と通信し Locally Synchronous Island へのデータの受信通知及びデータの転送終了通知を行うとともに，Local Clock Generator の制御を行う．Local Clock Genrator は外部との通信が行われないサイクルでは一定の周期のクロック信号を送るが，外部との通信が行われ，外部との通信の結果，データがメタステーブル状態になる可能性がある場合はクロック信号の周期を一時的に長くするなどの処理を行う．

Mutterbach の非同期同期インターフェイスはメタステーブルを回避するため調停回路を用いる点が特徴である (図 5.2)．この回路は図 5.1 の Port 及び Local Clock Genrator に相当する．図中左の回路では図中の非同期で送信されてくる Data を FIFO に入力している．Data はバッファに入力された後 FIFO へ入力される．Data がメタステーブル状態を起こさないタイミングで送信されて来る場合は，図中左下の Delay Line，C-element，及び，インバータにより作り出される一定の周期のクロック信号によりバッファが駆動され読み込まれる．Data がメタステーブル状態を起こすタイミングで送られて来た場合 Arbiter により信号の流れが変えられ，Delay Line，C-element，及び，インバータによるクロック信号の生成が 1 サイクル止まり，バッファが駆動されないためメタステーブル状態を回避できる．図中右の回路では FIFO からの出力を接続されている Locally Sychronous Island に送信している．FIFO と Locally Sychronous Island では動作周波数が異なるため，メタステーブル状態が発生するのを防ぐための回路が必要になる．Locally Sychronous Island は図中右の Delay Line，C-element，及び，インバータにより生成されたクロック信号により動作している．FIFO からデータが出力されると Arbiter により Delay Line，C-element，及び，インバータのループが遮断され FIFO に接続されているバッファに駆動信号が伝えられる．そして，アービタが元の状態に戻り，Delay Line，C-element，及び，インバータのループから Locally Sychronous Island のクロック信号が再び出力される．このクロック信号に駆動され Locally Sychronous Island に接続されているバッファへデータが入力され，メタステーブル状態を起こさずに Locally Sychronous Island へデータが転送される．

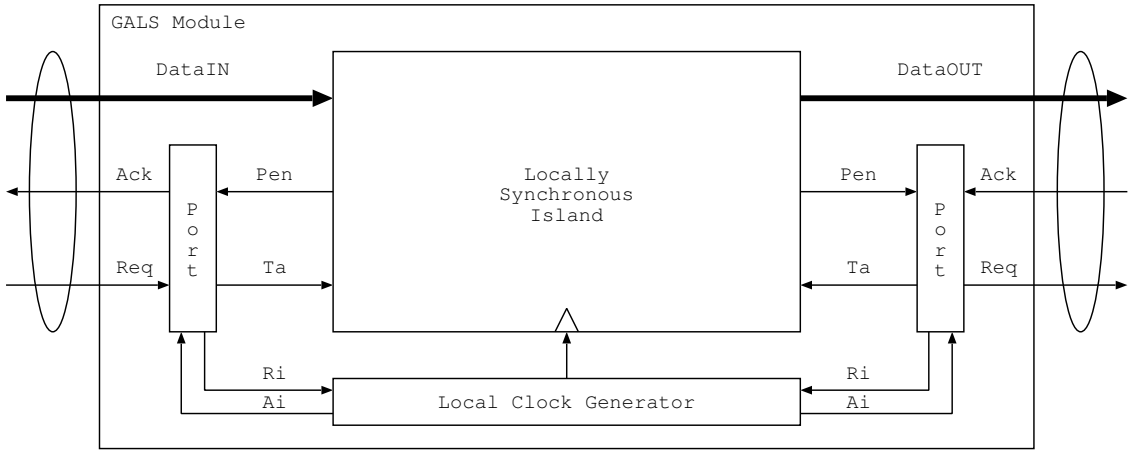

図 5.1: GALS Module

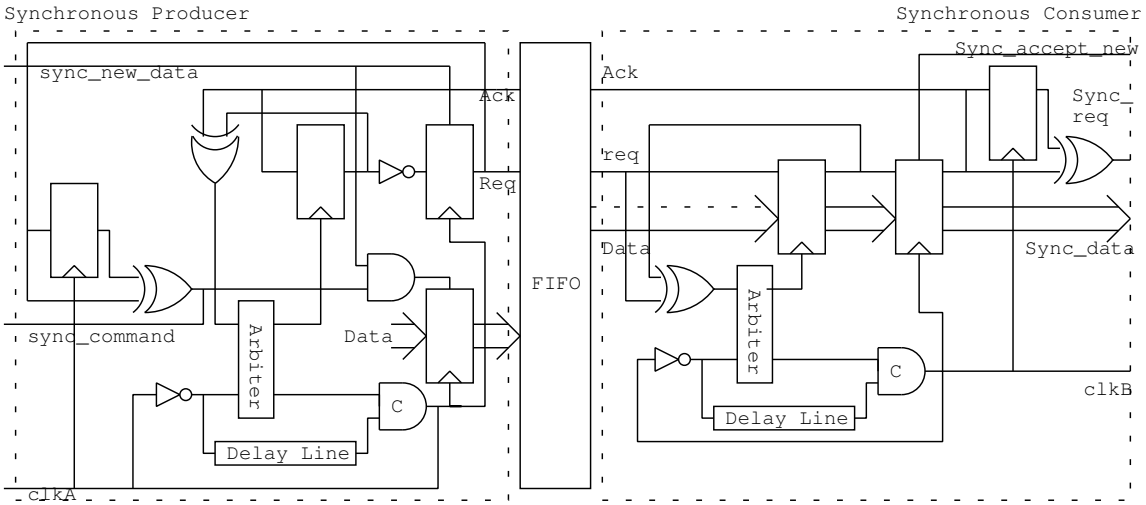

図 5.2: Mutterbach Interface

5.1.1.2　Clock generator

GALS を設計に取り入れるには同期式で動作するモジュールにクロック信号を与える必要がある．一般に各モジュールは異なった周波数で動作するが，各モジュールに適したクロック発生源を設計するためには多大な労力がかかる．そのため与えられた条件に合った周期のクロックを発生できる回路が必要になる．チップ上でクロックを発生させる手法として，[ONM+00] のようにリングオシレータによりクロック信号を発生させる方式が提案されている．

[ONM+00] の論文では図 5.3 のような回路を使い，元となる低周波数の信号から高周波のクロック信号を作り出している．

Delay Control Counter では発生させるクロック信号の周波数に応じて Digitally Controlled Oscillator へ信号を送る．Digitally Controlled Oscillator ではリングオシレータに

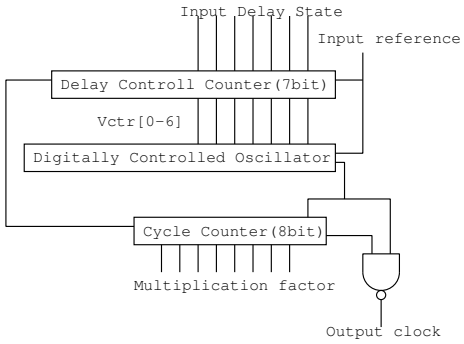

図 5.3: Clock Generator Block Diagram

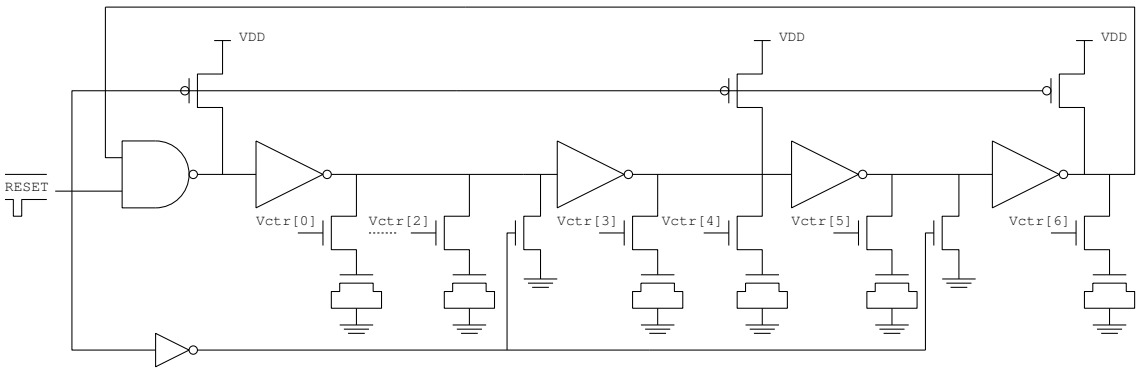

図 5.4: Olsson Oscillator

よりクロック信号を発生している．Cycle Counter では Digitally Controlled Oscillator の結果をカウントアップしていき，実際にクロック信号として出力する信号を選ぶ．

中心となる回路である Digitally Controlled Oscillator の構成を図 5.4 に示す．図中のコンデンサはそれぞれ別の容量を持っており，Delay Control Counter からの制御信号により有効となるコンデンサが選択される．この機構により安定性を確保しつつ様々な周波数のクロック信号を作り出す．この回路により 0.35μm プロセス，電源電圧 3V の環境でシミュレーションでは 556MHz から 864MHz，実機では 790MHz から 1.08GHz のクロック信号を発生させることに成功した．この回路は Delay Control Counter からの制御信号に対し非常に良いリニアリティを示す．この回路の出力を Cycle Counter に入力し，必要な信号のみを出力することにより，より周波数の低いクロック信号の出力が可能である．

この研究により GALS を実現するために必要な様々な周波数のクロック信号を安定して供給できる回路が実現可能であることが示された．この回路は元となる信号が低い周波数の同期信号であるため，消費電力の観点からも優れている．

5.1.1.3　CPU に適用させた場合の評価

GALS により設計した回路の性能評価としては CPU に適用した例が挙げられる [IM02][SAM+04].

[IM02] では Alpha21264 を想定し，同期式及び GALS を適用したものをシミュレーションによって評価している．GALS による設計では

- 分岐予想とフェッチ
- デコードとリネーミング
- 整数演算キューと整数演算ユニット
- メモリアクセス命令キューとデータメモリキャッシュ
- 浮動小数点演算キューと浮動小数点演算ユニット

の各機能ブロックにわけ，それぞれを非同期回路でつないでいる．

ベンチマークの結果，GALS の CPU は同期式 CPU と比較し 10%性能が低下することが分かった．また，時間当りの消費電力は 10%程度低下するが，同期式と比較し，処理時間が長いため，同じ処理を実行する場合約 1%電力を多く消費することが分かった．

[SAM+04] に於いても同じ Alpha21264 の各機能モジュールを次のように分離し，それぞれ異なった周波数のクロック信号で動作させ (Multiple Clock Domain)，非同期回路により接続している．

- フェッチとディスパッチ
- 整数演算ユニット
- 浮動小数点演算ユニット
- メモリアクセスユニット

この研究では異なった周波数のクロックシグナルで動作するモジュール同士を FIFO でつなぐ際，2 種類の FIFO を使用する．通常の 1 サイクル毎に同期を取り FIFO の状態を更新するタイプ及び，FIFO が Empty 状態になった場合または Full 状態になった場合を除き同期を取らずにデータを転送するタイプの FIFO を使い分けることにより高速化を図っている．この結果，SimpleScalar によるシミュレーションにより，同期式の Alpha21264 と比較し動作速度は 2%程度低下するに留まることが示された．

これらの研究により，GALS の考え方に基づく回路はシングルコアの CPU のように各機能モジュールが緊密に結び付いている場合においても低い性能低下率で実現できることが示された．機能モジュールが緊密に結び付いている場合，GALS による回路設計を導入することによる性能低下が顕著になる．機能モジュールが密接に結び付いている CPU において低い性能低下率で GALS の考え方に基づいた回路を実現できることが示されたことは GALS の適用可能範囲が広いことを示している．

5.1.1.4　AES に適用した場合の評価

また，変わった応用例としては AES 暗号化チップに適用した場合の評価がある．この場合，GALS 技術を適用した目的は長い配線によりクリティカルパスが長くなるのを防ぐ

ためではなく，電力消費パターン，電磁波照射，実行時間，表面温度等 (Side Channel) から暗号が解読されるのを防ぐためである．

一般に暗号キーのある bit が 0 であるか 1 であるかにより暗号化，復号化の処理は変化する．そのため，Side Channel の変化を精密に測定することにより暗号キーを解読することが可能になる．同期式の場合，演算処理を行い始めるタイミングは常にクロック信号に同期し，一定であるため，Side Channel 情報を集めるのはたやすい．また，動作速度が外部のクロックジェネレータに依存するため，動作速度を落とし，更に情報を集めやすくすることも可能である．

これらの問題はチップを非同期化することにより解決できるが，チップ全体を非同期化するには周辺ツールが今だに成熟していない，動作速度が遅くなるといった欠点がある．

そのため，GALS を適用することで成熟した同期式回路設計ツールを使うことにより効率良く高速な回路を設計することを可能にし，各回路が外部からのタイミング信号によらずにそれぞれのタイミングで動作するといった特徴を活かし Side Channel 攻撃に強い回路を設計可能にする．

この研究で GALS を適用して設計された AES 暗号化チップ Acacia では，AES 暗号化処理の中でも Key Generation，AddRoundkey 及び ShiftRows(David) と SubBytes 及び MixColumns(Goliath) に分け，それぞれの回路を非同期回路で接続した．また，比較のため同様に機能ブロックを分けて設計した同期式の暗号化チップが作成された．

GALS を適用したため，クリティカルパスは David は 5.43ns から 3.98ns，Goliath が 5.84ns から 5.27ns と短くなっているが David と Goliath の通信部でレイテンシが 1 クロック多く必要とするため，比較用同期式チップでは 40.88ns で処理が終了するのに対し，Acacia では 42.38ns 必要とする．

この研究では同期式の回路の特徴と非同期式の特徴を GALS により組み合わせることが可能であることが示された．このような GALS の応用例はこれからも増えていくことが予想される．

5.1.1.5　EDA を用いた設計

GALS による回路を設計する際，大部分の回路をレイアウトエディタで記述する必要がある，既存の EDA ソフトは全く使用できずオリジナルツールを使用しなければならないといった状況では設計者の負担が大きく，一般的な設計方法として使われることは難しくなる．そのため，[OGV+03] の研究など既存の EDA を最大限に利用した設計手法の研究が行われている．

[OGV+03] の研究で設計された回路は 25 のテスト用同期回路を非同期回路で結ぶ構成となっている．設計に際しては最大限既存の EDA を用いており，self-timed library 作成を除くと，SynopsysDC，Silicon Ensemble，Calibre DRC/LVS 等既存の同期式向け EDA ツールが使用された．この研究ではシミュレーションによる評価に留まっているが，今後，実機を作り評価することが予定されている．

この研究は GALS 特有の基本的な素子を除けば，既存の EDA ツールによる設計が可能であることを示した．市販の開発ツールが存在しないことは非同期式普及の大きな障害となっており，市販の同期式開発ツールが使用可能であることは非同期式による回路開発の

普及を促す.

5.1.1.6　Synchro-Tokens System Architecture

GALS の一種として Synchro-Tokens System Architecture[HH03] があげられる．この方式も同様に同期式のモジュールを非同期回路でつなぐことにより性能の低下を防ぐがメタステーブル状態を回避するためにトークンリングを使用する点が特徴である.

Synchro-Tokens System Architecture を採用した回路は図 5.5 に示す構造をしている．このうち，非同期式のデータ転送回路と同期式の演算回路を図 5.6 に示す．データ転送回路は多段 FIFO となっておりパフォーマンスの向上を図っている．Node は Request 信号と Acknowledge 信号を元にデータ転送回路のインターフェイス部分にデータの有効信号を出力している．コミュニケーションを行う Synchronous Block 同士はトークンをやり取りしており，通常のトークン方式による通信と同様に，トークンを持った Synchronous Block のみが信号を送信できる．トークンのやり取りはデータ通信よりも時間を必要とするように設計されている．この機構により，データがメタステーブル状態で受信しないようにし，また，有効ではないデータを受信することを防いでいる.

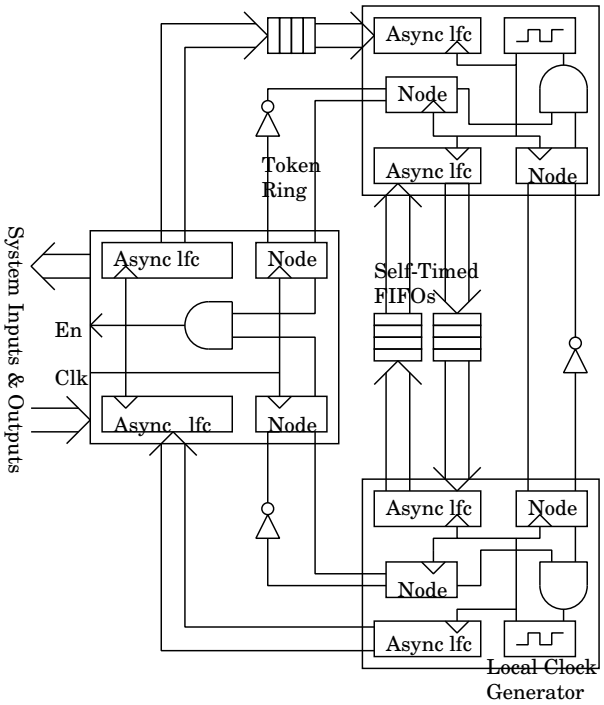

図 5.5: Synchro-Tokens System Architecture circuit

このアーキテクチャは Out-of-Order の簡単な CPU の Verilog シミュレーションによって評価が行われ，動作が確認された.

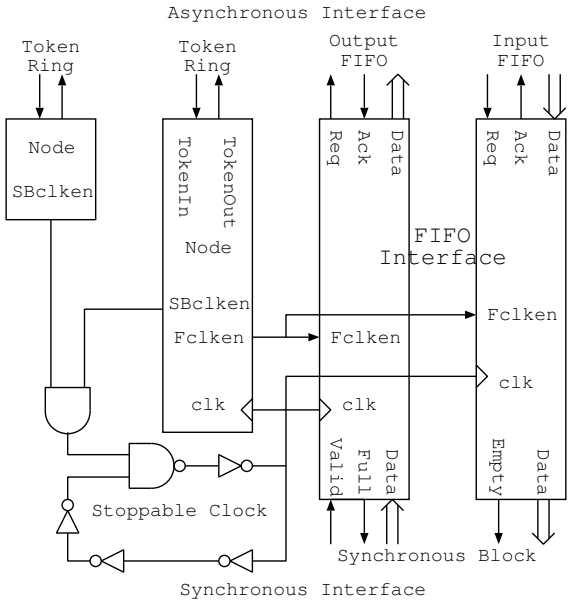

図 5.6: Synchro-Tokens System Architecture interface

5.1.1.7　まとめ

GALS は今後のプロセス技術の発展に伴う配線遅延の増大に対する有効な対応策の一つであり，非同期式回路の設計手法の同期式回路への導入として最も注目されている技術である．

様々な研究の結果，現在シミュレーションによる評価の段階から実機を作成する段階に移行しており，様々な分野での応用が期待されている．

5.1.2 RAPPID

RAPPID(Resolving Asynchronous Pentium Processor Instruction Decoder) は INTEL 社の Pentium-II プロセッサ向けのデコーダを非同期化したものである [RSG+99]．Pentium-II など CISC プロセッサでは命令長が一定ではないため，デコード回路は様々な命令長に対応できなければならない．同期式ではクリティカルパスになる処理に合わせて処理時間が決まるため，デコード回路は命令長の長い命令をデコードする際の処理時間によってデコード処理に必要な時間が決まる．このため，実行効率は低く，命令の大部分を占める短い命令長のものをデコードする際には処理時間に大きな無駄が生じている．この問題を解決するため，非同期式の回路を導入し，命令長が短い場合は短い処理時間でデコードを終わらせ，命令長が長い場合は時間をかけてデコードを行うことを可能にした．その結果，0.25μm プロセスを用い，1ns あたり 2.5-4.5 命令をデコード可能であることが分かった．同じ面積，同じプロセスの 400MHz 動作の同期式回路と比較し，3 倍の性能，半分のレイテンシ，半分の消費電力を達成した．

RAPPID は合計 16Bytes の命令を並列に処理することが可能であり，最も効率良く処理が行われた場合，4 命令を並列にデコードすることができる．IA-32 のプログラムに含まれている命令の長さは 4Bytes 以下が全体の約 8 割を占めているため，最も効率的な実行が可能である場合が多い．

また，命令長が 11Bytes 以上の命令の場合は異なったプロトコルでデコードを行う設計とし，命令長が短い場合の処理速度に影響を与えない構造を取るなど，最も多いケースで実行時間が短くなるように工夫されている．

RAPPID プロジェクトは非同期式回路は適した処理を実行する場合，同期式回路と比較し高性能になることを実証した．

5.2 Speculative Completion

Speculative Completion [Now96] は本論文の研究において採用されている束データ方式を採用した非同期式回路の性能を向上させる技術である．

通常の束データ方式においては処理のための素子遅延が小さい Request 信号と処理のための素子遅延が大きいデータ信号のタイミングのずれを防ぐために Request 信号線上に遅延線を置く (図 5.7)．この遅延線はデータ信号が回路で処理されている時間以上の遅延を作る長さが必要になる．このため，代表的な非同期設計法である 2 線方式で設計された回路はそのサイクルで必要な処理時間で実行可能であるのに対し，一般的な束データ方式で設計した場合，常に最も時間のかかる処理に合わせた処理時間が必要となる．

この問題を解決するため，Speculative Completion では 1 つの回路に複数の遅延線を使用する (図 5.8)．そして，行われる処理，扱われるデータなどから遅延判定回路で必要な遅延を判定し，最適な実行時間となる遅延線を選ぶ．この手法により通常の束データ方式と比較し，効率の良い実行を可能にする．

Nowick らは加算器に適用することにより，値をランダムに与えた場合，64bit 幅の値を与えた場合 29%，32bit 幅の値を与えた場合 19%高速になることを示した．又，実際のプログラムで実行される加算を抽出し，効果を評価した．その結果，分岐のための加算では 5%から 12%，アドレス計算のための加算では 2%から 16%，加算命令で扱われた加算では

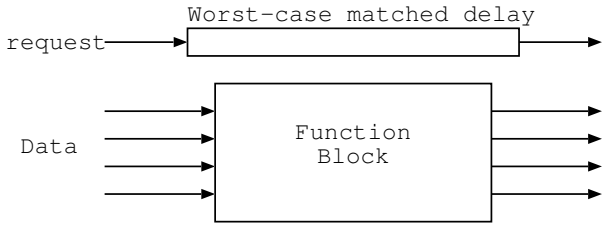

図 5.7: standard bundled datapath

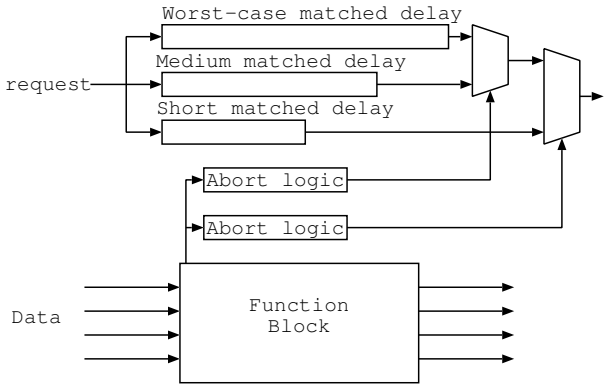

図 5.8: speculative completion

1%から 3%の性能向上が確認された.

Speculative Completion は束データ方式を利用した非同期回路において有用な高速化手法である.

5.3　束データ方式を用いた CPU

本論文で次章で述べる同期 Speculative Completion は 5.2 節で紹介した非同期式向けの高速化技術 Speculative Completion を同期式の回路に適用できる技術に変更したものである．本論文では CPU に適応する場合の評価を行ったため，束データ方式の非同期式 CPU を調査する必要がある．この節では束データ方式を利用した非同期式 CPU の代表例として AMULET1，AMULET2e について述べ，それらで使われている高速化技術マイクロパイプラインを紹介する.

5.3.1　AMULET シリーズ

5.3.1.1　AMULET1

AMULET1 [FDG+94b] [FDG+94a] [Fur95] [GFC99] は ARM6 互換非同期プロセッサである．非同期の省消費電力性に着目した Manchester 大学の研究グループが，完全非同期 CPU を作り，商用に耐えることを示す事を最終目標に製作した．1990 年の終わりから

設計を始め 1993 年に 5 年人の労力で完成した．

AMULET1 は制御方式として 2 相式の Sutherland のマイクロパイプラインを採用している．大まかな構造は図 5.9 のようになっている．命令のレジスタ依存関係を解決するためにフォワーディング等は使用せず，レジスタ依存命令の実行時にはレジスタに書き込まれるまで待つように設計されている．実行ユニットは同期式に比べシンプルで高速なものを作成することに成功している．分岐命令は現在の PC の値から ALU で次の命令アドレスを計算し，古い PC と交換する事により実行される．分岐先が命令読み込み処理を行う回路に非同期的に送られるため，読み込みの初期化命令に時間を必要とする．このため，同期式と比べスピード的に優位にはならない．ロード命令は 3 サイクル，ストア命令は 2 サイクル実行にかかる．この時間を有効に使うため，より複雑で高性能なアドレッシングモードをサポートしている．また，命令はバッファから供給されるためメモリアクセス命令は命令の読み込みには影響を与えない．また，ARM6 には複数のレジスタに対するメモリロード，ストア命令が存在する．これらの命令が実行される際には PC を増加させるハードウエアをアドレスの計算に利用することで，ハードウエアの有効利用を図る．

同期式 ARM6 との比較は表 5.1 のようになった．非同期回路は 2 倍の面積，トランジスタが必要になることがわかった．非同期式の省消費電力性に着目して始まった研究だが，性能対消費電力比においても AMULET1 は ARM6 より劣る．だが，AMULET1 は非同期設計が実現可能である事を確認するために製作された回路であり，性能改善の余地はある．また，多くの CPU ではピーク時の消費電力ではなく Idle 状態の消費電力の方が重要である．AMULET1 では待機電力をほぼ 0 に抑えることが可能であり，同期式と比較し有利である事が実証された．

AMULET1 では比較的深いパイプラインを採用しているが，デコーダがパイプラインの busy 状態を維持する数の命令を発効できていないため効率が悪くなっている．マイクロパイプラインを用いた回路は簡単にステージ数を増やすことができるが，ステージを無計画に増加させることによるパフォーマンスの改善はないことがわかった．また，コンパイルされたコードに多くのレジスタの依存関係があるためパイプラインが頻繁にストールする．この原因としてレジスタフォワーディングの使用が困難である為，行われていない点が挙げられる．

この研究を通じて判明した改良点として以下のようなものがあげられる．

(a) ARM 命令は複雑なためデコードステージが遅くなりボトルネックとなっている．この点を改良するためデコードステージの制御を 2 相式ではなく 4 相式で行う．

(b) AMULET1 ではフォワーディングを行っていない．ALU での実行結果を保存することにより，一つ前の命令の実行結果のフォワーディングを可能とする．

(c) locking mechanism の回路を変更し，レジスタへ書き込んでいる値をバイパス可能にする

(d) レジスタバンクの周辺回路として Write Enable Latch を付け加えることにより，レジスタの中身が書き込まれた後すぐに lock FIFO をクリア可能にする．また，write word line を無効化すると同時にレジスタを読み込むことを可能にする．

(e) パイプラインのステージを整理し，深いパイプライン構成をやめる．

表 5.1: AMULET1 vs ARM6

	AMULET1	ARM6
Process	$1\mu m$ DLX CMOS	$1\mu m$ DLM CMOS
Cell Area	5.5mm x 4.1mm	4.1mm x 2.7mm
Transistors	58374	33494
Performance	9K Dhrystones	14K Dhrystones @ 10MHz
Power	83mW	75mW @ 10MHz

(f) パイプラインを制御するためのラッチを改良する.

(g) MMU 及びキャッシュメモリを完全非同期化し搭載する.

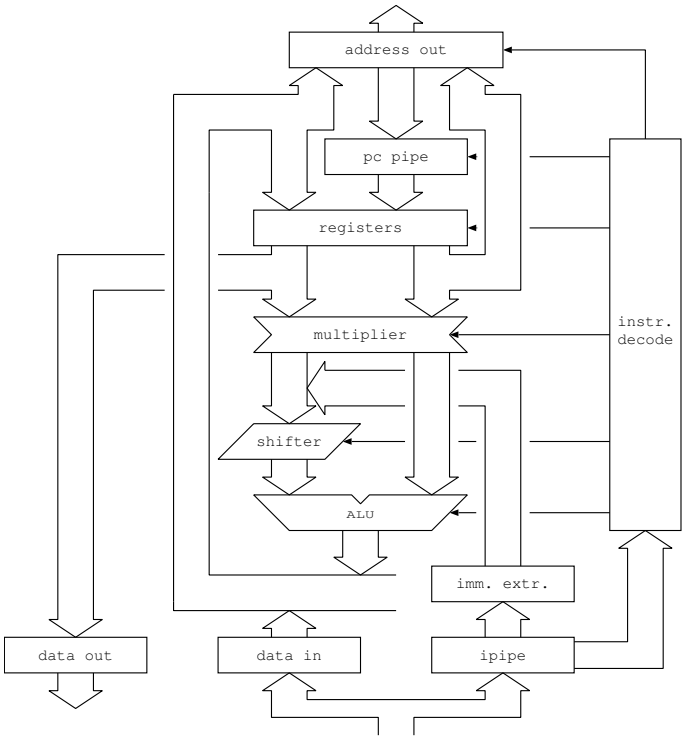

図 5.9: AMULET1

5.3.1.2　AMULET2e

AMULET1 の経験を活かし作られたのが AMULET2 及び AMULET2e　[FDG+96] [FGT+97]　[WDF+97]　[FGR+99] である．AMULET2 は AMULET1 と同様 ARM6 互換プロセッサである．1996 年 10 月にファーストシリコンが完成した．AMULET2e は CPU コア AMULET2 に 4KByte のキャッシュRAM，メモリインターフェイスを搭載して

いる．AMULET1 から数々の改良がなされており，代表的なものとしては以下のようなものがあげられる．

(a) 図 5.10 のように，無駄にステージ数を増やす事をやめ，データの流れを整理する．

(b) 図 5.11 のように 1 つ前の命令の結果を LRR(last result register) に保持し，次に実行される命令が 1 つ前の命令に依存している場合，保持した結果を読み込むことによりストールを回避する．また，最後にロードしたメモリの内容を LLV(last loaded value) に保持し，ストールを回避する．フォワーディングは図 5.12 のような回路で行われる．

(c) 無駄な命令読み込みを回避し，パフォーマンスと消費電力を改良するため，ジャンプ先をキャッシュする BTC（Branch target cache）を搭載する．

(d) stall 時にほぼ全ての回路が停止する設計にする．

キャッシュRAM は 4KBytes のものが 1KBytes ごとに管理され，64-way セットアソシエイティブで制御される．メモリインターフェイスではクロックを使用せずに現行の同期式向け RAM にアクセスするためチップ外に reference delay 回路と呼ばれる遅延を発生させる回路を置いている．

実機が製作されテストボードでテストされた．ほぼ全ての機能が設計通りに動作したが，abort 処理中に BTC(Branch Target Cache) をアクセスした場合，デッドロックが発生することがわかった．実機の評価の結果，3.3V の動作電圧時に 42MIPS の性能があることがわかった．また，ピーク性能動作時の消費電力は，150mW（I/O を除く）であることがわかった．同期式の ARM710 は 120mW 消費時に 23MIPS，ARM810 は 500mW 消費時に 86MIPS の性能である．AMULET2 は両者の中間の性能であるが，消費電力あたりの性能は最も高い結果となった．改良点に関する評価をを表 5.2 に示す．

電圧の変化に対する耐性もテストされ，ボード全体は 2.5V〜4V の間でのみ動作したが，AMULET2e は 2.0V で動作し，CPU コアである AMULET2 は 1.1V での動作が確認された．アーキテクチャの改良による性能向上は表 5.2 のようになった．フォワーディングによる性能向上があまり見られないのは実行ユニットの処理に比べ，他の処理が重く，ボトルネックになっているためである．また，BTC の効果も小さいが，これは大きいループには BTC がバッファするジャンプ先の数と比較し多くのジャンプ命令が存在するため BTC のエントリが不足するためである．CPU 停止時の省消費電力動作機構は設計意図通り働き，消費電力を 0.1mW に抑える．また，EMC 特性も非常に良いことが確認された．

AMULET2e は非同期 CPU の利点である，電力効率の良さ，EMC 特性の良さを示す一例である．だが，純粋な非同期設計では処理性能は高くならないことも同時に示している．

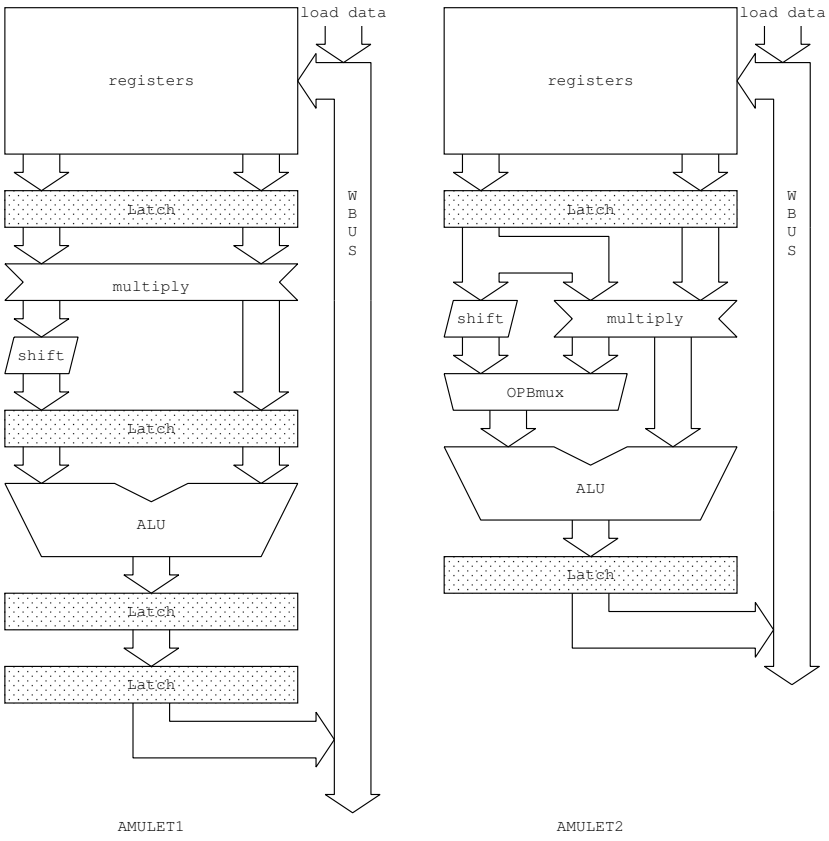

図 5.10: AMULET1 vs AMULET2

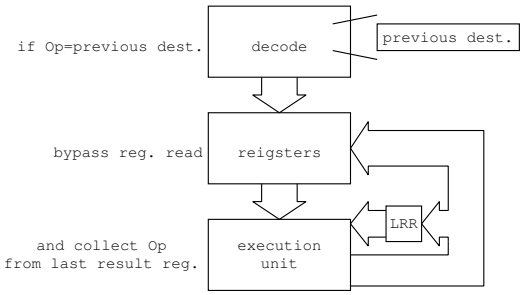

図 5.11: LRR(AMULET2)

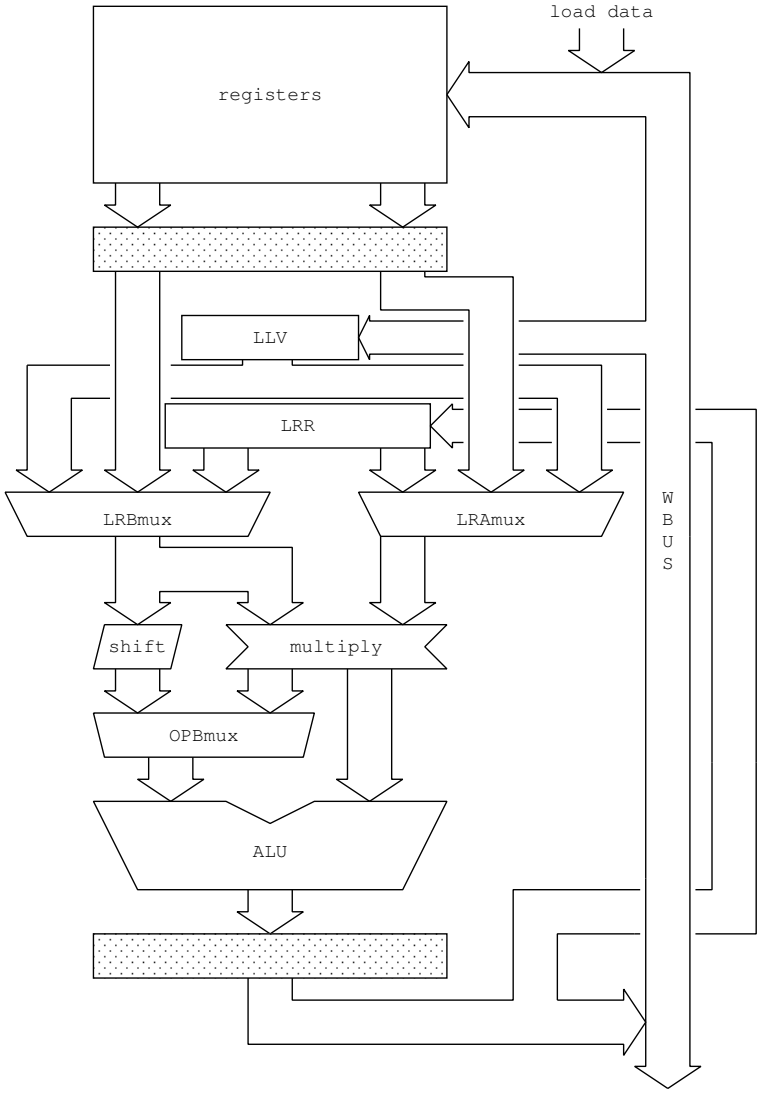

図 5.12: Forwarding(AMULET2)

Memory	Internal	Cache	External
Idle Power	76mW	162mW	66mW
Idle Power 'Halt' on	0.1mW	0.1mW	0.1mW
BTC MIPS	+6%	+3%	+7%
BTC power-efficiency	-5%	-3%	-4.5%
Fwd MIPS	+2%	+0.5%	0%
Fwd power-efficiency	0%	0%	0%
All features MIPS	+8%	+4%	+7%

表 5.2: 改良点の評価

5.3.2　関連技術：マイクロパイプライン

マイクロパイプライン [Sut89] は束データ方式を用いた非同期式回路の制御法の一つである．一般的な名前のため，考案者の名を冠して，Sutherland のマイクロパイプラインと書かれることが多い．この方式を用いることにより，柔軟かつ高速な非同期回路を製作することができるため広く使われている．

マイクロパイプラインの制御回路は図 5.13 のような回路になる．陰となっている部分がパイプラインの 1 ステージを表している．C-element には要求信号 R 及び応答信号 A が入力されており，前のステージからの要求信号と次のステージからの応答信号が（High, Low）もしくは（Low，High）の組み合わせの時に動作信号を出す．信号がこの組み合わせになることは前のステージの処理が終わり，次のステージが処理を行っていないことを表す．REG 素子は CP-latch と同じ構造をしている．C（Capture）信号及び P （Pass）信号によって制御されており，双方の値が同じであれば入力データをそのまま出力する．Cd 及び Pd はそれぞれ C 信号，P 信号を増幅し，遅延をつけ出力する．

処理は次のように進む．ここでの説明を付けた部分の回路に対するものである．初期状態として C-element への Request 信号と Acknowledge 信号は Low であるとする．

(a) C-element への Request 信号入力が High となる．

(b) C-element への Acknowledge 信号入力は Low であるので C-element は High を出力する（C-element への Acknowledge 信号入力はインバータによって反転される）．

(c) REG 素子の C 入力が High になり，REG 素子はデータの流れを遮断し，現在の値を保つようになる．

(d) REG 素子の Cd 出力が High となる．

(e) LOGIC が実行される．

(f) LOGIC 実行に必要なだけ遅延をされ，次のステージの C-element の Request 入力に High が入力する．

(g) 次のステージの REG 素子の Cd 出力が High となる．

(h) REG 素子の P 入力が High となり，REG 素子が入力したデータを透過するようになる．

(i) REG 素子の Pd 出力が High となる．

(j) C-element の Acknowledge 入力が Low となる．

(k) 前のステージの処理が終わり，C-element への Request 信号入力が Low となる（反転する）．

(l) REG 素子の C 入力が Low となり，REG 素子はデータの流れを遮断し，現在の値を保つようになる．

このマイクロパイプラインを用いることによりパイプラインを簡単に細分化し，高速化することができる．マイクロパイプラインではパイプラインを細かく区切るため，実行結果のフィードバックを行って演算を行う CPU のような回路の場合，空のステージが多く存在することになり効率が悪くなる場合がある．制御信号を処理するための回路の遅延が問題となるため，現在でも処理のオーバーヘッドを軽減するための研究が行われている．

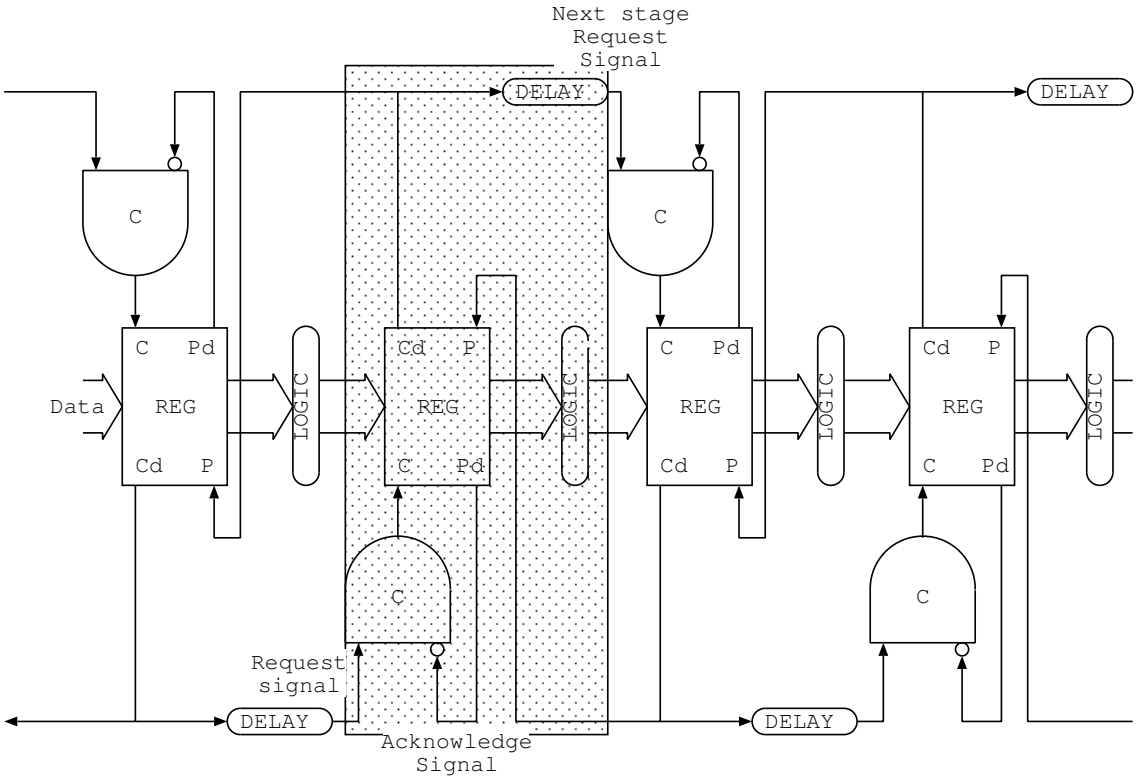

図 5.13: マイクロパイプライン

5.4　まとめ

非同期式の同期式への導入例として GALS と RAPPID，そして，束データ方式を利用した CPU の例として AMULET シリーズとそこで使われた高速化技術を取り上げた．これらから明らかなことは次の通りである．

- 非同期処理に適した処理の場合は同期式と比べ，処理能力が向上し，消費電力が下がる．
- 非同期処理に適さない処理の場合，処理能力，消費電力とも同程度か劣る．
- 同期式の回路と非同期式の回路を組み合わせることにより性能を向上できる可能性がある．

以上を踏まえ，本論文では非同期式の考え方を同期式に適用させて高速な処理を目指す同期 Speculative Conmpletion を提唱する．同期 Speculative Completion は非同期式の各サイクルで処理に必要な時間のみで実行可能である特長を同期式に応用した技術である．この技術により，非同期式の利点と同期式の利点を兼ね備えた回路を実現する．

第6章　同期Speculative Completion

6.1　導入

2線方式の非同期式回路は実際の処理に必要な時間のみで実行を完了可能な特徴がある．また，先に述べた通り本研究で用いられる束データ方式の非同期回路においてもSpeculative Completionを導入することにより必要な実行時間に近い時間で1サイクルの実行を完了できる．これらの特徴は非同期式に特有であり，クロック信号に合わせて処理を行う同期式には無い利点である．

同期式では通常，外部のクロックジェネレータからの信号，もしくはそれをベースにPLL等で作り出された信号に合わせて動作する為，各サイクルの実行時間は最も実行時間を必要とする処理に合わせる必要がある．各サイクルの処理の実行時間に合わせ，クロック信号の周期を変化させることによりより高速な処理が期待できる．これを同期Speculative Completionと呼ぶ．

Speculative Completionでは，パイプラインの各ステージにそれぞれのステージにおいて実行される処理時間に合わせた周期の動作信号を作り出す複数の遅延線と処理回路が用意される．それに対して，同期Speculative Completionでは，全ステージの中で最も実行に時間を必要とするステージに合わせた周期の動作信号をリングオシレータによって作り出し，同じ動作信号を全ステージで使用する．

同期Speculative Completionは全ステージの処理内容情報を一回路に集め，最も実行時間が長いものを判断し，それを元に動作信号を作り出し，全ステージに送る必要がある．このため，高速な回路には向かない．だが，動作時間5ns-10ns程度の中速の回路において，稀に極端に実行時間の長い処理を1サイクルで実行する必要がある場合，高い効果が期待できる．

本章では提案手法の有効性を示すため，この方式を用いたCPUコアを設計し，その動作速度を，ほぼ同機能の非同期CPUコア，Speculative Completionを用いた非同期CPUコア及び，同期CPUコアと比較，評価を行う．

6.2　同期Speculative Completion

6.2.1　同期Speculative Completion

本節では，Speculative Completionを同期式回路に応用する手法である同期Speculative Completionを提案する．ここでは，説明の都合上，パイプライン構造を前提とするが，基本的にはどのような回路にも適用可能である．

通常の同期式ではすべての回路のクロックは一定の周期で送られる．それに対して，同

期 Speculative Completion では，パイプラインの最も時間がかかる処理を行うステージの実行時間を全体で判断し，これに合わせて全ステージが共通の遅延を決定する．この方式を用いることで，それぞれの状況に応じた遅延時間で処理することができる．共通の遅延の決定は以下の手順により行われる．

(a) サイクルの開始を示すクロックパルス (Start signal パルス) が入力されると，パイプラインの各ステージで，入力データと内部状態より最も時間のかかる処理を検出し，それに応じた遅延を決定する．この部分は Speculative Completion と同様である．

(b) 決定された遅延の長さを示すデータを，それぞれのステージから判定回路に送る．

(c) 送られて来た遅延の中で，もっとも長いものを判定回路で検出し，対応する遅延線を選択して，クロックパルスを遅延させ，パイプラインの全ステージに送る．このクロックパルスにより次のサイクルが始まる．

この処理を繰り返すことにより，それぞれのサイクルに適合した遅延のクロックにより処理を行う．

最も時間のかかる処理を検出する方法はそのステージによって実行される処理によって異なる。CPU の実行ステージのように複数の処理を実行可能だが，選択された処理のみ実行される場合，選択された処理に応じて遅延を決定することが考えられる．また，そのステージの入力データによって処理に必要な時間が異なる場合，入力データに応じて遅延を決定することが考えられる．

6.2.2　Delay Line の実装方法

Delay Line の実装法としては，インバータチェーンを用いる方法とカウンタを用いる方法がある．インバータチェーンを用いる方法は，多数のインバータに連続して接続することで遅延を実現する方法であり，カウンタを用いる方法は，クロックジェネレータからの高周波数のクロック信号をカウントアップし，必要な遅延を生成する方法である．

カウンタを用いる方法は，インバータチェーンを用いる方法に比べて，多数のインバータを必要とせず，パルス幅も安定している．しかし，インバータチェーンを使用する方法では，0.2ns 単位で遅延を調整できるのに対して，カウンタを用いる方法は，遅延を細かく制御するためには元のクロック信号を高周波数にする必要がある．例えば 1nsec 刻みに遅延を制御するためには，1GHz のクロックジェネレータが必要になる．

本論文でテスト用に作成する CPU 程度の動作速度であればインバータチェーンを用いる方法，カウンタを用いる方法，どちらでも対応可能であるが，適用する動作周波数が高くなるにつれカウンタを用いる方式は非現実的になり，インバータチェーンを用いる方式が有利になる．このため，本論文では Delay Line の実装方法としてインバータチェーンを用いることにする．

インバータチェーンを用いる方法には 2 種類の実装方法が考えられる．全ての種類の Delay Line を用意する方法と 1 本の Delay Line を用意し，Delay Line の途中で信号を取り出す事を可能にする方法である．後者の方が必要なインバータが少なく，面積が小さくなるが，選択される Delay Line の長さが大きく異なる場合，正常に動作しない．短い Delay

Line が選択された場合，次のサイクルのパルスが入力された後であってもその前のサイクルのパルスが Delay Line を伝搬している可能性があるためである．インバータチェーンの実装方法は選択される Delay Line の長さに合わせて選択する必要がある．

6.2.3　Common Delay circuit

同期 Speculative Completion を実現する際に最も問題になるのは，各ステージからの遅延情報を受け取り，全体の遅延時間を決定して次のクロックパルスを生成する回路の実装法である．ここでは，サイクル毎にクロックの遅延の選択を可能とする実装法の一つとして Common Delay circuit を提案する．この回路は図 6.1 に示すように各ステージからの遅延情報を受け取り，次のクロックパルス信号を必要時間遅延させ各ステージへ送る働きをする．

電源が投入され，回路が安定した後，最初の Start signal が High になると Start signal パルスが生成され，Common Delay circuit に送られる．Start signal パルスは Pulse Generator で一定の長さに調整された後，Base Delay の分だけ遅延される．Pulse Generator の遅延と Common Delay circuit における遅延を合わせたものが，各ステージが必要な遅延を判断して判断結果を判定回路 (Common Delay Control circuit) に送る時間に等しくなるように調整する．

Common Delay Control circuit では各ステージで必要となる遅延の中で最も長いものに合わせて Delay Line を選択する．選択された Delay Line にパルスが入力され，遅延されたパルスはクロック信号として，各ステージへ送られ，出力バッファへの書き込みの制御に用いられる．同時に，パルスは，Pulse Generator へ戻され，次のサイクルのパルスを発生する．

この方式では，パルスの幅を一定に維持するための工夫が施されている．Pulse Generator(図 6.2) では，信号が遅延無しで流れる線と信号を遅延させる線に分けた後，OR 素子で合流する．この処理により，信号が短くなりすぎ，消えてしまうことを防ぐ (図 6.3)．そして，再度，信号が遅延無しで流れる線と信号を遅延させる線に分けた後，AND 素子で合流する．このとき遅延が加えられる線を流れる信号はインバータ素子によって反転する．この処理は信号の長さが一定の値を超えないように制限する働きがある (図 6.4)．以降，Pulse Generator の入力パルスの立ち上がりから，次のパルスの立ち上がりまでを，1 サイクルとして定義する．

この回路は，電源投入時に回路が安定する前に Start signal を与えるとジッタの問題が生じる可能性がある．したがって十分安定させた後にリセットを解除して Start signal を与える必要がある．

同期 Speculative Completion では全ての回路のクロック信号が一箇所の回路から送られるためスキューの問題が発生する．スキューを考慮する必要がない小さな回路では問題ないが，スキューを考慮しなければならない大きな回路の場合，同期式のクロックスキューと同様の解決策を取る必要がある．

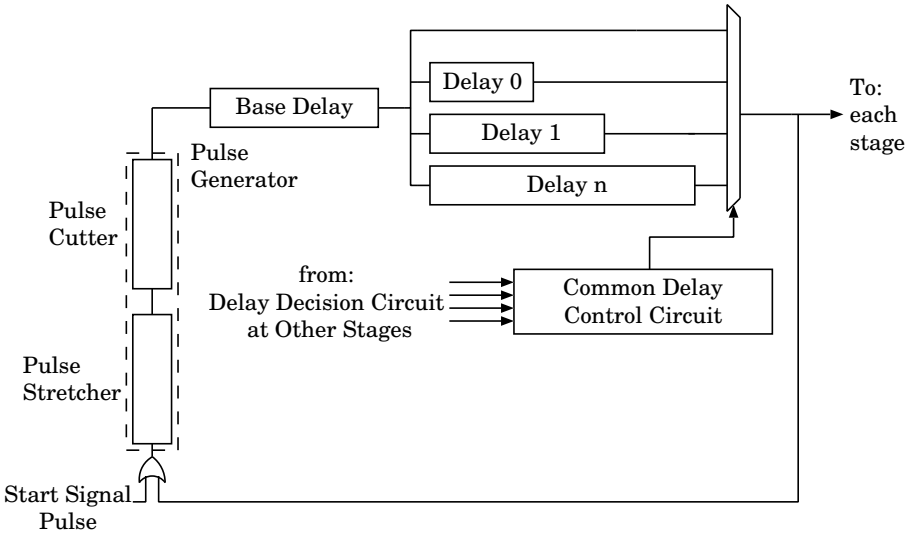

図 6.1: Common Delay circuit

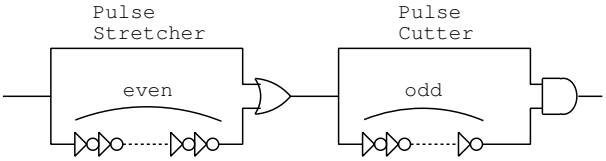

図 6.2: Pulse Generator circuit

6.3　評価

6.3.1　評価用 CPU(Sync) のアーキテクチャ

同期 Speculative Completion の評価のため簡単な CPU コアを設計した．評価用 CPU(以下 Sync) は DLX ISA [PH96] に整数乗算命令を加えた命令セットを Instruction Fetch, Instruction Decode, Execution, Memory Access, Write Back の標準的な 5 段パイプライン構成で実行する．キャッシュメモリはデータキャッシュ，命令キャッシュとも 32KB, Direct map 方式，データキャッシュは Write Back 方式とした．命令キャッシュとデータキャッシュはそれぞれ独立にアクセス可能で，キャッシュアクセス時の構造ハザードは生じない．

Sync は DLX 同様，メモリからデータを読み出す際に，次の命令がその結果を利用する場合ストールする．また，分岐命令は遅延スロット 1 の遅延分岐となっている．また，付け加えた整数乗算器は Wallace tree 型乗算器を用い，1 クロックで乗算を実行する．

6.3.2　Sync のクリティカルパスと実装面積

Sync を Verilog-HDL で記述し，別に評価する回路 (遅延線，Pulse Generator）を除く回路について Synopsys 社の Design Compiler で論理合成を行い，同社の Apollo を使用

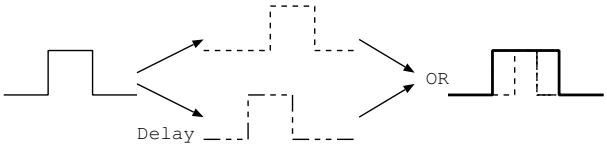

図 6.3: Signal Stretch circuit

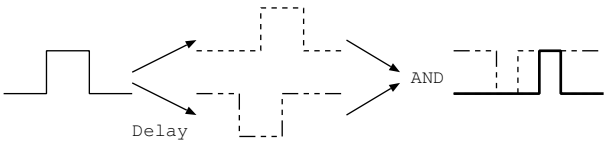

図 6.4: Signal Cut circuit

して配置配線を行った．プロセスは Rohm 社の 0.35μmCMOS メタル 3 層プロセスを想定した．

クリティカルパスは表 6.1 のようになった．

また，各ステージで必要な実行時間を決定し，判定回路で使用する遅延線を決定する際のクリティカルパスは 5.82ns となった．

配置配線後の面積は 6.30mm^2 となった．この値はキャッシュメモリを含んでいない．また，判定回路以外の Common Delay circuit の回路面積は後に評価するため，この値には含まれない．

6.3.3　同期 Speculative Completion

Sync のクリティカルパスを元に遅延を決定した．

命令およびデータに依存せず常に行われる処理の中でもっとも実行時間が長いのは Decode ステージである．この処理には 13.2ns かかる．

常に実行されるとは限らない処理で Decode ステージの実行時間よりも長いものについて考える．このような処理の中では，Execution ステージで 17bit 以上の値に対する処理が行われる場合が最も短く，9bit 以上の値の乗算が行われる場合が最も長い．それぞれ，14.1ns，23.0ns の処理時間を要す．Execution ステージで 16bit 以下の値に対する乗算以

表 6.1: クリティカルパス

stage	critical path(ns)
Instruction Fetch	2.2
Instruction Decode	13.2
Execution	23.0
Memory	4.1

表 6.2: Delay line

delay line name	delay time(ns)
OTHER	13.2
N-H	14.1
M-H	23.0
MEMORY	60.7

外の処理が行われる場合，及び，8bit 以下の乗算が行われる場合は Decode ステージの実行時間より短いため考慮する必要は無い．

メインメモリにアクセスする場合のメモリステージの実行時間は 60.7ns とした．この値は各処理の実行時間の差が長い場合において正しく動作することをテストするため，HSPICE でのシミュレーションがある程度の時間で終わる，できるだけ長い実行時間とした．

Sync では以上の結果を元に同期 Speculative Completion での遅延線を表 6.2 のように決定した．

6.3.4　Common Delay circuit の評価

前節で決定した遅延を元に Common Delay circuit を設計した．Verilog-HDL で記述し，Milkyway，Apollo を使い配置配線を行い，PDRACULA で結線情報を抽出し HSPICE で評価を行った．

本実装では，遅延線はインバータチェーンにより実現した．HSPICE による評価の結果，Common Delay circuit はクロックが立ち上がってから 9ns 以内に遅延線が決定すれば，正しくクロックパルスが発生可能であることがわかった．これより時間がかかると，動作信号が消える，一信号が割れピークが 2 個現れる，2 個の信号が一部融合するなど，波形が乱れる現象が起きた (図 6.5)．

HSPICE シミュレーションによって得られた波形の例を図 6.6-図 6.10 に示す．縦軸は電圧 (1 目盛 0.5V)，横軸は時間 (図 6.6，図 6.7，図 6.8 は 1 目盛 20ns，図 6.5，図 6.9，図 6.10 は 1 目盛 50ns) を表す．

インバータチェーンを用いているためシミュレーションを開始し電圧を上昇させる際に，波形に乱れが生じている．ループ状の回路になっているため，長いものでは 120ns 程度波形の乱れが続く．

図 6.10 は出力する周期を変更した場合の波形を表し，243ns までは 60.7ns 周期の信号を出力する設定となっており，243ns から 278ns までは 13.2ns の周期の信号を，278ns 以降は 23.0ns の信号を出力する設定となっている．各サイクルで遅延の長さが変化しないことが分かる．また，波形に乱れが生じていないことが分かる．

それぞれの遅延線による遅延の長さは表 6.3 のようになった．この表中の inverter で示されている使用したインバータの個数は各遅延線で使用された数のみ含み，Pulse Generator のインバータ素子は含まない[1]．

[1]Verilog 形式で表中の数インバータを記述し，論理合成，配置配線，結線情報の抽出を行い，HSPICE

表 6.3 中の Pow は動作信号出力が安定してから 100ns の間の Common Delay circuit の平均消費電力である．

配置配線後の回路の面積は $0.13mm^2$ となった．60.7ns の比較的長い遅延線が存在するにも関わらず，小さな面積となっている．

また，インバータチェーンの長さを調整することにより図 6.1 において示した Common Delay circuit から最大 256MHz 相当の信号を安定して作り出すことが可能であることを確認した．

図 6.5: Common Delay circuit のエラー出力例

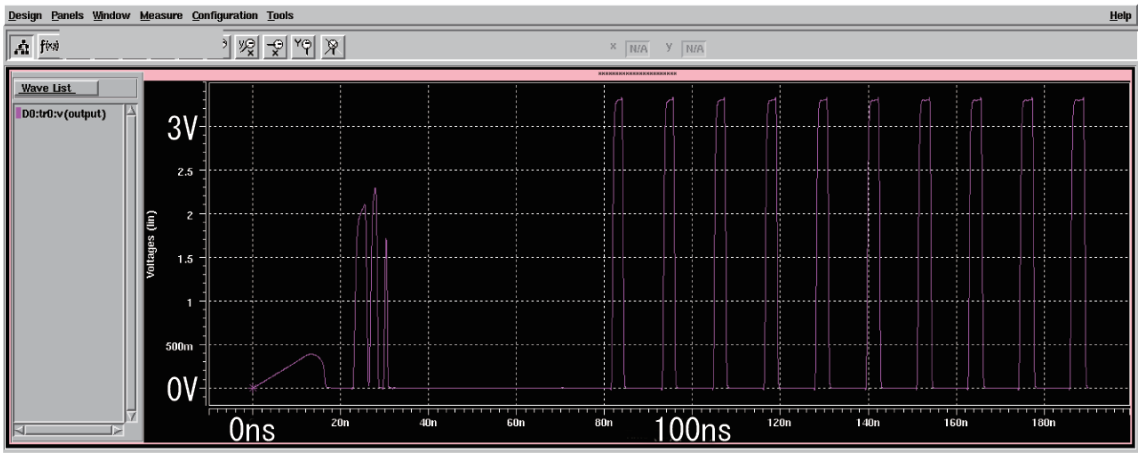

図 6.6: Common Delay circuit の出力例 (13.2ns)

形式の回路を得ているが，ツールのバグにより Verilog で記述した回路とは異なる回路が出力されるため，HSPICE 形式の回路の一部を書き換え，意図した回路を得ている．

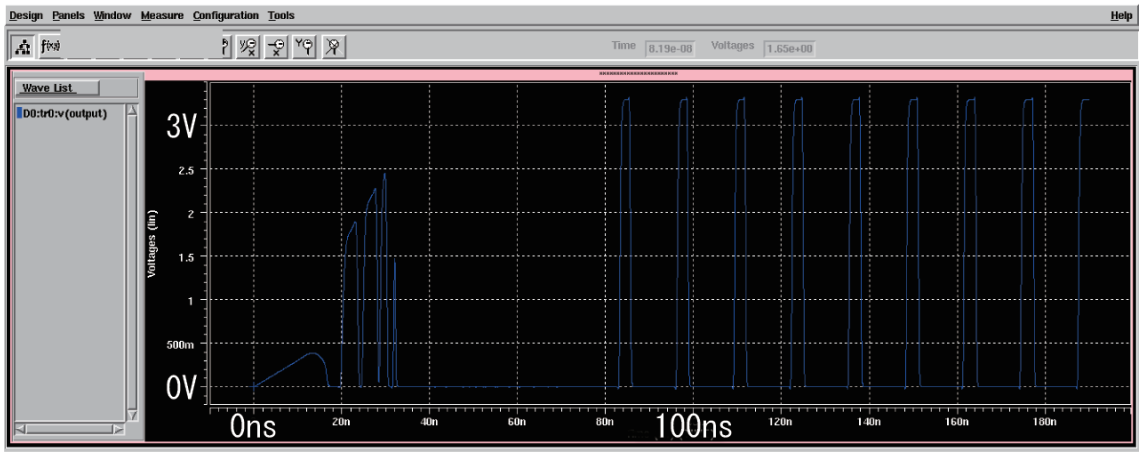

図 6.7: Common Delay circuit の出力例 (14.1ns)

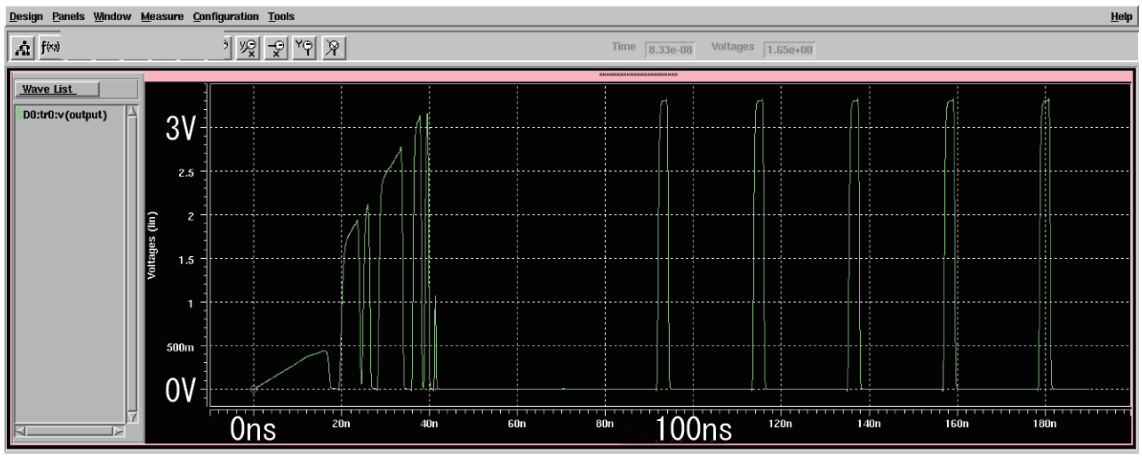

図 6.8: Common Delay circuit の出力例 (23.0ns)

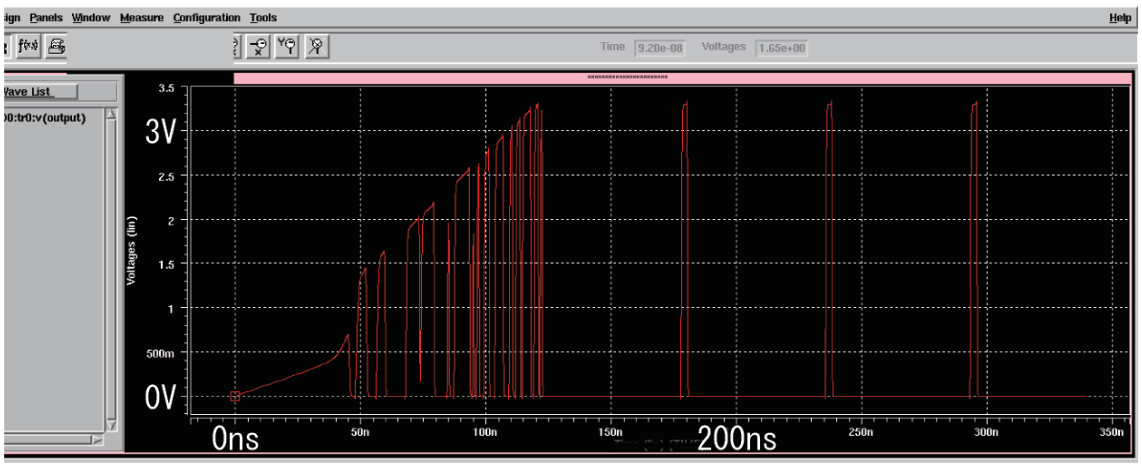

図 6.9: Common Delay circuit の出力例 (60.7ns)

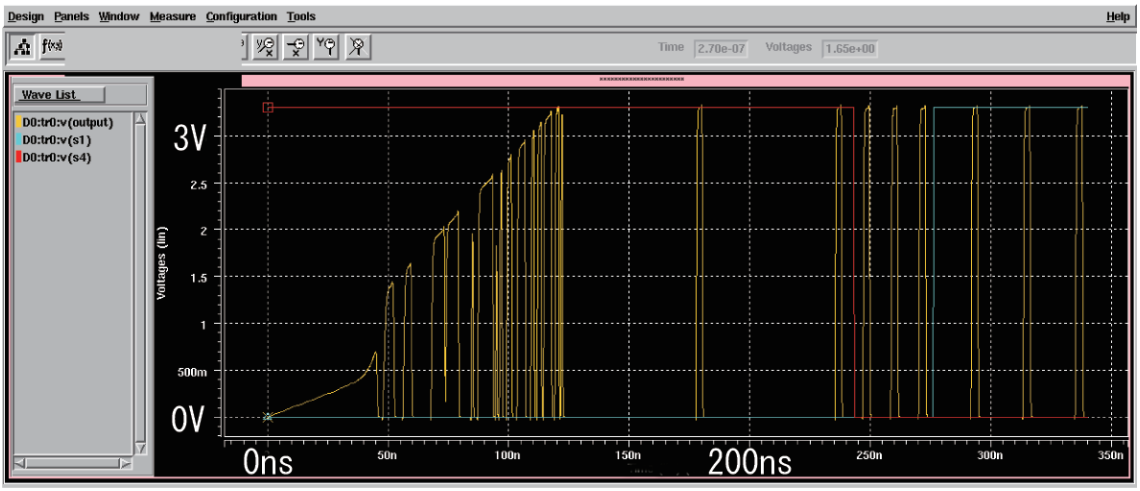

図 6.10: Common Delay circuit の出力例 (出力周期変更)

表 6.3: Delay line 別の遅延の長さ

name	interval(ns)	inverter	Pow(mW)
OTHER	13.2	14	3.96
N-H	14.1	32	3.88
M-H	23.0	150	3.48
MEMORY	60.7	624	2.52

6.3.5　評価用非同期 CPU コア (Async) の構成

提案した同期 Specualtive Completion を，従来型の非同期 Speculative Completion と比較するため，簡単な非同期 CPU コアを設計した．以下，この非同期 CPU コアを Async と呼ぶ．Async は AMULET2 [FDG+96] [FGT+97] [WDF+97] [FGR+99] のアーキテクチャを元に Sync を非同期化した．AMULET2 とは異なり性能向上のため Memory ステージから Execution ステージへフォワーディングを行っている．Async は前節に述べた Sync と以下の点で異なる．

- 各ステージは Request 信号と Acknowledge 信号をやり取りすることにより非同期で動作する．
- 非同期動作のため，異なるステージ間でフォワーディングを行う際にはハンドシェーク線を用い，同期を取る．

Async において，Instruction Decode ステージ，Execution ステージ，Memory ステージのパイプラインはループしているため，各ステージが同時に実行された場合，実行終了後，Instruction Decode ステージは Execution ステージの実行を待ち，Execution ステージは Memory ステージの実行を待ち，Memory ステージは Instruction Decode ステージの実行を待つため，デッドロックが発生する．そのため，Instruction Decode ステージは Memory ステージが Request 信号を出力した場合，Execution ステージの実行が終了するまで実行しない．

Async を Verilog-HDL で記述し，別途評価する回路 (遅延線，Pulse Generator，各ステージへの駆動信号を制御する制御回路）を除き Sync と同条件で評価を行った．

Async の配置配線後の面積は $5.53mm^2$ となった．この値は制御回路，キャッシュメモリを含んでいない．制御回路を含んでいないため Sync よりも小さくなっている．

Async のクリティカルパスを元に以下のように遅延を決定した．非同期式の制御に対応するため Sync とは一部異なった結果となっている．

- Fetch ステージの遅延はキャッシュヒット時には 7.7ns，キャッシュミス時には同期 Speculative Completion のメモリステージのメインメモリアクセス時の実行時間に合わせ 60.7ns かかるとした．
- Decode ステージの遅延はジャンプ命令・分岐命令実行の際に加算が行われない場合は 9.8ns，行われる場合は 10.4ns となった．
- Execution ステージの遅延は以下のようになった．
  - 16bit 以下の値同士の演算で乗算を行わない場合 12.1ns
  - 17bit 以上の値を含む演算で乗算を行わない場合 14.1ns
  - 8bit 以下の値同士の乗算の場合 12.2ns
  - 9bit 以上の値を含む乗算の場合 20.8ns
- Memory ステージの遅延は 3.8ns となった．メモリアクセスが行われる場合は，キャッシュヒット時は 9.3ns，キャッシュミス時には同期 Speculative Completion のメモリステージのメインメモリアクセス時の実行時間に合わせ 60.7ns かかるとした．

6.3.6　評価条件

Async 及び Sync を使い以下の 4 条件で評価した.

- Sync-SC:Sync を用い，同期 Speculative Completion を想定し，各サイクルの遅延を表 6.3 から取ったもの
- Sync-noSC:Sync を用い，通常の同期式 CPU を想定し，表 6.2 から 1 サイクルを 23.0ns，メインメモリへのアクセスタイムを 60.7ns としたもの
- Async-SC:Async を用い，非同期 Speculative Completion を想定し，各ステージの遅延を前節の値から取ったもの
- Async-noSC:Async を用い，非同期式 CPU を想定し，各ステージの遅延を前節の最も長い値から取ったもの

素子遅延，配線遅延を考慮した Verilog シミュレーションによって評価を行った.

評価用プログラムとして，SPECint95 より go と compress，暗号処理用プログラムの AES，imdct，および，LU 分解を使用した.

- go は碁の対局をコンピュータ同士で行うものであり，ボードサイズは 10，読みの深さを表す playlevel は 10 に設定した.
- compress はデータの圧縮を行うプログラムであり，圧縮を行うデータはシード及びスタートキャラクタを指定することにより自動的に生成される．ここではデータの大きさを表す count を 1，スタートキャラクタを a，シードを 713 とした.
- AES は 4096Bytes のデータを AES 方式で暗号化し復号化した後，元のデータと比較を行う．初期値は全て 0 である.
- imdct は MP3 の再生処理の一部である逆変形離散コサイン変換である．初期値は全て 0 とし 1152Bytes のデータの変換を行う.
- lu 分解では 32 × 32 の初期値が 0 である行列の LU 分解を行う.

全てのプログラムは処理すべきデータを内部に持ち，メモリ以外のデバイスとの入出力を行わない．go 及び AES は 6000000 命令を読み込んだ時点で実行を中断している.

6.4　シミュレーションによる性能評価

シミュレーションによる評価結果を表 6.4 に示す．単位は MIPS である.

ここで，α は Sync-SC の値を Sync-noSC の値で割った値，β は Async-SC の値を Async-noSC の値で割った値，γ は Sync-SC の値を Async-SC の値で割った値である.

表 6.5 から Sync-SC は Sync-noSC と比較し約 1.55 倍の性能であり，Async-SC は Async-noSC と比較し約 1.38 倍の性能である．このことから，同期式 CPU に対する同期 Speculative Completion の効果は非同期式 CPU に対する Speculative Completion の効果と比

表 6.4: Throughput

program	Sync-noSC	Sync-SC	Async-noSC	Async-SC
go	38.4	60.4	37.6	51.0
compress	44.1	67.7	34.6	44.5
AES	43.7	70.2	40.6	61.1
imdct	38.1	58.0	35.7	47.1
lu	42.6	65.3	37.8	53.1
Average	41.4	64.3	37.2	51.3

表 6.5: Ratio

program	α	β	γ
go	1.57	1.36	1.18
compress	1.54	1.29	1.52
AES	1.61	1.29	1.15
imdct	1.52	1.32	1.23
lu	1.53	1.41	1.23
Average	1.55	1.38	1.26

較し大きい事がわかる．これは Sync-noSC が，ほとんど実行されない 32bit 同士の値の乗算の遅延に合わせて全体が動くのに対して Sync-SC が入力に応じた遅延で動くことができるためである．

パイプライン処理の特定のステージのクリティカルパスが長い場合，そのステージをさらに複数のステージに分割することにより性能を向上することが可能である．すなわち，Sync においても乗算回路を 2 ステージに分けることにより動作周波数を向上させることができる．しかし，ステージ数の増加は，CPU のバックエンドにおいては，フォワーディングを複雑にし，フロントエンドにおいては分岐予測のオーバヘッドを大きくする等の問題点があるが，同期 Speculative Completion は，これを避けることができる利点がある．

また，Sync-SC は Async-SC と比較し約 1.26 倍の性能である．これは，Async-SC では Decode 処理で必要なレジスタ読み出しと，先行命令が Memory ステージを終えて演算結果を書き戻す Write Back 処理を非同期的に実行するためである．Decode 処理中の Write Back 処理を禁止すると，各ステージの処理結果を正しくフォワーディングできなくなる．このため，Decode 処理中に Write Back 処理が実行された場合，Decode 処理を再実行するように設計されており，このためのオーバーヘッドが生じている．

表 6.6 は Sync-SC の場合のそれぞれの Delay Line の使用回数である．Mem-I，Mem-D はそれぞれ命令／データを読み込むためにメインメモリにアクセスする処理に対応する長さの Delay Line が選択された場合を表す．M-H，N-H，OTHER は表 6.2 の delay line name に対応している．ALL は M-H，N-H，OTHER を合計した値，INST は実際に実行

表 6.6: 使用 Delay Line

	go	compress	AES	imdct	lu
Mem-I	9,841	1,139	1,707	1,370	106
Mem-D	31,020	71,714	232	23,926	2
M-H	30	0	0	28,968	20,858
N-H	4,137,900	4,469,432	2,220,381	871,908	71,865
OTHER	3,083,081	755,974	4,100,396	361,507	189,649
ALL	7,221,011	5,225,406	6,320,777	1,262,383	282,372
INST	6,000,000	5,000,907	6,000,000	1,054,550	260,945

された命令数を表している．ALL と INST の差はパイプラインがストールした回数を表す (以下，STALL と呼ぶ).

データ読み込み時のメインメモリに対するアクセス数が極端に少ないものがあるが，これはプログラムが処理開始時に使用するメモリに初期データの書き込みを行い，キャッシュメモリにデータが載った状態から処理を始める構造になっているためである．Sync-SC と Sync-noSC で各プログラムでの実行速度の改善割合はあまり変わらないが，Sync-SC において最も実行時間が短くなる OTHER の割合が高い go と AES の改善率が比較的高くなっている．

表 6.6 から，compress，AES プログラム実行時の性能が高い理由が，INST と比較し STALL が少ないためであることが分かる．これは go と imdct 等 INST と比較し STALL が多いプログラムは実行速度が遅くなっている事からも明らかである．

また，最も実行時間を必要とするメインメモリへのアクセス数が多い compress 実行時の処理速度より，imdct 実行時の処理速度の方が遅くなっていることから同期 Speculative Completion を用いる場合は，設計時には，最も時間を要するクリティカルパスを短くするために努力するよりも，最も良く用いられる回路の遅延を短くする努力をすべきであることがわかる．

6.5　結論

同期 Speculative Completion の評価を目的として作られた非同期 CPU コアの配線遅延付きシミュレーションを行った．シミュレーションの結果，通常の同期回路の約 1.65 倍と高速な動作が可能であることを示した．

本章では同期 Speculative Completion を CPU コアに適用して評価を行ったが同期 Speculative Completion の応用範囲はこれだけに留まらない．以下の条件を満たす場合、同期 Speculative Completion を導入することにより同期式と比較し高速な動作が期待できる。

- クリティカルパスとなっている処理の処理時間と処理が行われる確率により重み付けをされた平均処理時間の差がある．
- クリティカルパスにならない処理がクリティカルパスにならない事を予想回路により予め予想できる．

- 予想回路の予想に必要な時間，予想結果を Common Delay 回路に送る時間，及び，Common Delay 回路による判定時間を合計したものがクリティカルパスと比較し短い．

これら条件が満たされた上で同期 Speculative Completion 導入によるコストと比較し速度向上のメリットがあると判断された場合，同期 Speculative Completion を導入することを考慮にいれるべきである．特に各ステージの実行時間が大幅に変化することが予想される回路，ほとんど使わない機能のために動作周波数が低く押さえられる回路を同期 Speculative Completion を採用して作成した場合，大幅な動作速度の向上が期待できる．また，動的再構成可能なリコンフィギャラブルプロセッサのように演算内容に応じて数倍，実行時間が変化する回路への応用が期待される．

第7章　まとめ

1990年代より盛んになった非同期式回路に関する研究は2000年頃を境に大きく2つのフェーズにわけることができる．

第1のフェーズでは今まで同期式で実現していた全ての回路を非同期式回路によって実現し様々な面でのパフォーマンスの向上を図る事を目指し研究が行われた．そのため，簡単な回路からCPUなどに代表される複雑な回路まで非同期式の回路で実現され，パフォーマンスの評価が行われた．

この流れの中で，回路の基本的な構成要素であるスイッチに関する研究は非同期式を採用することにより大幅なパフォーマンス向上が期待されるにも関わらずほとんど研究は行われていなかった．

そのため，非同期式スイッチに関する研究を行った．この研究で作られた非同期式スイッチ (Simple Asynchronous Switch) は非同期式回路の利点を最大限活かすため，機能面ではスイッチとして最小限必要なアービトレーション機能のみを持つに留まるが非常に高速なデータ転送を可能にすることを目指した．

高速な転送を可能にするため，処理に時間を必要とするアービトレーション機能と処理が軽いデータ転送機能を可能な限り分離し，独自の高速連続転送に特化した転送プロトコルを採用した．

完全非同期のアービトレーション機能は独自設計の分散アービトレーション回路により実現した．データは転送時，素子5つのみを通過するため高速な転送を可能にしている．また，標準ライブラリの素子の5倍の周波数の信号をドライブ可能な独自設計の素子を用いることによりさらに高速な転送を可能にしている．

0.6μmプロセスを用い，シミュレーション，実機で評価を行った．シミュレーションでは全ポートを用いた場合最大32Gbpsの転送性能を示した．また，実機は評価ボードにより評価され，条件付きながら正しく動作することが確認された．

この研究は同期式を用いて作成されていた回路を非同期式回路で実現することにより，性能が向上する一例を示した．この研究では極めて単純な機能のみ実現したが，複雑な機能を持った回路も非同期式で実現可能である．複雑な回路を非同期式で実現した場合，同期式と比較し性能が劣る場合が多い．非同期式のスイッチに複雑な機能を搭載した場合，同期式と比較し性能面における評価を行うため研究を進める必要がある．

2000年頃に始まった非同期式研究第2フェーズでは従来の回路を全て非同期化するのではなく，同期式回路に非同期式の考え方を取り入れ，性能の向上を図る研究が盛んに行われるようになった．主に同期式回路の一部を非同期式回路に置き換えることにより高速な処理を目指す研究等が主に行われている．

そのような流れの中で非同期式の高速化技術を同期式に応用する研究を行った．

本論文で述べた同期Speculative Completionは非同期式回路の高速化技術であるSpec-

ulative Completion を同期式に応用した高速化技術である．同期式の回路を各サイクルにおいて同時に処理を開始する非同期式回路ととらえることにより非同期式の高速化技術を同期式に導入することを可能にしている．

同期 Speculative Completion を導入した CPU を Verilog-HDL 等を用いて記述し，素子遅延，配線遅延を考慮したシミュレーションによって評価した．比較対象として，ほぼ同様の機能を持った非同期式 CPU，Speculative Completion を導入した非同期式 CPU，同期式 CPU を用意した．

評価の結果，同期式 Speculative Completion を導入した CPU は

- 非同期式 CPU の 1.74 倍
- Speculative Completion を導入した非同期式 CPU の 1.26 倍
- 同期式 CPU の 1.55 倍

の処理性能があることが示された．

同期式 Speculative Completion は稀に実行される処理によってクリティカルパスが長くなり処理性能が低下する場合に特に有効に働く．クリティカルパスが長いステージが存在する場合，同期式ではステージを分割することによって対処するが，ステージが増え，フォワーディングなどが複雑になるなどの問題があった．同期 Speculative Completion はこの問題を解決するために有効な技術である．

本論文では同期 Speculative Completion を単純な CPU に適用した．だが，処理によって大きく動作クロックが変化する動的再構成可能な FPGA 等に適応する場合にも大きな効果が期待される．また，GALS と組合せ，GALS を適用した回路の一部に適用することにより性能を向上させる利用方法も考えられる．同期 Speculative Completion は数十 MHz から数百 MHz の低速から中速程度の動作周波数の回路に一般的に使用することが可能であり，様々な分野への応用が期待される．

非同期式回路は本格的な研究が始まってから日が浅く，今後も応用分野が広がっていくことが期待される．

特に近年注目を集めている低消費電力，低 EMI などの分野において非同期式は同期式に対して優れた特性を示すため，これからのさらなる研究開発が期待されている．

また，GALS，本論文で述べた同期 Speculative Completion に代表される同期式に非同期式の技術を導入するテクニックを用いることにより，同期式が直面している，もしくは直面するであろう問題に対処する研究は今後ますます盛んになることが予想される．

将来，非同期式は同期式と共存し，非同期式の特性が発揮可能な分野において発展することが期待される．

参考文献

[Bar00] A. Bardsley. *Implementing Balsa Handshake Circuits*. PhD thesis, Department of Computer Science, University of Manchester, 2000.

[BE00] A. Bardsley and D. A. Edwards. The Balsa asynchronous circuit synthesis system. In *Forum on Design Languages*, pp. 37–44, September 2000.

[Bre65] H. C. Brearley. Illiac ii: A short description and annotated bibliography. *IEEE Transactions on Computers*, Vol. 14, No. 6, pp. 399–403, June 1965.

[Cha84] Daniel M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, October 1984.

[Chu86] Tam-Anh Chu. On the models for designing VLSI asynchronous digital circuits. *Integration, the VLSI journal*, Vol. 4, No. 2, pp. 99–113, June 1986.

[Chu87] Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT Laboratory for Computer Science, June 1987.

[Cla67] Wesley A. Clark. Macromodular computer systems. In *AFIPS Conference Proceedings: 1967 Spring Joint Computer Conference*, Vol. 30, pp. 335–336, Atlantic City, NJ, 1967. Academic Press.

[CLJ+01] William S. Coates, Jon K. Lexau, Ian W. Jones, Scott M. Fairbanks, and Ivan E. Sutherland. FLEETzero: An asynchronous switch fabric chip experiment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 173–182. IEEE Computer Society Press, March 2001.

[EB02] Doug Edwards and Andrew Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, Vol. 45, No. 1, pp. 12–18, 2002.

[FDG+94a] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, S. Temple, and J. V. Woods. The design and evaluation of an asynchronous microprocessor. In *Proc. International Conf. Computer Design (ICCD)*, pp. 217–220. IEEE Computer Society Press, October 1994.

[FDG+94b] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and J. V. Woods. AMULET1: A micropipelined ARM. In *Proceedings IEEE Computer Conference (COMPCON)*, pp. 476–485, March 1994.

[FDG+96] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and S. Temple. AMULET2e. In C. Muller-Schloer, F. Geerinckx, B. Stanford-Smith, and R. van Riet, editors, *Embedded Microprocessor Systems*, pp. 23–25, September 1996. Proceedings of EMSYS'96 - OMI Sixth Annual Conference.

[FEG00] S. B. Furber, D. A. Edwards, and J. D. Garside. AMULET3: a 100 MIPS asynchronous embedded processor. In *Proc. International Conf. Computer Design (ICCD)*, September 2000.

[FGG98] Stephen B. Furber, James D. Garside, and David A. Gilbert. AMULET3: A high-performance self-timed ARM microprocessor. In *Proc. International Conf. Computer Design (ICCD)*, pp. 329–334, October 1998.

[FGR+99] Stephen B. Furber, James D. Garside, Peter Riocreux, Steven Temple, Paul Day, Jianwei Liu, and Nigel C. Paver. AMULET2e: An asynchronous embedded controller. *Proceedings of the IEEE*, Vol. 87, No. 2, pp. 243–256, February 1999.

[FGT+97] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, and N. C. Paver. AMULET2e: An asynchronous embedded controller. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 290–299. IEEE Computer Society Press, April 1997.

[Fur95] S. Furber. Computing without clocks: Micropipelining the ARM processor. In Graham Birtwistle and Al Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pp. 211–262. Springer-Verlag, 1995.

[Gag98] Hans van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller.* PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, September 1998.

[GBB+00] J. D. Garside, W. J. Bainbridge, A. Bardsley, D. A. Edwards, S. B. Furber, J. Liu, D. W. Lloyd, S. Mohammadi, J. S. Pepper, O. Petlin, S. Temple, and J. V. Woods. AMULET3i — an asynchronous system-on-chip. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 162–175. IEEE Computer Society Press, April 2000.

[GBvB+98] Hans van Gageldonk, Daniel Baumann, Kees van Berkel, Daniel Gloor, Ad Peeters, and Gerhard Stegmann. An asynchronous low-power 80c51 microcontroller. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 96–107, 1998.

[GFC99] J. D. Garside, S. B. Furber, and S.-H. Chung. AMULET3 revealed. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 51–59, April 1999.

[HGDT84] L. G. Heller, W. R. Griffin, J. W. Davis, and N. G. Thoma. Cascode voltage switch logic: A differential cmos logic family. In *International Solid State Circuits Conference*, pp. 16–17, February 1984.

[HH03] Matthew Heath and Ian Harris. A deterministic globally asynchronous locally synchronous microprocessor architecture. In *IEEE Microprocessor Test and Verification Workshop (MTV)*, pp. 119–123, 2003.

[Huf64] D. A. Huffman. The synthesis of sequential switching circuits. In E. F. Moore, editor, *Sequential Machines: Selected Papers*. Addison-Wesley, 1964.

[IM02] A. Iyer and D. Marculescu. Power and performance evaluation of globally asynchronous locally synchronous processors. In *International Symposium on Computer Architecture*, pp. 158–170, 2002.

[ION04] M. Imai, M. Ozcan, and T. Nanya. Evaluation of delay variance in asynchronous circuits based on the scalable-delay-insensitive model. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 62–71. IEEE Computer Society Press, April 2004.

[JBGK00] Hans Jacobson, Erik Brunvand, Ganesh Gopalakrishnan, and Prabhakar Kudva. High-level asynchronous system design using the ACK framework. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 93–103. IEEE Computer Society Press, April 2000.

[KIN+01] Ryusuke Konishi, Hideyuki Ito, Hiroshi Nakada, Akira Nagoya, Kiyoshi Oguri, Norbert Imlig, Tsunemichi Shiozawa, Minoru Inamori, and Kouichi Nagami. PCA-1: A fully asynchronous self-reconfigurable LSI. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 54–61. IEEE Computer Society Press, March 2001.

[KOTG99] Christoph Kern, Tarik Ono-Tesfaye, and Mark Greenstreet. A light-weight framework for hardware verification. In *The Fifth Annual Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'99*, Vol. 1579 of *Lecture Notes in Computer Science*, pp. 330–344, Amsterdam, The Netherlands, March 1999. Springer.

[Mar90] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In William J. Dally, editor, *Advanced Research in VLSI*, pp. 263–278. MIT Press, 1990.

[MB59] David E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pp. 204–243. Harvard University Press, April 1959.

[MBL+89a] Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, and Pieter J. Hazewindus. The design of an asynchronous microprocessor. In Charles L. Seitz, editor, *Advanced Research in VLSI*, pp. 351–373. MIT Press, 1989.

[MBL+89b] Alain J. Martin, Steven M. Burns, T. K. Lee, Drazen Borkovic, and Pieter J. Hazewindus. The first asynchronous microprocessor: the test results. *Computer Architecture News*, Vol. 17, No. 4, pp. 95–110, June 1989.

[McC62] E. J. McCluskey. Fundamental mode and pluse mode sequential circuits. In *In Proc. IFIP Congress*, pp. 725–730. Holland Publishing Co., 1962.

[MJCL97] Charles E. Molnar, Ian W. Jones, Bill Coates, and Jon Lexau. A FIFO ring oscillator performance experiment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 279–289. IEEE Computer Society Press, April 1997.

[MLM+97] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nyström, Paul Pénzes, Robert Southworth, and Uri Cummings. The design of an asynchronous MIPS R3000 microprocessor. In *Advanced Research in VLSI*, pp. 164–181, September 1997.

[MNPW01] Alain J. Martin, Mika Nyström, Paul Pénzes, and Catherine Wong. Speed and energy performance of an asynchronous MIPS R3000 microprocessor. Technical Report CSTR:2001.012, California Institute of Technology, 2001.

[MTMR02] Simon Moore, George Taylor, Robert Mullins, and Peter Robinson. Point to point GALS interconnect. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 69–75, April 2002.

[MVF00] Jens Muttersbach, Thomas Villiger, and Wolfgang Fichtner. Practical design of globally-asynchronous locally-synchronous systems. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 52–59, April 2000.

[Now96] S. M. Nowick. Design of a low-latency asynchronous adder using speculative completion. *IEE Proceedings, Computers and Digital Techniques*, Vol. 143, No. 5, pp. 301–307, September 1996.

[NTK+97] T. Nanya, A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno, F. Okamoto, H. Fujimoto, O. Fujita, M. Yamashina, and M. Fukuma. TITAC-2: A 32-bit scalable-delay-insensitive microprocessor. In *Symposium Record of HOT Chips IX*, pp. 19–32, August 1997.

[NTK+99] T. Nanya, A. Takamura, M. Kuwako, M. Imai, M. Ozawa, M. Ozcan, R. Morizawa, and H. Nakamura. Scalable-delay-insensitive design: A high-performance approach to dependable asynchronous systems. In *Proc. International Symp. on Future of Intellectual Integrated Electronics*, pp. 531–540, Sendai, Japan, March 1999.

[NUK+94] Takashi Nanya, Yoichiro Ueno, Hiroto Kagotani, Masashi Kuwako, and Akihiro Takamura. TITAC: Design of a quasi-delay-insensitive microprocessor. *IEEE Design & Test of Computers*, Vol. 11, No. 2, pp. 50–63, 1994.

[OGV+03] Stephan Oetiker, Frank K. Gurkaynak, Thomas Villiger, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner. Design flow for a 3-million transistor gals test chip. In *Proc. ACiD (Asynchronous Circuit Design) Workshop*, 2003.

[OIN02] Metehan Ozcan, Masashi Imai, and Takashi Nanya. Generation and verification of timing constraints for fine-grain pipelined asynchrounous data-path circuits. In *Proceeding of Advanced Research in Asynchronous Circuits and Systems*, pp. 109–114, April 2002.

[ONM+00] Thomas Olsson, Peter Nilsson, Thomas Meincke, Ahmed Hemani, and Mats Torkelson. A digitally controlled low-power clock multiplier for globally asynchronous locally synchronous designs. In *IEEE International Synposium on Circuits and Systems*, pp. 13–16, May 2000.

[PH96] D.A. Patterson and J.L. Hennessy. *Computer Architecture A Quantitative Approach Second Edition*. MORGAN KAUFMANN PUBLISHERS, 1996.

[RM03] V.S.P. Rapaka and D. Marculescu. A mixed-clock issue queue design for globally asynchronous, locally synchronous processor cores. In *International Symposium on Low Power Electronics and Design*, pp. 372–377, 2003.

[RSG+99] Shai Rotem, Ken Stevens, Ran Ginosar, Peter Beerel, Chris Myers, Kenneth Yun, Rakefet Kol, Charles Dike, Marly Roncken, and Boris Agapiev. RAPPID: An asynchronous instruction length decoder. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 60–70, April 1999.

[RSP+00] Marly Roncken, Ken Stevens, Rajesh Pendurkar, Shai Rotem, and Parimal Pal Chaudhuri. CA-BIST for asynchronous circuits: A case study on the RAPPID asynchronous instruction length decoder. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 62–72. IEEE Computer Society Press, April 2000.

[SAM+04] G. Semeraro, D. H. Albonesi, G. Magklis, M. L. Scott, S. G. Dropsho, and S. Dwarkadas. Hiding synchronization delays in GALS processor microarchitecture. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 159–169. IEEE Computer Society Press, April 2004.

[SF01] Ivan Sutherland and Scott Fairbanks. GasP: A minimal FIFO control. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 46–53. IEEE Computer Society Press, March 2001.

[SM00] Allen E. Sjogren and Chris J. Myers. Interfacing synchronous and asynchronous modules within a high-speed pipeline. *IEEE Transactions on VLSI Systems*, Vol. 8, No. 5, pp. 573–583, October 2000.

[Sut89] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, Vol. 32, No. 6, pp. 720–738, June 1989.

[TKI+97] Akihiro Takamura, Masashi Kuwako, Masashi Imai, Taro Fujii, Motokazu Ozawa, Izumi Fukasaku, Yoichiro Ueno, and Takashi Nanya. TITAC-2: An asynchronous 32-bit microprocessor based on scalable-delay-insensitive model. In *Proc. International Conf. Computer Design (ICCD)*, pp. 288–294, October 1997.

[Ung69] S. H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley-Interscience, John Wiley & Sons, Inc., New York, 1969.

[WDF+97] J. V. Woods, P. Day, S. B. Furber, J. D. Garside, N. C. Paver, and S. Temple. AMULET1: An asynchronous ARM processor. *IEEE Transactions on Computers*, Vol. 46, No. 4, pp. 385–398, April 1997.

[YD96] Kenneth Y. Yun and Ryan P. Donohue. Pausible clocking: A first step toward heterogeneous systems. In *Proc. International Conf. Computer Design (ICCD)*, pp. 118–123, October 1996.

論文目録

本研究に関する論文

公刊論文

石川 健一郎, 天野 英晴
非同期スイッチチップの設計と実装,
電子情報通信学会論文誌, 2003 Vol.J86-D-1, pp.711-720

石川 健一郎, 安達 義則, 天野 英晴
同期 Speculative Completion の提案と評価,
電子情報通信学会論文誌, 採録決定

その他の国際会議発表

K. Ishikawa, H. Amano
AN ASYNCHRONOUS SWITCHING FABRIC,
Asia and South Pacific Design Automation Conference,
Jan. 2002

K. Ishikawa, H. Amano
A SIMPLE ASYNCHRONOUS SWITCH,
Cool Chips,
Apr. 2002, pp. 163

K. Ishikawa, J. Maruyama, Y. Adachi, Hideharu Amano
CHRONOS:An Asynchronous CPU Core Using Speculative Completion,
Cool Chips,
Apr. 2004, pp. 79

研究会

石川 健一郎, 川上 大輔, 柴田 裕一郎, 天野 英晴
非同期スイッチの試作,
電子情報通信学会技術研究報告 VLD,
May. 2001, pp.15-22

石川 健一郎, 安達 義則, 丸山 淳一, 天野 英晴
非同期スーパースカラプロセッサにおける Speculative Completion の効果,
パルテノン研究会,
Dec. 2003, pp. 19-26

その他の論文

国際会議

Y. Adachi, K. Ishikawa, S. Tsutsumi, H. Amano
An implementation of the Rijdael on Async-WASMII,
IEEE Conference on Field-Programmable Technology,
Dec. 2003, pp. 44-51

その他の国際会議発表

Y. Adachi, K. Ishikawa, S. Tsutsumi, H. Amano
An implementation and evaluation of the Rijdael on Async-WASMII with PCA
Cool Chips,
Apl. 2003, pp. 83,

研究会

堤 聡, 石川 健一郎, 天野 英晴
PCA を用いた非同期 WASMII システム構築に向けての評価,
パルテノン研究会,
Dec. 2001, pp 22-30

堤 聡, 石川 健一郎, 天野 英晴
PCA を用いた汎用アクセラレータの検討,
電子情報通信学会技術研究報告 SLDM,
Nov. 2002, pp 121-126

安達 義則, 石川 健一郎, 堤 聡, 天野 英晴
非同期 WASMII 上での Rijndael の実装と評価,
パルテノン研究会,
Dec. 2002, pp.56-63

堤 聡, 石川 健一郎, 天野 英晴
PCA/プロセッサ混載システムにおけるアプリケーションモデルの提案,
パルテノン研究会,
Dec. 2002, pp. 8-15

丸山 淳一, 石川 健一郎, 安達 義則, 天野 英晴
Speculative Completion 用演算器の評価,
パルテノン研究会,
Jun. 2004 pp 71-76

天野 英晴, 安達 義則, 堤 聡,石川 健一郎
動的リコンフィギャラブルプロセッサにおける可変クロックの導入,
電子情報通信学会技術研究報告 VLD,
Jan. 2005, pp. 17-22