修士論文

Network-on-Chip におけるコアテストスケジューリング手法

平成 22 年度修了

三重大学大学院　工学研究科

博士前期課程　電気電子工学専攻

佐野　裕基

要旨

近年，半導体製造技術の進歩により LSI の高集積化が進み，システム全体が 1 チップ上に実現できるようになってきている. そのような大規模チップは SoC（System-on-Chip）と呼ばれている．しかし，SoC の大規模化により，機能ブロック（コア）間結合用のバスの配線遅延の影響からクロック数を上げることができない問題が出てきた．そこで効率のよい通信手段の SoC として，バス結合に代わりネットワーク結合によってコア間を結ぶ NoC（Network-on-Chip）の研究が行われている.

NoC の設計では，コアの既存設計を再利用するコアベース設計が行われている．そのため，製造の最終工程で行われるテストにおいて，コアごとに作成されたテストデータを用い，コアごとに独立してテストを行うコアテスト法が用いられる．また，従来はテストのために LSI 内部にテスト専用信号線を組み込んでいた．しかし，この方法はコスト高となるため，NoC においてはテスト専用信号線に代わり，NoC が通常の動作に使用する入出力ピン・ルータ・チャネル（NoC 内部ネットワーク）を利用するテスト法が研究されている．NoC 内部ネットワークを利用したコアテスト法においては，各コアのテスト開始時刻や，テストに使用するチャネル（テストルート）を指定するスケジュールの出来によってはテスト時間が増大してしまう（コアテストスケジューリング問題）．そのため，コアテストスケジューリング問題を解決し，最適なスケジュールを作成するスケジューリング手法を考案することが，テスト時間の短縮につながる．本論文では，NoC におけるコアテストスケジューリング手法を提案し評価実験を行う.

本論文のスケジューリングシステムは，NoC の回路情報と各コアのテストデータ量を入力することで，コアテストスケジュールを出力する．スケジューリングシステムでは，まず入力データからコアテスト優先順位決定手法に従いコアテスト優先順位を決定する．コアテスト優先順位とは，コア毎にコアテストの優先権を順位付けしたものである．このコアテスト優先順位を基に，スケジューラーによってコアテストスケジュールを作成する．この際，各コアのテストルートは，テストルート決定手法に従って決定される.

本論文では，スケジューリングシステム内の，コアテスト優先順位決定手法，および，テストルート決定手法の両方について提案を行う．コアテスト優先順位決定手法については，各コアのテストデータ量と周辺チャネル数からヒューリスティックに決定する手法と，多スタート局所探索法によって決定する手法の二つを提案する．テストルート決定手法については，チャネルのビット幅を活かすようにテストルートを決定する手法を，二つ提案する.

評価実験として，複数の NoC 回路に対して提案手法によるスケジューリングを行い，作成したスケジュールによる NoC のテスト時間の比較評価を行った．その結果，提案手法によってテスト時間を短縮することに成功し，提案手法の有用性が確認できた.

目次

図一覧

表一覧

第１章

はじめに

1.1　研究の背景

近年，半導体技術の進歩による LSI の高集積化が進み，システム全体が１チップ上に実現できるようになってきている．そのような大規模チップは，複数の IP コアを連結することで構成され，SoC(System-on-Chip)と呼ばれている．本論文において，IP コア（以後単にコア）とは，機能ブロック単位でまとめられた回路情報，および回路そのものを指す．

しかし，SoC の大規模化により，コア間結合用のバスの配線遅延の影響からクロック数を上げることができない問題が出てきた．そこで効率のよい通信手段の SoC として，バス結合に代わりネットワーク結合によってコア間を結ぶ NoC(Network-on-Chip)の研究が行われている[1][2].

NoC のような１システム全体が組み込まれた大規模 LSI を一つのメーカーがゼロからすべて設計することは，１メーカーが有する設計技術の限界及び市場投入までに許される開発期間などの問題から，困難な場合が多い．そのため，設計の再利用を行うために，コアが利用されている．NoC を設計するメーカー（システムインテグレータ）は，様々なコアプロバイダからコアを購入し，購入したコアと自社で設計したコアを組み合わせて NoC を構成する（コアベース設計）．これにより，自社が持たない技術を使った NoC の設計を比較的容易に行うことができ，設計コストの削減も可能となる．また，コアベース設計は市場の要求の変化に対して素早い対応が可能となる．

コアベース設計された NoC は製造時検査（以後単にテスト）方法にも影響を及ぼす．コアベース設計ではなく，NoC 全体を自社ですべて設計した場合，NoC の回路情報がすべて分かっているため，NoC に対するテスト用データ（テストデータ）を生成・作成することができる．しかし，コアベース設計では，特に，コアが回路内部情報の隠されたハードコアで提供される場合には，システムインテグレータは，そのコア部分のテストデータを生成・作成することができない．そのためコアプロバイダからは，コアの設計回路と共にそのコア部分のテ

ストデータも供給されることになり，システムインテグレータはそのテストデータをそのコア部分のテストに使うことになる．また，内部回路情報が開示されるソフトコアの場合でも，回路の巨大化や複雑化に伴い LSI 全体のテストデータをまとめて生成・作成することが困難になりつつあり，コアごとに分割してテストデータを生成・作成し，これを用いてコアごとに独立してテストを行う傾向がある．以上のような，コアごとにテストデータを用意し，NoC 内のコア毎に独立してテストを行う方法をコアテスト法と呼ぶ[3].

近年，NoC の大規模化に伴い，NoC のテストに要する時間・コストは増大しつつあり，テストの効率化が求められるようになってきた．近い将来，テストに要するコストは，LSI の製造コストよりも大きくなると予想されており，今後ますますテストの効率が重要となる．

1.2 研究の目的

従来，LSI はテスト時のテストデータ・テスト結果の伝送に用いるテスト専用の信号線が，内部に組み込まれていた．しかし，この方法ではテスト専用信号線を組み込むために NoC の面積を大きくする必要があり，チップコストが上がる．そのため NoC ではテスト専用信号線に代わり，NoC が通常の動作に使用する NoC 内部ネットワーク（入出力ピン・ルータ・チャネル）を利用するテスト法が研究されている．

このテスト法では，コアテストスケジューリング問題を解き各コアのテストを効率良く行うことが，テスト時間の短縮につながる．コアテストスケジューリング問題の詳細は，第 3 章で説明する．この問題はネットワークトポロジーが規則的であるホモジニアス結合型 NoC については研究されている[4]．しかし，ネットワークトポロジーが不規則なヘテロジニアス結合型 NoC については，あまり研究が行われていない．実際の NoC はホモジニアス結合型ばかりでなくヘテロジニアス結合型も数多くあるため，ヘテロジニアス結合型についてもコアテストスケジューリング問題を研究する必要がある．

そのため本研究では，ヘテロジニアス結合型 NoC を対象としたコアテストスケジューリング問題を解くアルゴリズムを提案する．

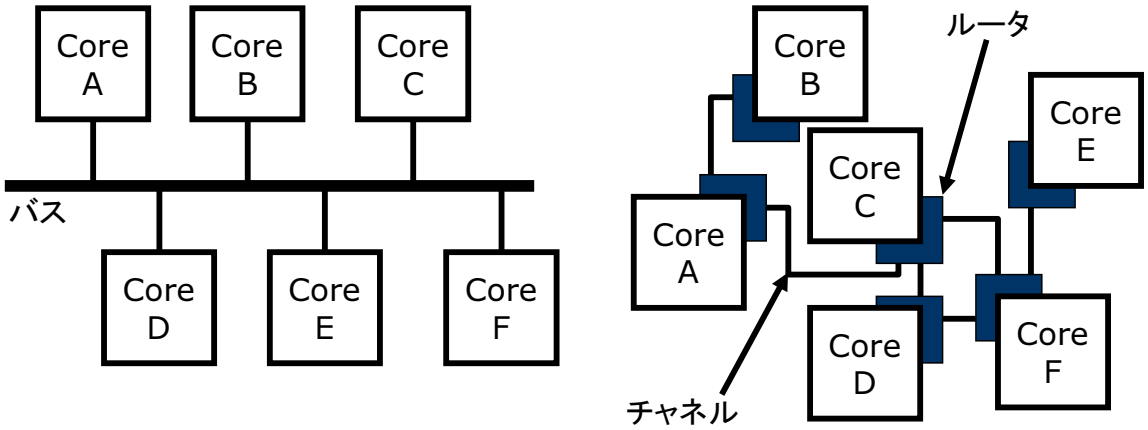
第２章

Network-on-Chip とテスト法

本章では，Networ-on-Chip の構成，設計法，本研究で想定しているテスト手法について説明する．

2.1　System-on-Chip と Network-on-Chip

System-on-Chip(SoC)はシステム全体を１チップ内に組み込んだ LSI である．SoC は, LSI 内のトランジスタや配線の微細加工技術の発達が主な要因となり，大規模・高集積化が進んでいる．しかし，従来のバス通信では，システムの微細化によってバスを共有するコア数が増加しても，配線遅延の影響からクロック数を上げることができないという問題がある．そのため，効率のよい通信手法としてコア間にバスの代わりにルータ（交換器）を配置する Network-on-Chip(NoC)が考えられている. 図１に従来のバス結合型 SoC と NoC の構成を示す.



(a)従来のバス結合型 SoC　　(b)NoC

図１　従来のバス結合型 SoC と NoC の構成

NoC は内部にルータを配置し，コア間ネットワークを構築することで効率的な通信を実現している. NoC におけるルータの一般的な内部構成を図 2 に示す.

このように複数の入出力ポートがそれぞれレジスタを持つことで，ルータは入力ポートが受信したデータを，スイッチによっていずれかの出力ポートから送出する．また，NoC は一般的にコアごとにルータが配置されているためルータ間をつなぐ個々のチャネルを短くでき，従来の SoC とは異なりクロック数を上げることができる．さらに，内部のレジスタが伝送されてきたデータを一時的に保持することでデータの中継を行い，コア間のデータ通信をパイプライン的に処理することができる．これらの理由により，NoC はコア間での高いデータ伝送処理能力を実現できる．

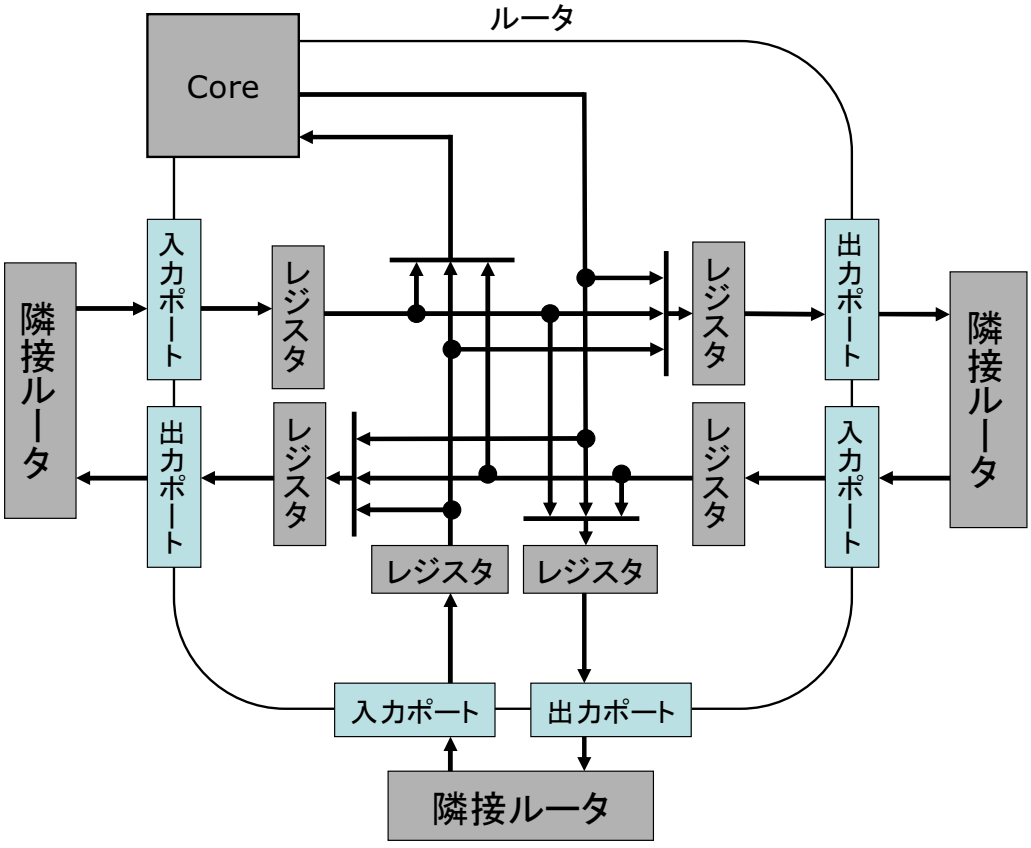


図 2　NoC におけるルータの内部構成

2.2　NoC におけるコアベース設計

NoC のような 1 システム全体が組み込まれた大規模 LSI を一つのメーカーがゼロからすべて設計することは，１メーカーが有する設計技術の限界及び市場

投入までに許される開発期間などの問題から，困難な場合が多い．そのため，NoC の設計にはコアベース設計が用いられることが多い．NoC を設計するメーカー（システムインテグレータ）は，様々なコアプロバイダからコアを購入し，購入したコアと自社で設計したコアを組み合わせて NoC を構成する（図 3）．

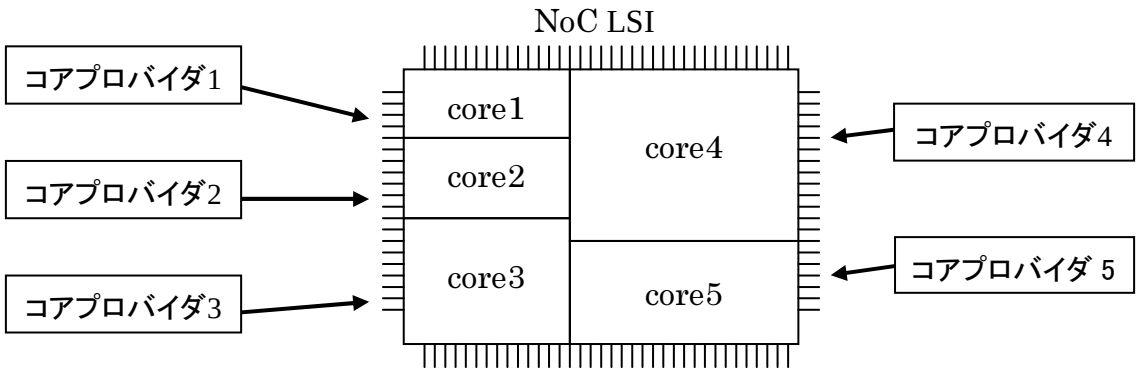


図 3　NoC のコアベース設計例

2.3　コアテスト法

コアベース設計された NoC，特に，コアが回路内部情報の隠されたハードコアで提供された場合には，システムインテグレータは，そのコア部分のテストデータを生成・作成することができない．一方，コアプロバイダはコアをゼロから設計しているため，回路内部をすべて把握しており，テストデータ生成用ソフト等を用いてテストデータを作成できる（図 4）．そのため，システムインテグレータは，コアプロバイダからコアの設計回路と共にそのコア部分のテストデータも供給され，そのテストデータをそのコア部分のテストに使用することになる．このようにして，システムインテグレータは各コアのテストデータを集め，NoC 全体のテストデータを用意する．NoC の製造時検査では，このテストデータを用いて各コアについてテストを行う（図 5）．コア間を結合するバスやネットワークなどの回路は別途テストする必要があるが，この部分のテストについては本研究の対象外とする．本研究では対象外としたが，この部分のテスト手法についても，研究が行われている[5].

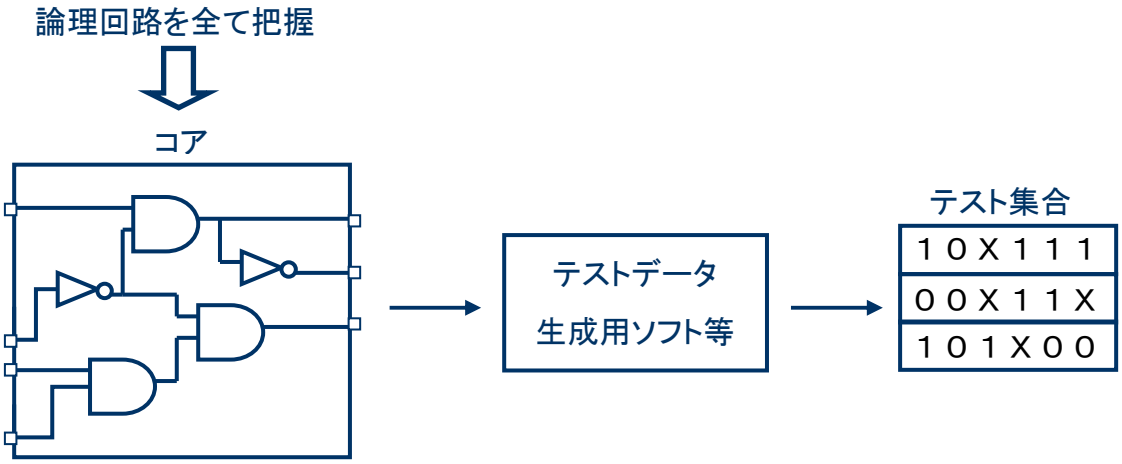


図 4　コアプロバイダでのテストデータ生成

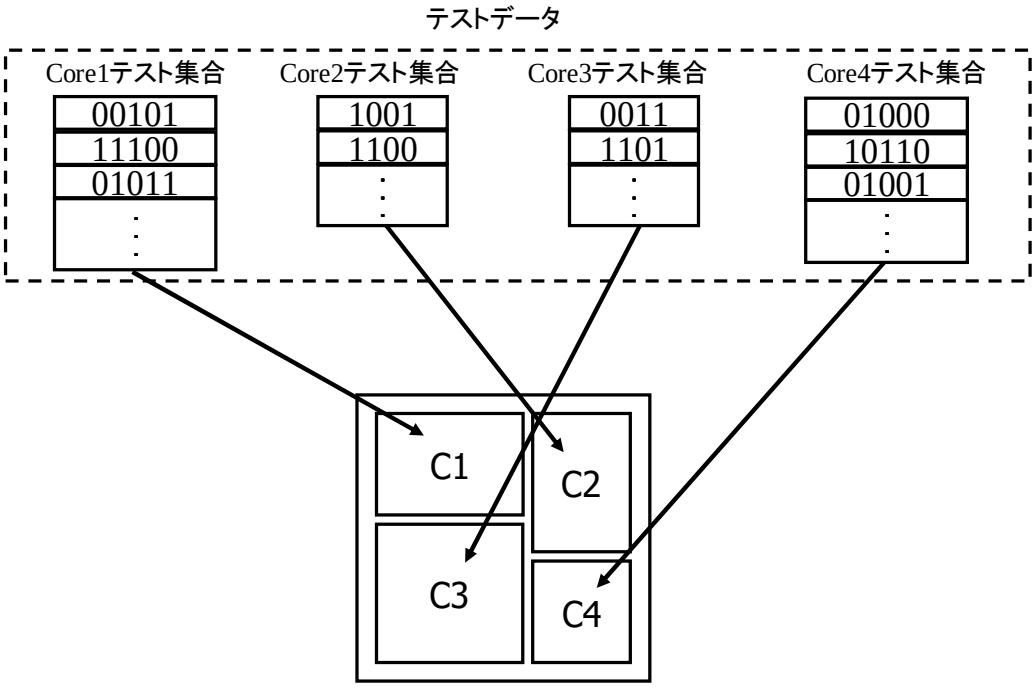


図 5　NoC のテスト

2.4　コアにおけるスキャンテスト

各コアは組み合わせ回路部と記憶素子である FF（Flip Flop）によって構成されている．本研究では，FF を連結したスキャンチェインと呼ばれる回路部を搭載しているコアを対象とする．スキャンチェインは外部から FF の値を任意の値に変更することが可能なシフトレジスタにより構成された回路である．スキャンテストのアーキテクチャを図 6 に示す．

スキャンテストのテスト手順について説明する．組み合わせ回路用のテストベクトルをスキャンチェインに 1 ビットずつ入力していき，一つのテストベクトルがすべてスキャンチェインに格納された後，被検査回路である組み合わせ回路を動作させ，テスト結果をスキャンチェインに格納する．その後，次のテストベクトルの入力と同時にテスト結果を外部に出力し，出力されたテスト結果が予定していた正しいテスト結果と一致しているかを確認することで，故障の有無を判断する．

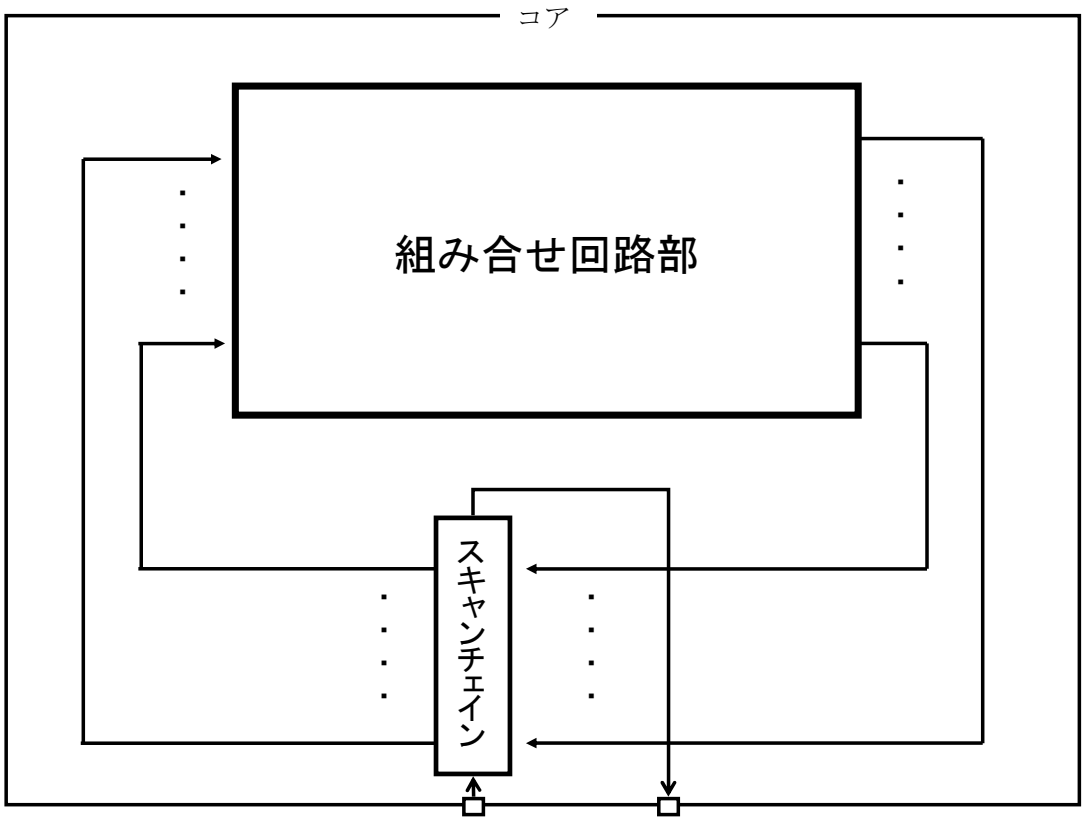


図 6　スキャンテスト

近年のコアは，スキャンチェインを多数本搭載したマルチスキャンチェインコアが多くなっている．図 7 に，マルチスキャンチェインを 2 本搭載したマルチスキャンチェインコアの例を示す．

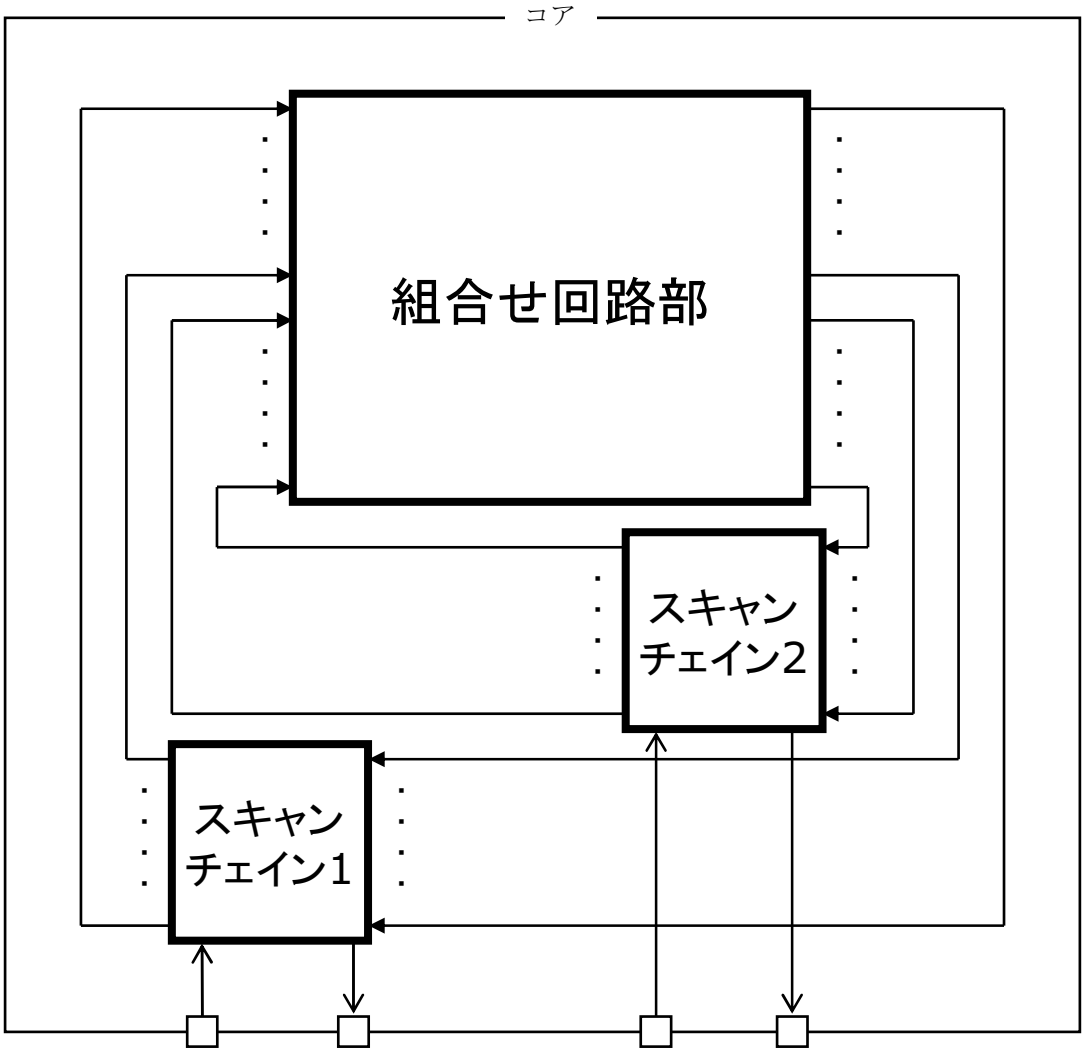


図 7　マルチスキャンチェインコア

2.5　NoC 内部ネットワークの利用

従来のコアテスト法では，各コアのテストデータとその結果を，図 8 のようにテスト用に設けた信号線で入力，伝送，出力していた．しかし，この方法ではテスト専用信号線を組み込むために NoC の面積を大きくする必要があり，LSI 上の配線コストが高くなるという問題が生じる．

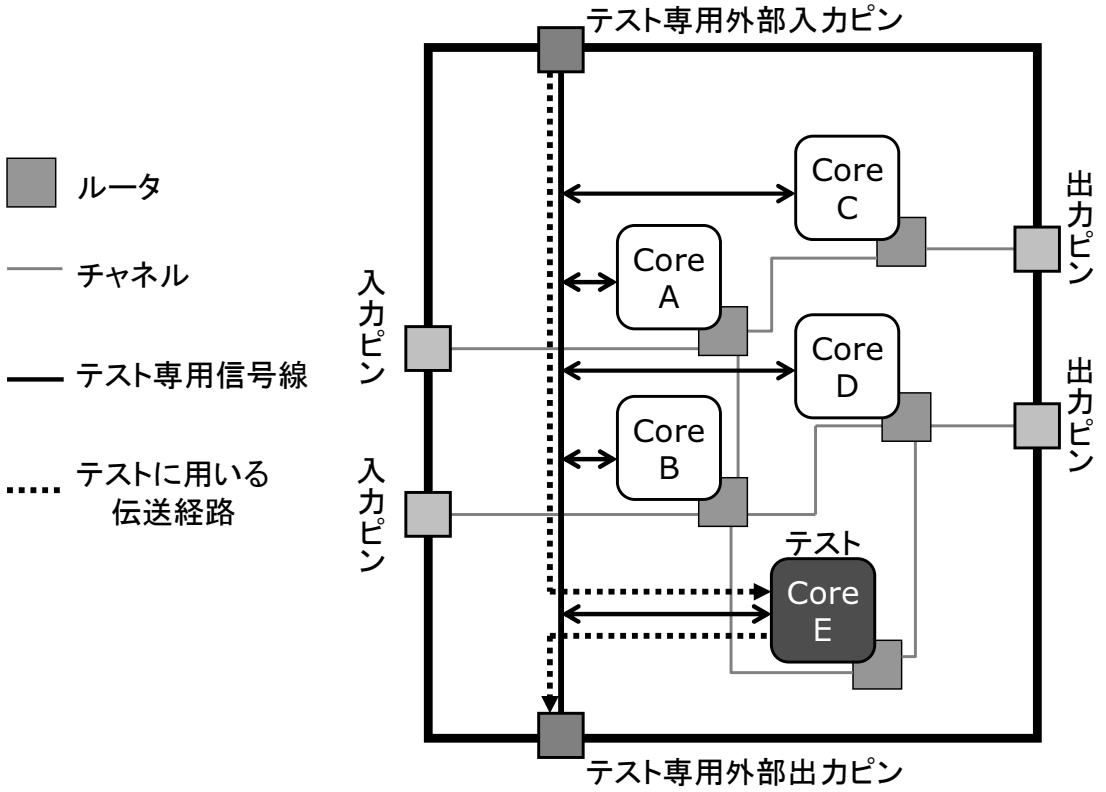


図 8　テスト専用信号線を組み込んだ NoC

この問題を解決するため，テスト専用信号線に代わり，NoC 内部ネットワークをテストに利用する方法がある．この方法では，テストデータは入力ピンから入力され，テスト結果は出力ピンから出力される．また，テストデータとテスト結果の伝送には，ルータとチャネルを用いる．この方法の例を図 9 に示す．

テストデータの入力・伝送・出力という一連の動作はパイプライン的に実行される．テストデータとその結果の伝送に使用するチャネルの組み合わせをテストルートと呼ぶ．本研究では，一つのコアのテストが開始してから終了するまで，テストルートは変更しないことを仮定する．これは，テストルートを変更するための機能をテスタに持たせるとテスタのコストが上がることから，テストルートを変更せず効率的にテストを行う手法が期待されているためである．

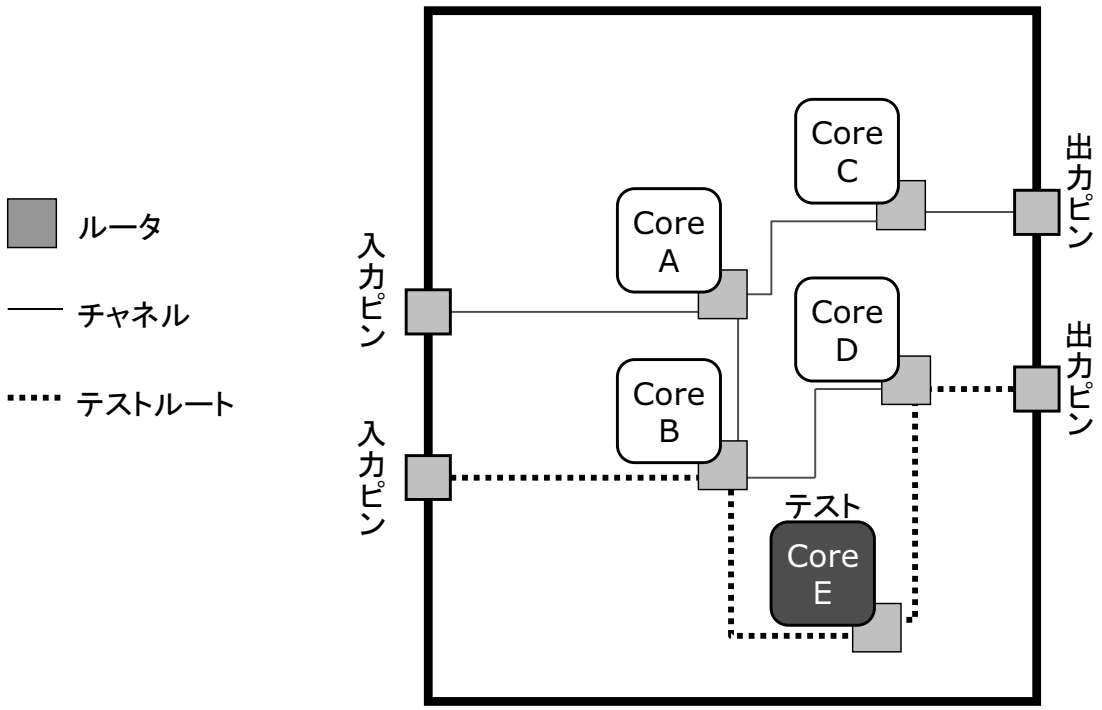


図 9　NoC 内部ネットワークを利用したテスト

第 3 章

NoC のコアテストにおけるスケジューリング

本章では，NoC のコアテストにおけるスケジューリング問題，本研究で構築したスケジューリングシステムの概要，本研究で対象としている NoC のネットワーク構成について説明する．

3.1　NoC のコアテストにおけるスケジューリング問題

NoC 内部ネットワークを利用したコアテスト法では，テストするコアの順序や，各コアのテストに使用するチャネル（テストルート）によっては，テスト時間が増大してしまう．このことを，図 10 に示す NoC に対して，図 11 に示すテストデータを用いてコアテストを行う場合を例にして，具体的に説明する．

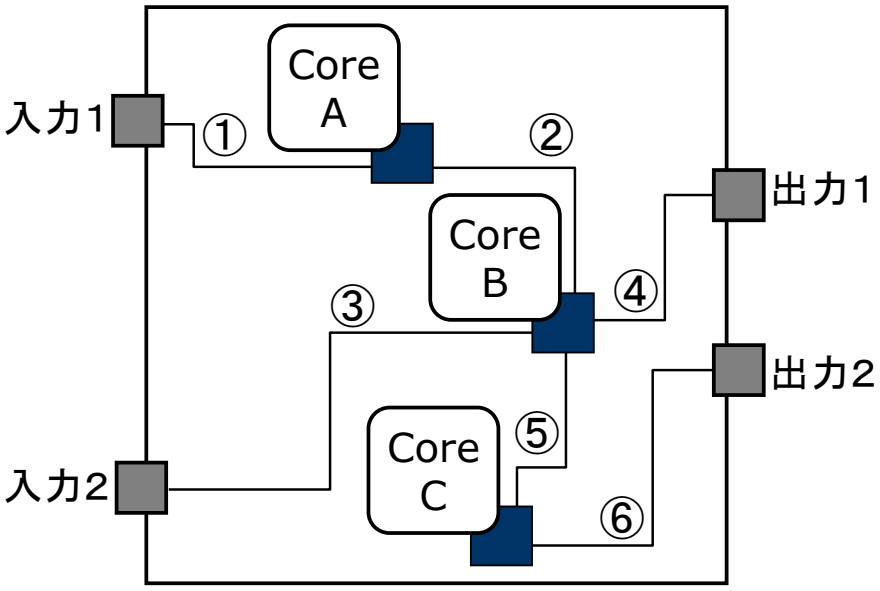


図 10　NoC の構成例 1

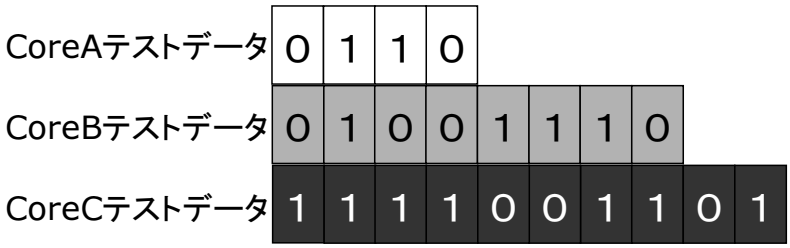


図 11　各コアのテストデータ例

ここでは，NoC のチャネルのビット幅は全て 1 ビットとし，テストルートは単純に，そのコアをテストする時点で使用可能なチャネルから，使用チャネル数が最小になるように決定するものとする．

まず，テストするコアの順序をコア A，コア B，コア C の順に設定した簡易スケジュール 1 を考える．この場合，図 12 のように最初に入力 1 へコア A テストデータ，入力 2 へコア B テストデータをそれぞれ入力し，コア A のテスト（テストルート：チャネル①，②，④），コア B のテスト（テストルート：チャネル③，⑤，⑥）を行う．

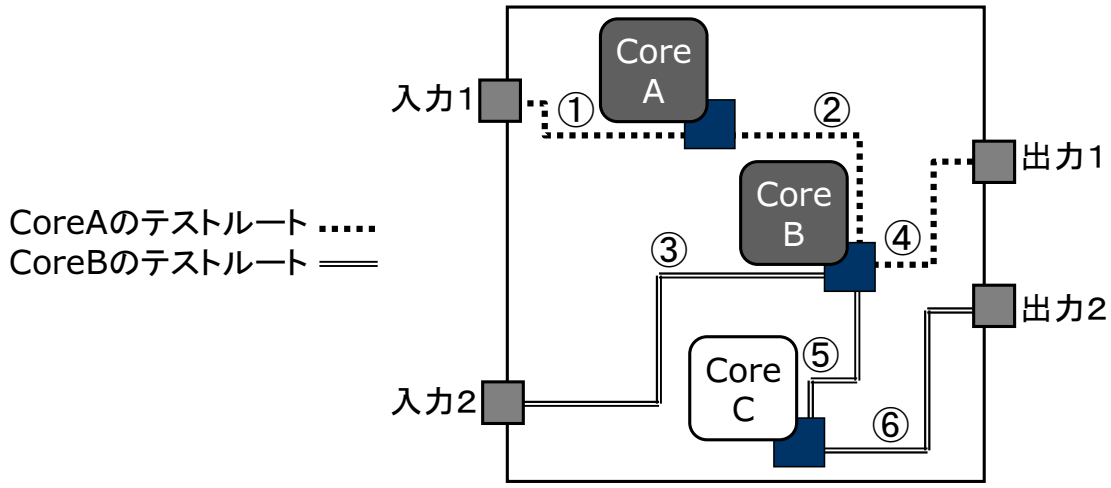


図 12　簡易スケジュール 1 のチャネル使用状況 1

図 13 に示すように 4 クロック後，コア A のテストは終了する．その直後，入力 1 を用いてすぐにコア C のテストを行いたいが，テストできない．これは，図 14 のようにコア C のテストに必要なチャネル⑤，⑥がコア B のテストで使

用されているためである．そのため，コア B のテスト終了後にコア C のテストを行う．このことから，簡易スケジュール 1 のテストイメージは図 15 のようになる．

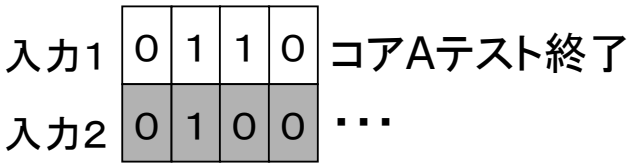


図 13　簡易スケジュール 1 のテスト経過イメージ

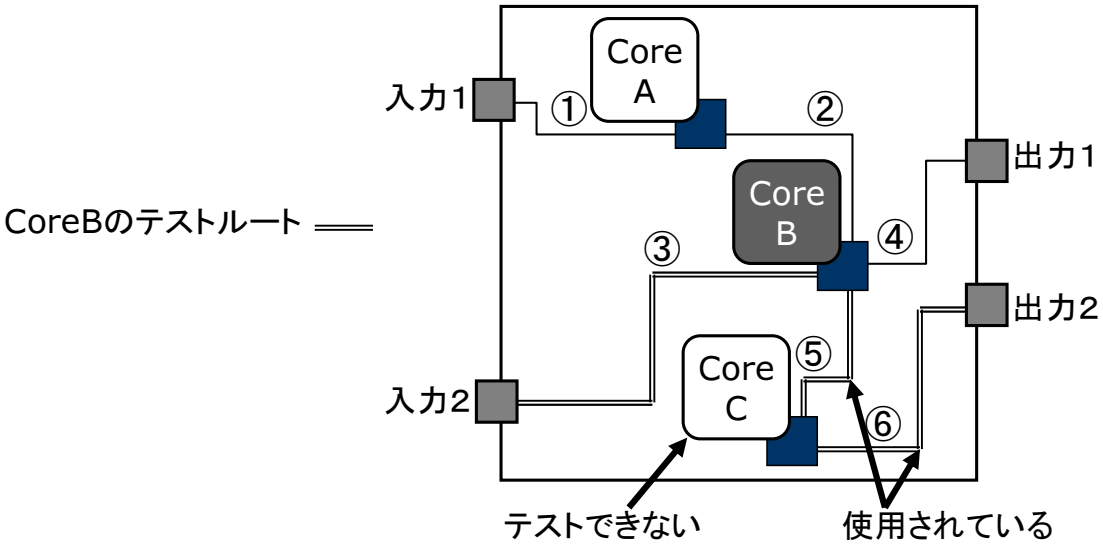


図 14　簡易スケジュール 1 のチャネル使用状況 2

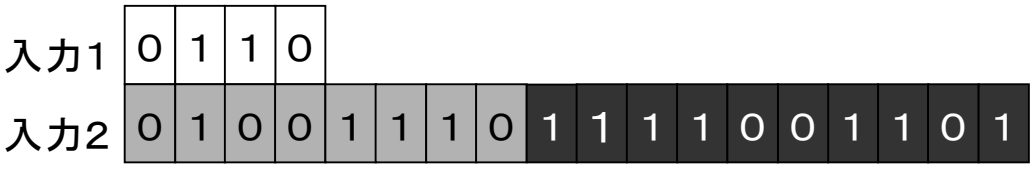


図 15　簡易スケジュール 1 のテストイメージ

次に，テストするコアの順序をコア A，コア C，コア B の順に設定した簡易スケジュール 2 を考える．この場合，図 16 のように最初に入力 1 へコア A テストデータ，入力 2 へコア C テストデータをそれぞれ入力し，コア A のテスト（テストルート：チャネル①，②，④），コア C のテスト（テストルート：チャネル③，⑤，⑥）を行う．

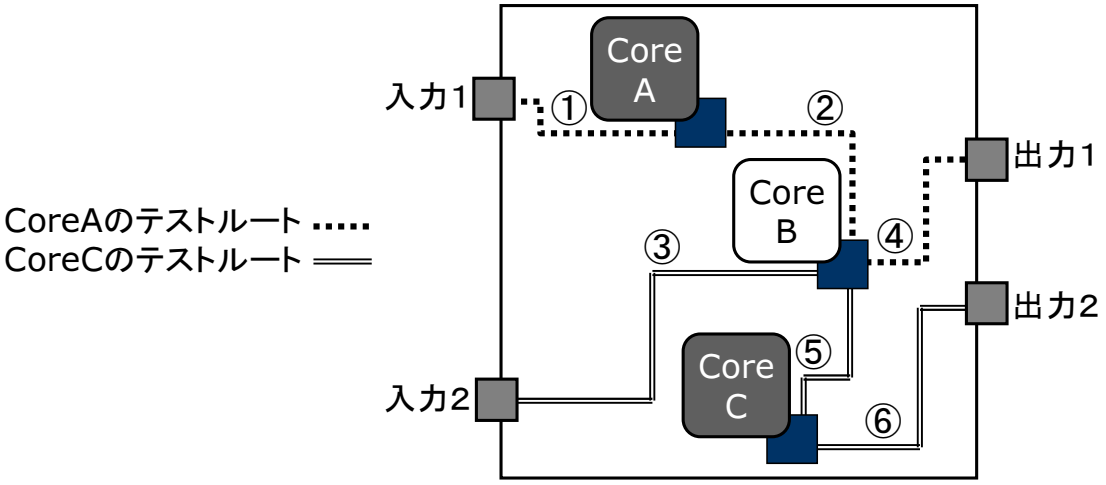


図 16　簡易スケジュール 2 のチャネル使用状況 1

図 17 に示すように 4 クロック後，コア A のテストは終了する．その直後，入力 1 を用いてすぐにコア B のテスト（テストルート：チャネル①，②，④）を行うことができる．これは，図 18 からも明らかである．このことから，簡易スケジュール 2 のテストイメージは図 19 のようになる．

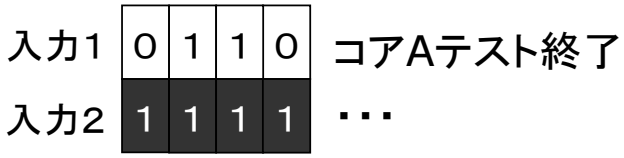


図 17　簡易スケジュール 2 のテスト経過イメージ

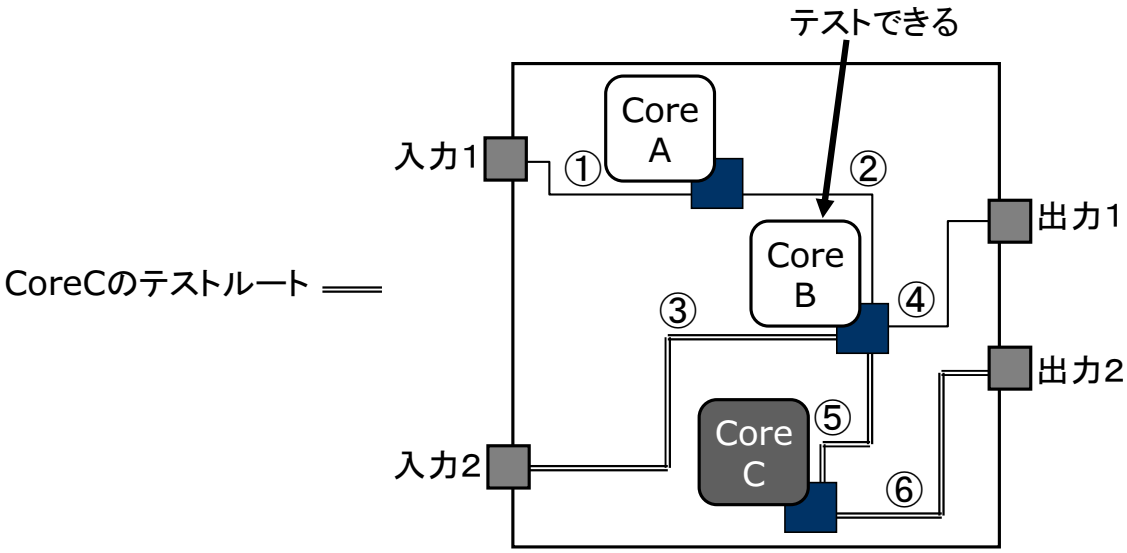


図 18　簡易スケジュール 2 のチャネル使用状況 2

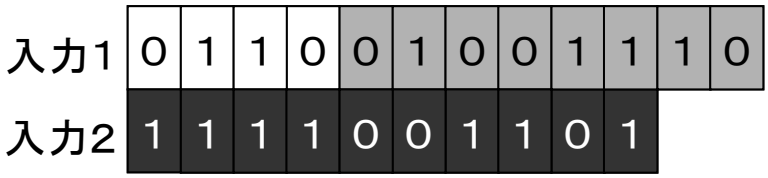
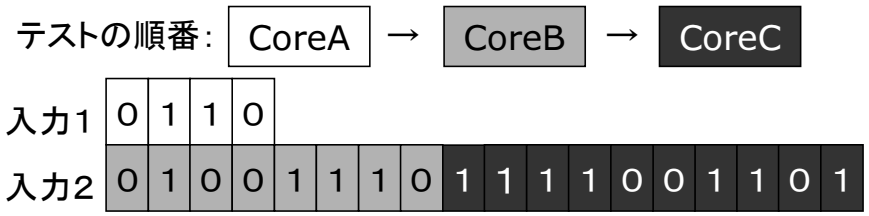
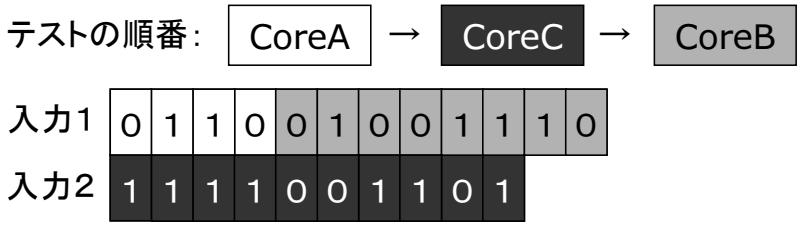


図 19　簡易スケジュール 2 のテストイメージ

図 20 は，簡易スケジュール 1 と簡易スケジュール 2 を比較したものである．この図から，スケジュール（この例ではコアをテストする順番）を変えるだけで，NoC 全体のテスト時間が大きく変わることがわかる．



(a)　簡易スケジュール 1 とテストイメージ



(b)　簡易スケジュール 2 とテストイメージ

図 20　簡易スケジュール 1 と 2 の比較

以上のことからわかるように，スケジュール次第でテスト時間は大きく変化する．そのため，テスト時間を短くするためには，最適なスケジュールを求める必要がある．このような問題をコアテストスケジューリング問題と呼ぶ．テスト時間の短縮には，コアテストスケジューリング問題を解決し，最適なスケジュールを作成するスケジューリング手法を考案することが重要である．

3.2　本研究のスケジューリングシステム

本研究のスケジューリングシステムの基本的な処理工程を図 21 に示す．このシステムは，NoC の回路情報と各コアのテストデータ量を入力とし，コアテストスケジュールを出力とする．コアテストスケジュールは，各コアのテストルート，テスト期間（開始時刻・終了時刻）から構成される．

スケジューリングシステムでは，まず入力データからコアテスト優先順位決定手法に従ってコアテスト優先順位を決定する．コアテスト優先順位とは，コア毎にコアテストの優先権を順位付けしたものである．このコアテスト優先順位に沿って，スケジューラーがコアテストスケジュールを作成する．この際，各コアのテストルートは，テストルート決定手法に従って決定される．

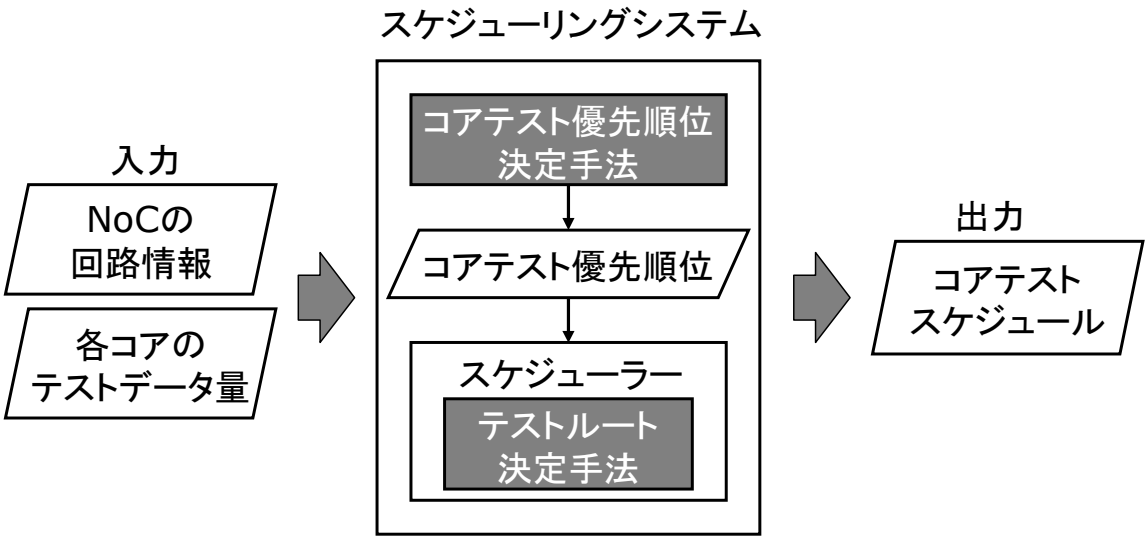


図 21　本研究のスケジューリングシステムの基本処理工程

3.2.1　本研究で用いるスケジューラー

本研究で用いるスケジューラーの役割は，各コアテストのテストルート，テスト期間（開始時刻・終了時刻）を決定することである．スケジューラーは，時系列に沿ってスケジュールを作成していく形式を採っており，スケジュール上のその時々における，全ての入力ピン・出力ピン・チャネルの使用状況と現在時刻を保持している．各コアテストのスケジュールは，コアテスト優先順位が高いものから優先的に作成される．ただし，その時点でそのコアテストのテストルートが作成不可能な場合，そのコアテストのスケジューリングは後回しにされる．また，各コアのテストルートは，その時点で使用されていないチャネルの中から，テストルート決定手法に従って決定される．以下で，図 22 に示すようなスケジューラーの処理の流れを，具体例を用いて説明する．

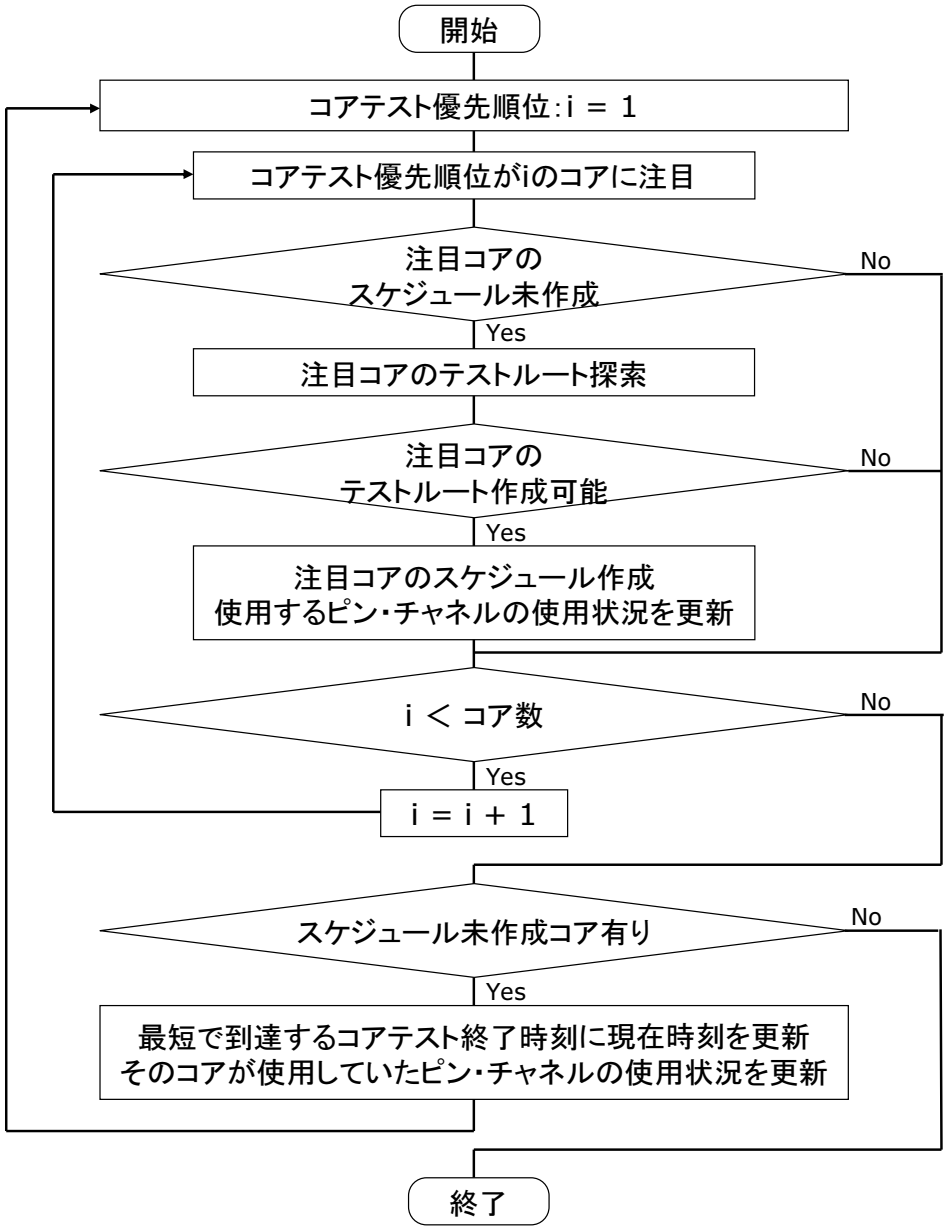


図 22　スケジューラーの処理の流れ

図 23 に示す NoC に対して表 1 に示すテストデータ量を用いてスケジューリングを行う場合を例にして，スケジュール作成工程を具体的に説明する．ここでは，コアテスト優先順位は 1 位：コア A，2 位：コア B，3 位：コア C と与えられ，テストルート決定手法は「そのコアをスケジューリングする時点で使用可能なチャネルの中から，使用チャネル数が最少になるように決定」（以後このテストルート決定手法を単純法と呼ぶ）に設定したと仮定する．また，NoC のチャネルのビット幅は全て 1 ビットとする．

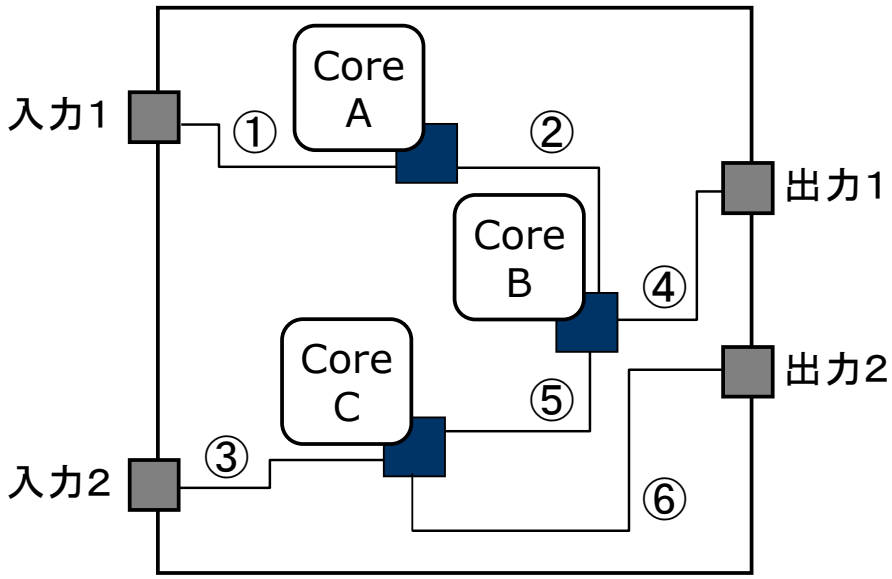


図 23　NoC の構成例 2

表 1　各コアのテストデータ量例

	テストデータ量[bit]
CoreA	21,014
CoreB	14,388
CoreC	34,740

まず，コア A のスケジュールを作成する．この時点でチャネルは全て使用可能であるため，テストルート決定手法（単純法）からテストルートはチャネル①，②，④と決定される．また，スケジュール作成開始時におけるスケジュール上の現在時刻は 0 であるため，コア A のテスト開始時刻は 0 となる．テスト終了時刻は，テスト開始時刻からテストデータ量クロック分経過した時刻になる．そのため，表 1 のテストデータ量から，テスト終了時刻は 21014 となる．以上からコア A のスケジュールは表 2 のように作成される．

表 2　スケジュール作成状況 1

	テストルート（使用チャネル）	テスト期間	
		開始時刻	終了時刻
CoreA	①, ②, ④	0	21,014
CoreB			
CoreC			

次に，コア B のスケジュールを作成したいが，図 24 からわかるようにコア B のテストルートは作成不可能なため，コア B のスケジュールは作成できない．そのため，コア B のスケジュール作成は後回しにし，コア C のスケジュールを作成する．図 24 からわかるようにコア C のテストルートは作成可能なため，コア C のスケジュールは作成することができる．テストルートはチャネル③，⑥と決定される．また，コア A と同様にテスト開始時刻は 0 となり，表 1 のテストデータ量からテスト終了時刻は 34740 となる．そのためコア C のスケジュールは表 3 のように決定される．

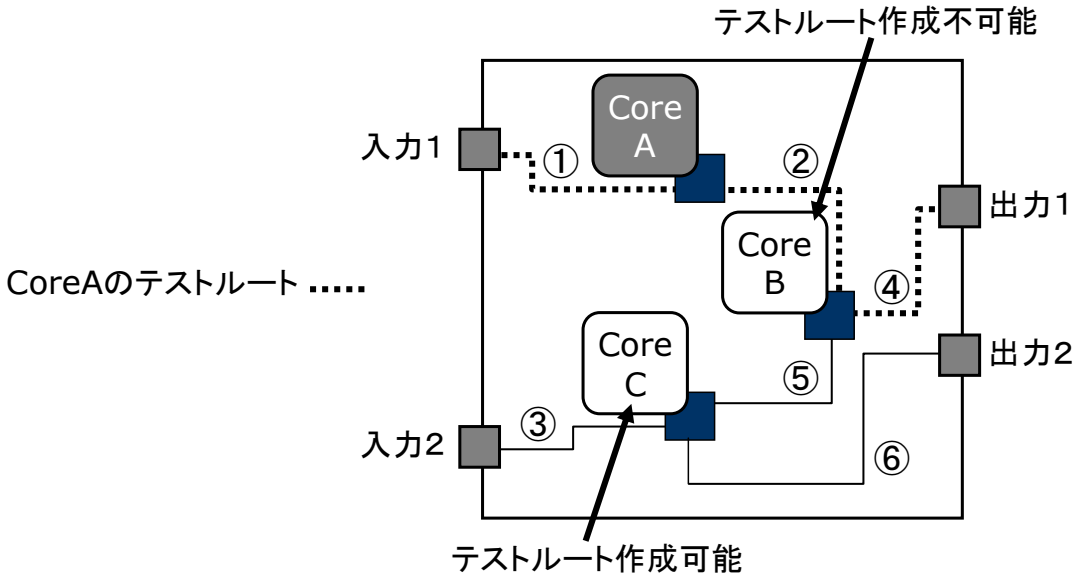


図 24　スケジューラーの保持するチャネル使用状況 1

表 3　スケジュール作成状況 2

	テストルート（使用チャネル）	テスト期間	
		開始時刻	終了時刻
CoreA	①, ②, ④	0	21,014
CoreB			
CoreC	③, ⑥	0	34,740

この時点で図 25 に示すように, コア B のテストルートは作成不可能であるため，コア B のスケジュールは作成できない．そのため，現在時刻をテストが先に終了するコア A のテスト終了時刻である 21014 に更新し，コア A の使用チャネル①，②，④を使用可能状態にする．このチャネル①，②，④をコア B のテストルートとすることで，コア B のスケジュールを作成する．テスト開始時刻は 21014 となり，テスト終了時刻は，表 1 のテストデータ量分の時間経過を考え 35402 となる．このようにしてコア B のスケジュールは決定され，表 4 のように NoC 全体のスケジュールが完成する．

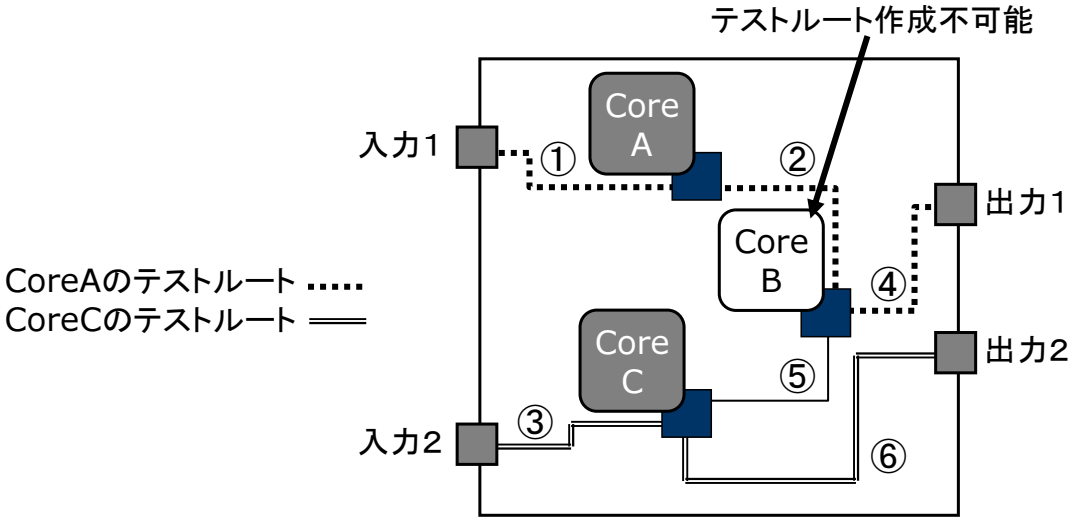


図 25　スケジューラーの保持するチャネル使用状況 2

表 4　スケジュール作成状況 3

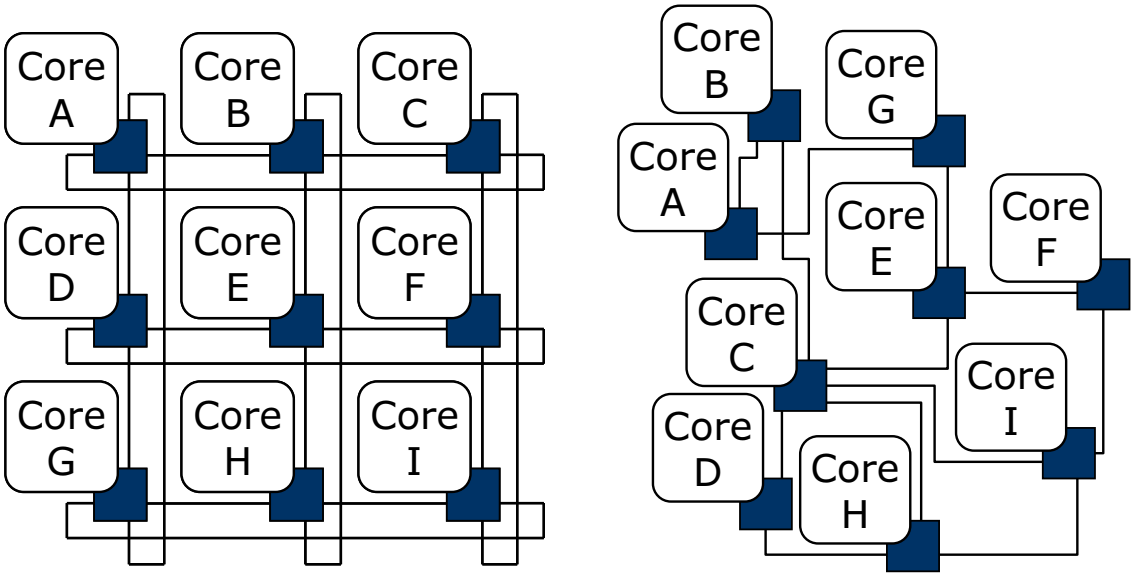
	テストルート（使用チャネル）	テスト期間	
		開始時刻	終了時刻
CoreA	①, ②, ④	0	21,014
CoreB	①, ②, ④	21,014	35,402
CoreC	③, ⑥	0	34,740

3.2.2　本研究のスケジューリングシステムにおける提案

本研究ではスケジューリングシステム内の，コアテスト優先順位を決定するための手法（コアテスト優先順位決定手法），テストルートを決定するための手法（テストルート決定手法）の二つの手法を提案する．この二つの手法次第で，出力されるスケジュールは大きく変化し，NoC 全体のテスト時間も大きな影響を受ける．コアテスト優先順位決定手法については，ヒューリスティックス法と探索法を提案・比較し，テストルート決定手法については，単純法とは異なる二種類のヒューリスティックス法を提案・比較する．コアテスト優先順位決定手法については第 4 章, テストルート決定手法については第 5 章で説明する．

3.3　本研究の研究対象

NoC 内部ネットワークの構成が図 26（a）に示す二次元トーラスのように規則的な形のものをホモジニアス結合型 NoC と呼び，図 26（b）のように不規則な形のものをヘテロジニアス結合型 NoC と呼ぶ[6].



(a)　ホモジニアス結合型(二次元トーラス)　　(b)　ヘテロジニアス結合型

図 26　NoC のネットワーク構成の違い

現在，ホモジニアス結合型 NoC を対象としたコアテストスケジューリング問題の研究は行われているが，ヘテロジニアス結合型 NoC を対象とした研究はあまり行われていない．しかし，実際の NoC はヘテロジニアス結合型も多くあるため，本研究ではヘテロジニアス結合型 NoC を対象とする．

第４章

コアテスト優先順位決定手法

コアテスト優先順位の順位付けパターン数は，NoC を構成するコア数の階乗になるため，全てのパターンを試すこと（全探索）は，コア数が多くなるほど計算時間の観点から困難になる．そのため，テスト時間が短いスケジュールを作成できるコアテスト優先順位を，全探索せずに決定する手法が必要である．

本章では，スケジューリングシステム内のコアテスト優先順位決定手法について，ヒューリスティックス法と探索法を提案し，実験により比較評価を行う．

4.1　ヒューリスティックス法

スケジューリングシステムに入力される NoC の回路情報や各コアのテストデータ量からコアテスト優先順位をヒューリスティックに決定する手法である[7].

この手法は，筆者の卒業論文に記述されているものである．詳細は，参考文献 7 で説明しているため，ここでは概要のみを記す.

ヒューリスティックス法では，(1) テストデータ量が多いコア，(2) コア周辺のチャネル数が少ないコアの，コアテスト優先順位を高くする．テストデータ量が多いコアの優先順位を高くする理由は，テストデータ量が多いコアを NoC のテスト終盤に残すと最後の帳尻が合わせられず，テスト時間が長くなると考えられるためである．コア周辺のチャネル数が少ないコアの優先順位を高くする理由は，そのようなコアが連結していた場合，その中のコアのどれか一つをテストすると，そのテスト中は他の連結したコアは全てテストできないため，このようなコア群がテスト終盤に残るとテスト時間が増大するためである．

4.2　探索法

探索法は，探索を繰り返すことでより良い解を見つけていき，最終的に優先順位を決定する手法である．本研究では，探索に多スタート局所探索法を用いる[8]. 多スタート局所探索法においては，コアテスト優先順位（リスト）を解，NoC 全体のテスト時間を評価値とし，近傍は挿入近傍を用いる．

4.2.1　コアテスト優先順位決定手法における多スタート局所探索法

コアテスト優先順位決定手法に多スタート局所探索法を用いた場合のスケジューリングシステムの処理工程を，図 27 に示し，以下で説明していく．

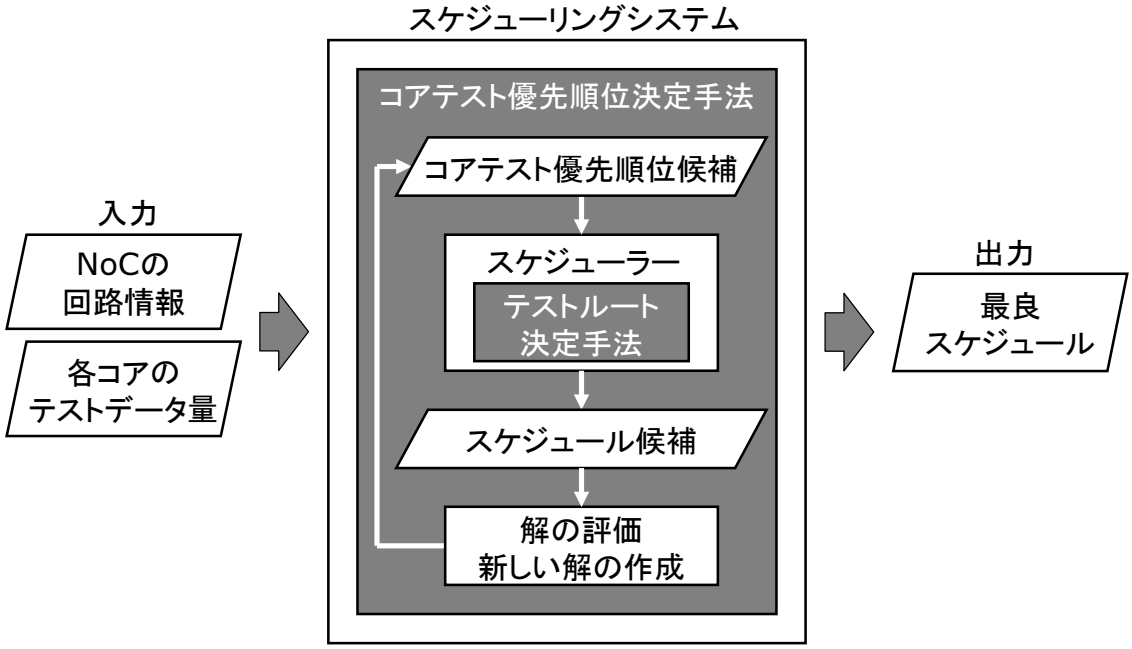


図 27　スケジューリングシステムの処理工程（多スタート局所探索法版）

① まず，入力データから NoC 内のコア数を把握する．その後，コアテスト優先順位（解）をランダムに設定する．この優先順位を基に，スケジューラーがスケジュールを作成する．スケジュールによって得られる，NoC 全体のテスト時間を評価値とする．

② 次に近傍操作によってコアテスト優先順位を少し変え，新しいコアテスト優先順位を生成する．今回の近傍操作で用いた近傍は挿入近傍である．本研究における挿入近傍は，図 28 に示すように，1 つのコアを別の優先順位に移し変え，その他のコアをずらすことにより得られる解集合である．

新しく生成したコアテスト優先順位を基に，スケジューラーがスケジュールを作成する．その後，新しく生成した優先順位による NoC 全体のテスト時間（評価値）を近傍操作前の優先順位によるテスト時間と比較する．新しく生成した優先順位によるテスト時間の方が短ければ，コアテスト優先順位を

新しいものに更新する．反対に，近傍操作前の優先順位によるテスト時間の方が短ければ，近傍操作前の優先順位を維持する．

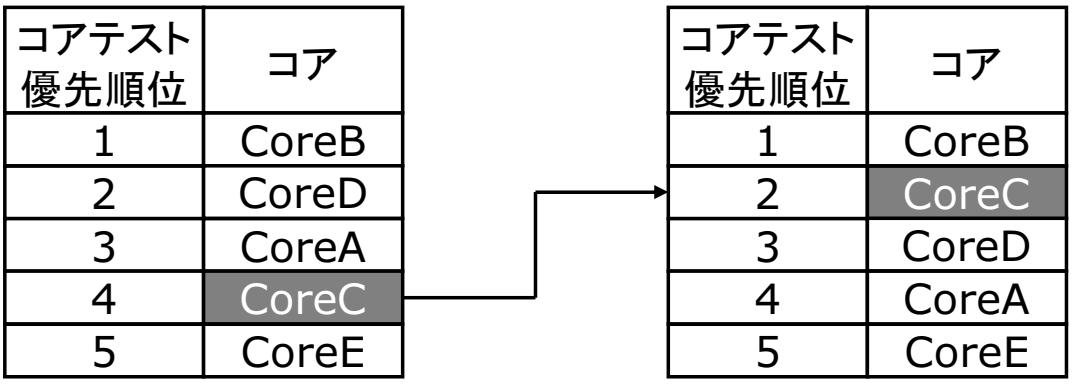


図 28　挿入近傍例

③　②の操作を，挿入近傍内の全てのコアテスト優先順位について実行し，近傍操作による新しいコアテスト優先順位の作成ができなくなった時点で，今回の初期解に対する探索を終了する．

④　③の後，これまでの各初期解に対する探索の合計近傍操作回数が設定回数未満であれば，再び①から同様の処理を行う．近傍操作回数が設定回数に到達していれば，全体の探索を終了する．

このようにして，多数のランダム初期解（コアテスト優先順位）に対して探索操作を行い，それらの中で，最もテスト時間が短いスケジュールの基になったコアテスト優先順位を，最終的なコアテスト優先順位として採用する．

4.3　コアテスト優先順位決定手法の評価実験

4.1，4.2 節で提案する手法をC言語で実装し，100 個の NoC 回路（コア数は 16 個，チャネルのビット幅は全て 1 ビット）に対してスケジュール作成実験を行った．各コアのテストデータ量は，表 5 のものを用いた．テストルート決定手法は単純法，すなわち，「そのコアをスケジューリングする時点で使用可能なチャネルの中から，使用チャネル数が最小になるように決定」する手法を用いた．

表 6，7，8，9 に実験結果を示す．ヒューリスティックス法・探索法の左側の数値は，スケジュールによって得られる NoC 全体のテスト時間[クロック数]を表している．探索法の右側は，ヒューリスティックス法のテスト時間を 100%としたときの，探索法のテスト時間の割合を示している．

表 5　実験で用いた各コアのテストデータ量（16 コア）

コア番号	テストデータ量[bit]
1	21,014
2	14,388
3	34,740
4	43,413
5	14,757
6	29,012
7	20,272
8	27,750
9	31,521
10	11,655
11	26,737
12	25,565
13	25,276
14	32,310
15	28,477
16	20,885

表 6　コアテスト優先順位決定手法の評価実験結果 1

回路No	ヒューリスティックス法	探索法	
1	118,103	108,399	91.78
2	77,647	70,150	90.34
3	90,291	74,927	82.98
4	108,748	83,209	76.52
5	73,316	66,690	90.96
6	131,044	89,795	68.52
7	78,600	73,132	93.04
8	74,638	69,400	92.98
9	75,897	70,150	92.43
10	81,672	81,672	100.00
11	79,932	70,150	87.76
12	84,024	76,878	91.50
13	91,502	75,407	82.41
14	103,212	97,464	94.43
15	83,209	69,602	83.65
16	98,743	69,634	70.52
17	87,146	70,150	80.50
18	89,760	75,486	84.10
19	106,448	79,171	74.38
20	83,424	78,153	93.68
21	113,089	92,891	82.14
22	86,352	70,137	81.22
23	77,761	73,187	94.12
24	81,319	70,137	86.25
25	83,823	70,615	84.24

表 7　コアテスト優先順位決定手法の評価実験結果 2

回路No	ヒューリスティックス法	探索法	
26	81,749	70,013	85.64
27	107,237	86,063	80.25
28	83,901	69,913	83.33
29	105,856	85,809	81.06
30	83,532	68,978	82.58
31	80,633	70,298	87.18
32	76,878	69,649	90.60
33	98,261	74,638	75.96
34	116,287	100,882	86.75
35	119,426	95,406	79.89
36	93,572	90,414	96.63
37	125,038	103,990	83.17
38	119,337	89,337	74.86
39	84,766	69,649	82.17
40	103,849	70,150	67.55
41	75,501	69,913	92.60
42	74,759	66,690	89.21
43	105,903	103,192	97.44
44	106,159	83,151	78.33
45	82,362	67,703	82.20
46	86,576	73,665	85.09
47	75,340	72,425	96.13
48	113,129	82,964	73.34
49	83,375	71,163	85.35
50	91,422	73,187	80.05

表８　コアテスト優先順位決定手法の評価実験結果３

回路No	ヒューリスティックス法	探索法	
51	105,038	72,188	68.73
52	96,426	80,052	83.02
53	81,012	70,142	86.58
54	76,026	76,026	100.00
55	102,064	85,563	83.83
56	89,574	72,425	80.85
57	162,124	150,422	92.78
58	93,900	83,393	88.81
59	90,778	75,372	83.03
60	78,153	69,913	89.46
61	83,151	66,770	80.30
62	126,969	110,024	86.65
63	78,815	73,187	92.86
64	84,102	69,913	83.13
65	115,302	105,355	91.37
66	84,229	78,345	93.01
67	85,563	67,703	79.13
68	95,199	68,978	72.46
69	89,466	71,163	79.54
70	81,749	75,407	92.24
71	103,247	83,534	80.91
72	86,063	70,615	82.05
73	74,849	69,913	93.41
74	90,826	78,578	86.51
75	96,906	76,021	78.45

表 9　コアテスト優先順位決定手法の評価実験結果 4

回路No	ヒューリスティックス法	探索法	
76	85,551	71,163	83.18
77	71,554	69,400	96.99
78	105,423	88,782	84.22
79	108,529	103,212	95.10
80	77,971	69,649	89.33
81	107,217	98,580	91.94
82	85,563	72,188	84.37
83	103,051	71,150	69.04
84	92,775	81,018	87.33
85	104,736	71,163	67.95
86	84,836	69,634	82.08
87	82,762	72,574	87.69
88	88,073	72,574	82.40
89	81,792	64,298	78.61
90	106,436	78,140	73.42
91	92,988	74,927	80.58
92	80,700	69,971	86.71
93	97,763	83,151	85.05
94	80,490	70,142	87.14
95	110,692	110,692	100.00
96	122,476	100,197	81.81
97	101,664	101,664	100.00
98	87,042	70,150	80.59
99	102,677	74,927	72.97
100	83,295	69,400	83.32

ヒューリスティックス法のテスト時間を 100%としたときの, 探索法のテスト時間の割合の 100 回路における平均値，最大値，最小値を表 10 にまとめた.

表 10　コアテスト優先順位決定手法の評価実験結果のまとめ

	探索法による テスト時間の割合[%]
平均値	84.89
最大値	100.00
最小値	67.55

4.4　考察

表 10 からもわかるように，以前に提案したヒューリスティックス法に比べ, 探索法は，平均で約 85%，最も短縮できた場合で 67.55%までテスト時間を短縮することができた．また，ヒューリスティックス法を用いた場合の計算時間は数ミリ秒であり，探索法を用いた場合の計算時間は数分であった．探索法を用いる場合，ヒューリスティックス法を用いる場合に比べ計算時間は長くなるが, この程度であれば NoC のスケジューリング解法としては問題にはならない．コア数が増加すると計算時間も長くなるが，現実的なコア数（数個～数十個）の範囲においては，許容範囲内に収まる．以上のことから，探索法はヒューリスティックス法よりも，コアテスト優先順位決定手法として有効な手法であると考えられる.

しかし，回路によっては，テスト時間を短縮することができなかった回路も存在した．これはネットワーク構成などの設計段階の物理的制約によって，テスト時間が決定してしまい，スケジューリングによってテスト時間を改善する余地がない回路ではないかと考えられる.

第 5 章

テストルート決定手法

テストルートのパターン数は，コアテスト優先順位の順位付けパターン数と同様で非常に多いため，全てのパターンを試すこと（全探索）は，計算時間の観点から困難になる．そのため，テスト時間が短いスケジュールを作成できるテストルートを，全探索せずに決定する手法が必要である．

本章では，スケジューリングシステム内のテストルート決定手法について，二つのヒューリスティックス法を提案し，実験により比較評価を行う．

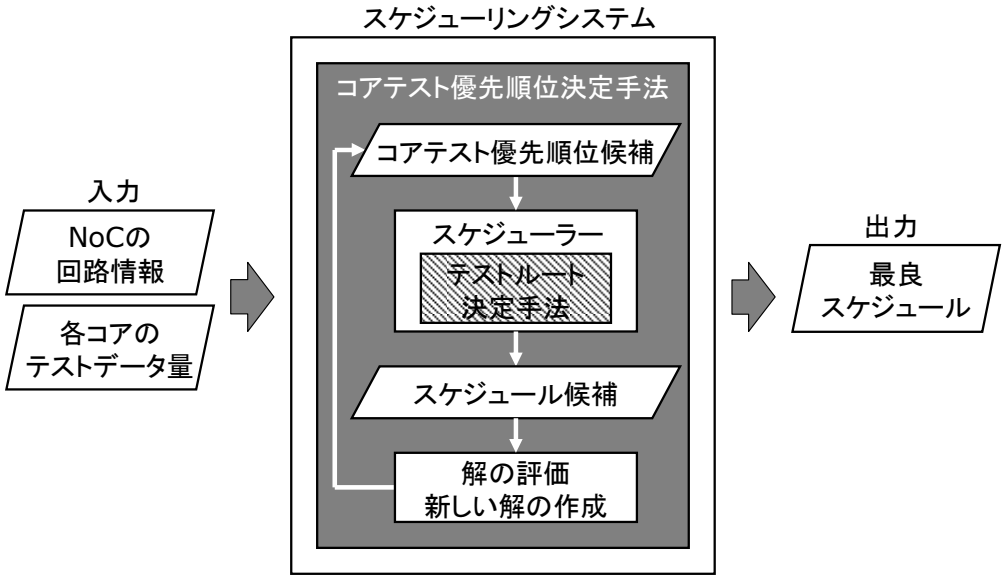


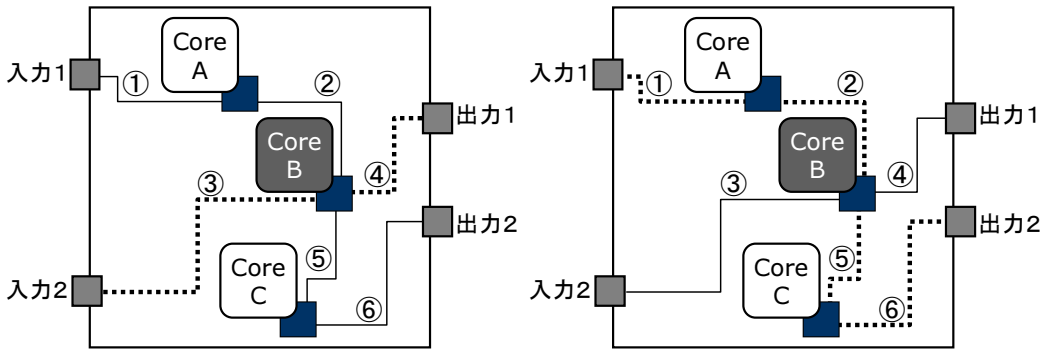
図 29　テストルート決定手法のスケジューリングシステム内での位置

5.1　チャネル使用数最小化法（単純法）

チャネル使用数最小化法は，第 3，4 章で用いた単純なテストルート決定手法である．この手法は，複数あるテストルート候補の中からチャネル使用数が最小となるテストルートを採用する．これは下記のような理由による．

NoC 内部ネットワークを利用したテストは，一つのコアテストで使用するチャネルが多いと，他のコアのテストの妨げになりやすい．このことを，図 30 に簡単な例を示して説明する．図 30（a）では，コア B のテストルートとしてチ

ャネル③，④を使用している．この場合，コア A またはコア C のテストルートは，チャネル①，②，⑤，⑥を用いて作成することができる．一方，図 30（b）では，コア B のテストルートとしてチャネル①，②，⑤，⑥を使用している．この場合，コア A またはコア C のテストルートは作成できない．



(a)　使用チャネル③，④　　(b)　使用チャネル①，②，⑤，⑥

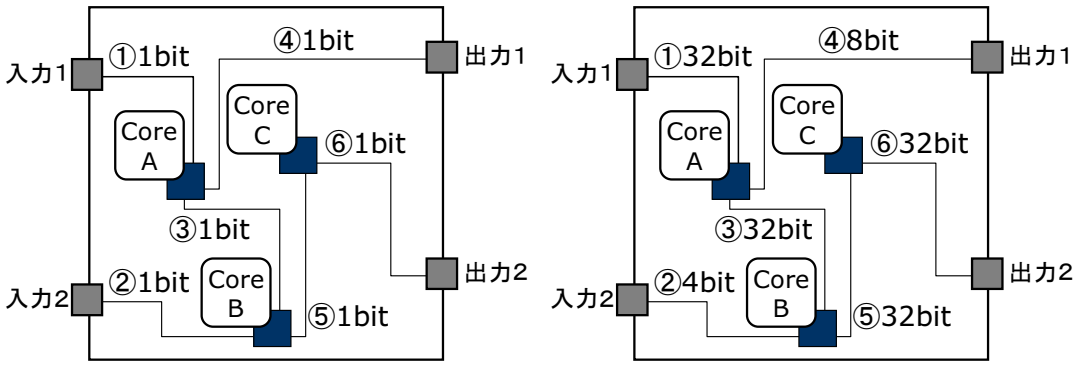
図 30　使用チャネル数が異なるテストルート

このことから，一つのコアテストで使用するチャネルが多いと，他のコアのテストの妨げになりやすいことがわかる．そのため，テストルートはチャネル使用数がなるべく少なくなるように決定するべきであると考えられる．第 3，4 章での説明・実験には，このチャネル使用数最小化法を用いた．

5.2　チャネル使用数最小化法の問題点

これまでは，NoC 内部ネットワークを構成するチャネルのビット幅が全て 1 ビットであるシングルビット幅チャネル NoC を前提として話を進めてきた．しかし，実際は，様々な複数ビット幅チャネルでネットワークを構成している複数ビット幅チャネル NoC が一般的である．図 31 にシングルビット幅チャネル NoC と複数ビット幅チャネル NoC の例を示す．

第 4 章で提案したコアテスト優先順位決定手法は，チャネルと直接関係するものでは無いため，シングルビット幅と複数ビット幅の両方の NoC に対して有効である．しかし，テストルート決定手法であるチャネル使用数最小化法は，シングルビット幅チャネル NoC に対しては有効であるが，複数ビット幅チャネル NoC に対しては必ずしも有効とはいえない．この理由を以下で説明する．



(a)　シングルビット幅チャネル NoC　　(b)　複数ビット幅チャネル NoC

図 31　シングルビット幅チャネル NoC と複数ビット幅チャネル NoC

まず，複数ビット幅チャネル NoC において，各コアのコアテスト時間（テストに必要なクロック数）がどのように決定されるかを説明する．複数ビット幅チャネル NoC の場合，各コアのテストに必要なクロック数は，各コアのテストデータ量を，テストルートを構成するチャネルの中でビット幅が最も小さいチャネルのビット幅（ルートビット幅と呼ぶ）で割った値となる．図 32 の例の場合，コア B のテストルートにチャネル②，③，④を使用しているため，最小ビット幅のチャネル②から，ルートビット幅は 4 ビットとなる．コアテスト時間は，コア B のテストデータ量 14388 をルートビット幅 4 で割るため 3597 クロックとなる．

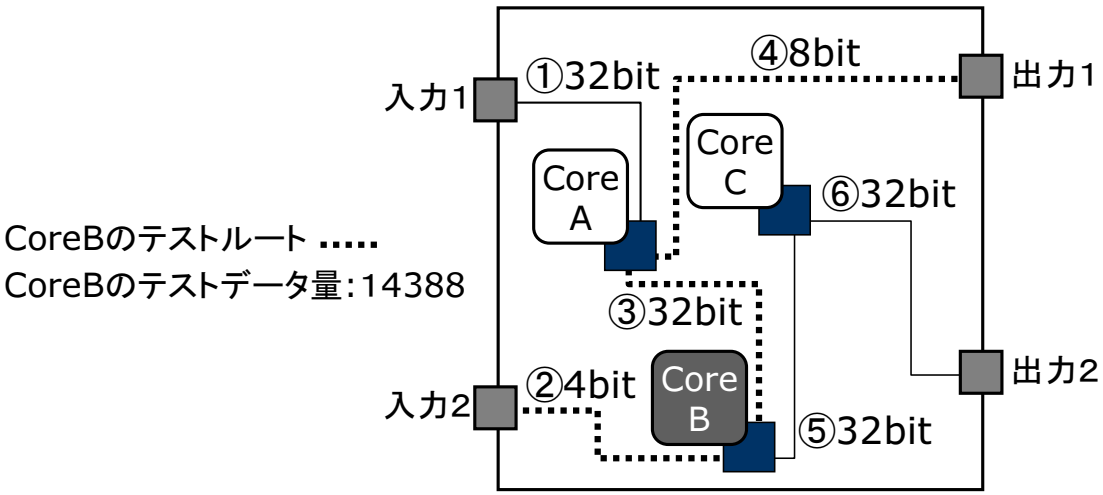


図 32　複数ビット幅チャネル NoC におけるコアテスト例

このように複数ビット幅チャネル NoC においては，コアテスト時間はルートビット幅に反比例する．そのため，複数ビット幅チャネル NoC に対してコアテストスケジューリングをする場合，ルートビット幅を考慮してテストルートを決定するべきである．しかし，チャネル使用数最小化法は，ルートビット幅を考慮せずにテストルートを決定する．そのため，複数ビット幅チャネル NoC に対しては，チャネル使用数最小化法は有効とはいえず，テスト時間が長いスケジュールを作成してしまう可能性が高い．これがチャネル使用数最小化法の問題点である．

5.3　ルートビット幅最大化法（提案手法 1）

複数ビット幅チャネル NoC に対応したテストルート決定手法として，ルートビット幅最大化法を提案する．この手法は，複数あるテストルート候補それぞれのルートビット幅に注目し，ルートビット幅が最大であるテストルートを採用するルーティング手法である．ルートビット幅が最大となるテストルート候補が複数ある場合は，その中でチャネル使用数が最小となるテストルートを採用する．この手法を用いることで，各コアテストにおけるルートビット幅が増大し，その結果テスト時間が短くなると考えられる．

図 31（b）の複数ビット幅チャネル NoC を例にして，ルートビット幅最大化法を説明する．コア A をテストする場合，表 11 に示すように，3 つのテストルート候補が考えられる．ルートビット幅最大化法を用いた場合は，チャネル使用数が多くても，ルートビット幅が最大であるテストルート 2 が採用される．

表 11　図 31（b）の NoC におけるコア A のテストルート候補

	使用チャネル群	チャネル使用数	ルートビット幅
テストルート1	①, ④	2	8
テストルート2	①, ③, ⑤, ⑥	4	32
テストルート3	②, ③, ④	3	4

5.4　ルートビット幅最大化法の問題点

5.3 節で説明したようにルートビット幅最大化法は，ルートビット幅（テストルートを構成するチャネル中でビット幅が最小のチャネルのビット幅）が最大となるようにテストルートを決定する．しかし，テストルートを構成する各チャネルのビット幅のばらつきは考慮していない．つまり，テストルートを決定する際，ルートビット幅を超えるビット幅のチャネル（過大チャネルと呼ぶ）に関しては考慮していない．過大チャネルを使用しても，そのコアテスト自体には，良い影響も悪い影響もない．しかし，ビット幅が大きいチャネルを無駄に使用してしまうことで，その後のコアのテストルートの決定において，ルートビット幅が小さくなってしまう可能性がある．この問題を図 33 に示す NoC を例にして，具体的に説明する．

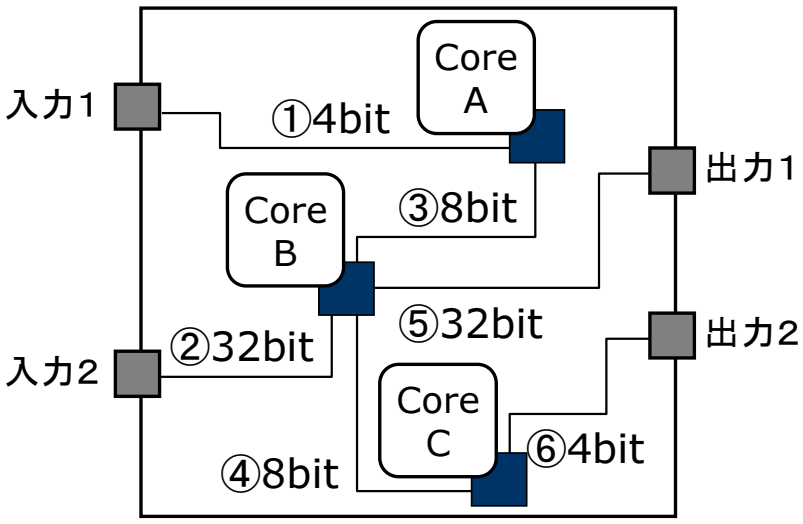


図 33　複数ビット幅チャネル NoC の構成例 2

図 33 に示す NoC のコア A のテストルートは，表 12 に示す二つの候補が考えられる．その際の過大チャネルのイメージを図 34 に示す．テストルート 1, 2 のルートビット幅は共に 4 であるが, テストルート 1 の方がチャネル使用数は少ない．ルートビット幅最大化法を用いた場合，ルートビット幅が最小となるテストルート候補の中から，チャネル使用数が最も小さいテストルートを採用する．その際，過大チャネルは考慮しない．そのため，テストルート 1 が採用され，NoC のチャネルの使用状況は，図 35 のようになる．

表 12　図 33 の NoC におけるコア A のテストルート候補

	使用チャネル群	チャネル使用数	ルートビット幅
テストルート1	①, ③, ⑤	3	4
テストルート2	①, ③, ④, ⑥	4	4

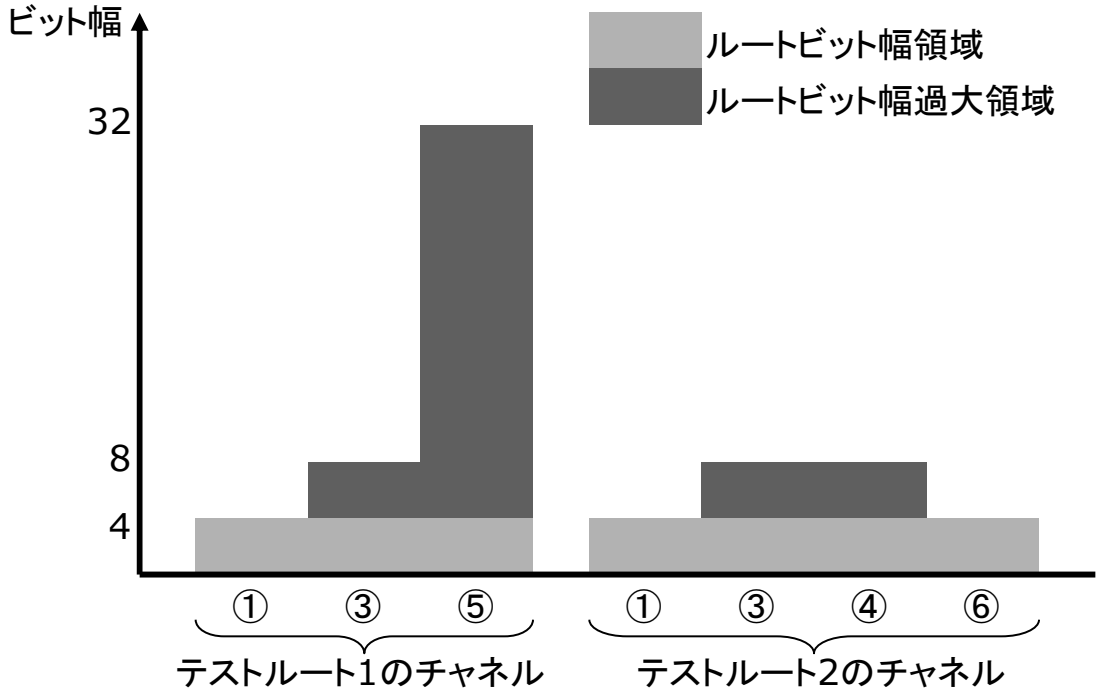


図 34　表 12 のテストルートにおける過大チャネルイメージ

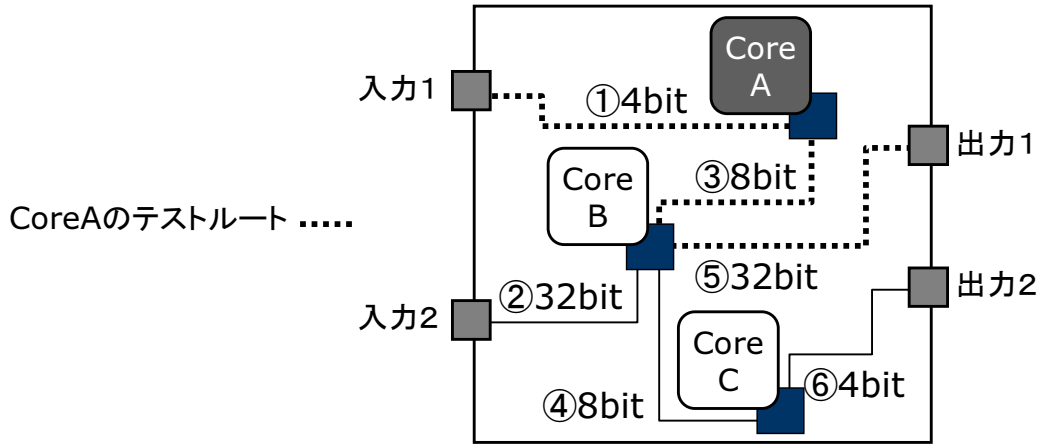


図 35　図 33 の NoC におけるチャネル使用状況 1

続けて，コア B のテストルートを考える．テストルートは，コア A のテストルートに使用されていない，チャネル②，④，⑥を使用することになる．チャネル⑥のビット幅が 4 であるため，コア B のテストルートのルートビット幅は 4 となる．

ここでコア A のテストルートに，テストルート 1 に比べ過大チャネルのビット幅が小さいテストルート 2 を採用する場合を考える．この場合，NoC のチャネルの使用状況は，図 36 のようになる．続いて，先ほどと同様にコア B のテストルートを考える．テストルートは，コア A のテストルートに使用されていない，チャネル②，⑤を使用することになる．チャネル②，⑤は共にビット幅が 32 であるため，コア B のテストルートのルートビット幅は 32 となる．

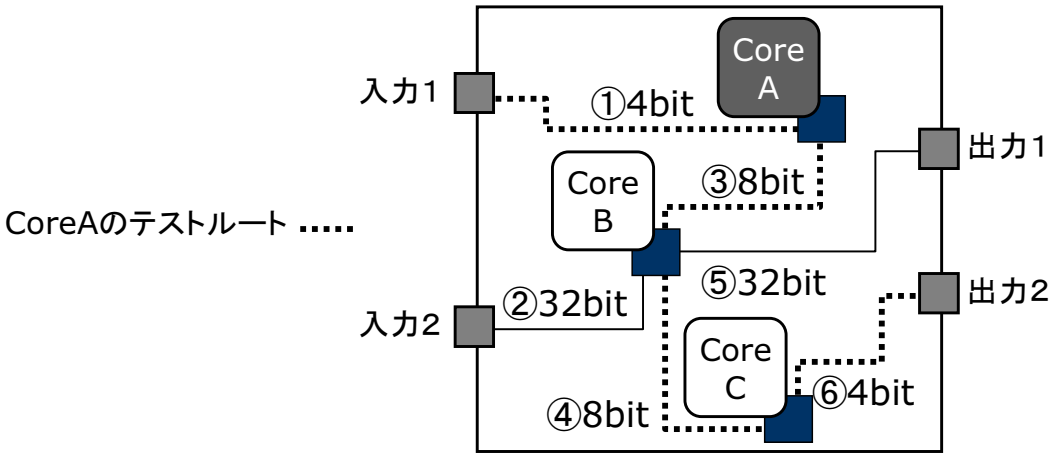


図 36　図 33 の NoC におけるチャネル使用状況 2

このように，コア A のテストルートに過大チャネルのビット幅が大きいテストルート 1 を採用すると，次に続くコア B のテストルートのルートビット幅は 4 となり，コア A に過大チャネルのビット幅が小さいテストルート 2 を採用すると，コア B のテストルートのルートビット幅は 32 となる．コア B のコアテスト時間は，ルートビット幅が 32 のときと 4 のときで，8 倍の違いが生じる．

このことから，テストルートを決定する際に過大チャネルを使用せず残しておいた場合，その後のコアのテストにおいて，ルートビット幅の大きいテストルートを採用できる可能性が高まることがわかる．そのため，過大チャネルはなるべく使用しない方が，その後のコアのテストにとって良いと考えられる．しかし，ルートビット幅最大化法は，ルートビット幅が最大となるテストルート候補が複数ある場合でも，過大チャネルについて何も考慮せずテストルートを決定してしまうため，不必要な過大チャネルを使用する可能性がある．これが，ルートビット幅最大化法の問題点である．

5.5　最大過大チャネル抑制ルートビット幅最大化法（提案手法 2）

5.4 節で説明したように，ルートビット幅最大化法には，過大チャネルについて考慮していないという問題点がある．その問題点を克服するため，過大チャネルを考慮したルートビット幅最大化法の一手法として，最大過大チャネル抑制ルートビット幅最大化法を提案する．テストルートの最大過大チャネルとは，そのテストルートを構成するチャネル中で，ビット幅が最大の過大チャネルを意味する．

この新手法は，ルートビット幅をなるべく大きくするという従来のルートビット幅最大化法の根本的な目的は引き継ぐ．つまり，複数あるテストルート候補からルートビット幅が最大となるテストルートを採用するという基本部分は同じである．新手法と従来手法の違いは，ルートビット幅が最大となるテストルート候補が複数ある場合のテストルートの決定方針である．この場合，従来のルートビット幅最大化法では，テストルートを構成するチャネル数が最小になるテストルートを採用していた．新手法では，最大過大チャネルのビット幅が最小になるテストルートを採用する．

図 34 のようなテストルート候補を例にして，新手法を具体的に説明する．図

37 に示すように，テストルート採用ラインを設定する．テストルート採用ラインはルートビット幅からスタートし（これは最大過大チャネルのビット幅が 0 であることを意味する），徐々にビット幅を大きくしていく．テストルートを構成する全チャネルのビット幅が，初めてこのライン以下に収まったテストルートを採用する．

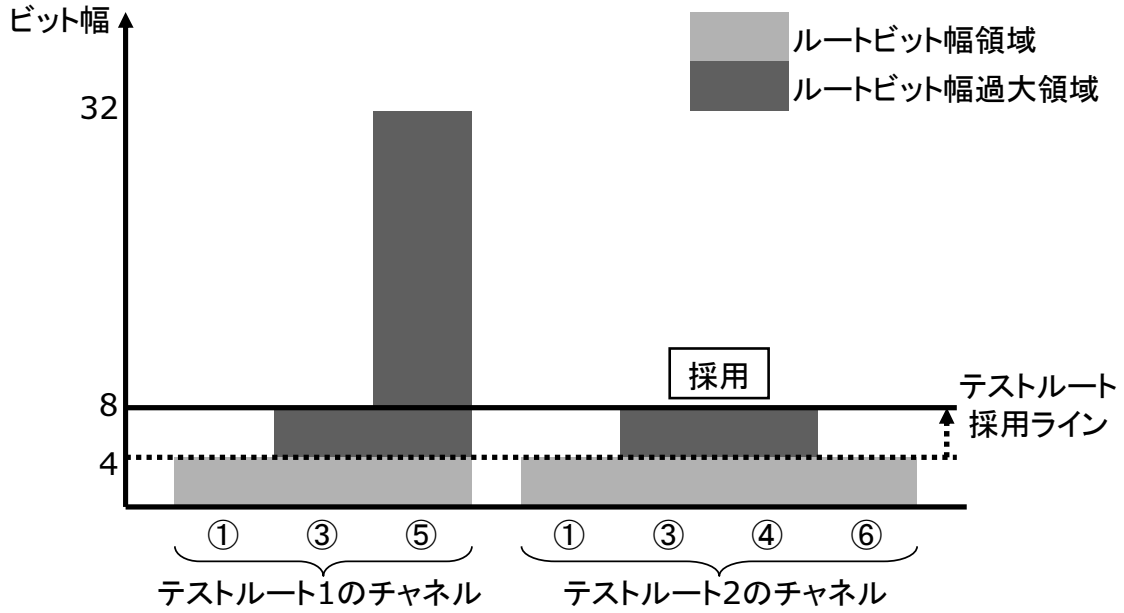


図 37　最大過大チャネル抑制ルートビット幅最大化法のイメージ

この新手法を用いることで，ルートビット幅の最大化は維持しつつ，最大過大チャネルを抑制することで，後のコアのために，ビット幅が大きい過大チャネルを使用せずに残すことができる．そのため，その後に続くコアのテストにおいて，ルートビット幅の大きいテストルートを採用できる可能性が高まり，よりテスト時間の短いスケジュールを作成できると考えられる．

5.6　テストルート決定手法の評価実験

5.1 節の単純法，5.3，5.5 節の提案手法をＣ言語で実装し，様々なタイプの複数ビット幅チャネル NoC 回路に対してスケジュール作成実験を行った．実験に用いた回路においては，チャネルのビット幅は，4，8，12，16，20，24，28，32 のいずれかの値をとる．また，実験に用いた回路は表 13 に示すように，コ

ア数 2 種類，チャネルのビット幅傾向 5 種類，コア数ごとに入（出）力ピン数 3 種類を用意し，合計 30 タイプを用意した．チャネルのビット幅傾向とは，回路にどのようなビット幅のチャネルが多く存在しているかという傾向を示している．例えば，チャネルタイプ「均一」は，4 から 32 ビットまでの全てのビット幅のチャネルが大方同じ割合で存在していることを意味し，チャネルタイプ「小ビット多め」は，ビット幅が 4 ビットのチャネルが多く存在していることを意味している．表 14 に，チャネルのビット幅傾向が意味する，コア間のチャネルにおけるビット幅別の存在割合を記す．コア数が 16 個の回路については，それぞれの回路タイプにつき 30 個の回路を実験に用い，コア数が 36 個の回路については，それぞれの回路タイプにつき 10 個の回路を実験に用いた．つまり，実験に用いた回路は合計 600 個（16 コア回路：450 個，36 コア回路：150 個）である．各コアのテストデータ量は，16 コアの場合は表 5 のように，36 コアの場合は表 15 のように設定した．コアテスト優先順位決定手法には，多スタート局所探索法を用いた．

表 13　実験に用いた回路のタイプ

コア数	チャネルのビット幅傾向	入（出）力ピン数	コア数	チャネルのビット幅傾向	入（出）力ピン数
16	均一	3	36	均一	3
		6			9
		9			15
	小ビット多め	3		小ビット多め	3
		6			9
		9			15
	中ビット多め	3		中ビット多め	3
		6			9
		9			15
	大ビット多め	3		大ビット多め	3
		6			9
		9			15
	小・大ビット多め	3		小・大ビット多め	3
		6			9
		9			15

表 14　チャネルのビット幅傾向

	コア間のチャネルにおける ビット幅別の存在割合
均一	全てのビット幅のチャネルが大方均一
小ビット 多め	4ビットチャネルが全体の約60%
中ビット 多め	16,20ビットチャネルが全体の約60%
大ビット 多め	32ビットチャネルが全体の約60%
小・大ビット 多め	4,32ビットチャネルが全体の約60%

表 15　実験で用いた各コアのテストデータ量（36 コア）

コア番号	テストデータ量[bit]		コア番号	テストデータ量[bit]
1	38,323		19	43,252
2	43,728		20	20,436
3	32,224		21	22,503
4	46,569		22	29,874
5	22,146		23	23,743
6	25,261		24	44,095
7	49,323		25	47,463
8	28,075		26	13,215
9	25,546		27	21,018
10	32,319		28	44,517
11	33,331		29	47,029
12	49,639		30	39,274
13	32,664		31	26,772
14	17,809		32	20,853
15	29,535		33	41,046
16	43,394		34	16,665
17	23,130		35	30,726
18	14,731		36	37,247

表 16，17 に実験結果を示す．各手法の左側の数値は，スケジュールによって得られる NoC 全体のテスト時間[クロック数]の平均値（16 コアは 30 回路，36 コアは 10 回路の平均）を表している．なお，提案手法のテスト時間の下の数値は，チャネル使用数最小化法のテスト時間の平均値を 100%としたときの，提案手法のテスト時間の平均値の割合[%]を示している．各手法の右側の数値は，ルートビット幅の平均値[bit]を示す．またルートビット幅最大化法に注目した，テスト時間の平均値の割合のグラフを図 38，39 に，ルートビット幅の平均値のグラフを図 40，41 に示す．

表 16　テストルート決定手法の評価実験結果（16 コア）

			単純法		提案手法			
			チャネル使用数最小化法		ルートビット幅最大化法		最大過大チャネル抑制ルートビット幅最大化法	
コア数	チャネルのビット幅傾向	入(出)力ピン数	テスト時間	ルートビット幅	テスト時間	ルートビット幅	テスト時間	ルートビット幅
16	均一	3	19,172.3	11.8	15,364.6 80.5	14.4	15,344.5 80.4	14.5
		6	9,730.6	13.3	8,287.7 85.0	15.7	8,295.3 85.0	15.6
		9	8,363.5	13.5	7,517.0 88.8	16.2	7,504.1 88.7	16.8
	小ビット多め	3	29,581.9	6.6	28,271.0 95.1	7.0	28,239.6 95.0	7.1
		6	16,169.4	7.9	15,679.6 96.8	8.5	15,943.8 98.4	8.6
		9	14,331.3	7.6	13,988.8 97.2	8.9	14,099.6 98.3	9.1
	中ビット多め	3	14,461.8	14.5	11,433.5 83.8	16.6	11,426.6 83.7	16.7
		6	7,632.6	15.0	6,297.2 81.7	17.6	6,331.3 82.2	17.7
		9	7,067.2	14.9	5,493.0 76.2	18.2	5,520.8 76.6	18.2
	大ビット多め	3	14,189.4	19.7	10,122.5 79.1	24.5	10,109.1 78.9	24.6
		6	7,042.1	16.8	5,599.5 79.5	20.5	5,613.1 79.6	20.3
		9	5,992.8	20.0	3,717.0 67.6	24.7	3,706.8 67.6	24.7
	小・大ビット多め	3	21,430.4	12.5	17,709.8 80.9	15.9	17,746.2 81.1	16.0
		6	11,336.9	13.7	9,371.2 82.0	17.2	9,390.0 82.3	17.8
		9	9,641.4	13.7	8,284.7 85.8	18.0	8,266.8 85.9	18.2

表 17　テストルート決定手法の評価実験結果（36 コア）

			単純法		提案手法			
			チャネル使用数最小化法		ルートビット幅最大化法		最大過大チャネル抑制ルートビット幅最大化法	
コア数	チャネルのビット幅傾向	入(出)力ピン数	テスト時間	ルートビット幅	テスト時間	ルートビット幅	テスト時間	ルートビット幅
36	均一	3	60,103.8	9.5	42,742.9 72.1	13.2	42,179.3 71.3	13.3
		9	31,311.3	10.4	22,574.3 75.3	14.0	22,322.1 74.3	14.1
		15	27,204.7	10.5	20,336.8 75.4	14.2	19,867.8 74.5	14.3
	小ビット多め	3	88,516.3	4.8	86,605.4 97.9	5.0	86,793.3 98.1	5.0
		9	47,715.5	5.9	47,123.1 98.9	6.5	47,604.6 100.0	6.5
		15	42,821.3	5.7	42,164.5 98.8	6.4	43,486.9 102.7	6.7
	中ビット多め	3	59,637.8	10.4	49,526.3 80.6	12.9	49,772.8 81.4	12.8
		9	21,487.5	12.2	16,714.0 78.0	14.4	16,721.8 78.2	14.4
		15	19,740.6	11.5	15,241.6 76.9	14.9	15,304.1 77.3	14.8
	大ビット多め	3	54,522.1	13.8	31,181.9 62.4	21.6	31,195.3 62.7	21.8
		9	28,712.7	14.2	21,839.3 74.0	20.2	21,173.6 70.9	20.3
		15	24,535.6	13.6	19,668.4 73.1	21.1	19,721.2 73.2	21.3
	小・大ビット多め	3	91,548.3	6.7	71,294.7 77.3	10.7	70,858.0 76.7	10.9
		9	41,342.2	8.4	36,297.4 88.0	11.0	37,272.3 90.5	10.8
		15	34,299.6	8.2	30,210.7 88.0	12.0	29,978.1 86.4	12.4

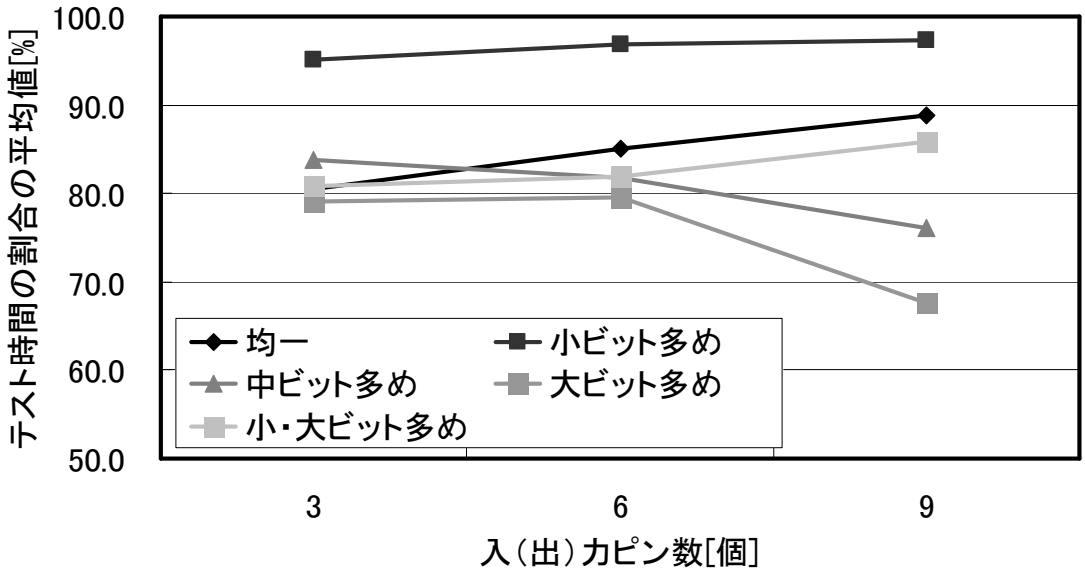


図 38　ルートビット幅最大化法のテスト時間の割合の平均値（16 コア）

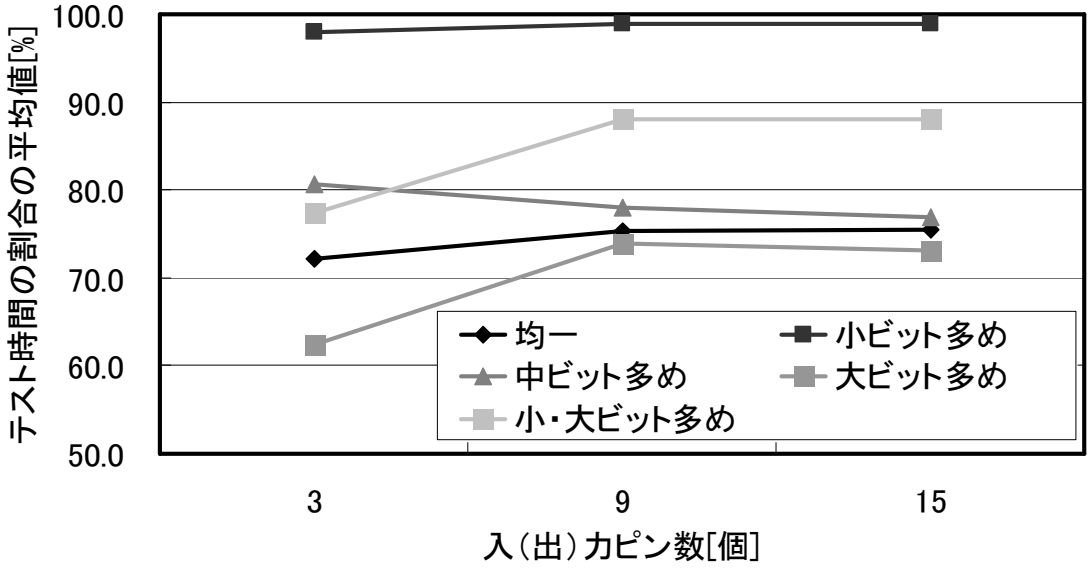


図 39　ルートビット幅最大化法のテスト時間の割合の平均値（36 コア）

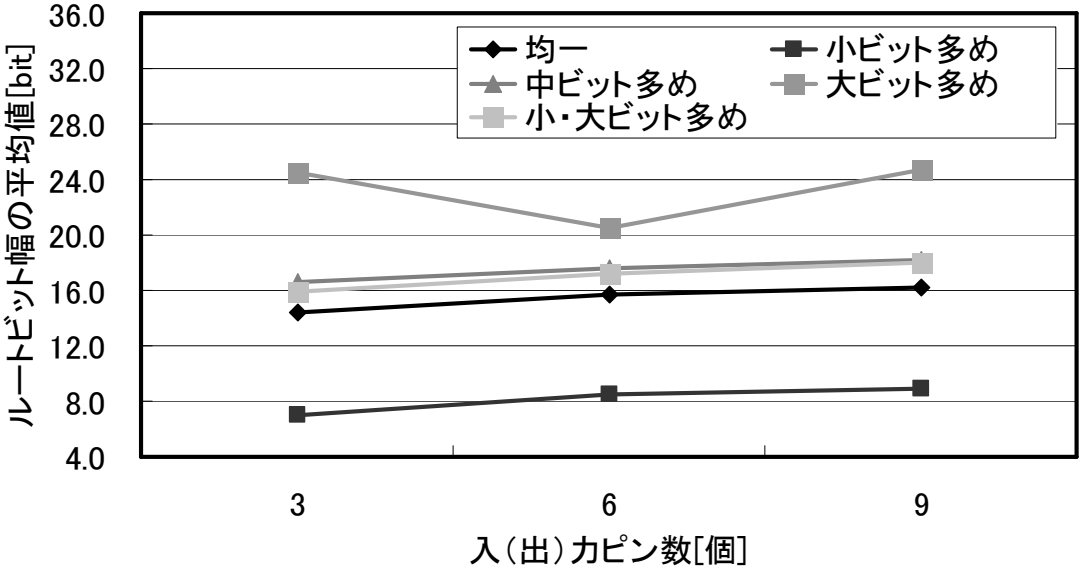


図 40　ルートビット幅最大化法のルートビット幅の平均値（16 コア）

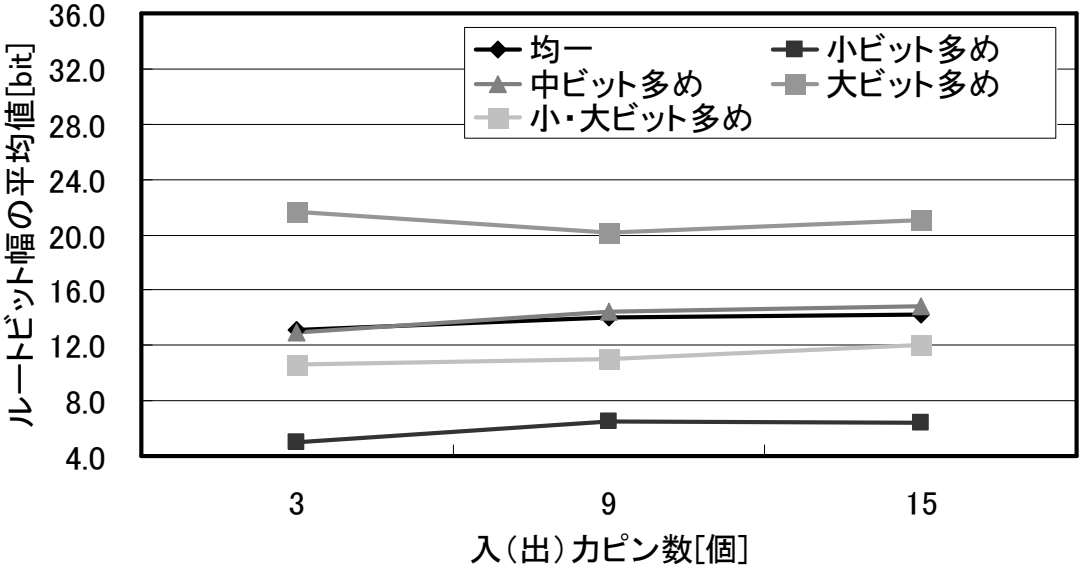


図 41　ルートビット幅最大化法のルートビット幅の平均値（36 コア）

5.7 考察

実験結果からわかるように，単純法に比べ，提案手法はテスト時間を短縮することができた．これは，表 16,17 からわかるように，提案手法を用いた場合，単純法を用いた場合に比べ，ルートビット幅が増加したためであると考えられる（ルートビット幅が大きいほど，テスト時間は短くなる）．

チャネルのビット幅傾向に注目すると，図 38，39 からわかるように，最もテスト時間を短縮できたのは「大ビット多め」で，60%台まで短縮できたタイプもあった．一方，「小ビット多め」に関してはあまりテスト時間を短縮することができず，100%に近い値となった．これは，ビット幅が大きいチャネルが多く存在する回路においては，ルートビット幅が大きいテストルートを作成することは容易であるが，ビット幅が小さいチャネルが多く存在する回路においては，ルートビット幅が大きいテストルートを作成することは，困難もしくは不可能であるためと考えられる．これは図 40，41 のルートビット幅の値からも明らかである．

次に入（出）力ピン数に注目する．実験前の予想では，入（出）力ピン数が増加すると，提案手法の効果は減少するのではないかと考えていた．その理由は，チャネルの使用数とコアテストの並列性の関係にある．提案手法はルートビット幅の大きいテストルートを採用するため，単純法（チャネル使用数最小化法）に比べ，一つのテストルートに使用するチャネル数が多くなりやすい．一つのテストルートに使用するチャネル数が多いほど，多くのコアを並列してテストすることは難しくなる．そのため，入（出）力ピン数が多い回路では，単純法の方が多くのコアを並列してテストし，提案手法の効果は減少すると予想していた．しかし，実際には，入（出）力ピン数が増加しても，提案手法の性能が極端に減少することはなかった．この理由は，入出力ピン数がある程度多いと，NoC 全体のテスト時間が最もテスト時間の長いコアのテスト時間とおおよそ等しくなる可能性が高いためであると考えられる．入（出）力ピン数が少ない場合，図 42 に示すように，各コアのテスト時間に比べ NoC 全体のテスト時間は長い．一方，入（出）力ピン数がある程度多い場合，図 43 に示すように，並列してテストを行えるコアの数が多いため，NoC 全体のテスト時間は短くなり，最もテスト時間の長いコアのテスト時間とおおよそ等しくなると考え

られる．そのため，NoC 全体のテスト時間を短縮するためには，各コアのテスト時間を短縮する必要がある．このことから，ルートビット幅を大きくし，各コアテストの時間を短くしようとする提案手法は，入（出）力ピン数が増加しても有効であったと考えられる．

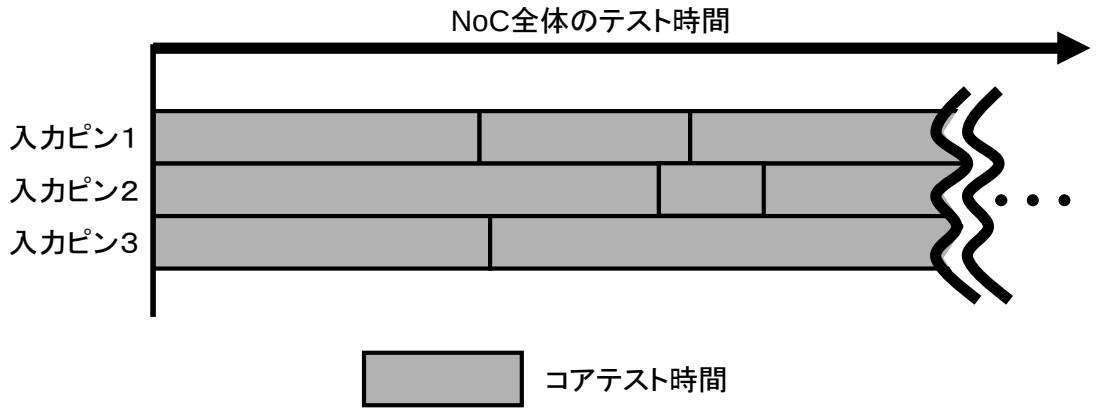


図 42　入力ピンが少ない場合の NoC 全体のテスト時間

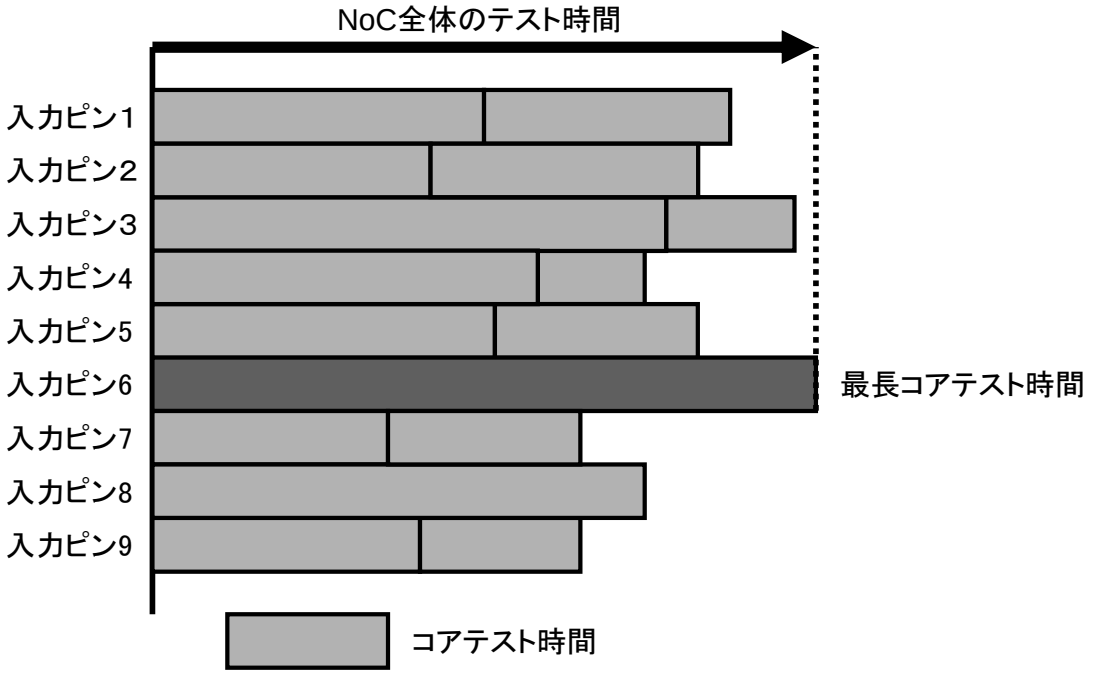


図 43　入力ピンが多い場合の NoC 全体のテスト時間

以上のように，単純法に比べ提案手法は効果が上がることがわかった．しかし，提案手法であるルートビット幅最大化法と最大過大チャネル抑制ルートビット幅最大化法のテスト時間には，顕著な差は見られなかった．これは，表 16，17 からわかるように，ルートビット幅最大化法と最大過大チャネル抑制ルートビット幅最大化法を用いた場合で，ルートビット幅がおおよそ等しい値となったためと考えられる．ルートビット幅に変化がなかった理由としては，回路自身のネットワーク構成などの設計段階の物理的原因によって，ルートビット幅が最大となるテストルートが限られ，最大過大チャネル抑制ルートビット幅最大化法による改善の余地がなかった可能性がある．また，最大過大チャネル抑制ルートビット幅最大化法によって，テストルート中のチャネルのビット幅のばらつきを改善しようとしたが，この方法では，ばらつきの改善が不十分であった可能性も考えられる．

第 6 章

まとめと今後の課題

本論文では，NoC における LSI テスト時間の短縮を目的として，ヘテロジニアス結合型 NoC におけるコアテストスケジューリング手法を提案した．具体的には，コアテスト優先順位を決定するための手法とテストルートを決定するための手法を提案した．コアテスト優先順位決定手法としては，各コアのテストデータ量と周辺チャネル数によるヒューリスティックス法と多スタート局所探索法を用いた探索法の二つを提案し，比較評価した．その結果，探索法によって作成されたスケジュールはヒューリスティックス法によって作成されたスケジュールに比べ，平均約 15%テスト時間が短いものとなった．また，テストルート決定手法としては，ルートビット幅最大化法と，その改良法である最大過大チャネル抑制ルートビット幅最大化法の二つを提案し，単純法であるチャネル使用数最小化法と比較評価を行った．その結果，提案手法によって作成されたスケジュールは, 単純法によって作成されたスケジュールに比べ, 平均約 17%, 回路のタイプによっては最大約 38%テスト時間が短いものとなった．しかし，二つの提案手法間では，テスト時間の差は見られなかった．

今後の課題は，テスト時間をさらに短縮するスケジューリング手法を提案することである．一つの方法として，テストルート決定手法において，よりビット幅のばらつきを小さくするルートビット幅最大化法を考案することが考えられる．

謝辞

本研究の遂行および修士論文の執筆にあたり，多くのご指導とご助言をいただきました元本学工学研究科電気電子工学専攻教授篠木剛先生，ならびに本学地域イノベーション学研究科の鶴岡信治教授，本学工学研究科電気電子工学専攻の高瀬治彦准教授，北英彦准教授，川中普晴助教授に深く感謝いたします．また，日頃熱心に討論して頂いた本学情報処理研究室の大学院生，卒業研究生の皆様方にお礼申し上げます．最後になりましたが，本論文をまとめるにあたり，お世話になった全ての方々に感謝いたします．

参考文献

[1] 林大輔，村井渉，中田明夫，木谷友哉，安本慶一，東野輝夫：ネットワークオンチップにおける回路面積と配線コストを考慮したチップ内通信構造最適化の一手法，電子情報通信学会技術研究報告，CPSY2007-35 (2007-11)

[2] 村井渉，林大輔，中田明夫，木谷友哉，安本慶一，東野輝夫：チップ内ネットワークの性能要求検証および最適化のための一手法，電子情報通信学会技術研究報告，CPSY2006-81,DC2006-95(2007-03)

[3] 鈴木亘，篠木剛，林照峯，川中普晴，鶴岡信治：コア間ベクトルオーバラッピングテスト法における最大消費電力削減手法について，第５７回FTC研究会資料，(2007-07)

[4] John Mark Nolen and Rabi N. Mahapatra : Time-Division-Multiplexed Test Delivery for NoC Systems, IEEE Design & Test of Computers, January/February pp.44-51, (2008)

[5] Jaan Raik, Vineeth Govind and Raimund Ubar : An External Test Approach for Network-on-a-Chip Switches, 15th Asian Test Symposium, (2006)

[6] Davide Bertozzi and Luca Benini : A Network-on-Chip Architecture for Gigascale Systems-on-Chip, IEEE Circuits & Systems Magazin, Second Quarter, pp.18-31, (2004)

[7] 佐野裕基：ヘテロジニアス結合型NoCにおけるコアテストスケジューリング問題，三重大学工学部電気電子工学科卒業論文，(2009-03)

[8] 柳浦睦憲，茨木俊英：組合せ最適化～メタ戦略を中心として～，朝倉書店，(2001)

発表論文

(1) 佐野裕基，篠木剛，:“ヘテロジニアス結合型 NoC におけるコアテストスケジューリング手法”，電気関係学会東海支部連合大会，0-426，2009

(2) 佐野裕基，篠木剛，:“ヘテロジニアス結合型 NoC におけるコアテストスケジューリング手法”，電気関係学会東海支部連合大会，D5-5，2010

(3) 佐野裕基，篠木剛，:“ヘテロジニアス結合型 NoC におけるコアテストスケジューリング手法　－複数ビット幅チャネル NoC への対応－”，計測自動制御学会中部支部三重地区計測制御研究講演会，A-05，2010