博士学位論文

パラメトリックモデルベースの
システムデザイン手法における
システム最適化に関する研究

村田　秀則

2015 年 1 月

大阪大学大学院工学研究科

目次

第1章　緒言

1.1 背景

本研究は、今日の大規模・複雑化する製品システムを最適にシステムデザインする基本的な手法の構築を目指したものである。

システムという言葉は現在、一般的に用いられる言葉であるが、その定義は様々ある。製品システム、システムデザインといった言葉に関しても同様である。そのため、まず、上記の 3 つの単語に関して明確に定義を行う。

まず、「システム」の定義は、JIS の定義である、「多数の構成要素が有機的な秩序を保ち、同一目的に向かって行動するもの(JIS Z 8121)」とする。これは、システムには、構成要素、構成要素間の連結、単一または複数の目的の 3 つが必要であることを意味している[1)]。また、システムには様々あるが、本研究では、その中でも製品システムを対象としたシステムデザインを行う。「製品システム」とは「実際に存在する物理的な部品であるハードウェアと、物理的には存在しないプログラムであるソフトウェアから構成されており、それらの構成要素が機械的、電気的に連結し、要求を満たすために、作られたもの」とし、「システムデザイン」とは、「与えられた要求を満たすシステム構成(部品の組み合わせ)を、理想構成の範囲内から探索、導出すること」と定義する。以降は、製品システム、システムデザインは上記の定義であることを前提とし、定義が必要なキーワードを本文中に用いた場合は、その都度定義を行うこととする。

システムデザインの歴史を概観すると、人類は現在に至るまで、様々な製品システムを開発・デザインし、発展させてきた。そして、この製品システム開発の歴史は設計・開発の問題点を解決してきた歴史でもある。人類は紀元前一万年前の石器時代から、火起こしや狩猟の道具、住居などのものづくりを行ってきた。大昔のものづくりはシステムやシステムデザインといった概念はなく、要求を実現するために、木材や石など基礎的な材料を加工するだけであった。その後、紀元前五千年ごろになると、古代文明が誕生し始め、この頃から、システムデザインが萌芽し始める。戦争のための道具である弓や投石器の開発は、従来の加工だけではなく、それらを組み合わせることで、要求を発現した、最初期のシステムであり、その開発の経緯はシステムデザインを行っていた可能性がある。さらに歴

史が進むと、ピラミッドを始めとし、大規模な建造物が作成され始める。ピラミッドの作成は、簡易な設計図や模型を用いて行われたと言われており、この時代から、人類は要求に対し、理想構成や概略構成をまず設計図、模型として創造し、そこからものを作る、現在のシステムデザインの基礎的な作成手順を踏むことになる。

歴史は進み、人類は産業革命により大きな産業化への転換が起こり、自動化が進展する。自動化とは、これまで人の手で製造を行っていたものを、機械などが代わりとなり、設計図通りの部品や製品を自動的に作成することである。産業革命の時代は、人類の人口の増加、文明の発展とともに大量生産が必須となっており、それにかかる人的コストの増大が問題となっていた。このことから、自動化による製品システム開発の効率化は必須であり、産業革命により、様々な機械装置の開発が始まることによりそれが可能となった。さらに 19 世紀に入り、電気が開発されることにより自動化はさらに進展し、現在では、多くの製品システム、あるいはその部品が自動的に生産されている。

しかし、現在、製品システムの発達とともに、製品システムに求める要求の多様化が進み、さらに、製品システム自体の大規模化も進んでいるため、現状の自動化技術だけでは要求を満たす製品システムを設計、開発することは非常に困難になってきっている。また、グローバル化による多様な国際的要求は、新製品開発サイクルの短期間化、設計・製造サイクルの短期間化を求めており、現在の大規模、複雑な製品システム内の膨大な設計項目を考慮し、なおかつ、設計期間の大幅な短縮を達成可能な革新的なシステムデザイン手法が求められている。

1.2 現状

サイモン [2)]によると、人は、自分が扱える問題の規模を超えた大規模な問題に取り組むとき、以下の手順をとる(Fig. 1.1)。まず対象とする問題の認識と定義を行い、次に問題をより理解しやすいサブ問題に分割し、これを直接解けるところまで続け、サブ問題の解を得る。さらに、これらの解の中から元の問題の解決に役立つ結果を試行錯誤的に選び出しながら、サブ問題の解を合成していき、問題の解決を図る。現状のシステムデザインは、まさに上記の大規模問題であり、こ

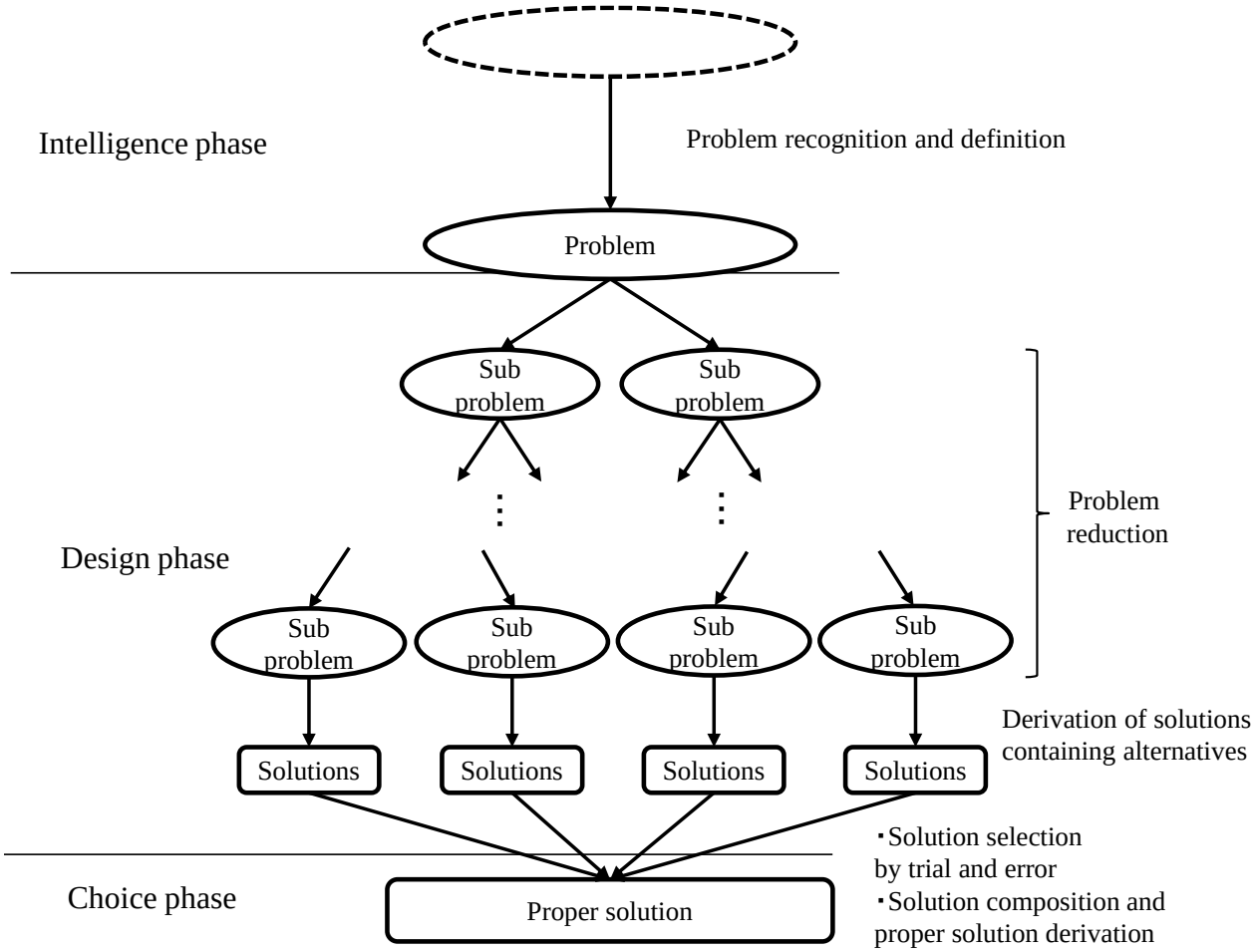


Fig.1.1 Solution process of general large scale problem.

れまでの製品システムのシステムデザインも、上記の大規模問題解決の手順の大規模問題を大規模製品システムに置き換えると、ほとんど同じである。現状の製品システムのシステムデザインの手順を Fig. 1.2 に示す。この手順は、以下に示す通りである。

① まず社会的・経済的・物理的要求から、熟練の設計者が自身の勘と経験を活かし、製品システムの理想構成を推定し、仕様を決定する。

② 次に推定された製品システムの理想構成から、設計者が製品システム全体の概略構成のモデリングを行う。

③ その後、製品システムをどの様なハードウェアとソフトウェアで実現するか、という大きな方向づけを行う、いわゆるアーキテクチャ設計を行い、このアーキテクチャに基づいて、具体的に実現するハードウェア・ソフトウェ

アのモデリングを行い、複数の設計ドメイン(機械系や電気系、等)を仮定し、設計ドメイン間の依存関係を考慮しながら、各設計ドメインでモデリングを行う。ここでの設計ドメインとは、ハードウェア・ソフトウェアよりも小さい設計単位のことである。

④このように作成された各設計ドメインのモデルをもとに、デジタルデザイン手法を駆使し、設計ドメインごとに解析・最適化手法を用い最適解の導出までを行う。

⑤そして、最終的に、熟練の設計者が勘や経験を活かし、各設計ドメインの依存関係を考慮しながら最適解群をすり合わせ、製品システムの適正解を導出

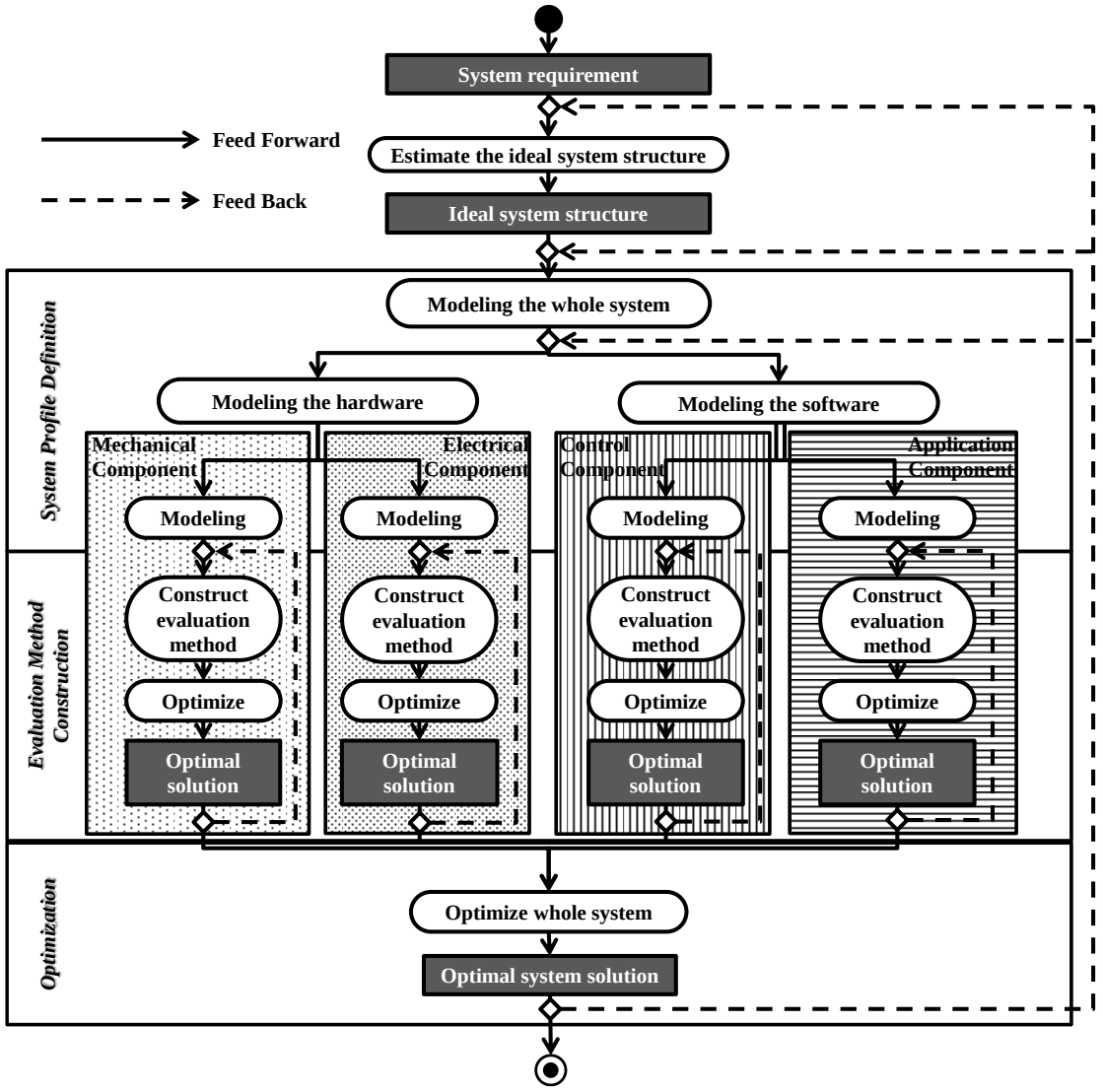


Fig.1.2 Conventional system design flow.

する。

⑥ただし、製品システムの適正解が要求・仕様を満足していない場合、各ドメインの再設計、あるいは、大きな修正が必要な場合は、製品システムの再モデリング、さらには、理想構成の再推定などを行う。

このように、製品システム全体(上記の①、②)・設計ドメイン(上記の③、④)の設計とすり合わせ(上記の⑤)、手戻り(上記の⑥)を繰り返すことで、要求・仕様を満足する製品システムの適正解を導出していた。

この従来のシステムデザインの特徴は、設計ドメイン毎に並行して設計することによる設計の高速化と設計ドメイン毎の部分最適解群のすり合わせに設計者の勘や経験を駆使することで、早急に要求を満たす適正な製品システムの解を求められることである。

1.3 課題

しかし、現在の製品システムの大規模化・複雑化に伴い、このすり合わせにかかる時間が増大しており、適正解を導出するまでの期間の長期化や、期限内に適正解が導出されない場合、要求を落とした製品システムを構築せざるを得ないといった問題点[3)]が発生している。この様なシステムデザインでは、新しい製品システムの開発・設計は難しく、その問題点は、以下のような事が考えられる。

(1) 依存関係の把握：

製品システムは、ハードウェア、ソフトウェアを始め、多様な設計ドメインに分かれており、現状の大規模、複雑な製品システム全体の依存関係を把握することは困難であること。

(2) 依存関係の分析：

製品システムの構築は、これら多様な設計ドメイン専門の異なる複数の設計者のすり合わせによる協働作業である。そのため、設計ドメイン間の依存関係を全て把握することは難しく、一部相反する依存関係が把握されない可能性がある。

(3) 依存関係の整理と解決：

設計ドメイン毎に設計を行うため、設計ドメイン間の依存関係に相反す

るものが存在する場合、すり合わせが困難になり、場合によっては大きな手戻りが発生すること。

(4) しかも、製品システムの規模が膨大になればなるほど、すり合わせが膨大になって適正解を求めることも不可能となり、その結果、要求を満たさず、妥協した製品システムを構築する。これを繰返すうちに、市場要求とは程遠い製品システムとなり、市場競争力を失って市場から撤退に追い込まれるケースが山積している。

(5) 現状のシステムデザイン手法では、各設計ドメイン(ハードウェア(機械系、電気系など))、ソフトウェア)の個別最適解を組合せた適正解であるため、製品システム全体の最適解を求めることはほぼ不可能であること。

この様に、これまでのすり合わせによるシステムデザインでは、上述してきたように、今後の大規模で、複雑な製品システムを適正にデザインすることが増々難しくなっている。

この原因は明らかに、設計者の勘や経験に大きく頼った設計を行っていることであり、これにより、製品システム全体の依存関係の把握、分析、整理、解決が適切になされていないことが本質的原因である。そのため、製品システム全体の依存関係の把握から解決まで行う手法を、システム化、デジタル化、することが必要である。ここで述べるシステム化は、経験論的な手法やプロセスを一般化・規格化し、システマティックに行えるようにすることの意味であり、デジタル化は、システム化した手法やプロセスをコンピュータ上で扱えるようにすることの意味である。

上記の依存関係に関する問題に対し、設計工学[4-6]やシステム工学[7-9]、モデル駆動エンジニアリング[10,11]等により、様々な手法が提案されている。提案されている手法や既に開発されているソフトウェアを、対象とする依存関係の問題に対し整理したものが Table. 1.1 である。

まず、依存関係の把握を行うための手法であるモデリング手法に関して述べる。モデリング手法は Fig. 1.2 に従い、システムレベル、設計ドメインレベルのモデリング手法に分けて述べると、システムレベルとしては、IDEF(Integration DEFinition)[12]、SysML(Systems Modelling Language)[13]などが提案されており、設

Table 1.1 Examples of design method classified with target dependency subject.

Target of dependency subject		Example of method	
Visualization	System modeling	IDEF	
		SysML	
		Modelica	
	Domain modeling	M-CAD	Mecanical modelling
		E-CAD	Electrical modelling
		UML	Software modelling
		LabVIEW	Control modelling & evaluation
		SystemC	Electrical modelling & Software modelling
Analysis	Multivariate statistics	Principal component analysis	
		Factor analysis	
		Clustering	
Arrangement	Graph theory		
	DSM		
	ISM		
Solution	Optimization method	Simplex method	
		Hill climbing method	
		Genetic algorithm	

計ドメインレベルとしては、各種 CAD(Computer Aided Design)や、ソフトウェアモデリングのために開発された言語である UML(Unified Modelling Language)[14] などがある。現状、設計ドメインレベル、あるいは個々の製品レベルで見ると、多様なモデリング手法が存在しているが、システムレベルでのモデリング手法は少なく、また、それぞれ、表記だけが目的であるもの、評価手法の連携だけが目的であるものなど、システムモデリング手法によっても目的は異なる。そのため、依存関係のモデリングが可能なシステムモデリング手法の選択と、そのモデリング方法の構築が課題である。

次に依存関係の分析を行うための手法としては、多変量解析[15]が挙げられる。シ製品ステムに含まれる依存関係は、製品システムに含まれる膨大な設計項目とその関係式で表現される。そのため、このような膨大なパラメータの分析には、

主成分分析、因子分析、クラスター分析などの多変量解析が用いられる。しかし、このような分析手法は数学的な手法であるため、分析する目的のために、どのように分析手法を用い、組み合わせ、結果を得るかということを考慮しなくてはならない。依存関係の分析では、設計ドメイン間の相反する依存関係を分析手法により、予め予測することが重要となる。そのため、この目的に合う、分析手法の選択と組み合わせが課題である。

依存関係の整理に関しては、DSM(Design Structure Matrix)[16]、ISM(Interpretive Structural Model)法[17]などの手法が挙げられる。これらは要素と要素の間の依存関係を表現、整理することが可能な手法であり、依存関係の整理はこの手法を用いれば、現状は可能である。但し、その前提として、要素間の依存関係が明確化されている必要がある。

最後に依存関係の解決方法としては、最適化手法が挙げられる。最適化手法は

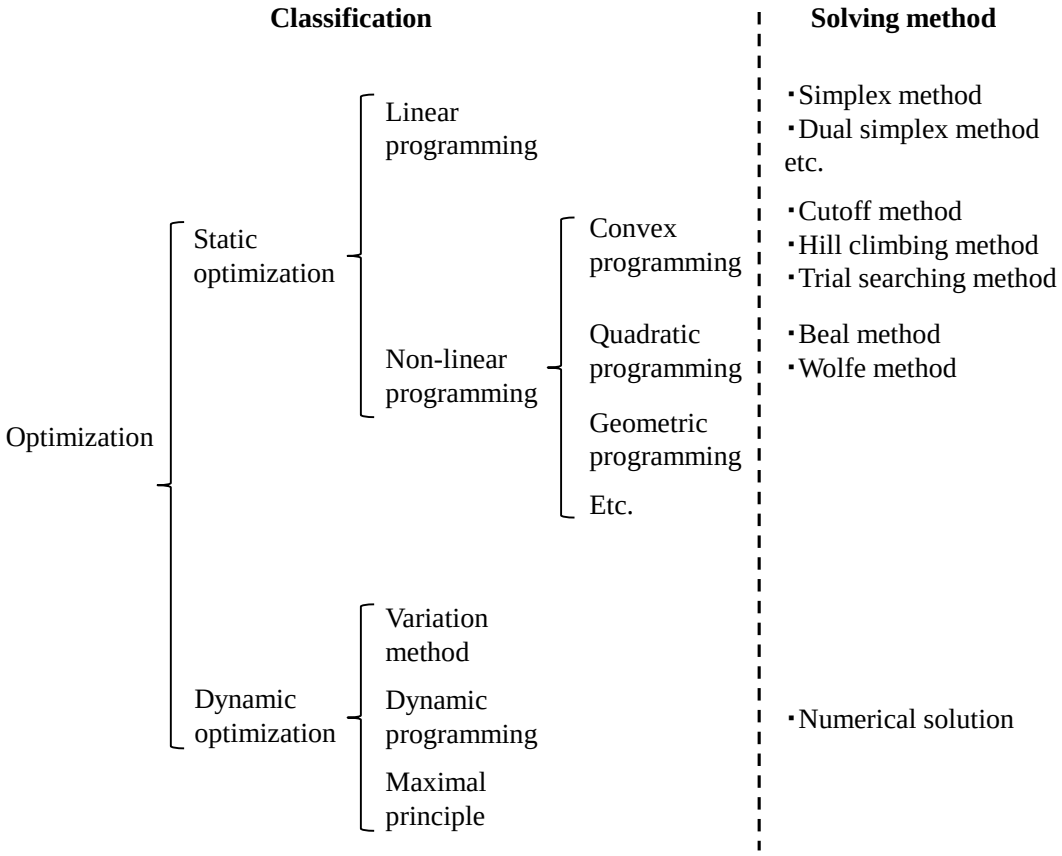


Fig.1.3 Classified optimization methods.

大規模･複雑な問題からより短時間で正確に最適解を発見するための手法であり、すり合わせを効率的に行う手法である。最適化手法は対象とする問題により様々な解法が存在する(Fig. 1.3)[18]。問題の分類としては、静的・動的、非線形・線形問題、多目的・単目的問題、など様々ある。しかし、大規模・複雑なシステムの場合、最適化問題の構造が明確に分かっていることは少なく、かつすり合わせの原因となる依存関係が不明な場合も多い。そのような場合は最適化を行うことは難しい。

このように、現状では、様々なシステム化、デジタル化手法が提案されつつあるが、基本的には、製品システムに含まれる依存関係が明確化されていることが前提であり、すり合わせの課題を解決するためには、製品システム全体の依存関係の把握と分析をベースに、整理、解決手法を連携させ最適解の導出まで行う新たなシステムデザイン手法の構築が必要である。

1.4 本研究の目的と構成

そこで本研究では、与えられた要求、理想構成に対し、製品システム全体をパラメトリックに表現することで、依存関係の明確化を行い、適切にコンピュータ上で依存関係の処理を行うことで、製品システムの最適解を導出する手法の構築と、その自動化手法の構築を目的とする。

本論文は 6 章から構成される。フローチャートを Fig. 1.4 に示す。第 1 章の緒言に始まり、第 2 章では、従来のシステムデザイン手法の課題を解決し、製品システムの最適解を導出するためのシステムデザイン手法の構築を行う。第 3 章では、第 2 章で構築したシステムデザイン手法の自動実行アルゴリズムの構築を行う。第 4 章では、第 2 章、第 3 章で構築したシステムデザイン手法を用いた応用展開として、次世代携帯電話用システム LSI を対象としたシステムデザインを行う。その後、第 5 章で本論文の考察を行い、第 6 章で本論文の結言を述べる。

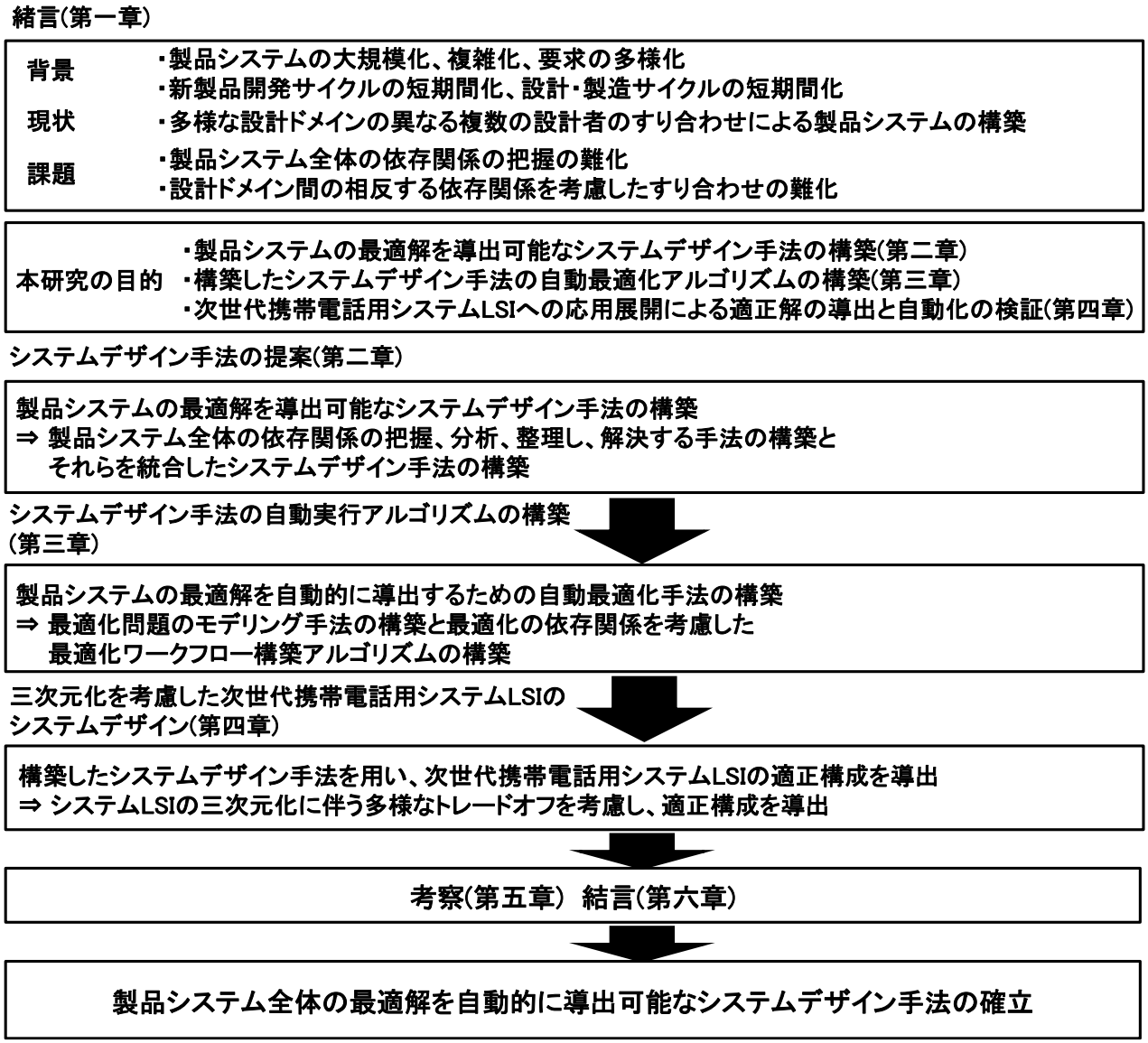


Fig.1.4 Flowchart of this research work.

参考文献

1) 赤木新介, “システム工学-エンジニアリングシステムの解析と計画-”, 共立出版株式会社, (1992).

2) ハーバート・A. サイモン, “システムの科学”, パーソナルメディア, (1996).

3) 佐藤了平, “イノベーション創出の鍵を握るシステムデザイン・インテグレーション(SD・SI)と課題”, Mate (Microjoining and Assembly Technology in Electronics) 2009 論文集, (2009) 1-6.

4) 大富浩一, “初歩から学ぶ設計手法―多彩なツールに振り回されないための、戦略的設計開発の考え方”, 工業調査会, (2007).

5) H. H. グッド, R. E. メイコール, “システム工学”, 日本科学技術連盟, (1960).

6) 小林孝, 吉村忍, M.J.Jeong, “上流設計の重要性とその手法”, 日本機械学会第13回 設計工学・システム部門講演会(2003), pp. 174- 17.

7) Nasa Headquarters, “NASA Systems Engineering Handbook”, www.Militarybookshop.Co.UK, (2007).

8) Tim Weilkiens, “Systems Engineering with SysML/UML:Modeling, Analysis, Design (The MK/OMG Press)”, Morgan Kaufmann, (2008).

9) INCOSE (International Council on Systems Enginerring), “A consensus of the Fellows of the INCOSE”, http://www.incose.org/practice/fellowconsensus.aspx.

10) Stuart Kent, “Model Driven Engineering”, Integrated Formal Methods, 2335 (2002) 286-298.

11) オブジェクトテクノロジー研究所, “欧米におけるモデル駆動組込みシステム開発と標準化の現状”, オブジェクトテクノロジー研究所, (2008).

12) Richard J. Mayer, Capt. Michael, K. Painter, Paula S. deWitte, “IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications”, http://www.idef.com/pdf/ideffami.pdf.

13) Sanford Friedenthal, Alan Moore, Rick Steiner, “A Practical Guide to SysML”, Morgan Kaufanm, (2009).

14) Object Management Group, ”OMG Unified Modeling Language Infrastructure”, http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/.

15) 永田靖, 棟近雅彦, “多変量解析法入門”, サイエンス社, (2001).
16) Steward, “D. V. The Design Structure System: A Method for Managing the Design of Complex Systems.”, IEEE Transactions on Engineering Management, 28 (1981) 71-74.
17) 池田將明, “システムズアプローチによる問題解決の方法”, 森北出版, (2013).
18) G. L. Nemhauser, M. J. Todd, A. H. G. Rinnooy Kan, “最適化ハンドブック” 朝倉書店, (1995).

第2章　システムデザイン手法の提案

2.1　緒言

これまでに様々な大規模製品システムのシステムデザインが行われてきた。それらは熟練の設計者による、膨大な専門知識、経験を用い、繰り返しすり合わせを行うことで達成されてきた。しかし、製品システムの大規模・複雑化に伴い、従来のすり合わせによるシステムデザインでは適正解を導出することが困難になってきている。本章では、すり合わせによるシステムデザインに代わる、製品システム全体を最適化可能なシステムデザイン手法の検討を行った結果を述べる。

2.2　従来の課題と解決策の提案

従来型のシステムデザインでは、すり合わせ設計であるため、製品システムの最適解が導出されることはほとんどない。すり合わせの原因は、製品システム内に含まれる依存関係を適切に把握、分析、整理、解決が行われていないためである。即ち、①要求を考慮した製品システム全体のモデリング(依存関係の把握)、②製品システムの設計ドメイン分割(依存関係の分析)、③設計ドメイン毎の設計・最適解の導出(依存関係の整理)、④すり合わせ最適化(依存関係の解決)、の 4 つに関して、システム化・デジタル化することが製品システムの最適解の導出につながる。

そこで本研究では、上記のすり合わせの問題を本質的に解決し、製品システム全体の最適解を求めることが可能なシステムデザイン手法(Fig. 2.1)を提案する。本手法の本質は、従来手法と比較して、各設計ドメインに分けたモデリングを行わずに、製品システム全体のモデリングを行い、デジタルに最適解を求めることである。これにより、製品システム全体の依存関係の把握が行えるため、最適化が可能となる。また、製品システム全体のモデリングを行ったとしても、依存関係を考慮し、大規模・複雑な製品システムの最適解を求めることは容易ではない。そのため、以下の 5 つからなる構成要素を連携させた手法を考案し、システムデザインを行う方法を提案する。

(1) システムプロファイル定義: 設計者が推定した理想構成をもとに、製品システム全体の依存関係のモデリングを行う。このモデリングを行うためには、ハードウェア・ソフトウェアを含む製品システム全体を定義可能なモデリング手法が必要となる。即ち、対象とする製品システムの構造のモデル、さらに製品システムに含まれる依

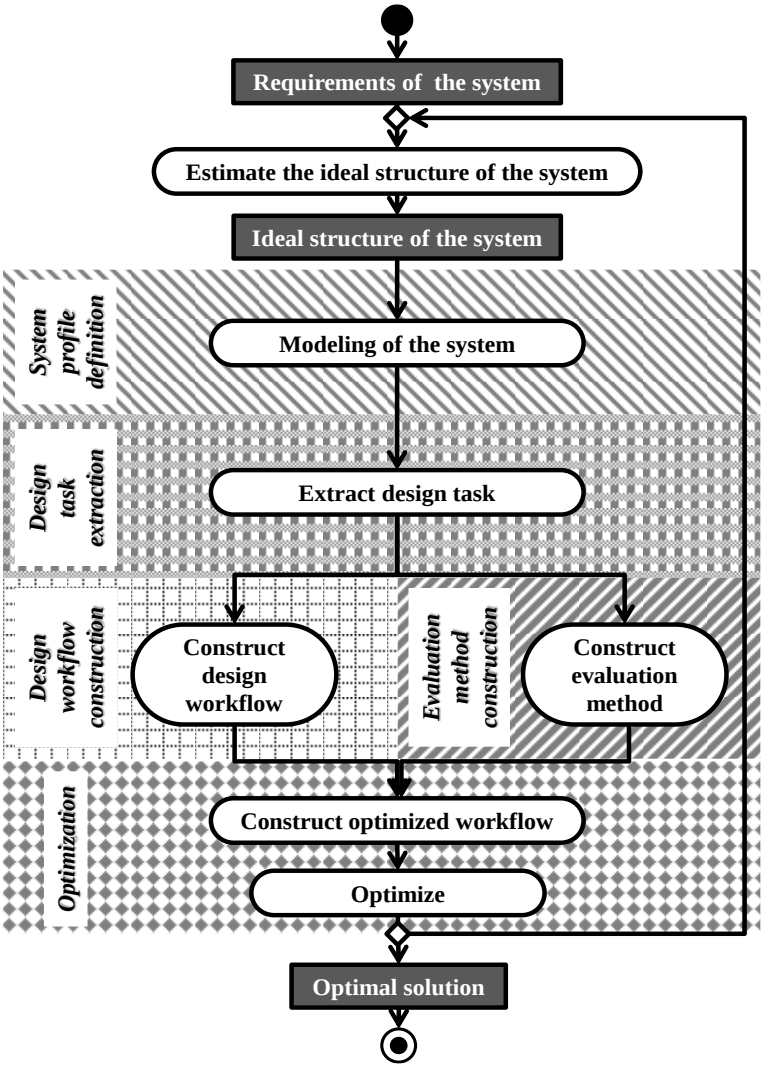


Fig. 2.1 Proposal system design flow.

存関係を表現し、評価手法と連携させるためのパラメトリックモデル、さらに、最適化のための設計者の設計意図である、評価項目や設計項目、即ち、目的関数や設計変数、制約条件といった最適化問題を定義できるモデリング手法が必要となる。しかし、従来のシステムモデリング手法では、構造モデル、パラメトリックモデルをそれぞれ定義できる手法は存在するが、最適化をモデリングできる手法はない。そのため、詳細は後述するが、本研究では、システム全体をモデリング可能なSysML(Systems Modeling Language)に着目し、それに最適化のモデルを追加した拡張SysML を構築した。この拡張 SysML を用いることで、複数のドメインに分けることなく、システムの要求や構造、機能、パラメトリックな関係の定義及び、拡張プロファイルを用いることで相反する問題(最適化問題)などのモデリングが可能である

ことが特徴である。

(2) 設計タスク抽出: 従来のシステムデザイン手法では、製品システムに含まれる多様な設計項目と依存関係から、設計者により、可能な限り独立になるように設計ドメインの分割が行われていた。しかし、製品システムの大規模化に伴い、設計項目間の依存関係の把握が困難になり、独立性の高い設計ドメイン間の導出が困難になるという課題があった。そのため、本研究では、製品システムに含まれる膨大な設計項目と依存関係の中から、関係性が深い、重要な設計項目を先に抽出することで、設計ドメインを可能な限り独立に抽出できるようにした。即ち、従来設計者が勘や経験で行っていた設計ドメイン分割(以降は設計タスクと呼ぶ)をシステテマティックに、コンピュータ上で実行できるようにした。これにより、各製品システムに適した設計タスクの抽出ができ、従来のシステムデザインより設計ドメイン間の依存関係が少なくなるため、手戻りが大幅に減ることが期待される。

(3) 評価手法の構築: コンピュータ上ですべての設計を行うため、製品システムをパラメトリックに扱える様にしている。ここで重要なのは、設計タスクごとにパラメトリックに扱うことが可能な評価手法を構築することである。この評価手法を用いることにより、後の手順で、設計タスク間の依存関係を考慮した製品システム全体での最適化の実行が可能となる。

(4) 設計ワークフロー構築: 抽出した設計タスク間には、依存関係が残っており、特に相反する依存関係が含まれている場合、そのまま設計を行うと大きな手戻りが発生する可能性がある。この課題に対し、最も設計の後戻りが少なく、最適な設計が可能となるように、設計タスクの依存関係を考慮し、整理を行うことで設計ワークフローを決めることができるようにした。

(5) 最適化実行: 従来の設計ドメイン毎の個別最適解のすり合わせではなく、最適化を実行する。即ち、導出された設計ワークフローに従い、構築された設計タスクごとの評価手法を実行し、定義された最適化問題に従い、遺伝的アルゴリズムなどの最適化手法を駆使して、最適化を実行する。これは、従来はシステム設計者しか製品システムの全体像を把握しておらず、設計タスク間の依存関係が不明瞭であったためすり合わせしかできなかったが、上記のようにコンピュータ上で製品システム全体の依存関係をモデリングし、最適化ワークフローの生成まで依存関係を考慮し、

デジタルに実行することにより最適化を可能にしたものである。

上記のように、システムデザイン手法はこれらの構成要素を、その対象と、発揮する機能の 2 つの軸から構成される 4 つの特徴ある面と入出力の 2 面で整理した。2 つの軸とは、時間軸、機能軸と対象軸である。機能軸は何をするかを表現する技術で、記述・分析する機能が主である技術と、実行処理する機能が主である技術に分けられる。対象軸は何を扱うのかを表現する技術で、製品情報と、工程情報に分けられる。6 つの面とは、システムプロファイル定義/設計タスク抽出(第 2 面)、評価手法構築 (第 3 面)、設計ワークフロー構築(第 4 面)、最適化実行(第 5 面)に、入出力(第 1 面、第 6 面)を加えた 6 つの面であり、これらを連携させる手法であることから、6 面を Cubic で表現し、SDSI(System Design System Integration)-Cubic と呼ぶこととした。Fig. 2.2 に 2 軸とそれにより構成される 4 つの面の詳細を、Fig. 2.3 に SDSI-Cubic の全体像を示す。

このように、新しいシステムデザイン手法 SDSI-Cubic を提案したので、各面の詳細な検討結果について以下に示す。

2.3 システムデザイン手法の構築結果

本節では、提案したシステムデザイン手法 SDSI-Cubic を構築するために開発した 6 つの面からなる内容について述べる。

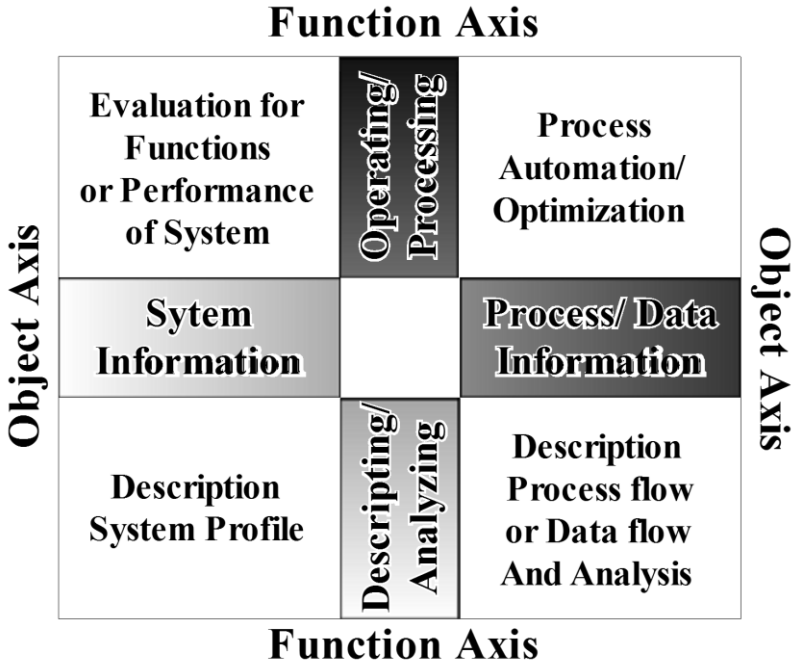


Fig. 2.2 The image of restructured design system with 2-axis and 4-faces.

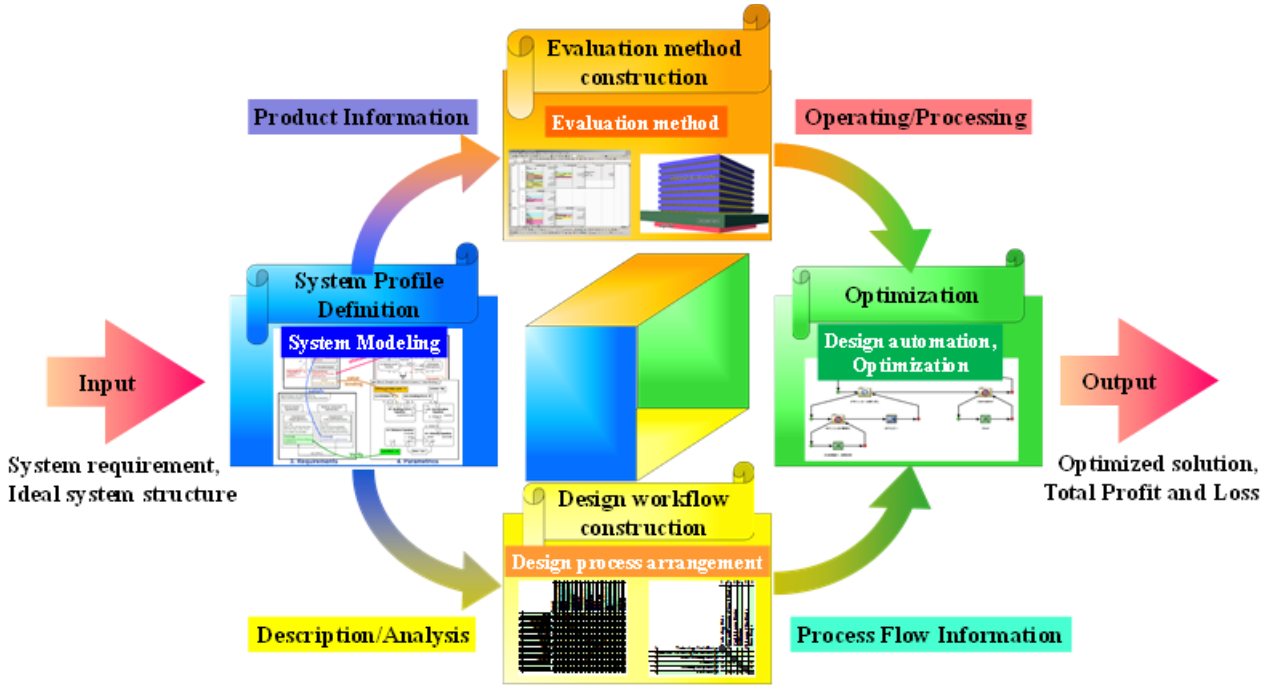


Fig. 2.3 SDSI-Cubic method.

以下では、製品システム全体を考慮して最適解を得るために必要と考えた、6 つの面の各要素について順に検討結果を述べる。

(1) 第 1 面(入力)

まず対象とする製品システムを実現する上で必要な情報を入力する。すなわち、この製品システムは何が要求されているか、その要求を実現する上での概略構成や、最適化に必要な重要相反問題の定義(目的関数、設計変数、制約条件)(これらは Fig. 2.1 の理想構成の推定で設計者が決定する項目である)について入力できるようにした。入力は第 2 面で用いる手法を実装したソフトウェアを用いて、モデリングすることで行われる。

(2) 第 2 面(システムプロファイル定義/設計タスク抽出)

次に要求や概略構成を決めた製品システム全体をコンピュータ上で扱えるように定義、モデリングを行う。これにより、①製品システムに含まれる多様な設計ドメインの依存関係をパラメトリックに定義し、コンピュータ上で製品システム全体を一括して扱え、②重要設計項目(設計タスク)の抽出、第 3 面、第 4 面での実行・処理を通して最適化が可能となると考えた。

このシステムプロファイル定義では、入力される項目である要求から想定された概略構

成と評価・最適化を行うための数理モデル、相反問題(最適化問題)の定義を、ハードウェア・ソフトウェアを含む製品システム全体に対して行う必要がある。ここでは、まず、モデリングすべき範囲を明確化するために、モデリングすべき製品システムの範囲(モデリングすべき設計ドメイン群)と、数理モデル、相反問題(最適化問題)の検討を行った。

まず、製品システムの範囲の検討結果に関して述べる。製品システムは大きくハードウェア・ソフトウェアに大別される。ハードウェアは実際に存在する物理的な部品であり、ソフトウェアは物理的には存在しないプログラムである。さらに分類すると、ハードウェアは機械系ハードウェアと電気系ハードウェアに分かれる。ソフトウェアはアプリケーションソフトウェア、制御系ソフトウェアに分かれる。機械系ハードウェアは、一般的に機械部品と呼ばれるものであり、機械的な動作を行うものである。それに対し電気系ハードウェアは電気的に動作するものである。アプリケーションソフトウェアは特定の目的のために作成されたソフトウェアであり、制御系ソフトウェアは制御目的に特化したソフトウェアである。従来は機械系ハードウェアと制御系ソフトウェアの組み合わせで製品システムは構築されてきたが、電子回路の発展とともに、製品システムの高機能化のために電子回路を用いた電気系ハードウェアや、それの上で動作するアプリケーションソフトウェアも含めた製品システムが構築されるようになってきた。そして近年、これら 4 つの設計ドメインは、一般的な大規模・複雑な製品システムには含まれるようになっている。本研究では、これら 4 つの設計ドメインを製品システムが包含すべきものと捉え、製品システム全体のモデリングは、これらをモデリング可能であることとする。

次に数理モデルに関して述べる。数理モデルはパラメータとその関係式から構成されるが、その関係式は大きく、定式化されたもの(パラメトリックな関係)と定式化されていないもの(ノンパラメトリックな関係)に分類[1)]される。これらは、一般的な物理法則などの中でも数式(運動方程式、マクスウェル方程式等)を用いて表されるものをパラメトリックな関係と呼び、実験や統計的情報をもとに、入力から出力を返すようなものをノンパラメトリックな関係と呼ぶ。このようなパラメトリックな関係、ノンパラメトリックな関係の両方を定義するためには、パラメータとその関係式を別々に定義し、さらに関係式には数式以外の情報を定義できる必要がある。また、以下では数理モデルをコンピュータで評価するモデルという意味で、パラメトリックモデルと呼称する。

相反問題(最適化問題)の分類は、パラメトリックモデルと最適化に関わるいくつかの要

素によってなされる。パラメトリックモデルによって決まる項目は、大きくは、線形か非線形かである。最適化に関わる要素とは、設計変数、制約条件、目的関数である。Table 2.1 に最適化問題の分類[2)]を示す。このことから、本研究では相反問題のモデル(以下では最適化モデルと呼ぶ)として、設計変数、制約条件、目的関数の 3 つを含むモデルと定義することとする。

現状、上記で述べた製品システム全体(機械系、電気系、アプリケーション系、制御系)を定義可能なモデリング手法として 3 つ挙げられる。一つは IDEF(Integration DEFinition)と呼ばれるアメリカ空軍で開発されたモデリング手法、二つ目は Modelica と呼ばれるオブジェクト指向でマルチドメインをモデリングするための言語、そして三つ目は標準化団体である OMG(Object Management Group)の開発したシステムモデリング言語 SysML である。

IDEF[3-5)]はアメリカ空軍の ICAM(Integrated Computer-Aided Manufacturing)と呼ばれるプロジェクトの中で開発されたもので、その目的は、航空機の調達の際に多数のメーカに対して部位別に分割して発注するために、全体の中でその特定のメーカへの発注分がどんな位置づけにあり、他社との関係がどうであるかを可視化するためであったと言われている。Table 2.2 に IDEF の図とその目的を示す。IDEF は何をモデル化するかによって Table 2.2 に示すように IDEF0 から IDEF14 まで存在している。現在は主に BPA(Business Process Analysis)でビジネスプロセスを可視化するためや、システムの機能・構造の定義に用いられており、その際に一般的には IDEF0、IDEF1X、IDEF3 を用いてモデル化が行われている。

Table. 2. 1 Classification of an optimization problems.

Classification conditions	Classification		Optimization problem
Design space	Linear		Linear programming problem
	Non-linear	Convex	convex quadratic program problem, etc.
		Non-convex	Non-convex quadratic program problem, etc.
Design variable	Continuous value		Continuous optimization problem
	Discrete value		Discrete optimization problem
Objective function	Single		single objective optimization problem
	Multi		Multi objective optimization problem

Table 2.2 List of IDEF diagrams.

	Outline
IDEF0	**Function Modeling**
IDEF1	**Infromation Modeling**
IDEF2	**Simulation Modeling**
IDEF1X	**Data Modeling**
IDEF3	**Process Description Capture**
IDEF4	**Object-oriented Design**
IDEF5	**Ontology Description Capture**
IDEF6	**Design Rationale Capture**
IDEF7	**Information System Audit Method**
IDEF8	**User Interface Modeling**
IDEF9	**Scenario-driven Info Sys Design Spec**
IDEF10	**Inplementation Architecture Modeling**
IDEF11	**Information Artifact Modeling**
IDEF12	**Organization Modeling**
IDEF13	**Three Schema Mapping Design**
IDEF14	**Network Design**

具体的には、IDEF は組織の機能や組織間の流れをモデル化し、簡単なシンボルで表す方法で、IDEF0 はシステムにおける決定、動作、活動のプロセスをトップダウンで階層的に詳細化してモデル化する手法、IDEF1X はリレーショナルデータベースの論理設計のためのデータモデル化手法、IDEF3 は IDEF におけるプロセスモデル詳細記述のための手法である。実際のモデル化はこれらの図を関連させて記述することによりモデルの記述を行っている。Fig. 2.4 に IDEF1 で構造を定義した例を示す。

Modelica[7-9]は、Modelica Design Group により開発されたオブジェクト指向型システムモデリング言語である。Modelica のモデリングは、微分・代数・離散方程式などの数学的な式をベースにシステムの要素をグラフィカルに表現するものである。Fig. 2.5 にモデリングの一例を示す。Modelica の特徴は、他のシステムモデリング言語とは異なり、表記法が式とその接続のみで表現されるため、複雑な表記法を習得する必要がなく、また、モデルの出力も式と接続関係のみであるため、容易にシミュレーションソフトウェアと連携できる点が挙げられる。ただし、式とその接続関係のみの表現であるために、ソフトウェアなど

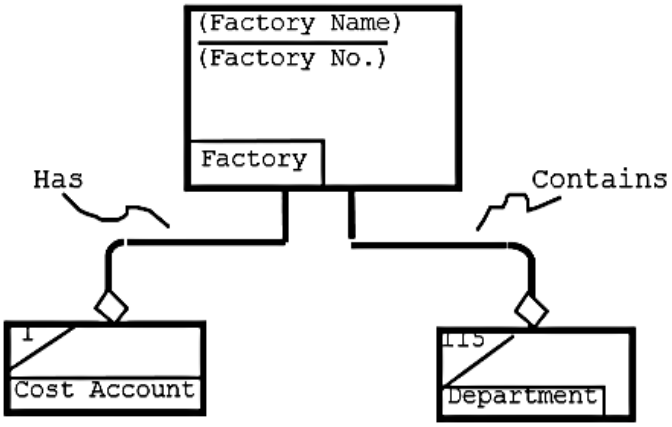


Fig. 2.4 Example of entity structure defined by IDEF1[6].

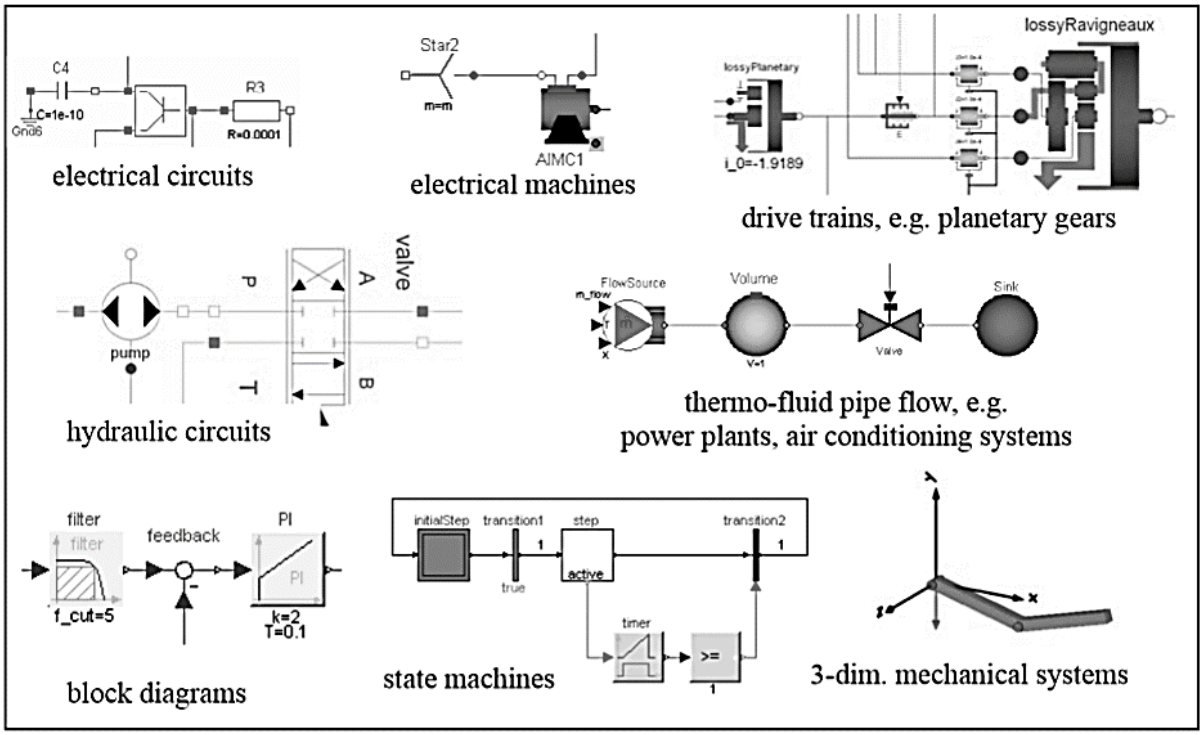


Fig. 2.5 Example of Modelica elements[7]

の物理的本質をもたず、式として表現できないものや、要求などの概念的なものに関しては表現できないといった欠点もある。

SysML[10-12]は、ハードウェア、ソフトウェア、データ、人、手続、設備を含むシステムの仕様化、分析、設計、確認、検証をサポートする目的で開発されたモデリング言語であり、システムの要求、振舞い、構造及び統合を工学的成果物を反映させつつモデル化するため

に必要な意味的基盤とその図式表現を提供している。SysML はソフトウェアのモデリングのために開発された言語である UML(Unified Modeling Language)[13,14]を基本としながら、ソフトウェア表記に特化した、システムを表記するために必要のない図を削り、また逆にソフトウェア表記には必要のない、システムに必要な図を追加し、構造、振舞い、要求、パラメトリックの 4 種の計 9 つの図でシステムの表記を可能にしている。Fig. 2.6 に SysML の全体像を、Fig. 2.7 に SysML でのシステムの定義例を示す。

この様に、3 つの代表的なシステムモデリング手法を示した。これらをさらに、システムデザインを行う上で必須と考えた、構造モデル、パラメトリックモデル、及び最適化モデルに関して定義が可能であるかを調べた。その結果を一覧表に示したものが Table 2.3 である。表に示すように、IDEF はパラメトリックモデル、最適化モデルが定義できず、Modelica はその性質上、パラメトリックモデル以外の構造モデル、最適化モデルが定義不可、SysML は最適化モデルが定義不可能であることがわかった。しかし、この中で SysML は、構造モデル、パラメトリックモデルの定義が可能であるとともに、独自の拡張手法を内包しており、不足している最適化モデルを、拡張手法を用いて定義できる可能性があり、最も実現可能性が高いと考えられるため、本研究では SysML をシステムプロファイル定義で用いる手法として選択することとした。さらに、最適化モデルに関しては、本研究の目的の一つであり、詳細な検討を行った結果、組込むことができた。その詳細は次章で述

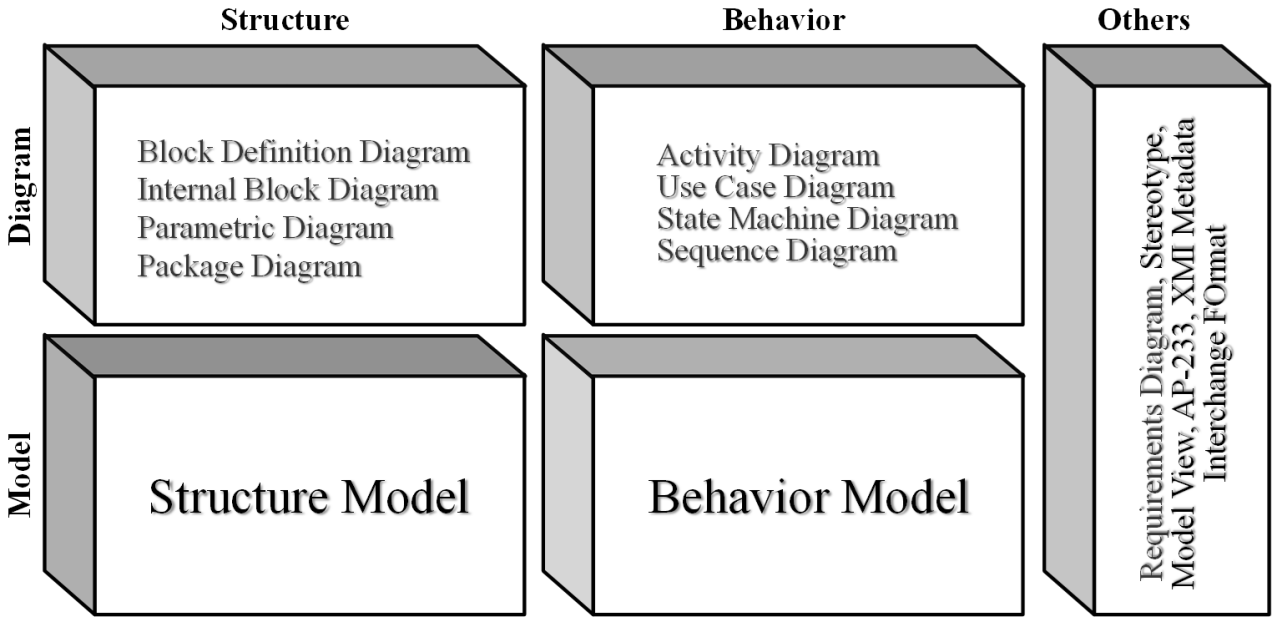


Fig. 2.6 The structure of SysML[15].

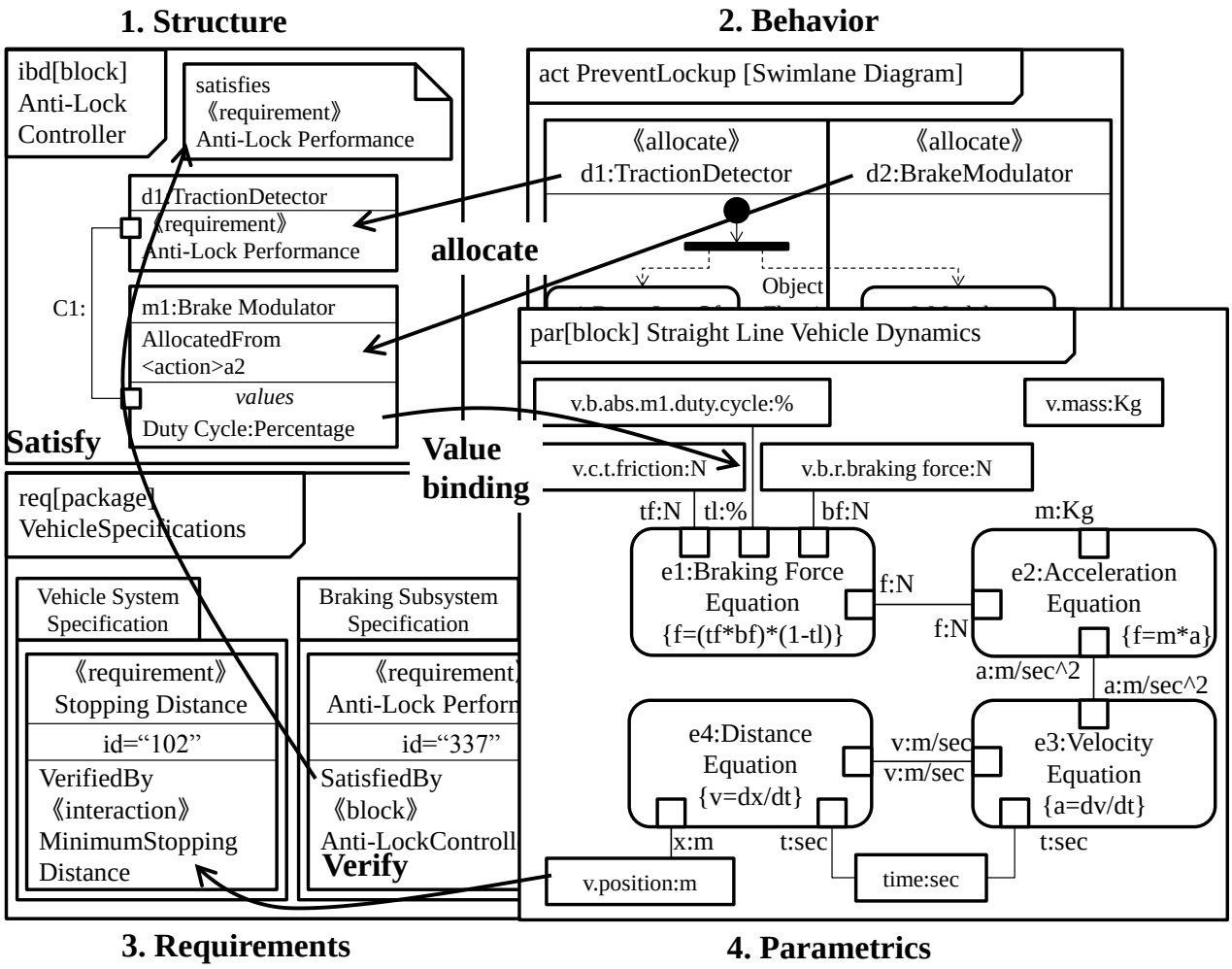


Fig. 2.7 The model example defined by SysML[16].

べる。以下では、SysML の詳細と、SysML を利用したモデリングプロセスを述べる。

SysML は、システム全体を統一した言語で表現し、連携処理を行うため、①要求、②振舞い、③構造、④パラメトリックを記述する合計 9 つの図から構成されている。9 つの図とは、①要求を記述するためのユースケース図(ステークホルダなど外部からのシステムへの要求を記述)、要求図(非機能要求を含めたシステムへの要求全てを記述)。②振舞いを定義するためのアクティビティ図(データの流れをフロー形式で記述)、シーケンス図(システム内の要素間の相互作用を記述)、ステートマシン図(状態とその遷移を記述)。③構造を記述するためのパッケージ図(要素や視点を整理するための図)、ブロック定義図(システムの構造を記述)、内部ブロック図(システムやサブシステムの内部構造を記述)、④パラメトリックモデルを記述するためのパラメトリック図(システムの制約を記述)である。また、これ

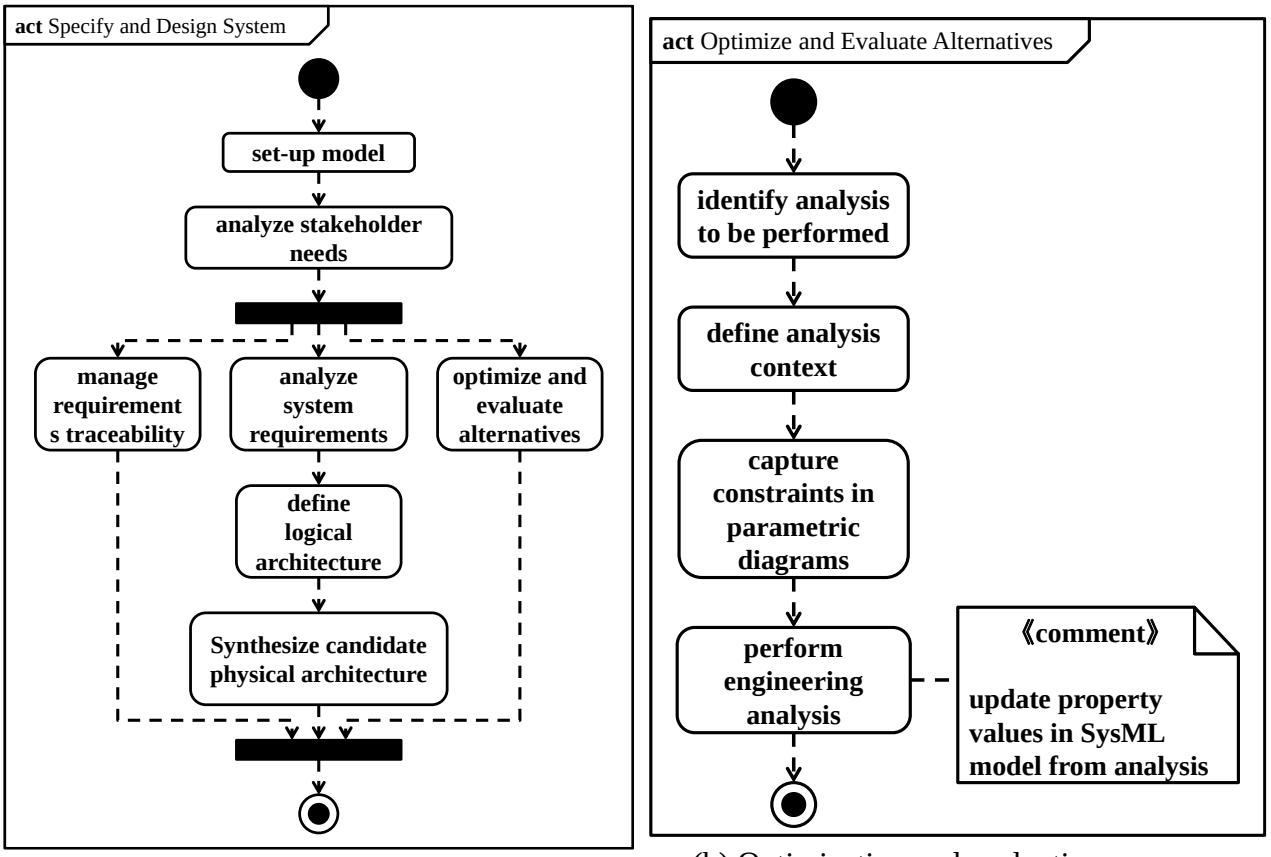
Table. 2. 3 Comparison among SysML and IDEF, Modelica.

	Structure model	Parametric model	Optimization model
SysML	○	○	×
IDEF	○	×	×
Modelica	×	○	×

らの図を用いてシステムの定義を行うためのプロセスは多く提案されている。その代表的なものとして INCOSE(International Council on Systems Engineering)の OOSEM(Object-Oriented Systems Engineering Method)で提案されているプロセスを Fig. 2.8(a)に、このプロセス内での評価・最適化部分の詳細なプロセスを Fig. 2.8(b)に示す。このプロセスでは要求から構造モデル、パラメトリックモデルの定義、さらには最適化と代替案の評価を行っている。しかし、このプロセスでは、最適化と代替案の評価は各評価・解析手法単位での最適化は対象としているが、システム全体での最適化は対象としておらず、システム全体の最適化問題をモデル化する検討は行われていない。そのため、要求モデル、構造モデル、パラメトリックモデルに関してはこのプロセスを用いて定義を行い、最適化モデルに関しては、SysML の拡張プロファイルを用いて、新たに最適化問題を定義する図を加え、最適なデザインを行う手法の検討が必要である。最適化モデルを含め、本研究での製品システム全体のモデリングに関しては次章で詳細を述べる。

次に、ここでは、設計タスクの抽出も行う必要がある。すなわち、SysML で定義した製品システムのモデルから第 3 面、第 4 面に連携させるために、製品システム内の膨大な設計項目や依存関係から、適切に設計タスクの抽出を行う必要がある。

設計タスク抽出は、従来の設計では、設計者が自身の勘と経験を活かし行っていた。その際に考慮されることとしては、会社による組織構造の制約が挙げられるが、それがない新規製品システムである場合は、設計ドメイン間の依存関係を考慮し、可能な限り独立となるように設計タスクは抽出されていたと考えられる。しかし、製品システムの大規模化に伴い、この方法のシステム化・デ



(a) Whole process (b) Optimization and evaluation process

Fig. 2.8 Typical system modeling process[10].

ジタル化が必須となっている。

設計ドメイン間の依存関係を考慮し、可能な限り独立となるように設計タスクを抽出するためには、パラメータによる依存関係を適切に整理・把握し、相互依存関係を可能な限り設計タスク間に残さないように設計タスクを決定する必要がある。そこで本研究では、パラメータネットワークの表現とその解析を行うためのソフトウェアである、古賀らが提案しているクリシュナシステム [17,18]を用いて設計タスク抽出を行うこととした。

クリシュナシステムは、独自のパラメータネットワーク表現とDSM(Design Structure Matrix)[19]と呼ばれる、グラフ理論を行列で表現する手法を用いた数学的処理を行うことでパラメータ間の関係を整理する手法である。本研究では、この手法の数学的処理として、パラメータ間の相互依存関係を抽出し、設計タスクとしてまとめるための方法を導入することとした。具体的には、製品システム内の設計項目間の関係式を一つの設計行為とみなし、依存関係の強さを主因子分析により分析し、クラスタリングを行うことで、複数の設計行為を一つにまとめた設計タスクを抽出することとした。クラスタリングには、主因子分析によりパラメータ間の依存関係の強さを導出し、導出された依

存関係の強さと SysML で定義された構造でクラスタリングすることにより、設計タスクとしてまとめることとした。また、以降ではこのような一連の処理を、設計タスク抽出手法と呼ぶ。Fig. 2.9 に主因子分析から設計タスク抽出までの流れを示す。

Fig.2.9 では、SysML で定義されたパラメトリックモデルをもとに主因子分析を行い、パラメータ間の依存関係を抽出する。抽出された依存関係に対し、特定の閾値でクラスタリング(以下、機能によるクラスタリング、と呼ぶ)を行う。その後、構造モデルによるクラスタリング(以下、構造によるクラスタリング、と呼ぶ)結果とマージを行う。マージはそれぞれのクラスタリング結果を加算的に組み合わせることで行う。また、マージの際には、「機能によるクラスタリング」結果をそのままマージすると、小規模な設計タスクが多数形成される可能性があり、設計時間の増加が懸念される。そのため、特定の閾値以下の規模のクラスタは、マージの際に設計タスクとしては抽出しないようにした。

Fig. 2.10 に実装後の設計タスク抽出手法の手順を示す。実装は opentalk を用いて行っており、以下の手順を踏むことで自動的に設計タスクの抽出を行っている。①SysML で定義されたモデルの情報の読み取り(構造モデルとパラメトリックモデル)、②主因子分析の実行、③主因子分析

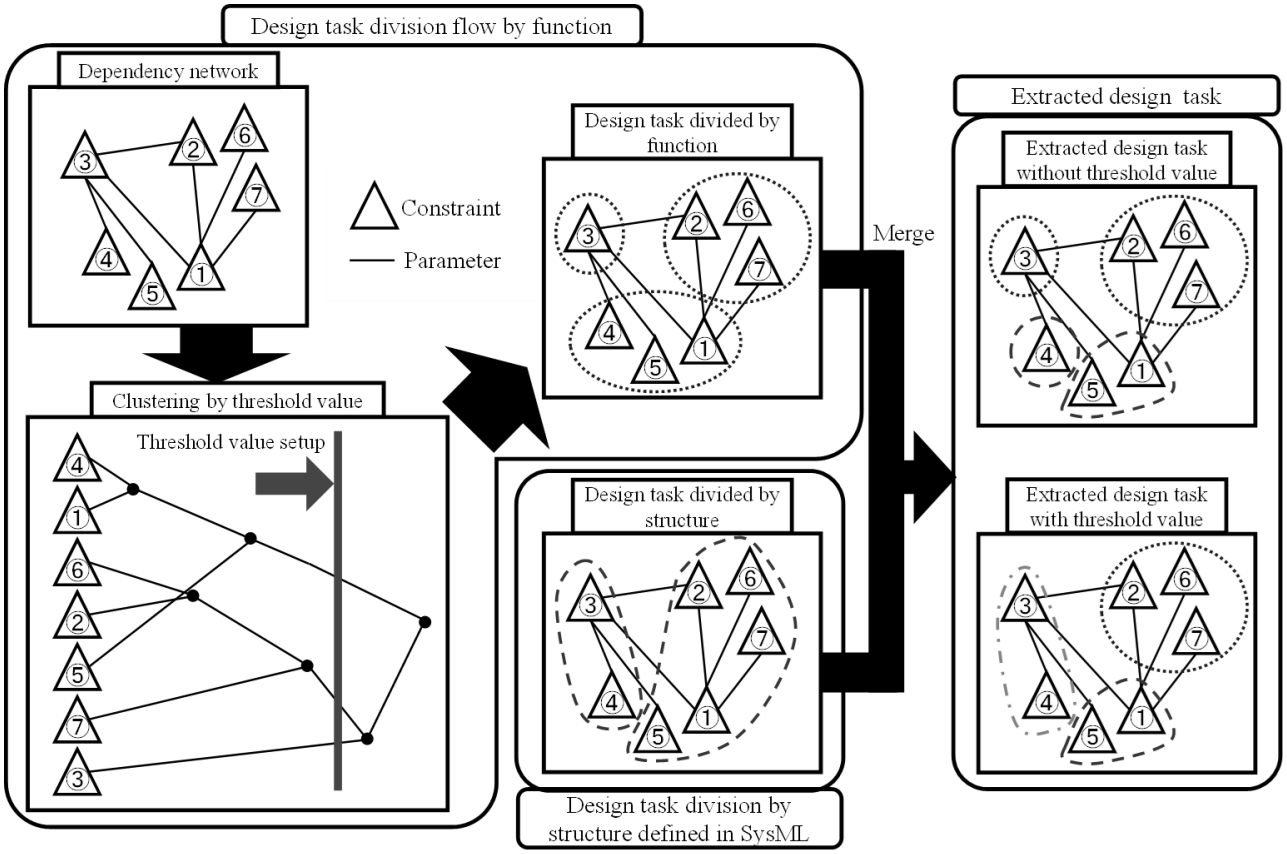


Fig. 2.9 Flow of design task extraction method.

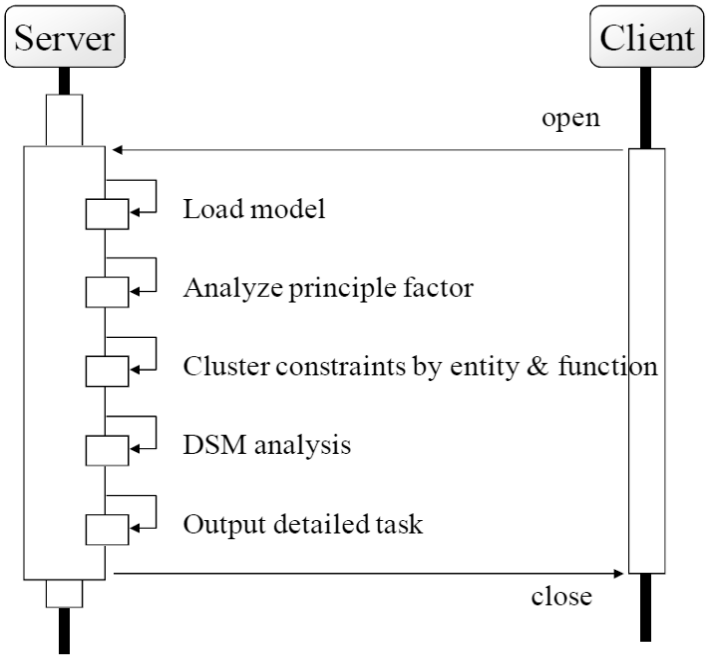


Fig. 2.10 Process flow of design task extraction method.

の結果とSysMLの構造モデルをもとにクラスタリング実行、④DSMにより設計タスクと依存関係の整理、⑤設計タスクの出力、の手順である。

このように、システムデザインを行う上で必須である、製品システムのモデリング、設計タスクの抽出をSysMLとクリシュナシステムを組み合わせることで、初めて可能とすることができた。これにより、システムデザインのためのシステムプロファイル定義が可能となった。

(3) 第3面(評価手法の構築)

第2面でSysMLを用いて製品システムのモデルを定義し、設計タスクを抽出した後、この面では、定義されたパラメトリックモデルをもとにパラメトリック・ノンパラメトリックな関係を評価可能な状態にする必要がある。この面では、そのための検討を行った。

まず、パラメトリックな関係の評価方法に関して述べる。パラメトリックな関係には大きく2つの種類がある。一つは、電卓や表計算などでも導出可能な単一解の存在する関係式の場合である。もう一つは、微分・積分方程式などの解が一意に決まらない関係である。

前者に関しては、拡張性や他のソフトウェアとの連携のしやすさから表計算ソフトウェアを用いるのが一般的であり、本研究でも同様に行う。後者に関しては、関係式に対応した解析ソフトウェアを用いる必要がある。しかし、関係式に応じて自動的に適切な解析ソフトウェアを選択し、評価

手法の構築を行うことは、関係式の多様さから困難であり、また関係式一つに対し、対応するソフトウェアの選択肢は複数あることから、本研究では、以下のような方法での実装を考える。

① 連携手法を使用するソフトウェアに合わせて追加できるように、ソフトウェアの入力となる関係式と、出力となる関係式の評価結果を、テキストフォーマットで作成する。これにより、テキストフォーマットの書式に関しては規定しておくことで、入出力ファイルとソフトウェアの連携部分を随時追加することにより、様々なソフトウェアと連携可能となる。

② ノンパラメトリックモデルを扱う際に一般的に用いられる、応答曲面[1,2)]を利用する。これは即ち、関係式の入力パラメータに対し、ソフトウェアを使用し評価した結果である出力パラメータをあらかじめ用意しておき、それをデータベース化しておくことでソフトウェアを介さずに評価を行う方法である。この方法は入力パラメータに対する出力パラメータのデータベースと、そのデータ群を補完するための簡単な補完式で評価できる。この場合は、本研究では、データベースと補完式を簡単に定義できる表計算ソフトウェアを用いて、評価を行うこととする。

次にノンパラメトリックな関係の評価方法に対して述べる。ノンパラメトリックな関係は、上記の②と同様、応答曲面を利用した方法をとり、実装も表計算ソフトウェアを用いて行うこととする。

式(1)にパラメトリックな関係かつ、解が一意に決まる場合の例を、式(2)に上記の②とノンパラメトリックな関係の際に用いる応答曲面を利用した式の例を挙げる。それぞれの式はシステムLSI(Large Scale Integration)を例としている。

① 性能式(演算性能)

$$P_{system} = P_d \times H_c + P_d \times (1-H_c) \times l_r \quad \cdots (1)$$

P_{system}: 性能(GIPS)P_d：汎用処理回路処理速度(GIPS)、H_c：キャッシュヒット率、l_r：総レイテンシサイクル(cycle)である。

② コスト式

$$C_{Die,Logic} = CoO(k, W_{Cut,Logic}, S_{C,L}, D_{Defect,Logic}) \times N_{Photo,L} + C_{Mask}/V_{Product} + C_{Design}/V_{product} \quad \cdots (2)$$

$C_{Die,Logic}$:ロジックチップコスト、 k:テクノロジノード(nm)、$W_{Cut,Logic}$:ロジックチップ切りシロ(mm)、$S_{C,L}$:ロジックチップ面積(mm^2)、$D_{Defect,Logic}$:ロジックチップ欠陥密度(falts/cm^2)、$N_{Photo,L}$:ロジックチップ露光回数、C_{Mask}:マスクコスト、$V_{Product}$:製造数、C_{Design}:設計コストである。

式(1)はシステム LSI の性能評価式の一部を表しており、これは式の左辺が一意に求まる。式(2)は CoO(Cost Of Ownership)を用いたシステム LSI のコスト評価式の一部を表しており、この中の $CoO(k,W_{Cut,Logic},S_{C,L},D_{Defect,Logic})$が応答曲面を利用したもの部分である。CoO に関しては第 4 章で述べる。この例では、k、$W_{Cut,Logic}$、$S_{C,L}$、$D_{Defect,Logic}$の 4 つの入力パラメータに対し、出力パラメータである CoO のデータベースを用意している。即ち、k、$W_{Cut,Logic}$、$S_{C,L}$、$D_{Defect,Logic}$ という 4 つの入力パラメータの複数の組み合わせに対し、出力パラメータである CoO の計算結果をデータベースとして用意し、足りないデータに関しては線形補完することで補い、応答曲面を作成している。

(4) 第 4 面(設計ワークフロー構築)

この面では、第 2 面のシステムプロファイルの定義より作成された設計タスクを、重要度や相反問題を考慮して最適設計するために、手戻りが少なくなるように、設計順序を整理し、設計ワークフローを決定する検討を行った。

設計順序を整理する手法として、DSM が挙げられる。

DSM は、クリシュナシステムでも表記のために用いている手法であり、設計タスクと、設計タスク間の依存関係を行列化し、この行列にパーティショニングを行って、依存関係の少ない順に並び替える手法 [19,20]である。Fig. 2.11 にタスク間の相互関係表現の例を、グラフ理論との対比で示す。

例として、Task A から Task L までの 11 のタスクを想定し、それぞれに関して Fig. 2.12 の有向グラフに示す相互関係を持っているシステムを考える。このとき、DSM は Fig. 2.13(a)のように書かれる。マトリックス中の「1」は、行の項目から列の項目へ受け渡しが

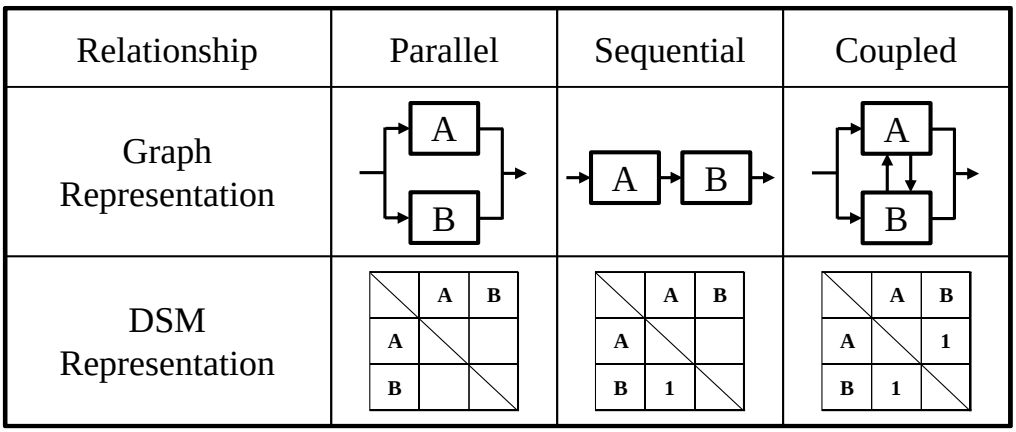


Fig. 2.11 Expression three configurations that characterize a system in DSM[5].

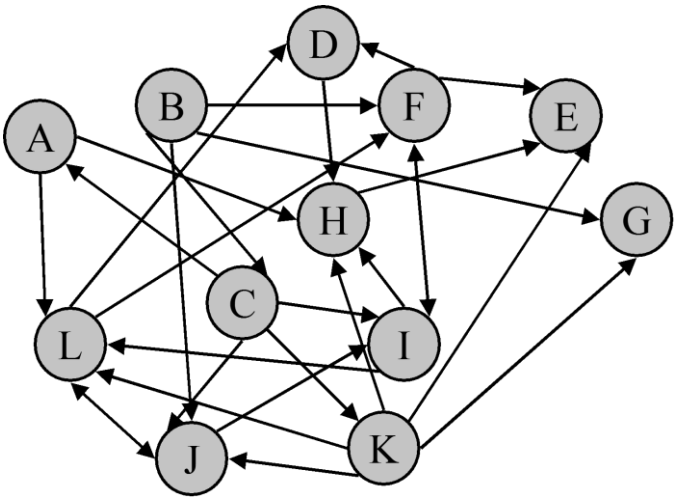


Fig. 2.12 A diagram structure of execution sample for DSM.

あることを示している。ここで、Task D に関して、これを決定するタスクは Task E、 F、L の 3 つあるが、Task D が決定するタスクは Task H の 1 つである。一方で、Task C が決定するタスクは 4 つあるが、Task C を決定するタスクは 1 つしかない。そのため、ここだけを見ると、Task C を Task D より先に決定する方が良いと考えられる。同様にして入出力の関係から、これを整理するために、右上の三角行列の中に存在する「1」、すなわちフィードバックを、できる限り減らすように項目の順番を並べ替える(この処理をパーティショニングと呼ぶ)事によって、フィードバックを減らすことができる(Fig. 2.13(b))。

このように DSM を用いることで実際に設計、製造を行うためのタスクのフローを整理することができ、次の実行段階での設計ワークフローを作成することが可能となる。また、DSM を実行するためのソフトウェアは Web 上で無料で配布されており、表計算ソフトウェアのアドインという形で利用できるので、本研究ではそれを利用することとし、検討を行った。

(5) 第 5 面(最適化実行)

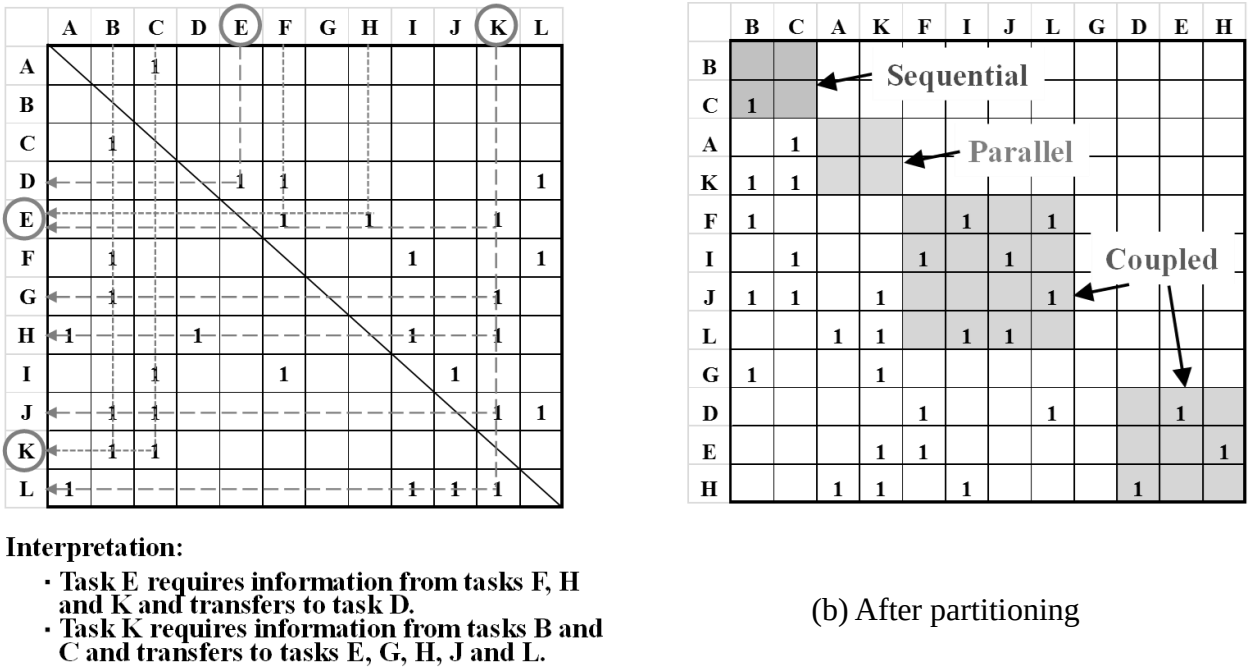
この面では、解の自動探索・最適化の検討を行った。具体的には、第 4 面で作成した設計ワークフローに、第 3 面で作成した製品システムの評価手法を実装し、第 2 面のシステムプロファイル定義で定義した最適化問題をもとに、最適な設計解を探索する検討を行った。

最適化ソフトウェアは、商用ソフトウェアの Isight を用いた。Isight は設計検討の評価やシミュレーションの繰り返し作業を自動化し、これらの自動化された個々のプロセスを統合し、全体シス

テムへ拡張することで、よりよい設計案を探索するソフトウェアである。また、この手法は表計算ソフトウェアやさまざまな CAD(Computer Aided Design)、CAE(Computer Aided Engineering)との連携が可能であり、かつ、様々な実験計画法、最適化手法を実行可能なことから、本システムデザイン手法に対して、最も適していると考え、これを用いて検討を行った。Fig. 2.14 に I-sight で作成されたワークフローの一例を示す。

(6) 第 6 面(出力)

この面では、第 5 面の最適化結果を受けて、必要となる情報(最適解、構造、経済性などの情



(a) Before partitioning　　(b) After partitioning

Fig. 2.13 DSM examples.

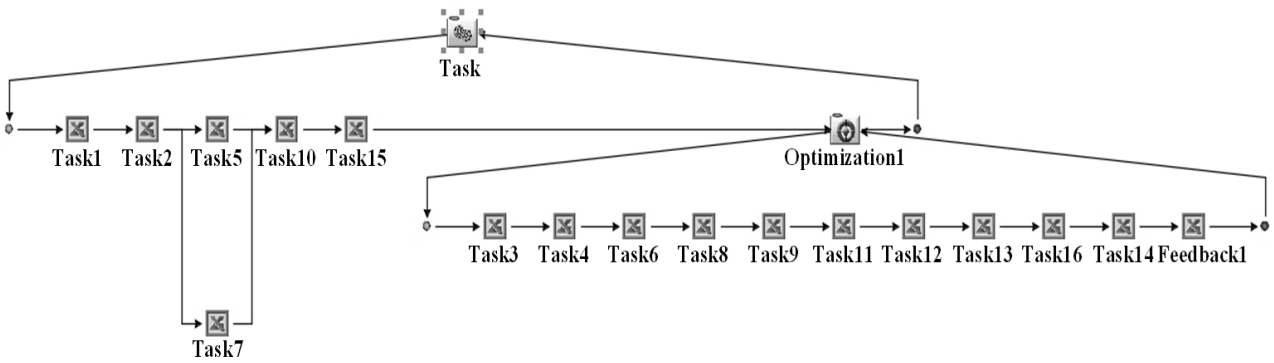


Fig. 2.14 I-sight workflow example.

報)を出力する検討を行った。出力される結果は、設計者が勘・判断力・意思を十分に用いることのできる、感覚的にも論理的にも分かりやすい結果でなければならない。そのため、CAD 図や各種解析の結果、または事業性評価のための累計損益の推移などを出力する検討を行った。

2.4 結果と考察

本章では従来のシステムデザインの課題を解決し、各手法をうまく組み合わせ、製品システム全体を最適化可能なシステムデザイン手法として、6 面からなる SDSI-Cubic を構築した。その結果、各面で必要な手法の明確化、また、各面の連携を行うために必要な情報を明らかにした。Fig. 2.15 に SDSI-Cubic の各面の手法と、連携のための情報を示したフローを示す。

このフローは、①対象とする製品システムの要求から理想構成の推定、②SysML を用いた製品システムのモデリング実行(製品システムの構造モデル(設計タスク抽出で利用)、パラメトリックモデル(設計タスク抽出、評価手法構築で利用)、最適化モデル(最適化実行で利用))、③設計タスク抽出実行(抽出された設計タスクを出力、設計順序の決定、評価手法の構築で利用)、④設計タスクごとの評価手法構築(設計タスクごとの評価手法を出力)と⑤DSM の作成とパーティショニング実行による設計ワークフローの構築(設計ワークフローを出力)、⑥最適化ワークフロー構築と実行、⑦最適解及び最適理想構成などを出力、という⑦つの手順を行い、⑦の実行後、最適解が存在しない場合は①に戻り、最初の要求から理想構成の推定を見直し、①から⑦を繰り返す、という手順を示している。

以上から明らかなように、本手法は、従来手法(Fig. 1.2)に比べ、個別最適解のすり合わせから製品システム全体の最適化を可能とするとともに、従来は多岐にわたる手戻りであったものを、最初の理想構成に戻る手戻りのみにした単純明快な手法であることから、これまでの課題を解決する、短期間で全体最適なシステムデザイン手法であると考える。

このようにして、システムデザイン手法を考案したので、次章で明らかにする、連携すべき手法群、連携のための情報を用いて、実際の連携手法の検討を行うとともに、SysML により定義された製品システムのモデルから最適化までを自動的に実行するためのアルゴリズムの構築を行う。

2.5 結言

大規模・複雑化する製品システムをデザインするための手法として、製品システム全体

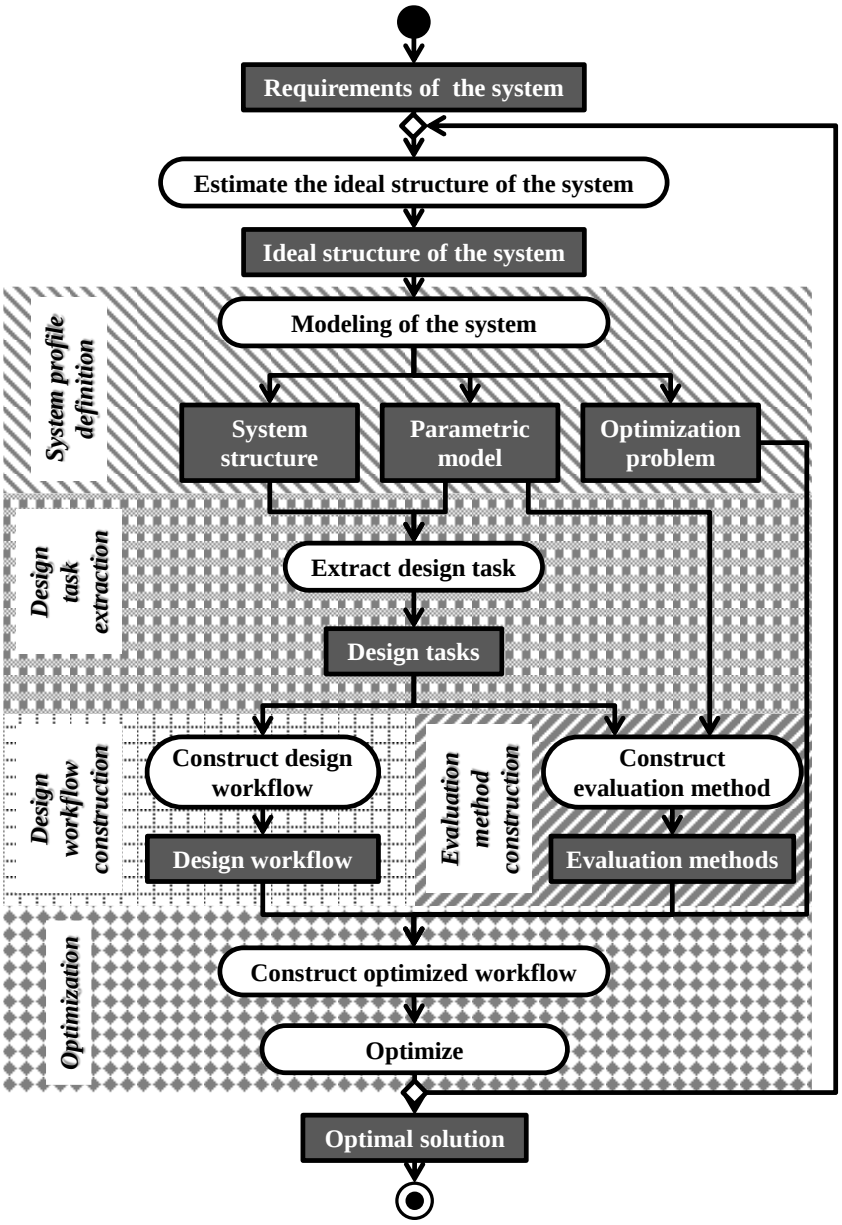


Fig. 2.15 Detailed proposal system design

を捉え、重要設計項目を抽出し、適正設計を行うシステムデザイン手法の検討を行い、以下の結論を得た。

(1) 従来のシステムデザインの課題であるすり合わせは､その原因が､個別のモデルによる、設計ドメイン間の相反問題や依存関係の不明瞭さと、設計ドメイン毎の個別最適化にあることを明らかにした。

(2) この課題に対し、要求や理想構成をもとに、製品システム全体の依存関係をパラメトリックモデルで表現、主成分分析とクラスタリングを組み合わせた設計タスク抽出手

法を用いて分析、設計タスクとして抽出することで、従来の設計ドメイン毎の最適解のすり合わせにもとづくシステムデザインの問題点を解決するシステムデザイン手法を提案した。

(3) 本手法は、製品システム全体の依存関係の定義、すなわちパラメトリックモデルを含むモデリングを行い、その中から可能な限り独立な設計タスクの抽出と、設計タスク間の依存関係を考慮した最適化ワークフローを作成して、最適解を求めることにより、適正設計を行うものである。

本章では、従来のすり合わせに代わる新たなシステムデザイン手法を構築した。次章では、本章で構築したシステムデザイン手法を自動実行するためのアルゴリズムの構築を行った結果を述べる。

参考文献

1) 清水良明, “最適化工学のすすめ ―賢い決め方のワークベンチ―”, コロナ社, (2010).
2) 穴井宏和, “数理最適化の実践ガイド”, 講談社, (2013).
3) Richard J. Mayer, Capt. Michael, K. Painter, Paula S. deWitte, “IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications”, http://www.idef.com/pdf/ideffami.pdf
4) 石橋耀,“IDEF:BPA のための手法”, オペレーションズ・リサーチ:経営の科学, 42-4 (1997) 201- 209.
5) 平澤章,“オブジェクト指向でなぜつくるのか”, 日経 BP 社, (2004).
6) Richard J. Mayer, Capt. Michael, K. Painter, Paula S. deWitte, “IDEF1 Information Modeling”, http://www.idef.com/pdf/idef1mr-part1.pdf
7) Modelica Association, “Modelica - A Unified Object-Oriented Language for Systems Modeling”, https://modelica.org/documents/ModelicaSpec32.pdf
8) Michael M. Tiller, “Modelica による物理モデリング入門”, オーム社, (2003).
9) 大富浩一, 羽藤武宏,“1DCAE によるものづくりの革新”, 東芝レビュー, 67-7 (2012) 7-10.
10) Sanford Friedenthal, Alan Moore, Rick Steiner, “A Practical Guide to SysML”, Morgan Kaufanm, (2009).
11) Mauro Prevostini, “An Introduction to SysML”, DATE -Design,Automation and Test in Europe-Workshop, (2007).
12) Laurent Balmelli, ”An overview of System Modeling Language for Product and System Development –Part1: Requirements, Use-Case, and Test-Case Modeling”, IBM, (2006).
13) Object Management Group, ”OMG Unified Modeling Language Infrastructure”, http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/.
14) 児玉公信, ”UML モデリングの本質”, 日経 BP 社, (2004).
15) Sanford Friedenthal, Alan Moore, Rick Steiner, “OMG Systems Modeling Language Tutorial”, International Council on Systems Engineering, (2008).
16) Tim Weilkeins, “Systems Engineering with SysML/UML: Modeling Analysis, Design”, Morgan Kaufmann, (2008).
17) Akihiro Hirao, Tsuyoshi Koga, Kazuhiro Aoyama, “Planning Support of Initial Design Process

Based on Grouping and Ordering of Tasks – Design Example of an Integrated Circuit", DSM 2010: Proceedings of the 12th International DSM Conference, (2010) 83-96.

18) 中野高聖, 古賀毅, 村田秀則, 岩田剛治, 青山和浩, "マルチドメインの設計解探索のための製品機能と構造に着目した設計プロセスの導出", 日本機械学会第22回 設計工学・システム部門講演会, (2012) 762-771.

19) Steven D. Eppinger, Tyson R. Browning, "Design Structure Matrix Methods and Applications", The MIT Press, (2012).

20) Steward, "D. V. The Design Structure System: A Method for Managing the Design of Complex Systems", IEEE Transactions on Engineering Management, 28 (1981) 71-74.

第3章　システムデザイン手法の自動実行アルゴリズムの構築

3.1　緒言

第 2 章で、製品システム全体の最適解を導出するための、6 面からなるシステムデザイン手法 SDSI(System Design System Integration)-Cubic を構築し、従来の課題であるすり合わせ設計の課題を解決し、製品システム全体の最適化を行う手法の明確化を行った。

本章では、第 2 章で提案した SDSI-Cubic の各面の手法を連携させ、自動的に最適解の導出を行うアルゴリズムの構築を行った。

3.2　自動最適化アルゴリズムの構築

3.2.1　自動最適化アルゴリズム構築のための課題

第 2 章では、以下のように 5 つの項目を考案し、システムデザイン手法 SDSI-Cubic の構築を行ったが、大規模製品システムのシステムデザインを行うためにはこれら各面の手法を連携させて自動実行させることが必要であり、大きな課題である。そこで、まず、各面の目的の整理と自動化に必要な課題の抽出を行った。

(1) システムプロファイル定義: 製品システム全体の依存関係のモデリングを行う。

(2) 設計タスク抽出: 製品システムを定義した SysML(Systems Modeling Language)には膨大な設計項目と依存関係が存在するため、その中から関係性が深い、重要な設計項目(設計タスク)を抽出し、可能な限り独立な設計タスクを抽出する。

(3) 評価手法の構築: コンピュータ上ですべての設計を行うため、製品システムをパラメトリックに扱える様にする。

(4) 設計ワークフロー構築: 抽出した設計タスクには、依存関係、特に相反する項目が含まれている可能性があるので、最も設計の後戻りが少なく、最適な設計が可能となるように、設計タスクの重要度を考慮し、設計の順序を決めることができるようにする。

(5) 最適化実行: 作成した設計ワークフローに従い、構築された設計タスクごとの評価手法を実行し、定義された最適化問題に従い、遺伝的アルゴリズムなどの最適化手法を駆使して、最適化を実行する。

この中で、自動最適化のために連携する必要がある項目は、①(1)システムプロファイル定義と(2)設計タスク抽出、②(2)設計タスク抽出と(3)評価手法の構築、③(2)設計タスク抽

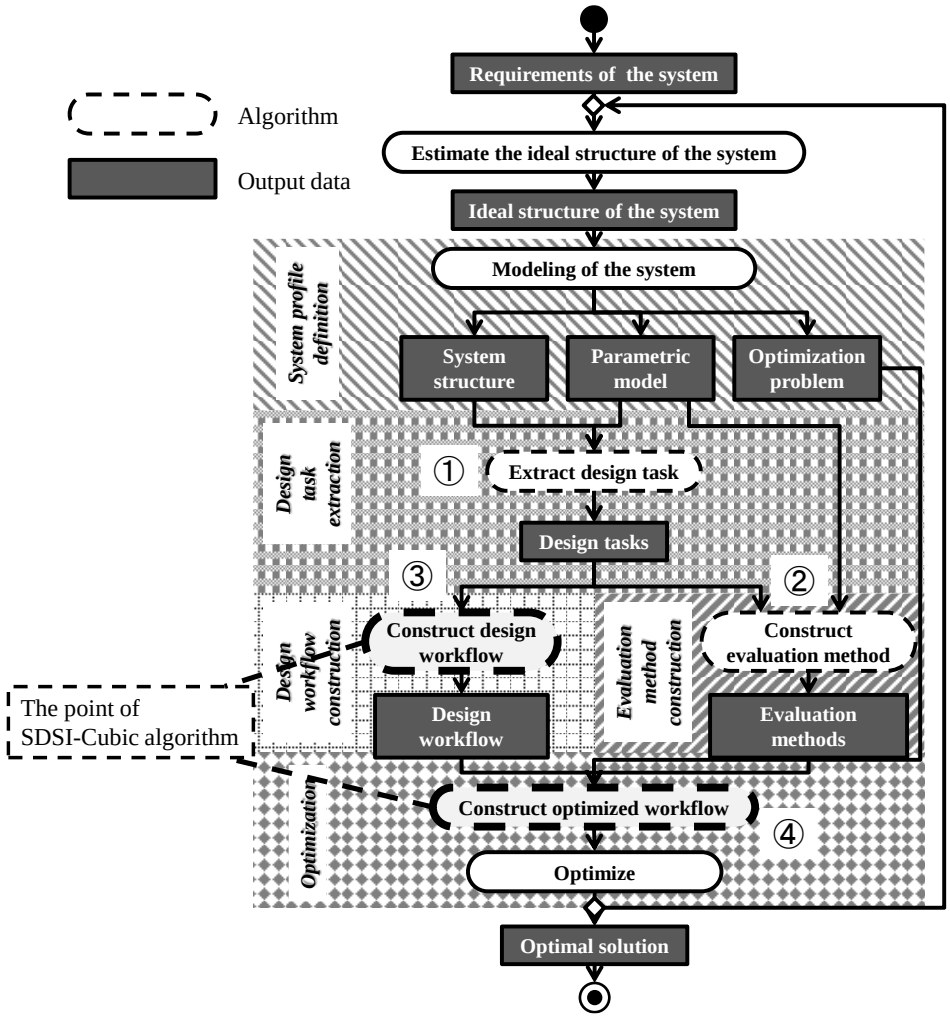


Fig. 3.1 Flow of SDSI-Cubic algorithm.

出と(4)設計ワークフロー構築、④(3)評価手法の構築、(4)設計ワークフロー構築と(5)最適化実行の 4 つである。また、これらの連携のためのシステムのモデリング手法の検討も必要となる。Fig. 3.1 は、これらの構築すべき自動最適化のためのアルゴリズムを含め、SDSI-Cubic 手法のフローを示したものである。以下、順を追って連携の課題について述べる

まず、システムのモデリング手法に関しては、第 2 章で述べたように、従来のシステムモデリング手法には、最適化問題を定義するためのモデルが存在していないのが課題であった。そのため、新たに最適化モデルのメタモデルを構築し、SysML を用いて、システムの最適化モデルを表現する手法の検討が必要である。また、SDSI-Cubic の各手法との連携のための構造モデル、パラメトリックモデルの構築も必要である。①に関しては、既に第 2 章で SysML によるシステムプロファイル定義から主因子分析、クラスタリングを用いて設計タスクを抽出できるように手法を構築したため、他手法との連携の

ための手法構築のみが必要となる。②に関しては、既に、本研究では、パラメトリック・ノンパラメトリックの両方の関係に対応可能とするため、設計タスクごとに、パラメトリックモデルを評価可能な形に表計算ソフトウェアに実装することにより、連携可能とすることができた。そのため、こちらも同様に他手法との連携のための手法構築のみが必要となる。③に関しては、④と関連して設計ワークフローに最適化を定義した最適化ワークフローを構築する必要がある。そのためには、SysML で拡張した、最適化問題としての情報を考慮する必要がある。しかし、DSM(Design Structure Matrix)上では、システムプロファイル定義で定義された依存関係をもとに抽出された設計タスクとその依存関係をもとに手戻りが少ないワークフローを導出するだけであるため、システムプロファイル定義で定義されていない最適化の依存関係を考慮したワークフローの構築は行うことはできない。そのため、この問題点を解決する設計ワークフロー構築のための新しいアルゴリズムが必要となる。④に関しては、③と関連して、最適化ワークフローに最適化問題の定義を加え、最適化を実行可能な形にするための連携手法の構築が必要である。

また、上記に加え、連携を自動的に行うために、各面での入力ファイル・出力ファイルの形式を決定する必要がある。その際に項目を考慮する必要がある。①各面で使用するソフトウェアに関わらず、一般的に必要な項目を包含すること、②様々なソフトウェアと連携するために最も一般的なファイル形式である、テキスト形式で表現できること、である。以上の検討から、SDSI-Cubic で自動最適化を行うための課題は、SysML によるシステムプロファイル定義手法の構築と自動最適化アルゴリズム及び、入出力ファイル形式を決めることであることが明らかになった。以下では、これらの課題に対して検討を行った結果を述べる。

3.2.2 SysML によるシステムプロファイル定義手法と連携手法の構築

3.2.2.1 SysML によるシステムプロファイル定義手法の構築

ここでは、SDSI-Cubic の入力である、理想構成の推定から、製品システムのモデルを定義する必要がある。

製品システムのモデルを定義する場合、Fig. 2.6 に示すように、要求モデル、構造モデル、パラメトリックモデルの順で定義を行う。要求モデルは、システムの要求や仕様を記述するためのモデルであり、製品システムのモデルの定義は、構造モデルやパラメトリックモ

デルが製品システムの要求、仕様を満たしているかを確認することが重要なため、要求モデルと連携をとりながら行う必要がある。最適化モデルも、意思決定を反映したモデルであるため、構造モデル、パラメトリックモデルと同様、要求モデルと連携をとりながら定義を行う必要がある。この点から、要求モデルは製品システムのモデルの核となるモデルであり、構造モデル、パラメトリックモデル、最適化モデルが要求、仕様を満たしているか、即ち、モデルレベルでの検証を行うために、製品システムのモデルの一部として定義することが必要である。

ここでは、まず要求モデルと構造モデルの定義方法を述べる。パラメトリックモデルに関しては、専用の図が用意されているが、SysML と連携させるソフトウェアや手法により細部の定義方法が異なるため、本研究では、SDSI-Cubic の評価手法として表計算ソフトウェアを用いることを前提にパラメトリックモデルの定義手法の構築を行う。最適化モデルに関しては、SysML の拡張プロファイルを用いた定義手法の構築を行う。

(1) 要求モデルの定義方法

まず SysML での要求モデルの定義方法に関して述べる。SysML では要求モデルの定義のために要求図が用意されており、それを補う形でユースケース図、さらに、内部ブロック図をカスタマイズしたコンテキスト図が用いられる。それぞれの図の例を Fig. 3.2 から Fig. 3.4 に示す。

Fig. 3.2 はコンテキスト図であり、これは対象とシステムとその外部を切り分け、明確に定義する図である。Fig. 3.2 では、「Target System」が設計対象のシステムであり、それ

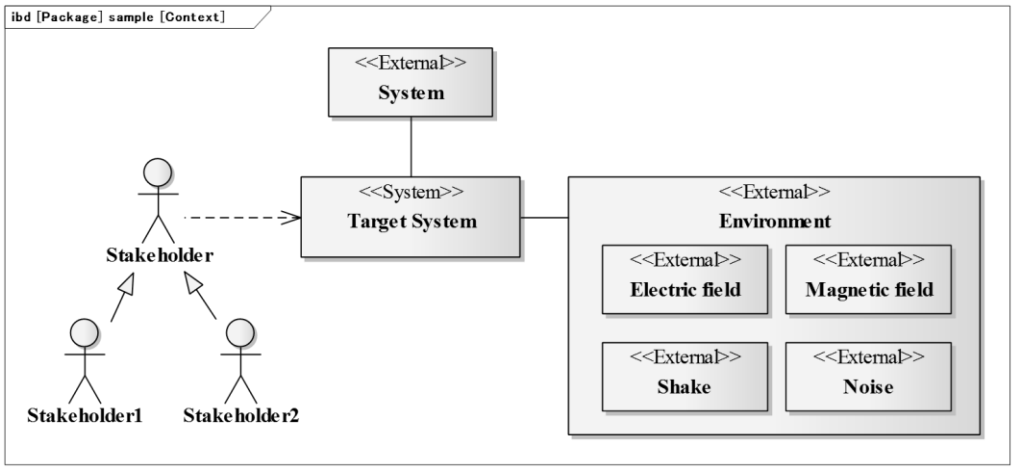


Fig. 3.2 Context diagram example.

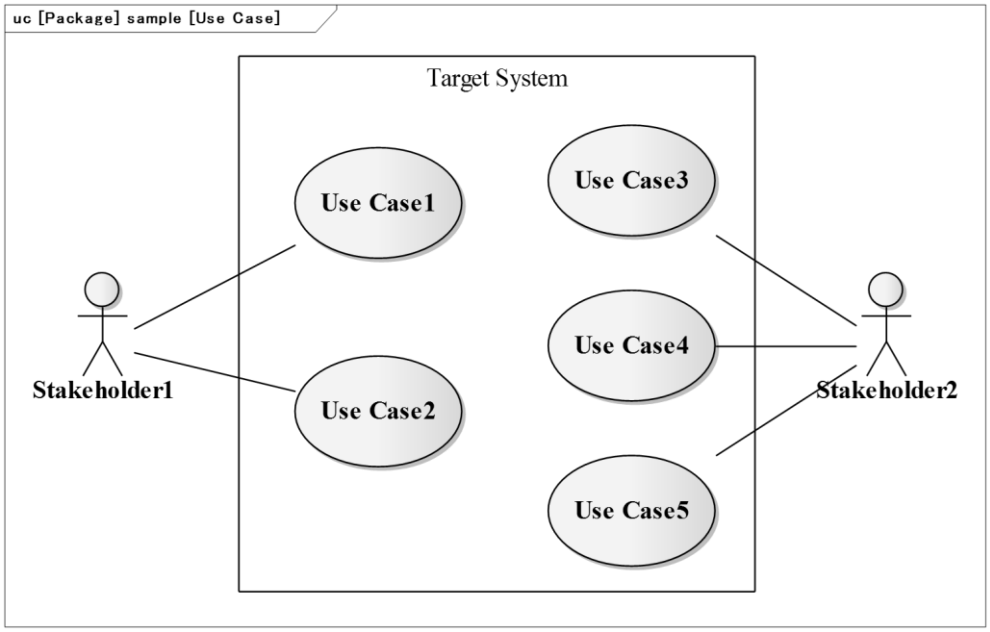


Fig. 3.3 Use case diagram example.

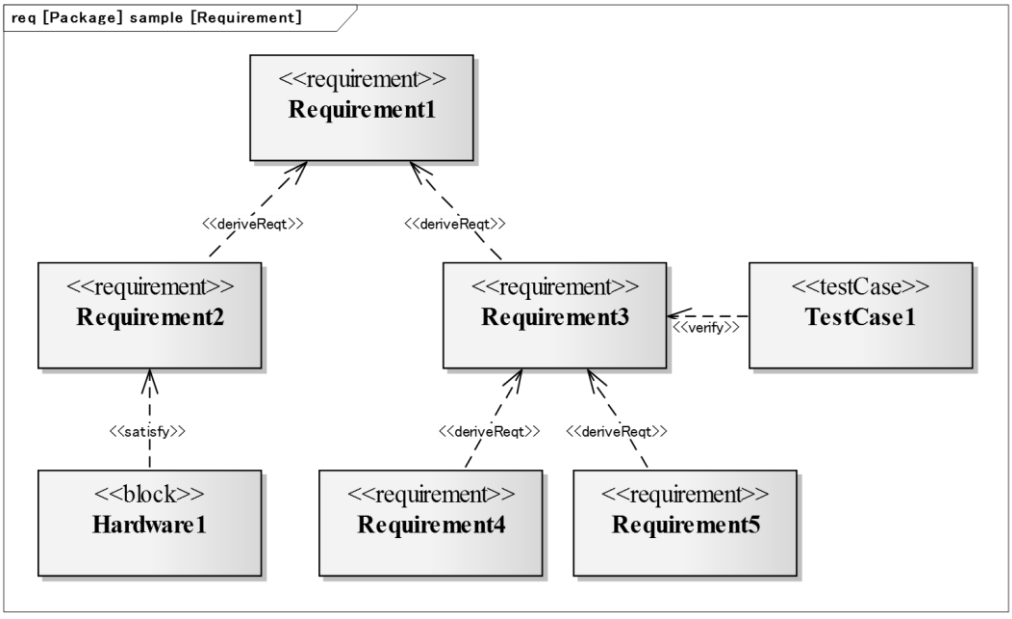


Fig. 3.4 Requirement diagram example.

以外が外部である。外部の例としては、ステークホルダ(図中の「Stakeholder」、「Stakeholder1」、「Stakeholder2」)や電磁場・振動などの外部環境(図中の「Environment」)、対象とするシステムを包含する上位のシステム(図中の「System」)などが挙げられる。

コンテキスト図を定義した後はユースケース図の定義を行う。ユースケース図は、ステー

クホルダが対象システムに対して要求する事項(ユースケース)を定義するための図であり、Fig. 3.3 に示すように、ステークホルダ毎の要求をユースケースとして記述する。図中では、「Stakeholder1」に 2 つのユースケース、「Stakeholder2」に 3 つのユースケースが定義されている。

これらのコンテキスト図、ユースケース図の情報をもとに、Fig. 3.4 の要求図を定義する。要求図は対象とするシステムに対する全ての要求を定義するための図であり、機能要求だけでなく、非機能要求も定義することができる。定義の際には、要求は階層的に記述し、要求の導出関係を明確に表現する。さらに、要求は構造モデル、パラメトリックモデル、最適化モデルが要求、仕様を満たしているかを検証するための機能を有しており、要求に対するテストケースの定義や、「satisfy」という関連がそれにあたる。図中では「Requirement1」から「Requirement5」が要求項目であり、「Testcase1」がテストケース、「Hardware1」が構造モデルのハードウェアの一部を表している。また、図中の「deriveReqt」と示された矢印が要求の階層構造を表現し、「verify」と示された矢印が要求に対するテストを、「satisfy」と示された矢印が、要求を構造が満たしていることを示している。上記のように、要求だけでなく、様々な要素、関連(図中の矢印)を用いることで、他のモデルとの検証を行い、整合性をとることができるようになっている。

このように、3 つの図を用い、上記の手順で定義することで要求モデルは定義できる。

(2) 構造モデルの定義方法

構造モデルは、SysML のブロック定義図を用いて定義を行う。Fig. 3.5 に一例を示す。

構造モデルでは、ハードウェア・ソフトウェアの両方を定義する必要がある。SysML のブロック定義図では、Fig. 3.5 のように、ブロックという構成要素を用いて階層的に表現することでシステムとサブシステムの関係を表現する。その際にブロックはハードウェアの一つの部品、または、ソフトウェアの一つのプログラムやオブジェクトを表現でき、これにより、製品システムをハードウェア・ソフトウェアの両面を同一の図で定義できるようになっている。図中では「Hardware1」から「Hardware4」がハードウェア、「Software1」、「Software2」がソフトウェア、「Object1」、「Object2」がオブジェクトを表している。これらのうち、「Hardware1」、「Harware4」、「Software1」、「Software2」は「Target system」と直接矢印で結ばれており、これは「Target system」のサブシステムであることを示している。ま

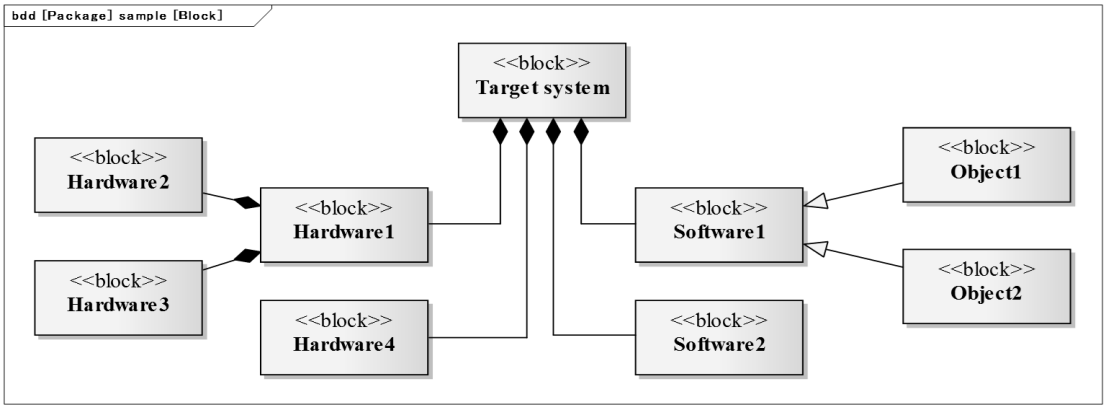


Fig. 3.5 Block definition diagram example.

た、「Hardware1」と「Hardware2」、「Hardware3」の関係も同様である。「Object1」、「Object2」は「Software1」と先ほどとは異なる矢印で結ばれており、これは、「Software1」を実現するためのオブジェクトが「Object1」、「Object2」であることを示している。このように、一部情報が異なるが、基本的にハードウェアとソフトウェアはブロック定義図を用いて同様に定義を行うことができる。

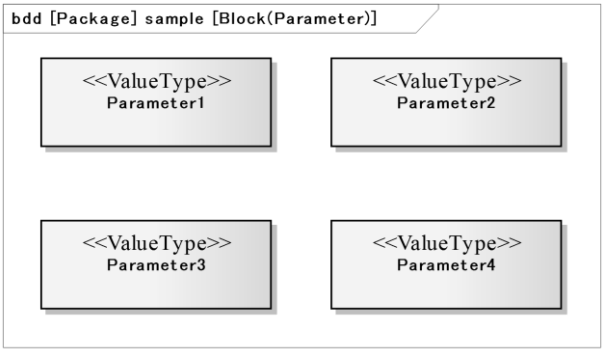
(3) パラメトリックモデルの定義方法

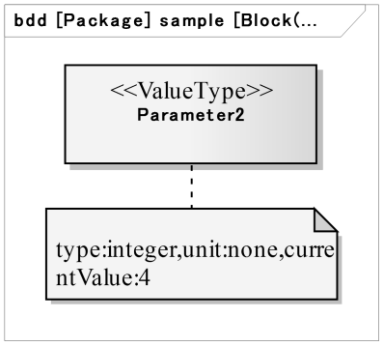
次にパラメトリックモデルの定義方法に関して述べる。パラメトリックモデルはブロック定義図、内部ブロック図、パラメトリック図の 3 つの図を用いて表現される。Fig. 3.6 から Fig. 3.9 に一例を示す。

パラメトリックモデルは、パラメトリック・ノンパラメトリックのどちらの関係も考慮するために、パラメータと関係式(データベース引用を含む)を別々に定義する必要がある。

SysML では、パラメータと関係式を定義する際に 2 つの手順を踏むことによりモデルを作成する。最初の手順は、ブロック定義図を用いて型を定義する手順、次の手順は、可視化の手順である。最初の手順で定義する型とは、パラメータや関係式の性質を定義するためのものである。型の定義方法、即ち、どのような性質を定義するかは SysML では決められておらず、プロセスに依存する。そのため、SysML と連携させるソフトウェアや連携方法によって定義方法を決める必要がある。

本研究では、最適化や設計タスク抽出・表計算ソフトウェアと連携することを考慮し、パラメータの性質として、数値の型(type)、単位(unit)、初期値(currentValue)の 3 つの項目を定義することとした。()内は実際の定義に使用した項目名である。数値の型は、評価を





(a)Parameter type definition example. (b)Parameter property definition.

Fig. 3.6 Parameter type definition example.

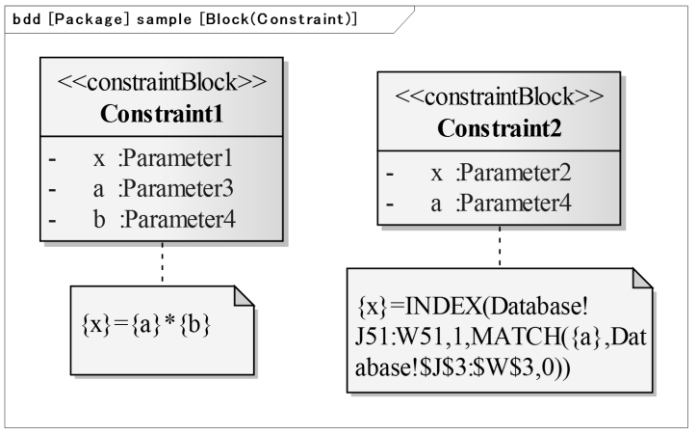


Fig. 3.7 Constraint type definition example.

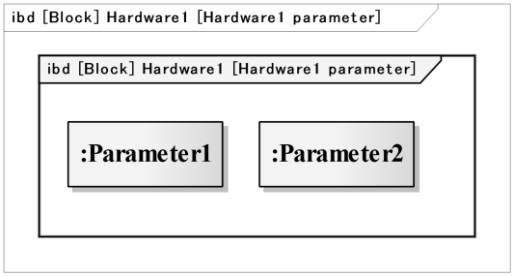


Fig. 3.8 Internal block definition diagram example.

行う際にそのパラメータが取りうる値であるかを判断するため、単位は基本的なパラメータの情報として、初期値は、定数をとるパラメータを定義するためである。Fig. 3.6(a)の「Value Type」と書かれた要素の「Parameter1」から「Parameter4」までがパラメータの型であり、Fig. 3.6(b)の「Parameter1」と「Parameter2」に接続されているテキスト部分がそのパラメータの性質定義である。「Value Type」はパラメータの型を表す要素である。

関係式の性質としては、表計算ソフトウェアで評価することを考慮し、パラメトリックな関係、かつ評価式がある場合は、対応する式を、そうでない場合、ノンパラメトリック

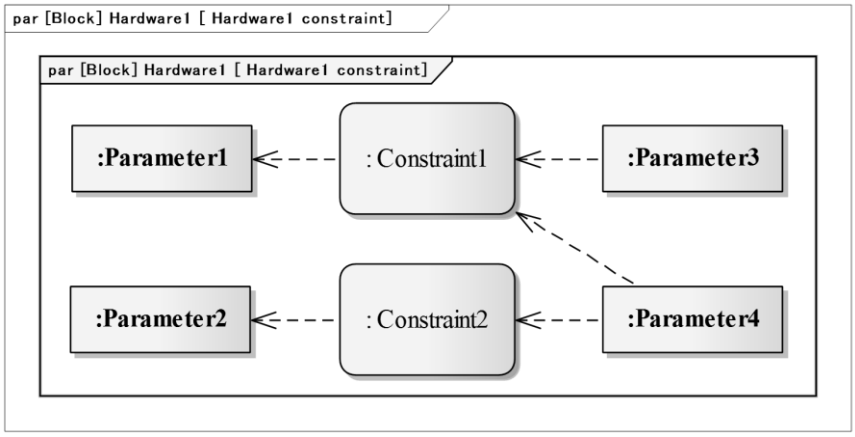


Fig. 3.9 Parametric definition diagram example.

な関係の場合は、データベースの引用を行うための式を定義する。Fig. 3.7 は「constraintBlock」と書かれた要素の「Constraint1」と「Constraint 2」が関係式の型を表したものであり、それぞれに接続されているテキスト部分がその関係式の性質、即ち式を表している。Fig. 3.7 では、「Constraint1」がパラメトリックな関係を定義したものであり、その式で用いるパラメータは先ほど定義したパラメータの型を利用し、「Constraint1」内に定義されている。x:Parameter1 以下 3 つのパラメータがそれに当る。また、「Constraint2」がノンパラメトリックな関係を定義したものであり、テキスト内にはデータベース引用式の一例が書かれている。式で用いるパラメータの定義に関しては「Constraint1」と同様である。

パラメータ、関係式の型定義を行った後、可視化を行う。ここでの可視化は、構造モデルとリンクさせ、パラメータや関係式がどの構造に属するものなのかを定義することを指す。可視化には、パラメータは内部ブロック図、関係式はパラメトリック図を用いる。Fig. 3.8 が内部ブロック図、Fig. 3.9 がパラメトリック図の一例である。Fig. 3.8 は、型を定義されたパラメータの内、「Parameter1」、「Parameter2」が Fig. 3.5 の「Hardware1」という構造に定義されていることを表している。Fig. 3.9 は、Fig. 3.5 の「Hardware1」という構造に対して、型を定義された関係式である、「Constraint1」、「Constraint2」の両方が定義されており、また、それぞれの式の入出力パラメータを矢印と Fig. 3.8 で定義されたパラメータを用いて表現している。

このような手順で 3 つの図を用いることによりパラメトリックモデルを定義可能とした。

(4) 最適化モデルの定義方法

自動最適化を実行するためには、これまでの SysML では考慮されていなかった、最適化問題を定義する必要がある。以下では、最適化問題、即ち最適化モデルの SysML での定義方法を検討した結果を述べる

最適化モデルで定義すべき項目は、第二章の検討結果より、設計変数・制約条件・目的関数である。これらは、第二章の検討結果より、現状の SysML では定義する方法はない。ただし、SysML には、ステレオタイプと呼ばれる拡張プロファイル[1,2]が用意されている。

ステレオタイプは、各設計ドメインや解析ソフトウェアなどで用いられる固有の概念を表現するために用意されているものである。これは即ち、現状の SysML にないモデル要素をステレオタイプを用いて加えることが出来るということである。ステレオタイプを用いた拡張の一例を Fig. 3.10 に示す。Fig. 3.10 は、SysML と Modelica を連携させるために OMG(Object Management Group)により公式に作成されたステレオタイプ群である。

本研究では、このステレオタイプを用いて、最適化の概念である、設計変数・目的関数・制約条件を表現するためのステレオタイプと、それを用いた図(以下では最適化図と呼ぶ)の定義方法を新たに構築した。Fig. 3.11 に新たに作成したステレオタイプの一覧を、Fig. 3.12 にこれらを用いて定義した最適化図の一例を示す。Fig. 3.11 は最適化を表す「Optimization」という概念(メタクラス)を新たに追加し、それを表現するための要素として「design parameter」、「objective function」「constraint condition」の 3 つのステレオタイプを追加したことを表している。Fig. 3.12 は、「Parameter1」に「objective function」というステレオタイプが追加されており、このパラメータが目的関数であることを示している。また、「Parameter3」、「Parameter4」には、「design parameter」というステレオタイプが追加されており、このパラメータが設計変数であることを示している。「Parameter5」には、「constraint condition」というステレオタイプが追加されており、このパラメータが制約条件であることを示している。

また、設計変数・目的関数・制約条件の定義の追加に伴い、設計変数であるパラメータに関しては、とりうる値の範囲を、目的関数であるパラメータに関しては、最小化、最大化のどちらの方向に最適化するかという最適化方向を、制約条件に関しては、パラメータに最小値、最大値を定義する必要がある。この定義には、パラメトリックモデルで用いた、パラメータの型の定義に新たな項目を追加することで対応する。

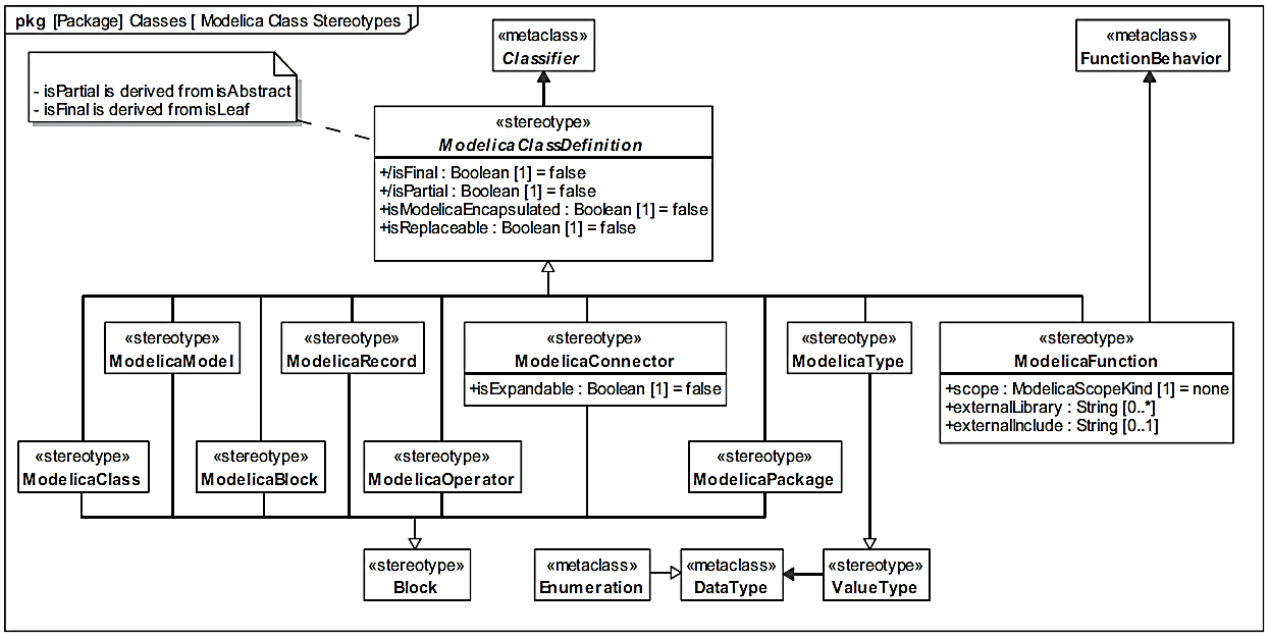


Fig. 3.10 Stereotype example[3)].

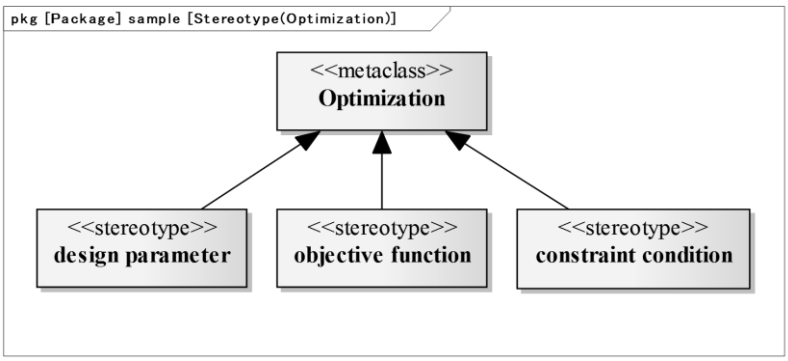


Fig. 3.11 Stereotype for optimization.

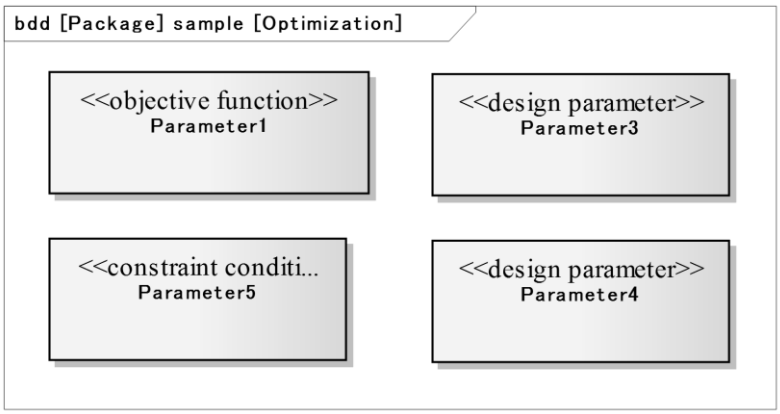


Fig. 3.12 Optimization diagram example.

設計変数に関しては、連続値・離散値の定義(discrete)、連続値である場合は、最小値(min)・最大値(max)、離散値である場合は、どのような値をとるかを定義する。目的関数に関しては最適化の方向(direction)を定義することとする。制約条件に関しては、最小値(min)、最大値(max)を定義する。()内は実際の定義に使用した項目名である。Fig. 3.13 に上記のパラメータの性質を追加したブロック定義図によるパラメータの型定義の例を示す。

このように、新たにステレオタイプとそれを用いた最適化の設計変数、目的関数を定義するための最適化図を構築し、最適化モデルを SysML で定義可能とした。これにより、自動最適化を行うための各モデルの定義を行うことができた。Fig. 3.14 に SDSI-Cubic での自

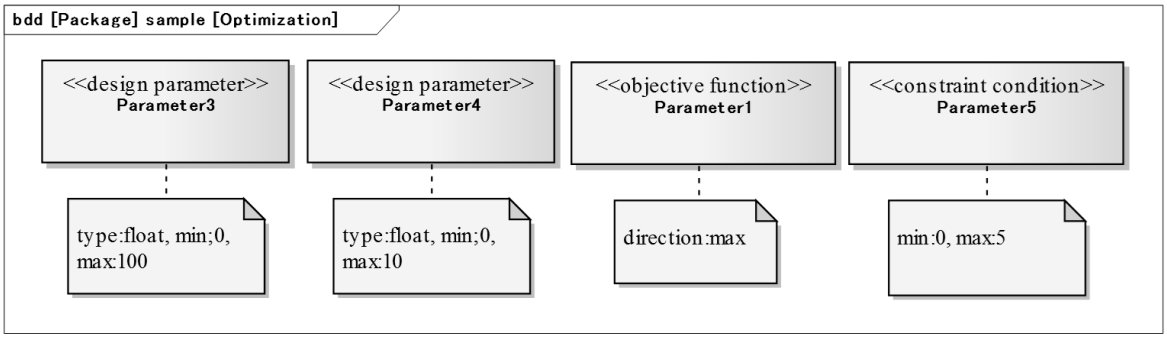


Fig. 3.13 Optimization property example.

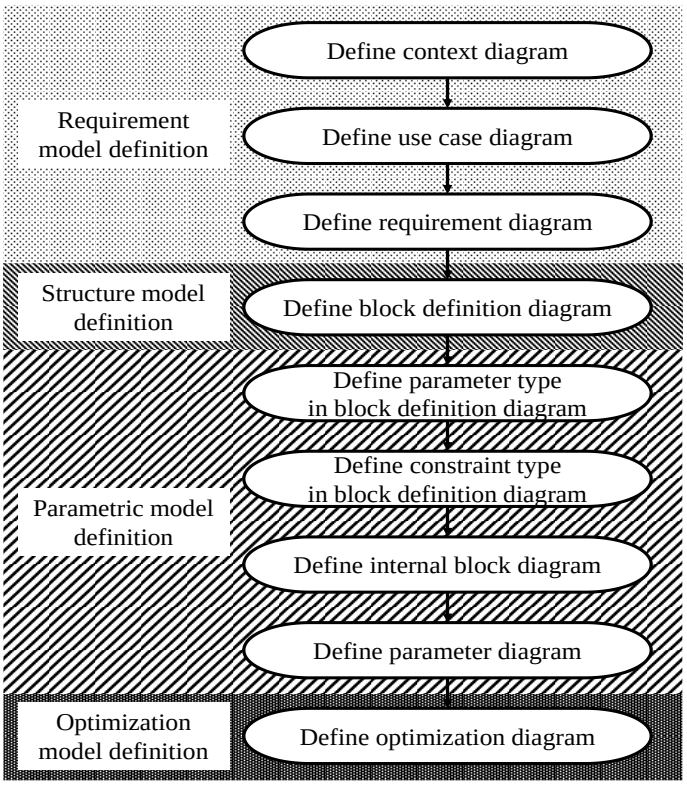


Fig. 3.14 System model definition flow for SDSI-Cubic with SysML.

動最適化を考慮した、SysML による製品システムのモデルの定義手順を示す。

以降の自動化は、上記の製品システムのモデルの内、以下の条件を満たしているという前提で構築を行った。

① 定義された関係式は 1 入力 1 出力、あるいは多入力 1 出力である。
② 設計変数であるパラメータから目的関数であるパラメータを導出するための関係式は定義されている。
③ パラメータの性質を定義するブロック定義図で、すべてのパラメータに対し、数値の型(type)が定義されている。
④ 最適化図で、少なくとも 1 つ以上の設計変数、目的関数が定義されている。
⑤ 最適化図で、すべての設計変数に対し、性質として、最小値(min)、最大値(max)、数値の型(type)、連続値・離散値の定義(discrete)、初期値(currentValue)が定義されている。
⑥ 最適化図で、すべての目的関数に対し、数値の型(type)、連続値・離散値の定義(discrete)、初期値(currentValue)、最適化方向(direction)が定義されている。
⑦ 最適化図で、すべての制約条件に対し、最小値(min)、最大値(max)が定義されている。

3.2.2.2 システムプロファイル定義の連携手法の構築

前節で、自動最適化を考慮した SysML での製品システムのモデルの定義方法の構築を行った。本節では、これらの定義したモデルを具体的に自動最適化に用いるための連携手法に関して検討を行った。

SysML で製品システムのモデルを定義した後、他のソフトウェアと連携させるためには、SysML で定義した内容を、適切なフォーマットでテキストファイルに出力する必要がある。

製品システムのモデルとしては、要求モデル、構造モデル、パラメトリックモデル、最適化モデルの 4 つが定義されているが、これらの内、要求モデルに関しては、製品システムの要求を定義しているだけであり、既に構造モデル、パラメトリックモデル、最適化モデルに反映されているため、出力の必要はない。そのため、ここでは、構造モデル、パラメトリックモデル、最適化モデルの出力フォーマットの検討を行った。

フォーマットを決定するために、それぞれの出力ファイルが、どの手法に用いられるかを明らかにする必要がある。

まず、構造モデルに関して述べる。構造モデルは、次の設計タスク抽出手法のアルゴリ

ズム中のクラスタリングの際に用いられるのみであり、以降は設計タスク抽出手法で出力される設計タスクに置き換えられる。

パラメトリックモデルに関しては、設計タスク抽出手法の主因子分析、第 3 面の評価手法構築で用いられる。ただし、第 3 面の評価手法構築後は、出力された評価手法が最適化実行に用いられるため、以降は必要ない。第 4 面の設計ワークフロー構築では、DSM の構築に設計タスク間の依存関係を定義する必要があるが、その情報は次の節で述べる、設計タスク抽出手法の出力である、設計タスク毎の出力ファイルの情報で作成できるため、第 4 面ではパラメトリックモデルの情報は必要ない。上記から、パラメトリックモデルの出力ファイルは、設計タスク抽出手法と、第 3 面の評価手法構築で用いられる。

最適化モデルに関しては、後に述べる、第 4 面での設計ワークフロー生成アルゴリズムを実行する際と、第 5 面で最適化ワークフローを生成する際に用いられる。

これらの出力ファイルの用途を考慮し、出力ファイルのフォーマットを以下のようにすることとした。Fig. 3.15 に構造モデルの出力ファイルの一例を、Fig. 3.16-Fig. 3.17 にパラメトリックモデルの出力ファイルの一例を、Fig. 3.18-Fig. 3.19 に最適化モデルの出力ファイルの一例を示す。

(1) 構造モデル

構造モデルはブロック定義図で階層的に定義されており、その階層情報を設計タスク抽

Fig. 3.15 Block definition diagram output example.

	A	B	C	D	E	F	G	H	I	J	K
1	attribute sheet										
2	entity name	attribute name	unit	type	current value	max	min	isDiscrete	discrete value1	discrete value2	discrete
3	Hardware1	Parameter1	W	float	1	10	0	FALSE			
4	–	Parameter2	none	integer	4	32	1	TRUE	1	2	
5											
6	Hardware2	Parameter3	W	float	10	100	0	FALSE			
7											
8	Hardware3	Parameter4	none	float	5	10	0	FALSE			
9											

AttributeSheet

Fig. 3.16 Internal block diagram output example.

	A	B	C	D	E	F	G
1	constraintSheet						
2		constraint name					
3		Constraint1					
4		equation					
5			={Parameter3}*{Parameter4}				
6		calculation result					
7							
8							
9		input attributes					
10		entity name	attribute name				
11		Hardware2	Parameter3				
12		Hardware3	Parameter4				
13							
14							
15							
16		output attributes					
17		Hardware1	Parameter1				

A001

Fig. 3.17 Parametric diagram output example.

出手法で用いる。そのため、システム・サブシステムの階層構造を「-(ハイフン)」を利用し表現することとした。Fig. 3.15 は、出力ファイルの一例として、Fig. 3.5 のブロック定義図の情報を出力したファイルを示している。図に示すように、Fig. 3.5 で階層関係として定義されていた「Target System」と「Hardware1」、「Hardware4」、「Software1」、「Software2」が「-」を用いて、親要素である「Target System」の下行に子要素である「Hardware1」、「Hardware4」、「Software1」、「Software2」が記述されている。このようなフォーマットで階層構造を表現し、構造モデルを出力することとした。

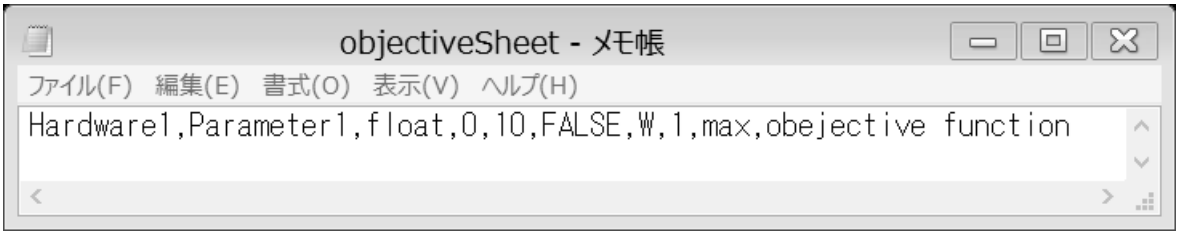


Fig. 3.18 Objective function output example.

Fig. 3.19 Design variable output example.

(2) パラメトリックモデル

パラメトリックモデルは、ブロック定義図でパラメータ、関係式の性質を、内部ブロック図でパラメータの所属する構造の定義と可視化、パラメトリック図で関係式の所属する構造の定義と可視化を行っている。これらの内、パラメータの性質の情報は、関係式の依存関係や設計タスクの依存関係に関係ないため、第 3 面の評価手法構築でしか用いられず、また、パラメータの所属する構造の情報は、関係式間の依存関係として設計タスク抽出で用いられる。設計タスク抽出後は、設計タスクに含まれる関係式とその入出力パラメータは、設計タスク毎に出力される情報に含まれるため、設計タスク抽出以降は必要ない。Fig. 3.16 は、一例として、Fig. 3.6 と Fig. 3.8 で定義されたパラメータの情報を出力したファイルを示している。図では、分かりやすいように表形式で表示しているが、ファイル形式はテキスト形式で出力されている。図中の「entity name」がパラメータの所属する構造(内部ブロック図の定義情報)を、「attribute name」がパラメータの名前を、「unit」、「type」、「current value」、「max」、「min」、「is Discrete」、「Discrete value X」がそれぞれ、ブロック定義図や最適化図でパラメータの性質として定義された単位、型、初期値、最大値、最小値、連続値・離散値の定義を表している。Fig. 3.16 では、Fig. 3.8 で定義された「Parameter1」と「Parameter2」が「Hardware1」に属しているという情報が 3 行目と 4 行目の 1、2 列で表しており、「Parameter1」と「Parameter2」の性質に関しては 3 行目と 4 行目に示されている。

また、関係式の情報は、設計タスク抽出後、第 3 面の評価手法構築を行う際に用いられる。即ち、設計タスク抽出手法で各設計タスクに分類された関係式が、設計タスクごとに

評価手法として構築されるため、関係式はそれぞれ個別に出力する必要がある。システムプロファイル定義で定義されていた関係式の所属が、設計タスク抽出により、再編成されるためである。Fig. 3.17 は、Fig. 3.7 と Fig. 3.9 で定義された関係式の情報の内、「Constraint1」に関して出力したファイルを示している。この図に関しても、分かりやすいように表形式で表示しているが、ファイル形式はテキスト形式で出力されている。図の「constraint name」が関係式名、「equation」が式の詳細、「input attributes」、「output attributes」がそれぞれ入力パラメータと出力パラメータを表している。Fig. 3.17 では、Fig. 3.7 、Fig. 3.9 で定義されているように、「Constraint1」が「Parameter3」、「Parameter4」を入力とし、「Parameter1」を出力する関係式であることが示されている。また、SDSI-Cubic のソフトウェアの処理上、高速化を図るため、関係式のファイル名を振り(Fig. 3.17 は「A001」であり、A の後に 001 から始まるようにファイル名を付けている)、関係式名とファイル名の対応表を作成、インデックスとして利用している。

このようなフォーマットでパラメトリックモデルを出力することとした。

(3) 最適化モデル

最適化モデルは最適化図で設計変数、目的関数、制約条件が定義されており、その内、設計変数と目的関数はそれぞれ個別に用いられるため、設計変数のファイルと目的関数のファイルを別々に出力することとした。Fig. 3.18 は、Fig. 3.12 で定義された最適化図の情報の内、目的関数を表したもの、Fig. 3.19 は、Fig. 3.12 で定義された最適化図の情報の内、設計変数を表したものである。目的関数に関しては、最適化の際に、最適化方向と制約条件を定義する必要があるため、最適化方向を含むパラメータの性質と所属の情報を、設計変数に関しては、設計変数であるパラメータとその所属の情報、性質として最小値、最大値、連続値・離散値の定義を出力することとした。また、設計変数、制約条件であるパラメータの性質に関しては、パラメトリックモデルの出力で述べたとおりであり、Fig. 3.16 内に示されている。

このようなフォーマットで最適化モデルを出力することとした。

3.2.3 システムプロファイル定義と設計タスク抽出の連携手法の構築

次に、出力された製品システムのモデルのテキストファイルと設計タスク抽出との連携

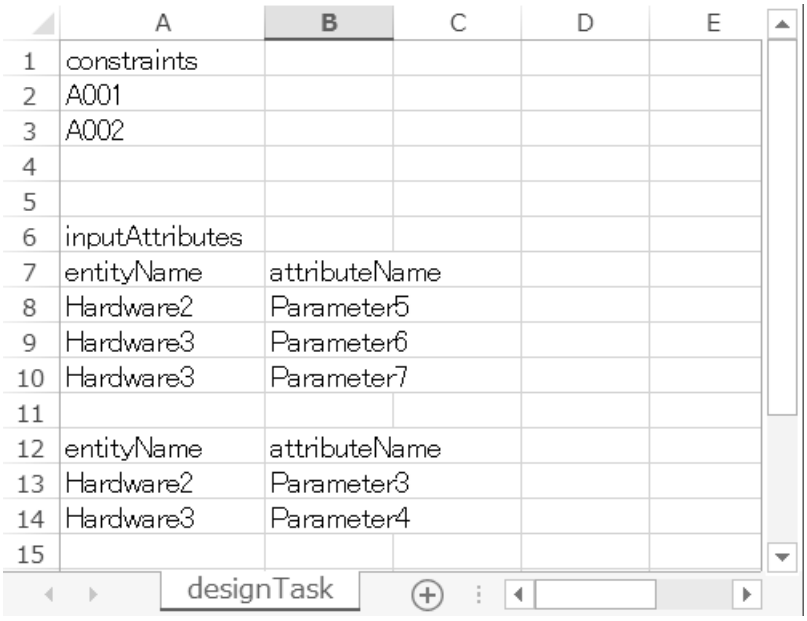
方法に関して述べる。

設計タスク抽出は、第 2 章で述べたように、構造モデル、パラメトリックモデルを用い、実行するためのアルゴリズム開発済みである。そのため、ここでは出力ファイルのフォーマットの検討のみ行う必要がある。

設計タスク抽出手法は実行することにより、構造に所属している関係式が、新たに設計タスクとして再編成される。これにより、関係式の所属が、構造から設計タスクに変更されるため、どの設計タスクにどの関係式が含まれているか、という情報を出力する必要がある。

Fig. 3.20 に設計タスク抽出の出力ファイルの一例を示す。この図に関しても、分かりやすいように表形式で表示しているが、ファイル形式はテキスト形式で出力されている。Fig. 3.20 に示すように、設計タスクに含まれる関係式が「constraints」に、関係式の入力パラメータ群が「inputAttributes」に、関係式の出力パラメータ群が「outputAttributes」に記述されている。ただし、関係式に関しては、システムプロファイル定義の出力であるファイル名を使用している。

この出力ファイルを用い、第 3 面の評価手法の構築、第 4 面の設計ワークフロー構築と連携を行う。



	A	B	C	D	E
1	constraints				
2	A001				
3	A002				
4					
5					
6	inputAttributes				
7	entityName	attributeName			
8	Hardware2	Parameter5			
9	Hardware3	Parameter6			
10	Hardware3	Parameter7			
11					
12	entityName	attributeName			
13	Hardware2	Parameter3			
14	Hardware3	Parameter4			
15					

designTask

Fig. 3.20 Design task output example.

3.2.4　設計タスク抽出と評価手法の構築の連携手法の構築

ここでは、設計タスク抽出で出力された設計タスク毎のテキストファイルと、システムプロファイル定義で出力されたパラメトリックモデルの出力ファイルを用いて、評価手法の構築を行う。

システムプロファイル定義でも述べたが、第 2 章の検討結果より、評価手法として表計算ソフトウェアを選択し、実装を行う。ノンパラメトリックな関係や、パラメトリックな関係の一部で使用する応答曲面を定義したデータベースに関しては、表計算ソフトウェアを用いてあらかじめ定義しておくこととする。Fig. 3.21 に式(2.2)で一例として挙げたCoO(Cost Of Ownership)のデータベースファイルの一部を示す。この例では $W_{Cut,Logic}$ を固定し、k を 3 つのパラメータで振り、$S_{C,L}$ と、$D_{Defect,Logic}$ とを複数の値で振った場合のデータベースを用意し、足りないデータに関しては線形補完することとした。以下では、他のソフトウェアとの連携も考え、テキスト形式の入力ファイルを作成するため、まず、出力された設計タスクのテキストファイルから、評価手法を構築するための入力ファイルを作成する方法に関して述べる。

入力ファイルとして必要となるのは、設計タスクとパラメトリックモデルの情報を合わせた、設計タスク毎のパラメータと関係式の情報を記述した情報である。これらをシンプルにテキスト形式にまとめることで、連携をとれるようにする必要がある。Fig. 3.22 に表計算ソフトウェアとの連携用の入力ファイルの一例を示す。図に示すように、設計タスク

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
53	90nm		Die Size (cm2)													
54			0.01	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00	2.00	3.00	4.00
55	Defect Density (/cm2)	0.00	0.000102	0.001019	0.002038	0.003058	0.004077	0.005096	0.006115	0.007135	0.008154	0.009173	0.010192	0.020385	0.030577	0.040769
56		0.01	0.000102	0.001021	0.002044	0.003071	0.0041	0.005132	0.006168	0.007206	0.008247	0.009291	0.010338	0.020966	0.031885	0.043095
57		0.02	0.000102	0.001022	0.00205	0.003084	0.004123	0.005169	0.00622	0.007277	0.00834	0.009409	0.010483	0.021548	0.033194	0.045422
58		0.03	0.000102	0.001024	0.002056	0.003097	0.004147	0.005205	0.006272	0.007348	0.008433	0.009526	0.010628	0.022129	0.034503	0.047749
59		0.04	0.000102	0.001025	0.002062	0.00311	0.00417	0.005242	0.006325	0.00742	0.008526	0.009644	0.010774	0.022711	0.035812	0.050077
60		0.05	0.000102	0.001026	0.002068	0.003123	0.004193	0.005278	0.006377	0.007491	0.008619	0.009762	0.010919	0.023293	0.037121	0.052405
61		0.06	0.000102	0.001028	0.002073	0.003136	0.004216	0.005314	0.006429	0.007562	0.008712	0.00988	0.011065	0.023875	0.038431	0.054734
62		0.07	0.000102	0.001029	0.002079	0.003149	0.00424	0.005351	0.006482	0.007633	0.008805	0.009997	0.01121	0.024457	0.03974	0.057063
63		0.08	0.000102	0.001031	0.002085	0.003162	0.004263	0.005387	0.006534	0.007705	0.008898	0.010115	0.011356	0.025038	0.04105	0.059392
64		0.09	0.000102	0.001032	0.002091	0.003175	0.004286	0.005423	0.006586	0.007776	0.008991	0.010233	0.011501	0.02562	0.04236	0.061722
65		0.10	0.000102	0.001034	0.002097	0.003189	0.00431	0.00546	0.006639	0.007847	0.009084	0.010351	0.011646	0.026203	0.043671	0.064053
66		0.11	0.000102	0.001035	0.002102	0.003202	0.004333	0.005496	0.006691	0.007918	0.009177	0.010469	0.011792	0.026785	0.044981	0.066384
67		0.12	0.000102	0.001037	0.002108	0.003215	0.004356	0.005532	0.006744	0.00799	0.009271	0.010586	0.011937	0.027367	0.046292	0.068715
68		0.13	0.000102	0.001038	0.002114	0.003228	0.004379	0.005569	0.006796	0.008061	0.009364	0.010704	0.012083	0.027949	0.047603	0.071047
69		0.14	0.000102	0.00104	0.00212	0.003241	0.004403	0.005605	0.006848	0.008132	0.009457	0.010822	0.012228	0.028531	0.048914	0.073379
70		0.15	0.000102	0.001041	0.002126	0.003254	0.004426	0.005641	0.006901	0.008203	0.00955	0.01094	0.012374	0.029114	0.050225	0.075712
71		0.16	0.000102	0.001042	0.002132	0.003267	0.004449	0.005678	0.006953	0.008275	0.009643	0.011058	0.012519	0.029696	0.051536	0.078045
72		0.17	0.000102	0.001044	0.002137	0.00328	0.004472	0.005714	0.007005	0.008346	0.009736	0.011176	0.012665	0.030279	0.052848	0.080379
73		0.18	0.000102	0.001045	0.002143	0.003293	0.004496	0.00575	0.007058	0.008417	0.009829	0.011294	0.01281	0.030861	0.05416	0.082713
74		0.19	0.000102	0.001047	0.002149	0.003306	0.004519	0.005787	0.00711	0.008489	0.009922	0.011411	0.012956	0.031444	0.055472	0.085048
75		0.20	0.000102	0.001048	0.002155	0.003319	0.004542	0.005823	0.007162	0.00856	0.010015	0.011529	0.013101	0.032026	0.056784	0.087383
76		0.30	0.000102	0.001063	0.002213	0.00345	0.004775	0.006187	0.007686	0.009273	0.010947	0.012708	0.014557	0.037856	0.069917	0.11076

Fig. 3.21 Database example.

に含まれる関係式の「入力パラメータ」、「出力パラメータ」の 2 つの項目を用意し、「出力パラメータ」に関しては、関係式も記述することとした。パラメータの所属に関しては、評価の際には必要のない情報であるため、記載しない。また、本研究では、表計算ソフトウェアを用いる前提であったため、この項目群としているが、今後、他のソフトウェアと連携する場合、どの関係式をどのソフトウェアで評価するのかを定義し、入力ファイルに記述する必要があると考えられ、今後の課題である。

この入力ファイルをもとに、表計算ソフトウェアへの実装を行う。Fig. 3.23 に実装された表計算ソフトウェアの一例を示す。図は Fig. 3.22 を表計算ソフトウェアに実装した例であり、図に示すように、「Input」、「Output」の 2 つの項目があり、それぞれ Fig. 3.23 の「Input」、「Output」と対応している。また、「Output」の値の定義されているセルには、Fig. 3.22 の「Output」の項目に記述されていた関係式に対応する式が定義されており、このファイルで評価が可能となっている。データベースファイルに関しては、実装の際に取り込むこと

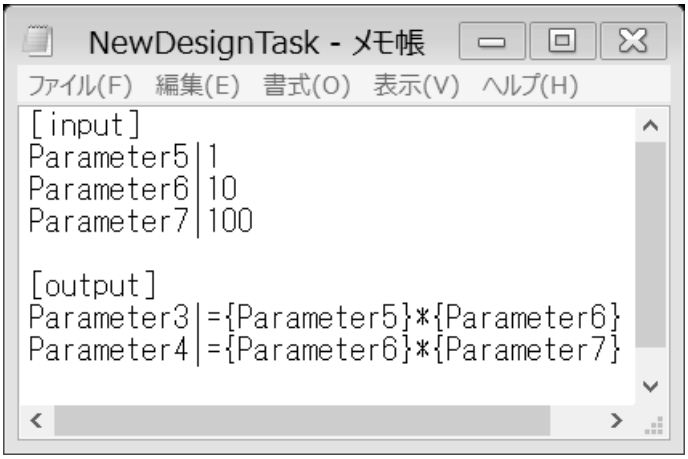


Fig. 3.22 Evaluation method input example.

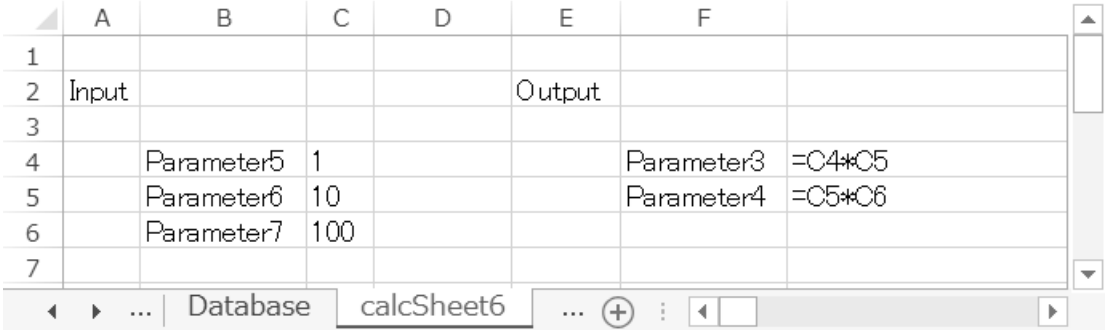


Fig. 3.23 Evaluation method output example.

としており、「Output」の関係式がデータベース引用の式であったとしても評価可能としている。

この面では、上記の表計算ソフトウェアの評価ファイルを設計タスク毎に作成、出力する。

3.2.5　設計タスク抽出と設計ワークフロー構築の連携手法の構築

ここでは、設計タスク抽出で出力された設計タスク毎のファイルと、システムプロファイル定義で出力されたパラメトリックモデル・最適化モデルの出力ファイルをもとに、DSM を用いて設計ワークフロー構築を行う。また、そのための入出力ファイルのフォーマットの検討を行う。

(1) 最適化を考慮した設計ワークフロー構築アルゴリズムの検討

ここでは、まず最適化を考慮した設計ワークフロー構築アルゴリズムの検討を行う。

この面では第 2 章の検討結果より、タスク DSM を用いる。タスク DSM は、設計タスク内の関係を考慮し、手戻りが少ない設計ワークフローを導出することができる手法である。しかし、SDSI-Cubic では、最適化を行うために、設計ワークフローに最適化を加える必要がある。その際に考えられる最もシンプルな方法として、導出された設計ワークフロー全体を一つの最適化とし、最適化モデルの設計変数、目的関数をもつ最適化問題とすることであるが、最適化計算は膨大な繰り返し計算であるため、大規模な設計ワークフロー全体を最適化とすることは計算時間を鑑みると現実的ではない。そのため、可能な限り、最適化を行う設計タスクを減らすためのアルゴリズムが求められる。

上記を行うために、まず、タスク DSM の現状技術の調査、検討結果を述べる。

タスク DSM を用いた製品適用事例[4-6]は Table. 3.1 に示すように多々あり、この中で、設計タスク間に関係性のあるなしという 2 値でのタスク DSM を検討した例が表中の網掛け部である。また、2 値でのタスク DSM での事例は 2 つのタイプに分けられ、一つはタスク DSM を実際の製品開発に適用、分析を行い、その結果をどのように製品開発にフィードバックするかという、タスク DSM の製品開発への適用方法や結果の解釈方法を検討した基礎的なもの、もう一つは、タスク DSM に対し、タスク DSM 作成前、あるいは分析中に処理を追加することで手戻りを減らし、製品開発の効率化を図るものである。前者の具体的な事例として、Intel の例[5]を取り上げる。Fig. 3.24 は、Intel がシステ

Table 3.1 Task DSM applied examples.

Reference	Description system	Analysis tool	Application target	Summary
Browning. T.R (2002)	Binary	Partithioning	Virtual example	It shows DSM as a tool which describes the complicated process of product development and the methodology which attains optimization of a process.
Clarkson and Hamilton (2000)	Binary	DSM Description	The rotor of a helicopter	It proposed description of the dependency of the design tasks by DSM for constructing the procedure of the optimal design development work in complicated product development.
Kusiak (2002)	Binary	Parthithoning/ Clustering	Virtual example	It shows the methodology for realizing modularity and pays attention to the interdependence of the product model, the process model, and the resource model.
Intel Inside	Binary	DSM Descripti	System LSI	Analysis of the possibility of feedbacks, and examination of the feedback method to product development.
Smith, R.P. and S. D. Eppinger (1997)	value	Partithioning	Virtual example (Electric vehicle)	The algorithm which aims at shortening of a lead time is developed. It was attained by description of a repetition of successive work, reexamination of sequence of operation, etc. in complicated product development.
Yassine, et al. (2000)	value	Partithioning	Automobile (Bonnet)	It shows the way of reducing the repetition tasks in complicated product development. It has succeeded in reducing repetitions more sharply than simple partitioning of DSM by inserting an easy verification task in the task block including a repetition.
Seki, et al.	Binary	Parthithoning/ Clustering	System LSI	It examines the reduction method of the feedbacks by the addition using optimization of design tasks and dependencies.

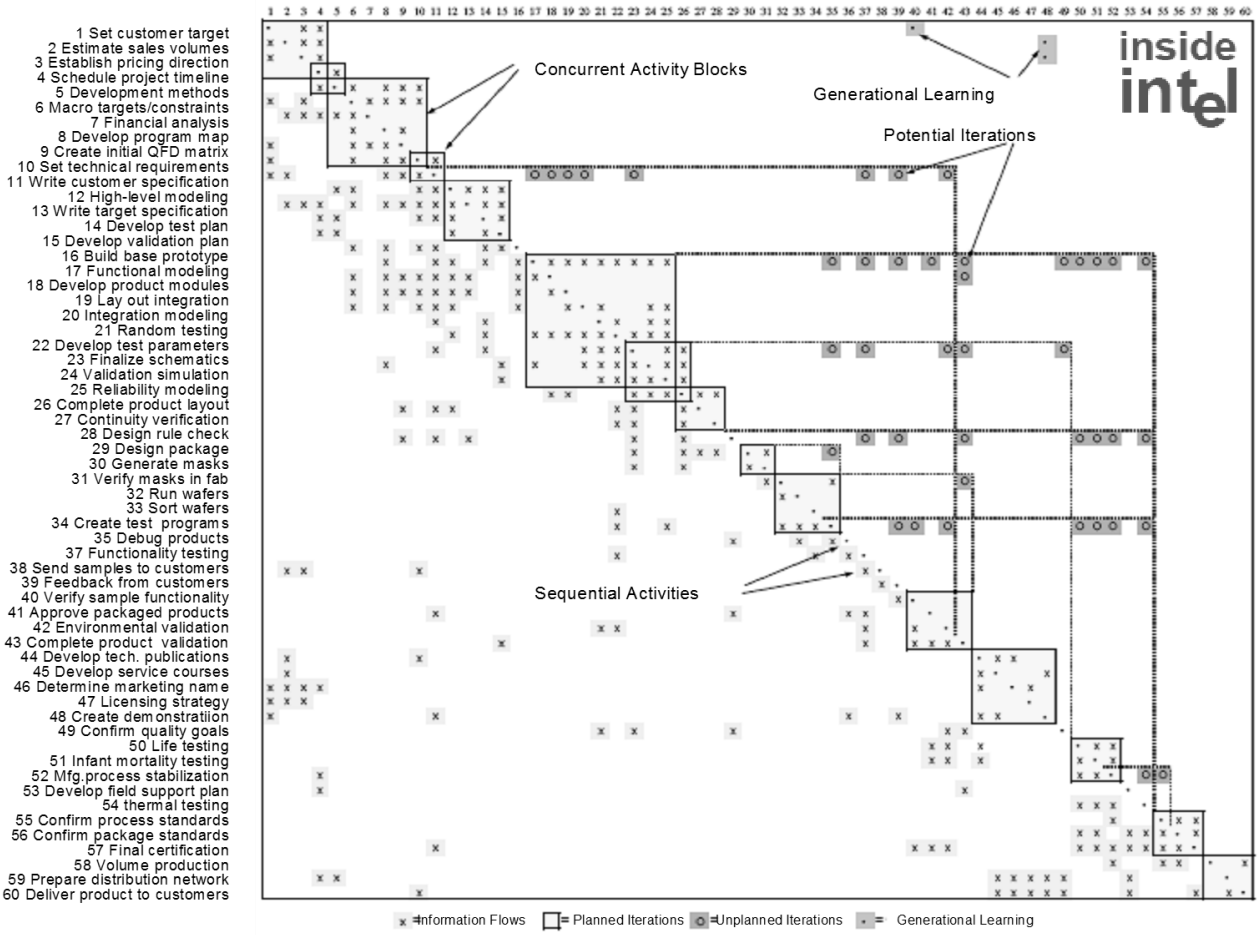
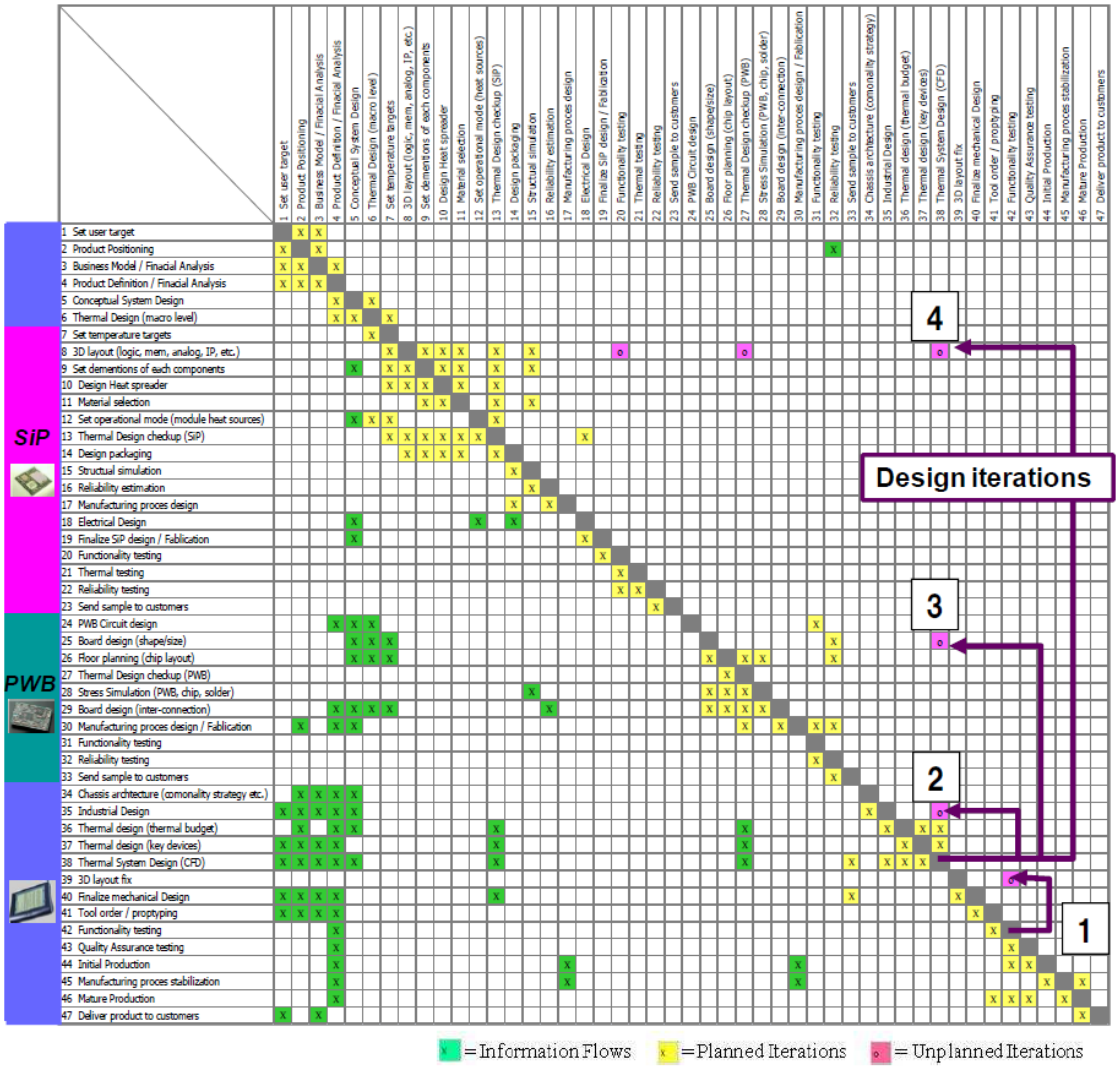


Fig. 3.24 The example of applying DSM to system LSI development of Intel.

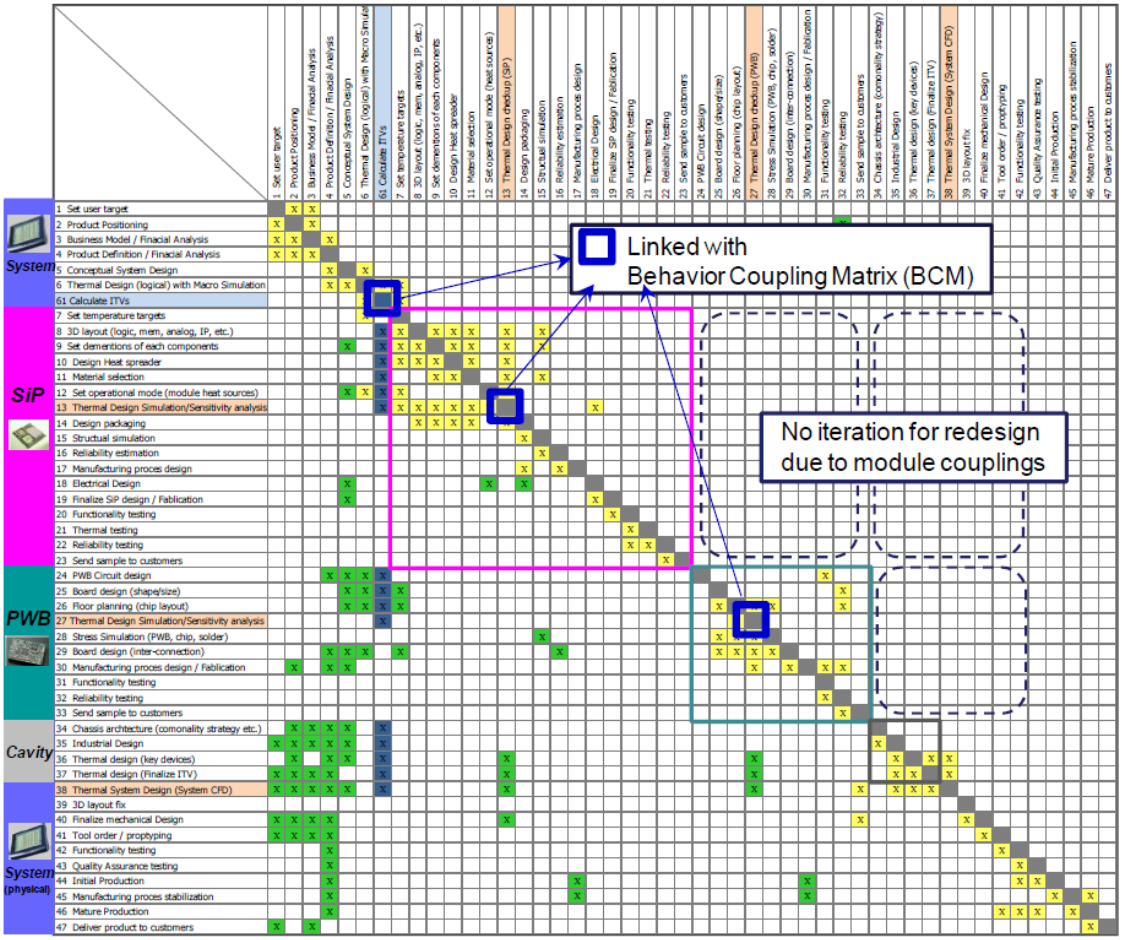
ム LSI の要求から製造までの設計プロセスをタスク DSM を用いて整理したものであり、タスク DSM を用いることであらかじめ手戻りを想定し、協調設計や繰り返し設計を行うことで設計の効率化を行った例である。

後者の具体的な事例として、関ら [6]の行った、システム LSI を対象とし、グローバルな設計の効率化を試みた事例を取り上げる。この研究では、グローバルな設計を効率的に行うために、キャビティと呼ぶ物理的な制約を考慮した最適化をあらかじめ行い、各設計タスクに適切に設計目標(ITVs)を与えることにより、設計の手戻りを減らすことを行っている。Fig. 3.25 (a)が手法の適用前のタスク DSM、Fig. 3.25 (b)が手法の適用後のタスク DSM である。



(a) Before considering ITVs.

Fig. 3.25 The example of applying DSM to system LSI development of Seki and et al.



(b) After considering ITVs.

Fig. 3.25 Continued.

このようにタスク DSM の製品適用と設計の効率化に関しては現在研究が行われているが、これらは、実際の製品開発前に適用するための手法であり、パーティショニング実行後の設計タスクのクラスタリングや、手戻りの製品開発のフィードバックは、各設計タスク毎に個別であり、適用する製品毎に異なっている。即ち、タスク DSM の解釈はその時々で設計者が行うものであり、また、タスク DSM 分析後の設計タスクの実行自体も個人、またはチームで個別に行うものであるため、タスク DSM 分析結果の解釈と設計タスク評価の自動化の検討は行われていない。

そのため、本研究では、最適化を考慮したタスク DSM の生成とその分析結果の解釈の自動化を行うことで、最適化を考慮した設計ワークフローの自動生成アルゴリズムの構築を試みた。

Fig. 3.26 に従来手法と本研究で提案する手法のフローチャートを示す。本研究では自

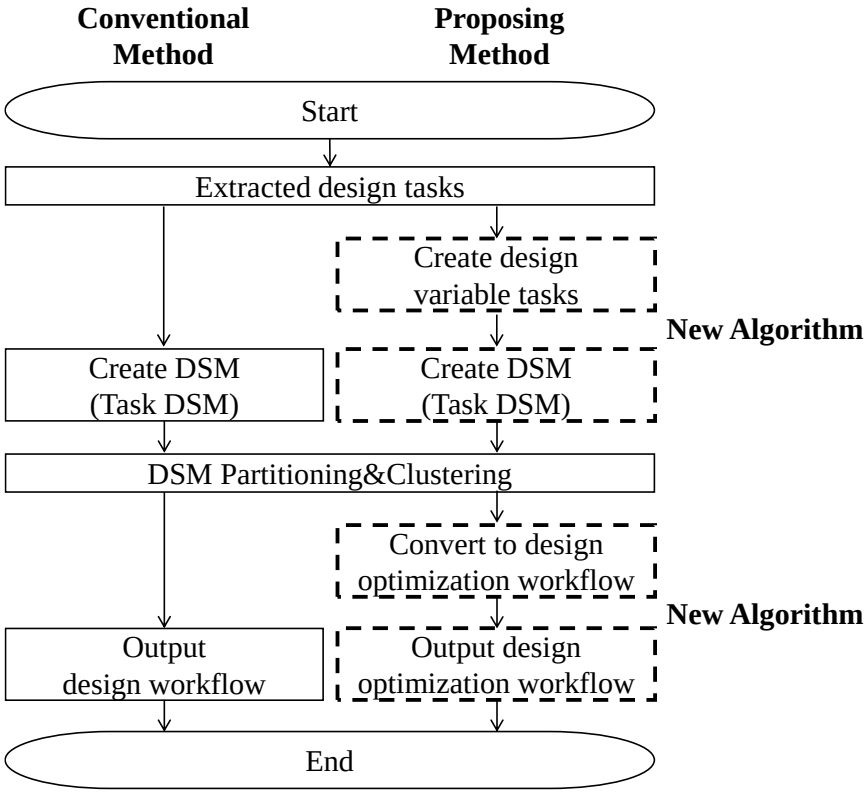


Fig. 3.26 Flowchart of proposal algorithm.

動化の課題である、①SysML で拡張した、最適化を考慮した設計ワークフロー構築(新アルゴリズムの検討 1)、②最適化問題の定義を加えた設計ワークフローを構築し最適化を実行可能にすること(新アルゴリズムの検討 2)に対し、新しいアルゴリズム(Fig. 3.26 の太線部)を考案した。

以下では①SysML で拡張した、最適化を考慮した設計ワークフロー構築(新アルゴリズムの検討 1)、②最適化問題の定義を加えた設計ワークフロー(最適化実行ワークフロ―)を構築し最適化を実行可能にすること(新アルゴリズムの検討 2)、の 2 つの新しいアルゴリズムの検討を行った結果を示す。

まずシステムプロファイル定義(SysML)から最適化を考慮した設計ワークフローを自動的に生成するための手法に関して述べる。SDSI-Cubic では、システムプロファイル定義(SysML)から自動的に設計タスクが決定され、その設計タスクをもとに DSM を生成し、パーティショニングを行うことでワークフローを構築することを提案している。しかし、この手法ではシステムプロファイル定義で定義された最適化問題が考慮されておらず、このままでは最適化を考慮した設計ワークフローは構築できない。その理由を、Fig. 3.27 に

示す、従来手法での設計ワークフロー構築手法をもとに説明する。

従来手法では、システムプロファイル定義で設計変数が定義されていたとしても、Fig. 3.27 左の設計タスク抽出手法では設計変数と目的関数をパラメータと同様とみなし設計タスクを決定するため、作成された DSM、ワークフローにも設計変数と目的関数からのフィードバックの考慮が行われておらず、最適化を考慮した設計ワークフローの導出は行えていなかった。しかし、設計変数は実際の設計として重要なパラメータであり、様々な設計タスクのパラメータとして含まれている可能性が高い。それにより、設計変数が設計タスク間の依存関係や相反問題の原因となり、手戻りやすり合わせの原因となる可能性が高い。そのため、本研究では、設計タスク抽出により決定された設計タスクに、設計変数を決定するための設計タスク(以下では設計変数タスクと呼ぶ)を追加し、それをワークフローの最初に実行することとした。また、設計変数タスクを追加しただけでは、従来の DSM と大きな変化はなく、最適化を考慮した設計ワークフローはまだ作成できない。そのため、本研究ではさらに設計変数タスクに対して目的関数を導出する制約式を含む設計タスクから依存関係を定義する。その理由は、SysML で定義された設計変数が設計変数でないパラメータと異なる点が、目的関数の値によって値が変更さ

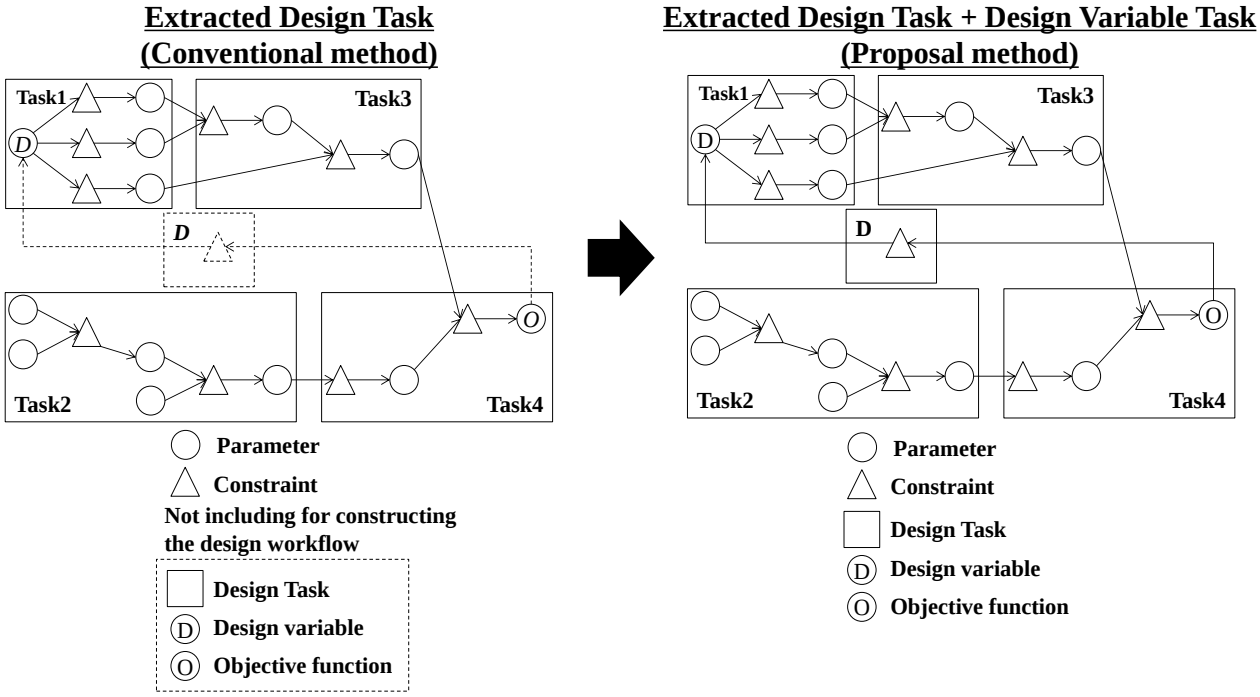


Fig. 3.27 The difference between conventional extracting design task method and proposal method.

れるかどうか、即ち、目的関数と依存関係があるかどうかであるためだと考えたためである。このように DSM を作成することで設計者が暗黙的に行っていたことを自動的に導出できるようにした。

Fig. 3.28 に本研究で提案する、最適化を考慮した設計ワークフロー構築手法を示す。Fig. 3.28 の DSM 中の「D」が設計変数タスクであり、目的関数を導出する制約式を含む設計タスク「Task4」からフィードバックをかけている。また、この DSM をパーティショニングした後、Fig. 3.28 に提案する手法を用いることにより最適化を考慮した設計ワークフローの構築を行う。Fig. 3.29 は DSM のパーティショニング結果から最適化を考慮した設計ワークフローへの変換ルールを定義したものである。パーティショニング結果の並列、連続、結合関係は Fig. 3.29 の右図に示すように、そのまま最適化を考慮した設計ワークフローの並列、連続に変換することとし、結合関係に関しては一つの最適化問題として取り扱う必要があるため、最適化問題に変換することとする。

次に DSM のパーティショニング結果からの最適化実行ワークフローの構築方法に関して述べる。

前節で述べた手法で構築される最適化を考慮した設計ワークフローには設計タスクの

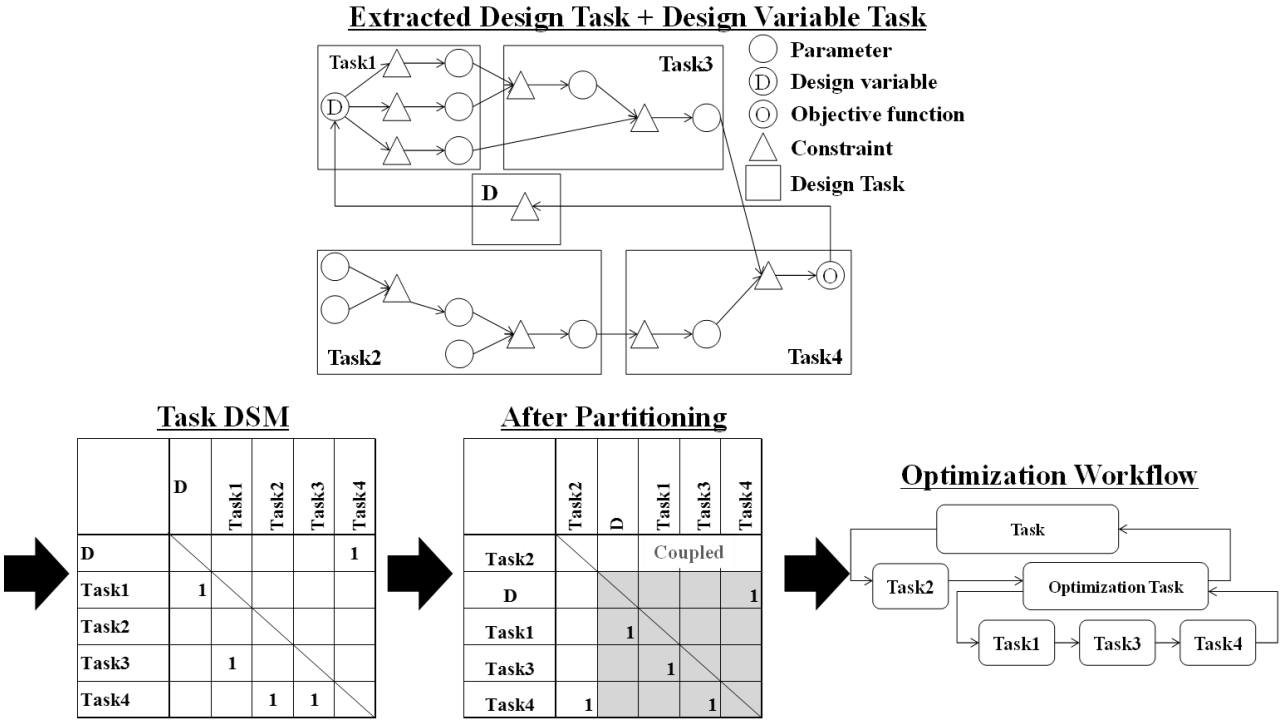


Fig. 3.28 Proposal method of generating optimization workflow from system profile

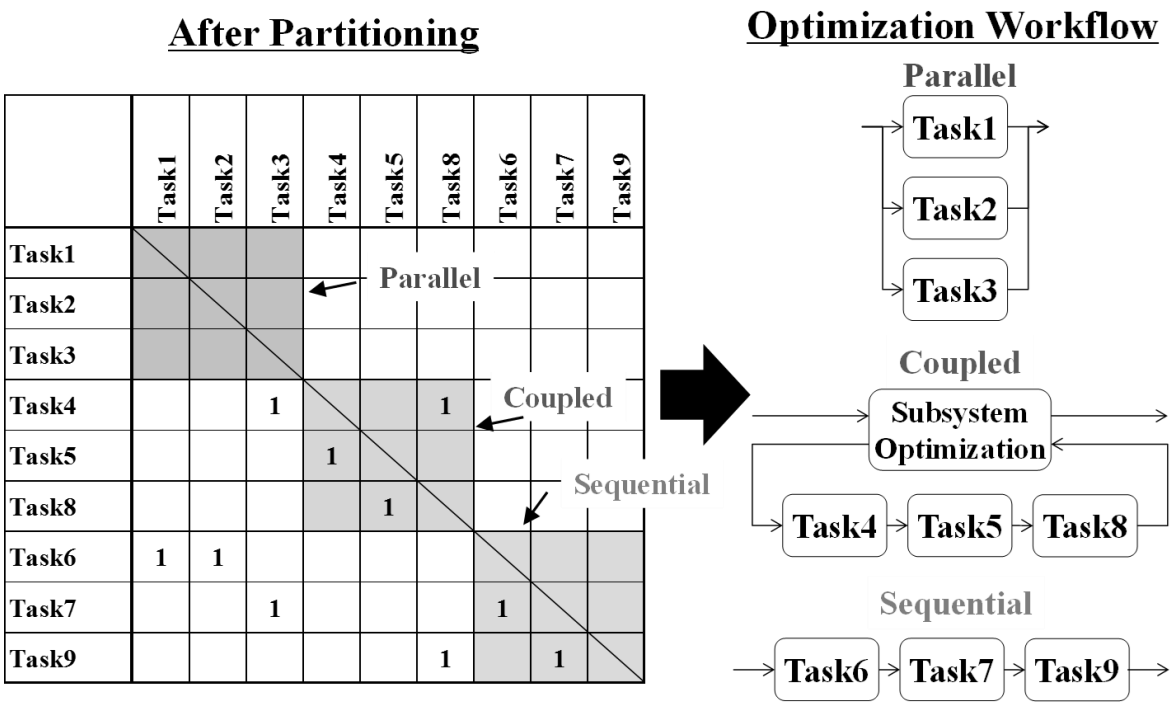


Fig. 3.29 Proposal conversion method from partitioned DSM to system design workflow.

ワークフローと最適化は定義されているが、最適化問題は定義されていない。即ち、最適化に設計変数、目的関数など、最適化問題を決定するものが定義されていない。そのため、最適化を考慮した設計ワークフローに、DSM のパーティショニングの結果をもとに最適化問題を定義する手法の構築を行う。

本研究では、Fig. 3.29 で示したように結合関係を最適化問題と変換している。ここで、結合関係に関しては、設計タスク間で相互にパラメータを参照しているため、最適化の際に設計順序を決定することが難しい。そのため本研究では、設計タスク間でフィードバックとなっているパラメータに注目し、そのパラメータを設計変数に置き換えることでこの問題の解決を図った。Fig. 3.29 の例で説明すると、 Task4 は Task8 から出力されるパラメータを入力とすることで、初めて計算可能となるが、本研究では、そのパラメータを設計変数として最適化を行う。またその際に、設計変数としたパラメータと、実際に Task8 から出力されるパラメータが一致しているかの確認を行う必要がある。そのため、一致を確認するタスクとして Fig. 3.30 の右図に示す「F」のタスクを追加する。また、Task4 が設計変数のタスクである場合、フィードバックは SysML で定義された設計変数に起因するため、それを設計変数に追加し最適化を行う。このような手順を行う理

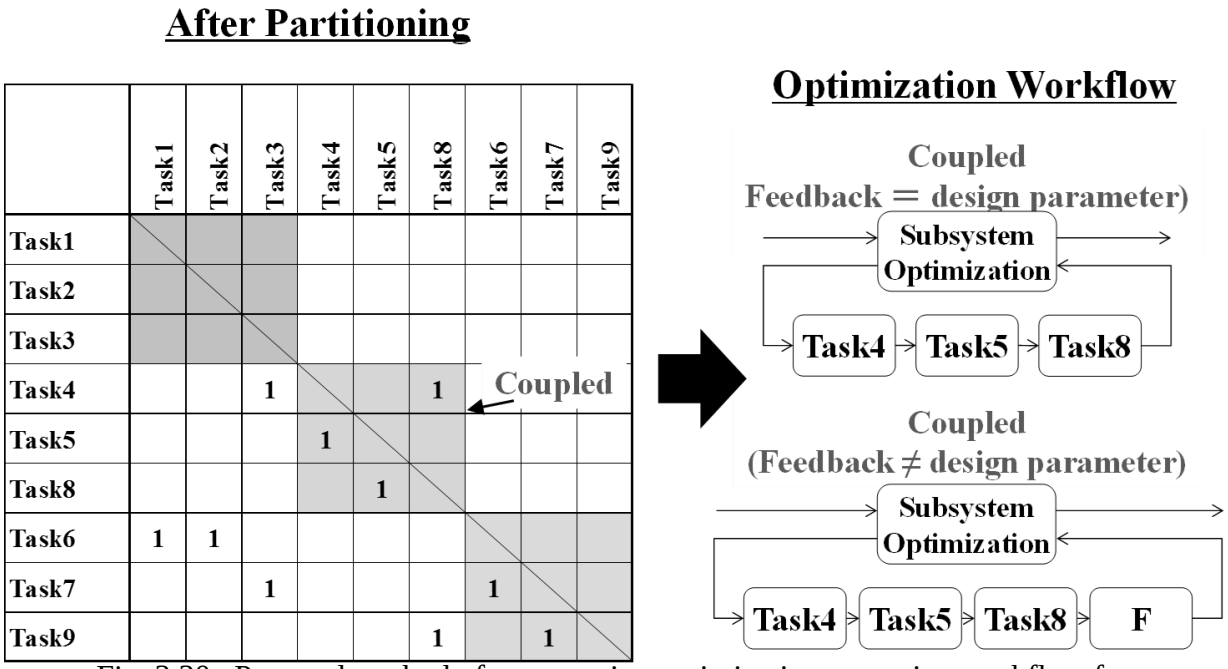


Fig. 3.30 Proposal method of constructing optimization execution workflow from partitioned DSM.

由としては、SysML の最適化図で定義された設計変数の場合、設計変数への依存関係は式としては定義されておらず、実際に設計変数の値が導出されているわけではない。それに対し、他のパラメータは導出する制約が存在するため、設計変数としてしまうと、導出後の値との一致を取る必要がある。そのために一致を確認するタスクを追加する。また最適化タスクの目的関数に関しては、SysML の最適化図で定義された目的関数とする。最適化図で定義された目的関数が存在しない場合は、DSM のすべての設計タスクを最適化タスクとし、SysML の最適化図で定義された設計変数、目的関数に加え、結合関係のフィードバックとなっているパラメータを設計変数に追加することで最適化実行ワークフローにすることが出来ると考えたものである。

(2) 入出力ファイルの検討

(1)で構築したアルゴリズムを実行するための入力ファイルのフォーマットの検討を行う。

入力ファイルとして必要なものは、設計タスクファイルを再編成し、設計変数タスクを加えた新設計タスクファイルと、それらをもとに作成された DSM の定義されたファイルである。この内、前者に関しては、設計タスクファイルと同様のフォーマットを用い、後

者に関しては、設計タスクと依存関係をテキスト形式で定義する。Fig. 3.31 に設計変数タスクの入力ファイルの一例を、Fig. 3.32 に DSM 入力ファイルの一例を示す。また、以降の入力、出力ファイルは、これ以前の入力、出力ファイルとは関係のない、別の例であるの

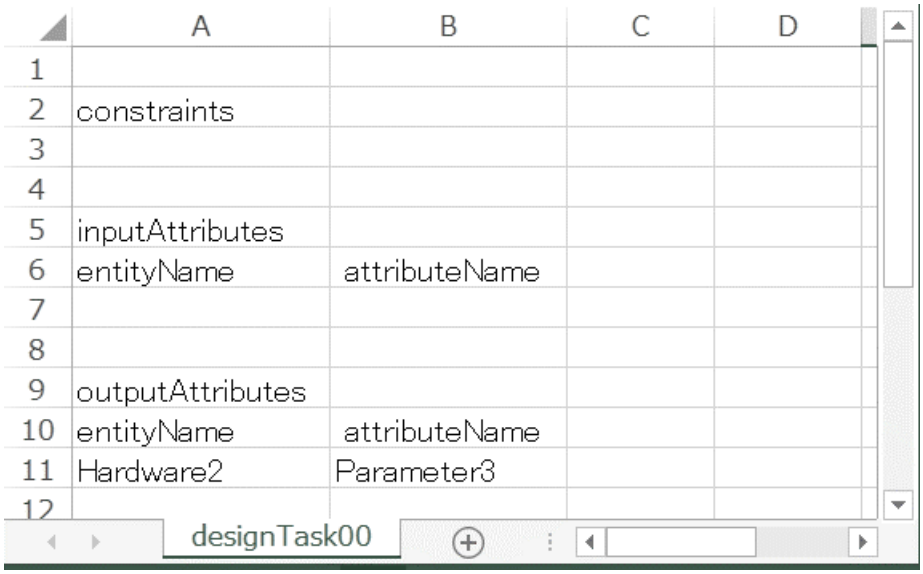


Fig. 3.31 Design variable task input example.

	designTask00	designTask01	designTask1	designTask2	designTask3	designTask4	designTask5	designTask6	designTask7	designTask8	designTask9	designTask10	designTask11	designTask12	designTask13	designTask14	designTask15	designTask16
designTask00																1		
designTask01																1		
designTask1																		
designTask2			1															
designTask3				1														1
designTask4				1											1			
designTask5			1															
designTask6		1	1															
designTask7			1															
designTask8		1	1				1											
designTask9	1		1						1									
designTask10			1	1					1									
designTask11				1						1								
designTask12	1		1						1									
designTask13			1			1				1			1	1				
designTask14		1	1					1							1			1
designTask15				1					1			1						
designTask16			1	1	1				1		1	1					1	

Fig. 3.32 DSM input example.

で注意されたい。

この入力ファイルをもとに、第二章で述べた DSM の分析ソフトウェアを用い、パーティショニングとクラスタリングを行い、最適化実行ワークフローを出力する。

出力ファイルとして必要なものは、構築された最適化実行ワークフローの情報である。即ち、どの手順で設計タスクの評価を行うか、最適化を行うのはどの設計タスク群なのか、最適化の目的関数と設計変数は何かという情報を出力する必要がある。Fig. 3.33 に設計ワークフローの出力ファイルの一例を、Fig. 3.34 に最適化の出力ファイルの一例を示す。また、システムプロファイル定義で設計変数として定義されたパラメータと異なるパラメータが設計変数となる場合、設計変数としたパラメータと、出力されるパラメータが一致しているかの確認を行うタスクを作成し、出力する必要がある。その出力ファイルの一例を Fig. 3.35 に示す。

Fig. 3.33 に示すように、本研究では、評価を優先すべき設計タスクから順に記述し、最適化タスクは「Optimization」という表現をすることとした。また、最適化に含まれている設計タスクは、「-(ハイフン)」を用いて表現し、最適化内で優先すべき設計タスクか

Fig. 3.33 Design workflow output example.

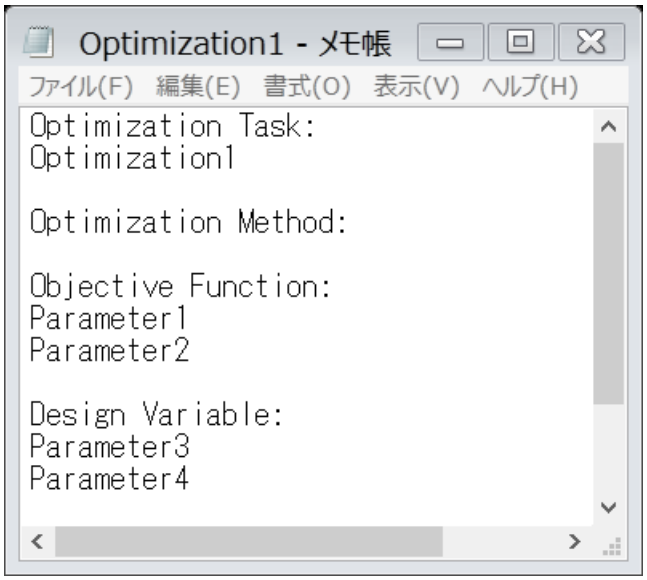


Fig. 3.34 Optimization task output example.

	A	B	C	D	E	F
1	Parameter6dp		Parameter6		Parameter6df	=ABS(B1-D1)/D1
2	Parameter7dp		Parameter7		Parameter7df	=ABS(B2-D2)/D2
3	Parameter8dp		Parameter8		Parameter8df	=ABS(B3-D3)/D3
4						

Feedback1

Fig. 3.35 Feedback task output example.

ら順に記述することとした。図は、Fig. 3.32 のパーティショニング結果から最適化ワークフローを構築した結果を出力したものであり、「designTask1」、「designTask2」、「designTask5」、「designTask6」、「designTask10」、「designTask15」、「Optimization1」の順で評価を行い、「Optimization1」は、「designTask00」、「designTask01」に含まれる設計変数をもとに、「designTask3」、「designTask4」、「designTask6」、「designTask8」、「designTask9」、「designTask11」、「designTask12」、「designTask13」、「designTask14」、「designTask16」、「Feedback1」の順で評価・最適化を行う最適化であることを表している。「Feedback1」は設計変数としたパラメータと、出力されるパラメータが一致しているかの確認を行うタスクである。

また、Fig. 3.34 に示すように、最適化と定義された設計タスク群の最適化問題、即ち、

設計変数、目的関数、最適化手法を出力することとした。図中の「Optimization Task」は、このファイルで定義された最適化問題を適用する最適化を表しており、「Optimization Method」は、この最適化に適用する最適化手法とその詳細、「Objective Function」は目的関数となっているパラメータ、「Design Variable」は設計変数となっているパラメータを表している。Fig. 3.34 では「Optimization Task1」に対して、「Objective Function」として「Parameter1」、「Parameter2」が定義されており、「Design Variable」として「Parameter3」、「Parameter4」、「Parameter5」が定義されていることを表している。「Optimization Method」に関しては、次の最適化実行の際に、設計変数、目的関数の詳細設定とともに、入力ファイル生成の際に定義される。

Fig. 3.35 は設計変数としたパラメータと、出力されるパラメータが一致しているかの確認を行うタスクの出力例であり、図に示すように、設計変数としたパラメータ、出力されるパラメータ、それらの一致をとる式とその結果を表すパラメータの 3 つのパラメータと一つの式をパラメータ毎に定義、評価できるように表計算ソフトウェアに実装することとした。図では、「Parameter6」、「Parameter7」、「Parameter8」が一致をとるべきパラメータであり、パラメータ名の後に「dp」のついたものが設計変数としたパラメータ、パラメータ名そのままのものが、出力としたパラメータ、パラメータ名の後に「df」のついたものが一致をとった後の結果を表したものである。

このように設計ワークロー構築の入力ファイルと出力ファイルを規定した。

3.2.6　評価手法の構築・設計ワークフロー構築と最適化実行の連携手法の構築

この面では、設計ワークフローの構築で出力された最適化実行ワークフローを定義したファイルと、評価手法構築で出力された評価手法毎のファイルをもとに、最適化を実行する。また、この面では第二章の検討結果より、I-sight という自動化・最適化支援ソフトウェアを用いる。

まず、入力ファイルとして、構築された最適化実行ワークフローの最適化に対して、最適化問題を設定したファイルが必要となる。Fig. 3.36 にその入力ファイルの一例を示す。

Fig. 3.36 に示すように、入力ファイルは、設計ワークフロー構築の出力ファイルである、Fig. 3.34 の最適化の出力ファイルに、最適化手法とその詳細、目的関数の制約条件と最適化方向、設計変数の取りうる値の範囲である、最小値、最大値、連続値・離散値の定義、

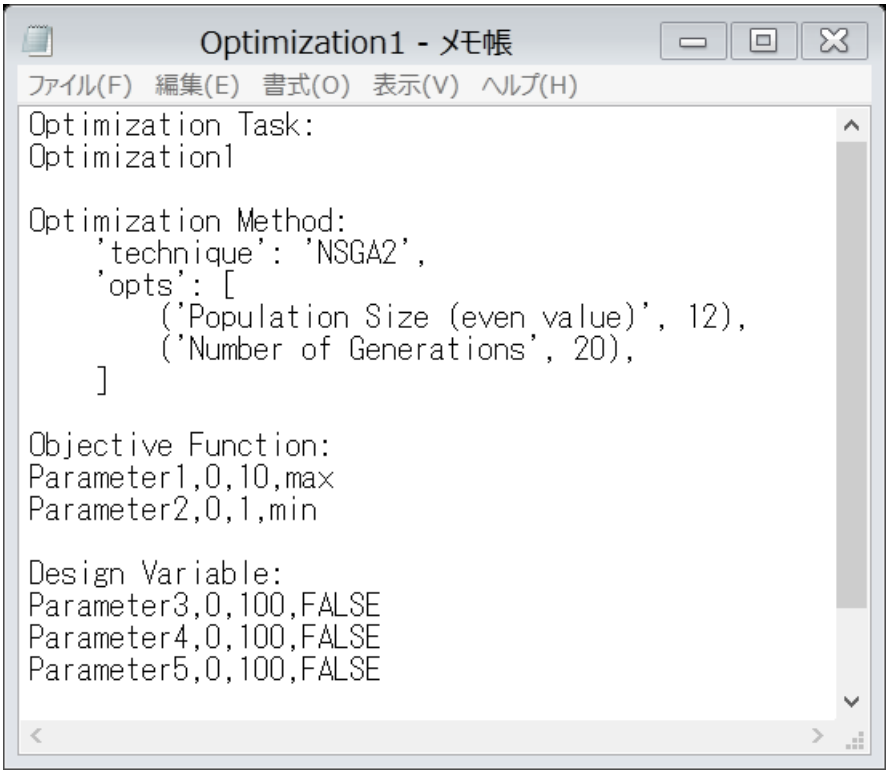


Fig. 3.36 Optimization definition input example.

を新たに追加したものである。この出力ファイルで最適化手法を定義しているが、最適化手法は Fig. 1.4 に示すように様々あり、また、問題の分類により、用いるべき手法は異なる(Table 2.1)。しかし、SDSI-Cubic で Table 2.1 のように問題の分類を細かく行うことは非常に困難である。そのため、本研究では、目的関数の数に注目し、目的関数が単一の場合と複数の場合で最適化問題を分類し、最適化手法を選択することとした。この理由は、目的関数による分類が最適化手法の選択に大きな影響を与え、かつ、少なくとも、この分類により手法を決定することで、最適解の導出が可能であるからである。具体的には目的関数が複数の場合は、NSGA-II [7]を、一つの場合は、DownhillSimplex 法 [8]をデフォルトとして使用することとした。ただし、オプションとして最適化手法とその詳細の選択は可能とした。Fig. 3.36 では、「Optimization1」に対して、「Optimization Method」として「NSGA-II」が定義されており、その詳細として、集団の個体数と世代数が定義されている。「NSGA-II」とは、遺伝的アルゴリズムを目的関数が複数の場合にも適用できるように改良したものである。また、目的関数「Objective Function」と設計変数「Design Variable」に関して、Fig. 3.34 で定義されていたパラメータに、目的関数は、制約条件として最小値、最大値と最適化方向を、設計変数は、最小値、最大値、連続値・離散値の

定義を加えている。

この入力ファイルと Fig. 3.33 の設計ワークフローの出力ファイル、Fig. 3.23 の設計タスク毎に表計算ソフトウェアに実装された評価ファイルの 3 種類のファイルをもとに I-sight 用の入力ファイルを作成する。

I-sight は、Fig. 2.11 の例のように、GUI(Graphic User Interface)を用いて、以下の手順で最適化が実行される。

① ワークフローの作成

② 評価手法の実装

③ 最適化問題の定義

④ 最適化の実行

⑤ 最適化結果の出力方法の設定と出力

本研究では、これらの手順を GUI なしで自動的に行うために、I-sight の機能の一つである、カスタマイズスクリプトを用いることとした。これは「Python」というプログラミング言語を基本とし、I-sight 独自の拡張がなされた「Jython」というプログラミング言語を用いてプログラムをテキストファイルなどに記述し、I-sight のコンパイラでコンパイル、実行することで上記の①～⑤の手順を自動的に行える手法である。本研究ではこの手法を用

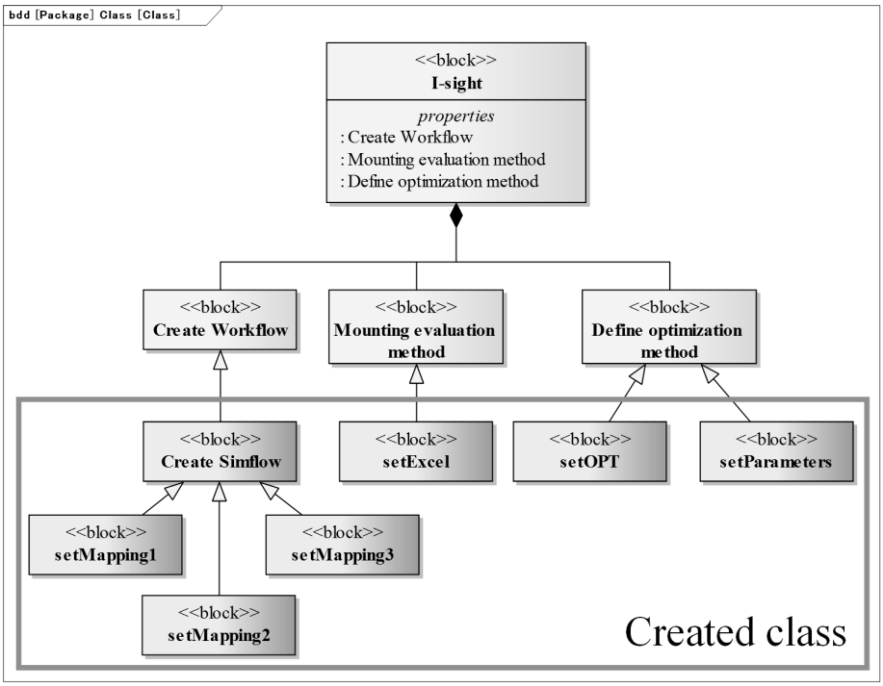


Fig. 3.37 Class definition for Isight customize script.

い、「Jython」で Fig. 3.37 に示すクラスを作成し、これを用い、上記の①～⑤の手順でアルゴリズムを実装することで、I-sight の入力ファイルを自動的に作成、最適解の実行までを行えるようにした。また、最適解の出力はテキスト形式で出力することとした。最適解の出力例を Fig. 3.38 に示す。Fig. 3.38 では、「出力値」の「Parameter1」、「Parameter2」が目的関数の最適化結果の出力、「Parameter3」、「Parameter4」、「Parameter5」が設計変数の最適化結果の出力である。

このように入力ファイルの作成と、最適化の実行、最適解の出力をできるようにした。

3.3　システムデザインフレームワークの構築

上述したように、提案した SDSI-Cubic 手法の各面で実行するための方法を明らかにし、これをコンピュータ上で自動実行するためのアルゴリズムの検討を構築した。

Fig. 3.39 に本章で構築された自動実行アルゴリズムを考慮し、最終的に構築されたシステムデザインのフローを示す。

このフローは、Fig. 3.1 に本章で開発したアルゴリズムと入出力ファイルを追加し、整理しなおしたものである。このアルゴリズムに従って、自動実行を行うため、各面の手法と連携するためのソフトウェアやファイル群を開発・規定し、自動最適化実行を可能とした。その際に実装や評価の関係上、評価手法構築で用いる評価手法は表計算ソフトウェアに限定し、SysML で定義するパラメトリックモデルも表計算ソフトウェアで計算

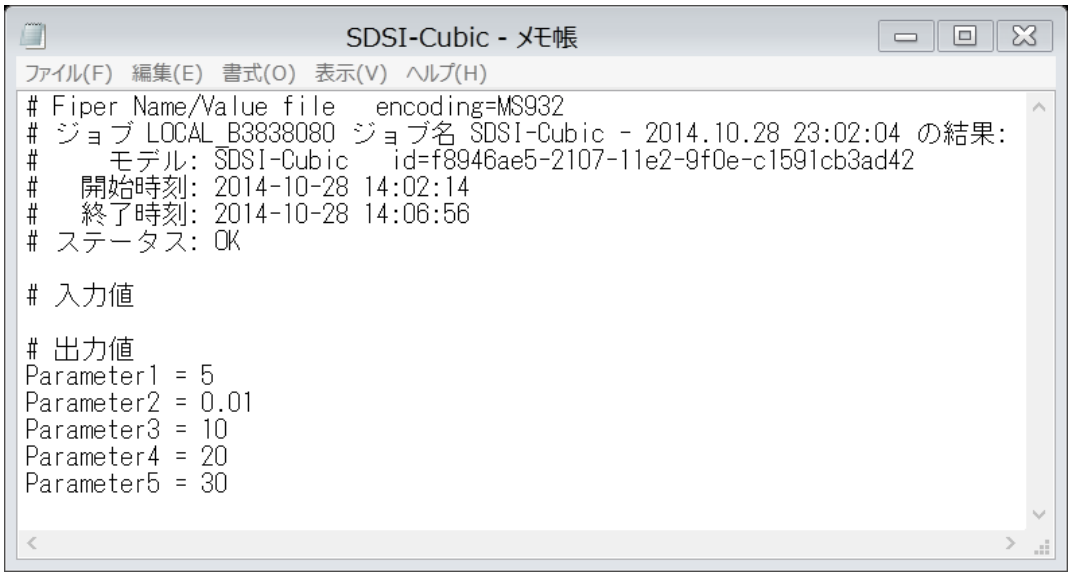


Fig. 3.38 Optimization result output example.

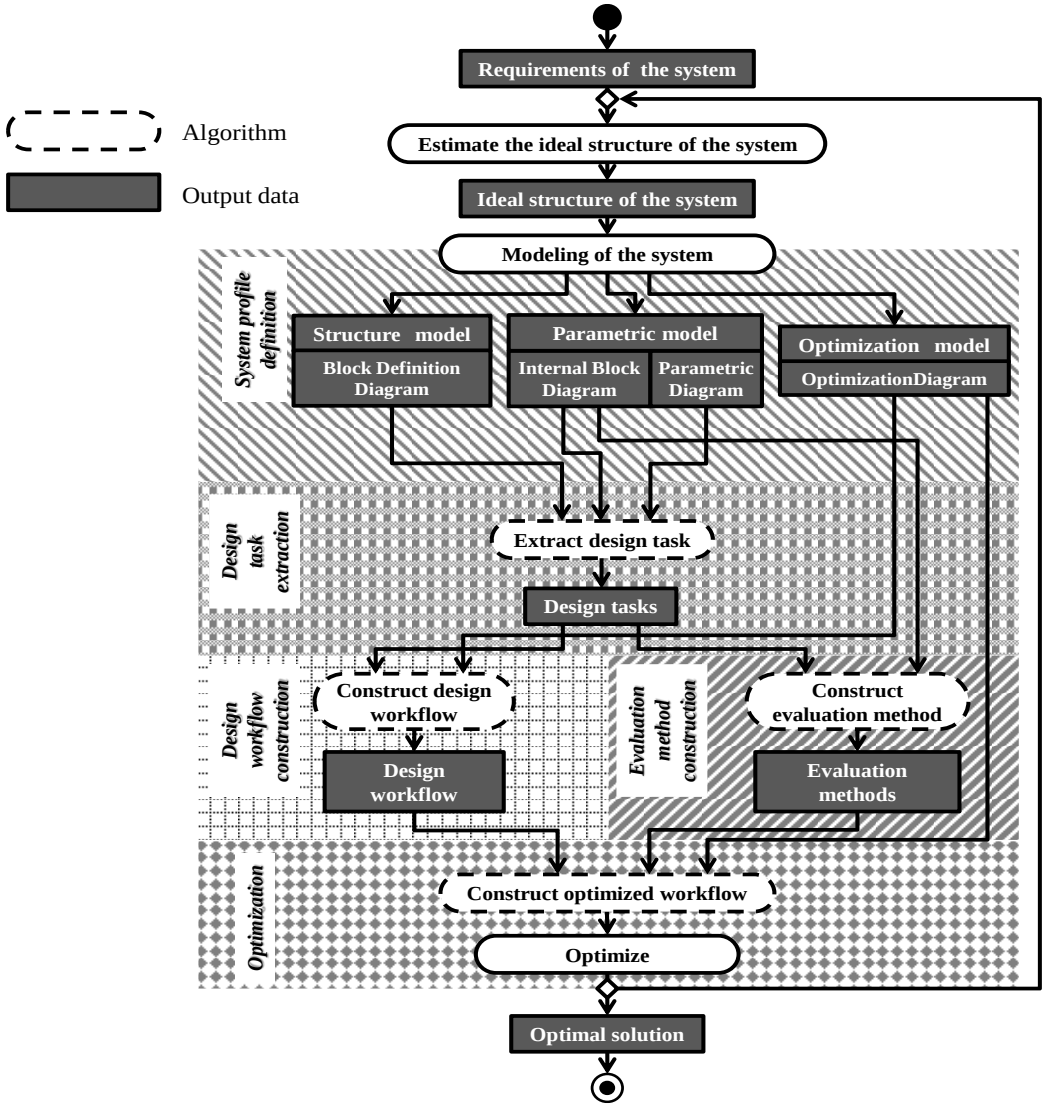


Fig. 3.39 SDSI-Cubic system design flow.

可能なものに限定して行うこととした。実装は汎用性を考慮し、多様な OS 上で動作するソフトウェアプラットフォームである Eclipse プラットフォーム上にプラグインの形で行った。その際にメンテナンス性や後の応用を考慮し、SDSI-Cubic 各面の入力作成ソフト、出力作成ソフトをそれぞれ個別に作成、また、SDSI-Cubic プラグイン上で各面の手法が個別に動作する形で実装を行った。これにより、各手法や入出力のアップデートやメンテナンスを容易に行うことが可能となった。作成した SDSI-Cubic の GUI を Fig. 3.40 に示す。

このように、SDSI-Cubic で自動最適化を行うためのアルゴリズムの開発を行い、実際に Fig. 3.41 に示すように、Eclipse プラットフォーム上に実装することで、自動最適化を可能とした。

Fig. 3.40 SDSI-Cubic software graphic user interface.

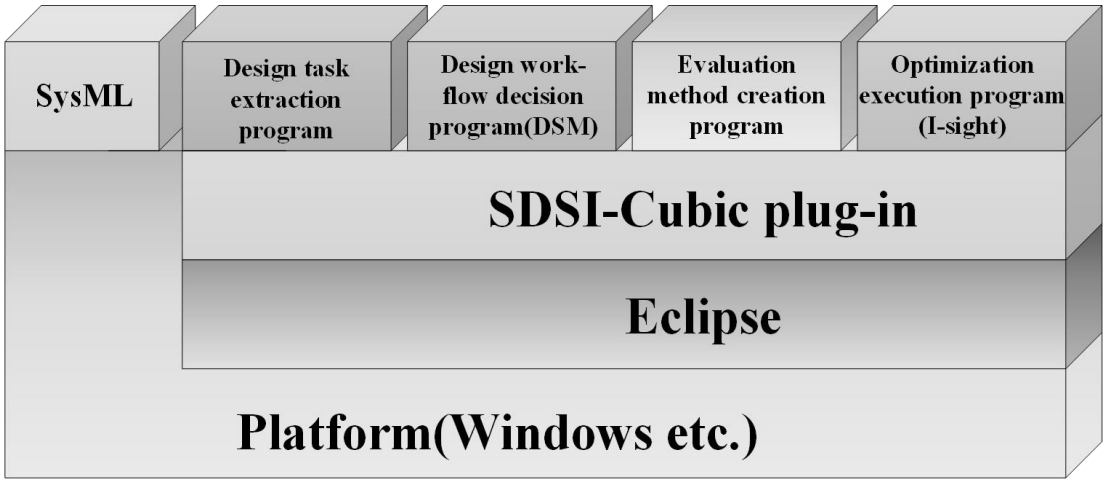


Fig. 3.41 SDSI-Cubic platform.

3.4 結言

製品システム全体を最適化するための、システムデザイン手法 SDSI-Cubic での自動最適化のために、各面を連携させるためのアルゴリズムの構築を行った結果、以下の結論が得られた。

(1) これまでの手法では、設計タスク抽出から最適化の依存関係を考慮した DSM の生成、DSM のパーティショニング結果から最適化ワークフローの導出の自動実行方法の構築

と、これらを連携させるための製品システムのモデリング手法の検討が、自動最適化のために必要であることを見出した。

(2) 連携させるための製品システムのモデリング手法として、SysML を用いた自動化のための製品システムの構造モデル、パラメトリックモデルの定義方法を構築した。また、従来のシステムモデリング手法の課題である、設計者の設計目的である最適化問題を定義するために、目的関数、設計変数、制約条件からなる最適化のメタモデルを構築し、SysML の拡張プロファイルを用いて SysML での最適化モデルの定義手法を構築した。

(3) 設計タスク抽出から最適化の依存関係を考慮した DSM の生成の自動実行の課題を解決するため、抽出された設計タスクに、最適化問題を構成する設計変数自体を設計タスクとし、目的関数と設計変数の間の依存関係を追加する新しいアルゴリズムを構築した。

(4) DSM のパーティショニング結果から最適化ワークフローの導出の自動実行の課題を解決するため、DSM のパーティショニング結果である並列、連続、結合関係を最適化ワークフローに変換するための一般的な解釈方法を提案し、それを行うためのアルゴリズムを構築した。

(5) (2)の自動化のためのモデリング手法と(3)(4)のアルゴリズムを本システムデザイン手法に組み込み、入出力ファイル群のフォーマットを規定することで、自動最適化フレームワークを構築し、工学的な適用性、応用性を考慮し、Eclipse 上にプラグインの形でソフトウェアの実装を行った。

参考文献

1) Sanford Friedenthal, Alan Moore, Rick Steiner, “A Practical Guide to SysML”, Morgan Kaufanm, (2009).
2) Yue Cao, Yusheng Liu , Hongri Fan, Bo Fan,“SysML-based uniform behavior modeling and automated mapping of design and simulation model for complex mechatronics”, Computer-Aided Design, 45 (2013) 764–776.
3) Object Management Group, ” SysML-Modelica Transformation”, http://www.omg.org/spec/SyM/1.0/PDF/
4) 目代武史, “製品開発マネジメントの分析ツールとしての設計構造マトリックスに関する考察”, 地域経済研究, 17 (2006) 25-42.
5) Tyson R. Browning, “Process Modeling with Design Structure Matrices (DSMs)”, INCOSE INSIGHT, (1999) 15-17.
6) 関研一, 西村秀和, 朱紹鵬, Laurent Balmelli, “民生機器開発における機能・構造モデルを用いた分散協調設計(SysML 製品モデルと DSM を利用したモジュール設計プロセスの計画)”, 日本機械学会論文集(C 編), 78-785 (2011) 187-200.
7) Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II”, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, 6-2 (2002) 182-197.
8) G. L. Nemhauser, M. J. Todd, A. H. G. Rinnooy Kan, “最適化ハンドブック” 朝倉書店, (1995).

第4章　三次元化を考慮した次世代携帯電話用システム LSI のシステムデザイン

4.1　緒言

第 2 章で、製品システム全体を最適化するための、6 面からなるシステムデザイン手法 SDSI(System Design System Integration)-Cubic を構築し、第 3 章で自動最適化のためのシステムモデリング手法の構築、アルゴリズムの構築、連携手法の構築を行い、ソフトウェアの実装まで行った。

本章では、構築したシステムデザイン手法 SDSI-Cubic を用いて、大規模・複雑かつ、要求の多様化が叫ばれている次世代携帯電話用システム LSI(Large Scale Integration)に応用展開を行い、本システムデザイン手法による適正設計の可能性を明らかにしたものである。

4.2　次世代携帯電話用システム LSI への適用結果と考察

本研究では、SDSI-Cubic の応用展開の対象として次世代携帯電話用システム LSI をとりあげる。以下では、まず、次世代携帯電話用システム LSI をとりあげた理由を述べ、その後、実際に適用を行い、適正解の導出とその考察を行う。

4.2.1　次世代携帯電話用システム LSI の背景と本手法の適用理由

携帯電話用システム LSI は、携帯という観点から、小型化、オールインワン化が求められている [1)]。現在は、通信技術として第 3 世代から第 4 世代の通信方式を採用した携帯電話が多く使われており、最先端の物では第 3.9 世代である LTE(Long Term Evolution)や WiMAX(Worldwide Interoperability for Microwave Access)を用いたものが流通し始めており、通信速度の高速度化が進んでいる。そのため、Fig. 4. 1 に示すように、第 4 世代以降では、より高性能（通信の速度の向上)、多機能な携帯電話が必要とされると考えられる。また電池の進化とともに改善しつつあるが、まだまだ携帯電話の稼働時間は短く、端末の価格も多機能になるにつれ、高価格化している。そのため、将来の携帯電話には、小型高性能、低価格かつ低消費電力が求められる。

このように携帯電話はわずか 105mm^3(100cc)、100g 前後の超小型、軽量で高度な機能を実現する究極の製品システムであるが、その製品システムを実現するキーデバイスがシステム LSI である。このシステム LSI は CPU (Central Processing Unit)、各種制御素子、各種メモリなどの多くのモジュールから構成された複雑系の製品システムである。システム

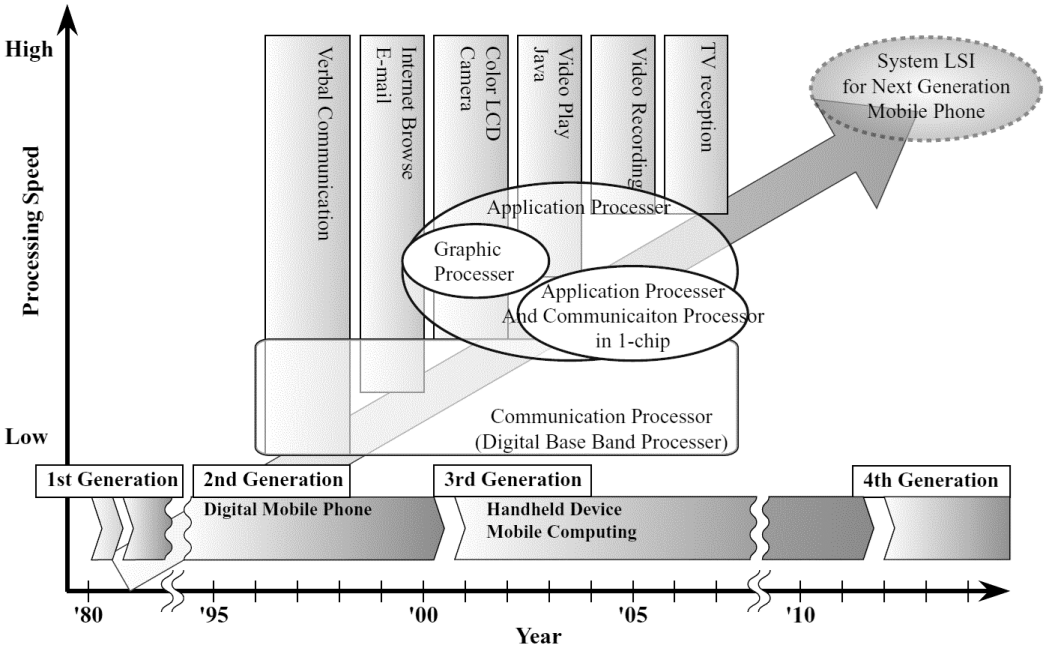


Fig. 4.1 Roadmap of mobile phone and processor unit in mobile phone[2,3].

LSI は携帯電話という製品システムの大半を LSI の中に凝縮させたものである。

また、システム LSI は高度情報化社会のコアとして、著しい進歩をしてきた。その要因となるのがムーアの法則 [4](Fig. 4.2)と呼ばれる集積度増加に関する経験則であり、またそれに伴う設計のゴールデンルールであるスケーリング則 [5](Table 4.1)である。しかし近年、微細化に伴う開発・製造費の高騰や、リーク電流による消費電力の増加と発熱量の増加、配線遅延時間の増加、低電圧化の技術的障壁など様々な問題 [8-10]を抱えている。特に、性能、機能の増大に伴って、論理回路規模やメモリ規模が増大し、チップサイズの大型化、グローバル配線の増加と微細化とのトレードオフ [10,11]が問題となっている。また、それらの課題を乗り越えたとしても半導体素子のサイズが 10 数 nm になると、物理的な動作限界を迎えるといわれている。

それに加え、近年のシステム LSI に対する要求は、高性能化だけではなく、高信頼性や環境性、携帯電話などのモバイル向けには低消費電力化や低コスト化など様々あり、これらを満たす設計はますます困難になってきている。

このような背景から、SoC(System on Chip)に代表される二次元集積のみでは要求仕様を満足するための最適な解を与えることが困難になってきており、SiP(System in Package)と呼ばれる、三次元に LSI を集積する新しい集積方式 [11-16]が検討され始めている。しかし、3D-SiP は新規技術であるため設計の方向性が決まっておらず、様々な構成が提案されてい

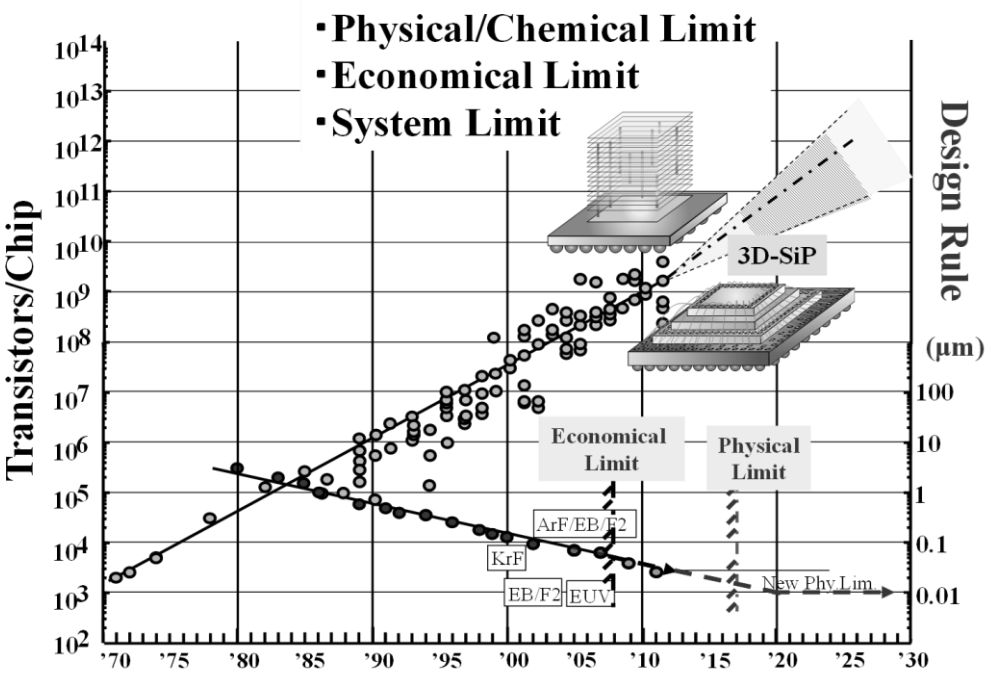


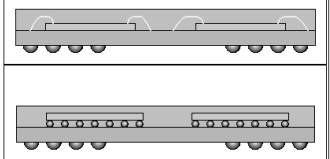
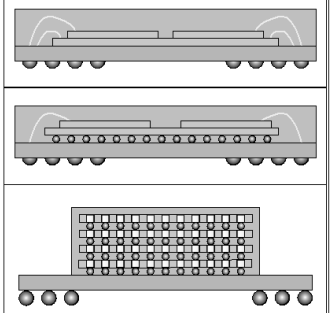
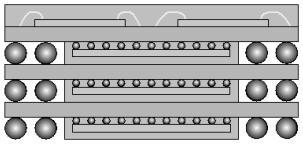
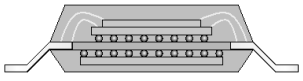
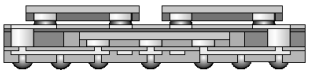
Fig. 4.2 Physical, system and economic limitation of miniaturization.

Table 4.1 Scaling Factor [6,7].

MOSFET Device or Circuits Parameters		Scaling Factor (k > 1)
Device Size	x	1/k
Voltage	V	1/k
Electric Field	E∼V/x	1
Area	S∼x^2	$1/k^2$
Circuit Density	1/S	k^2
Current	I∼$V^{1.4}/x$	$1/k^{0.4}$
Capacitance	C∼eSx	1/k
Circuit Delay	d∼CV/I	$1/k^{1.6}$
Power	P∼CV^2/d^2	$1/k^{1.4}$
Power Density	p∼P/S	$k^{0.8}$
Wire Resistance	r∼L/WT	k
Wire Capacitance	c∼LW/H	1/k
Wire Delay	rc∼L^2/TH	1
Transistor Delay	rc/d	$k^{1.6}$
etc. (Miniaturizing technology (ArF, EUV))		

る(Table 4.2)。このような様々な構成に対し、評価を行い、構成を決定するためには、様々なトレードオフが存在している。例えば、3 次元化による実装コストの増大、1 枚当たりのチップサイズの削減、歩留りの向上によるチップ当たりコストの減少やチップ内配線の外部配線化によるチップ間遅延時間の増加とチップ内配線距離の減少によるチップ内遅延時間の減少、チップ毎に IO(Input Output)部を追加することによるコストの増加や消費電力の増加などである。このように考慮すべきトレードオフが多々あり、適正な製品システム構成を決定することは非常に困難な現状があるため、本手法を適用するには適切な対象であると考える。

Table 4.2 Comparison specification of various SiP structures [17,18].

Type	Aspect of Structures	Advantage	Downsizing High density	Cost	Radiation	Reliability
Plate Structure		High radiation performance and reliability	△	△	◎	◎
Stack Structure		Downsizing and increase density in one package	◎	○	○	◎
Package on Package		Amount of large memory and available to shrink	○	△	○	○
Quad Flat Package		Easy to package and low cost	△	◎	○	◎
Embedded Structure		Available to embedde various modules	◎	△	○	△

まとめるとシステム LSI は、①ゲート規模の増大による大規模複雑化、②要求の増加による設計の困難さの増加、③3D-LSI の設計ノウハウの未蓄積により多様なトレードオフ評価の困難さという 3 つの設計上の問題点を持っている。しかし、2D-LSI の際に使われてきた回路設計、熱・応力評価、信頼性評価などの様々な技術があり、これらの設計資産は有用であると考える。

また、システム LSI の設計は、Fig. 4.3 に示すようなフロー[9]で設計されるが、ハードウェア、ソフトウェアを含む詳細な設計は数か月を要する膨大な設計である。そのため、本研究では、要求仕様から製品システムの仕様を決める構想設計段階に絞ってシステムデザインの検討を行った。その際に、構想設計段階での次世代の携帯電話用システム LSI の最適なシステムデザインとして、2D-SoC と 3D-SiP の選択が重要な課題となっているため、論理構成やソフトウェアは同じと仮定し、この問題に絞ったシステムデザインを、本手法

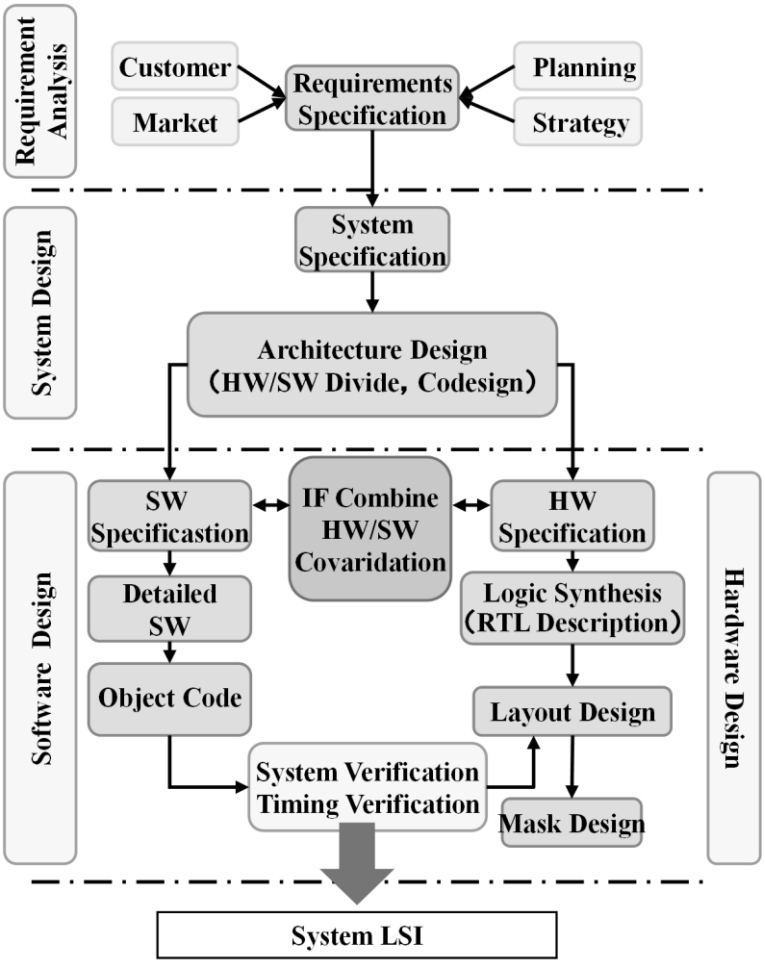
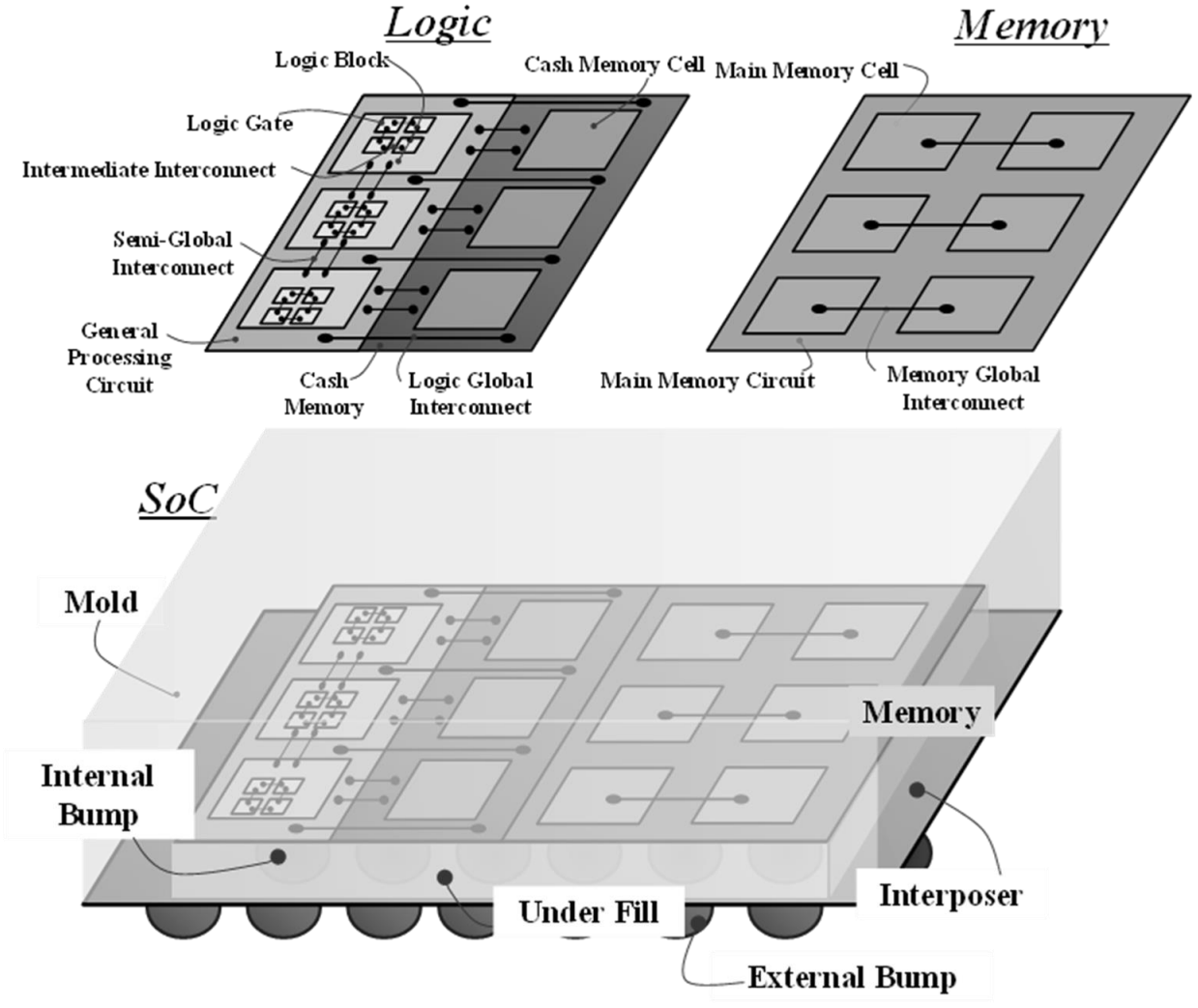


Fig. 4.3 System LSI design flow.

を用いて試みた。以下でその手順の詳細を説明する。

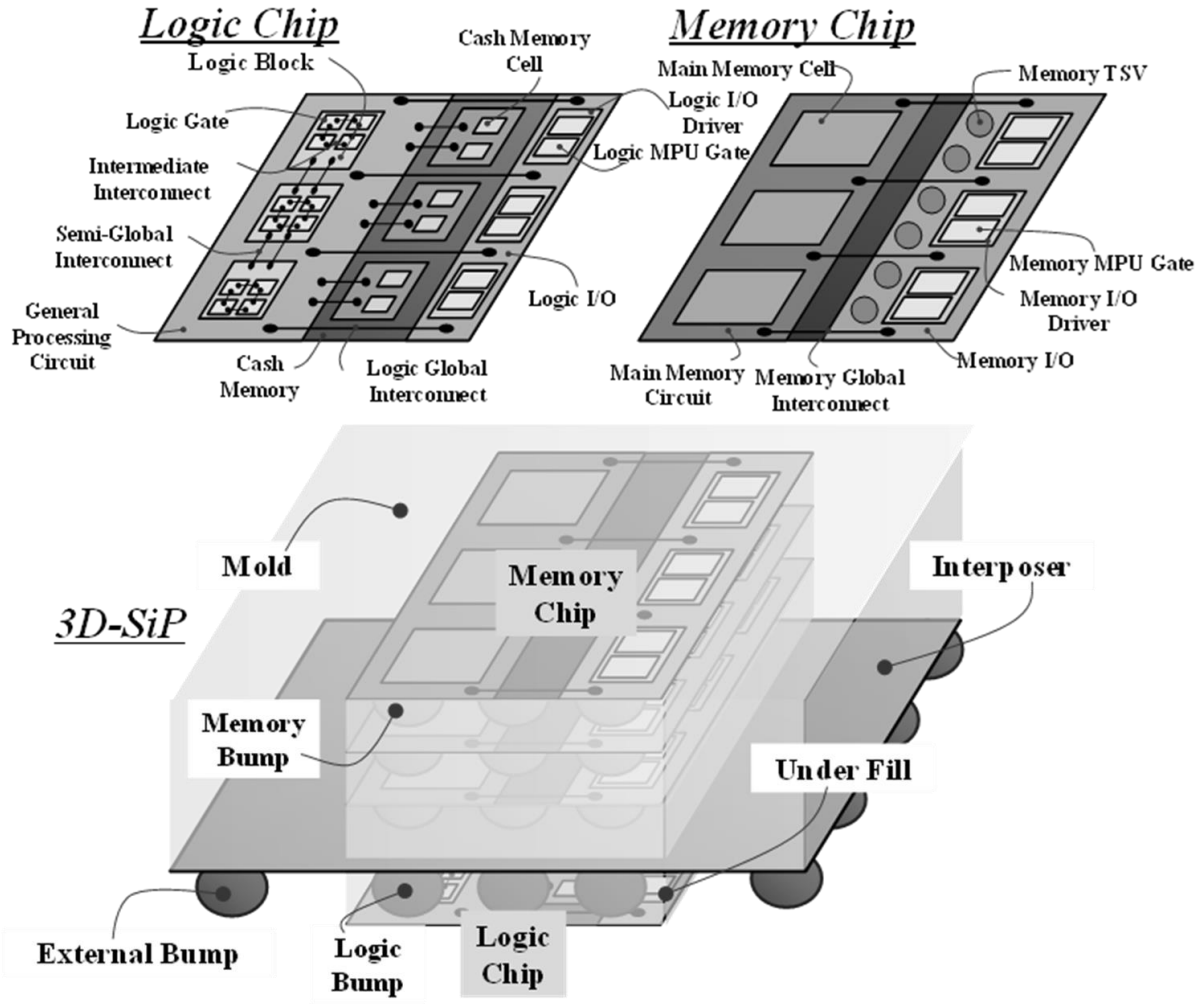
4.2.2　第 1 面(入力)の実行結果

本研究で想定したシステム LSI の構成を Fig. 4.4(SoC で構成した場合(Fig. 4.4(a))と 3D-SiP で構成した場合(Fig. 4.4(b)))に示す。本研究では、システム LSI の主要構成要素として、SoC のロジック、メインメモリ、キャッシュメモリ、といった機能ブロックを考慮し、分割を行った。SiP では、これを三次元化するために、さらに実装部として I/O 部、インターポーザ部を追加し、図の様なロジックとメモリチップからなる三次元構成を想定した。その際に、次世代携帯電話用システム LSI を対象としているため、三次元化に関わ



(a) SoC structure.

Fig. 4.4 Image of system model in this study.



(b) 3D-SiP structure.

Fig. 4.4 Continued.

る技術（TSV (Through Silicon Via)形成技術、三次元チップ積層技術など）に関しては実用化されており、2D-SoC と同等レベルの製造歩留まりがあるという前提、かつ、前工程設計や放熱も可能であるという前提をおいた。設計変数は半導体のマクロな重要設計項目である、テクノロジーノード、信号ピン数、メモリチップ分割数を入力した。入力した設計変数を Table 4.3 に示す。目的関数は、上記より、市場の要求が、コストパフォーマンス、すなわち機能の増加を含む性能の向上と、低消費電力および低コストであることから、これらの相反問題を考慮し、式(4.1) に示す、SDSI-C という係数が最大になることが理想と考えた。

$$SDSI\text{-}C=Performance(GIPS)/PowerConsumption(W)/Cost(\$) \quad (4.1)$$

4.2.3　第 2 面(システムプロファイル定義/設計タスク抽出)の実行結果

ここでは想定した理想構成(Fig. 4.4)に対し、システムプロファイル定義として、第 3 章で述べた SysML(Systems Modeling Language)によるシステムモデリング手法を用いて要求モデル、構造モデル、パラメトリックモデル、最適化モデルの定義を行った。Fig. 4.5-Fig.4.7 に、要求モデルを定義したコンテキスト図、ユースケース図、要求図を、Fig. 4.8 に構造モデルを定義したブロック定義図を示す。要求モデルは、入力である理想構成導出の際に考慮したものを記載している。コンテキスト図には、システム LSI の外部環境として、使用するユーザ、筐体、周辺回路、電源、また、システム LSI に実装されるソフトウェアを考慮し、定義した。ユースケース図は、コンテキスト図で定義した外部環境からの要求として、性能、コスト、消費電力、体積、汎用性を考慮し、定義した。要求図は、ユースケースの要求項目を考慮し、システム LSI の要求の細分化を行った。構造モデル(SoC の構造モデル Fig. 4.8(a)と SiP の構造モデル Fig. 4.8(b))は、要求モデルから想定された理想

Table 4. 3 List of design variables for this study.

Design Variable Name	Value
Number of Memory Chips	1,2,4,8,16,32,64
Number of Signal I/O Pins	64,128,256,512,1024,2048
Technology Node	45nm,65nm,90nm

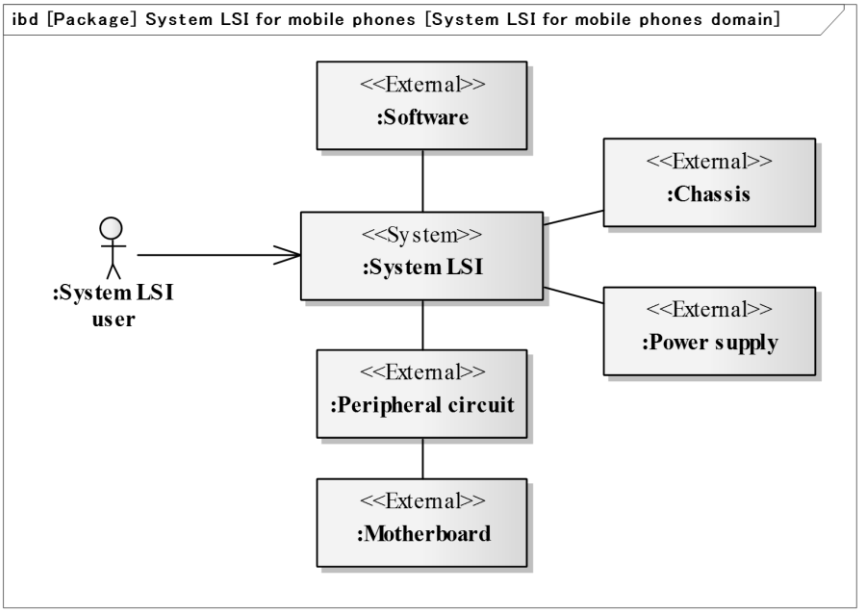


Fig. 4.5 Context diagram.

構成、即ち、Fig. 4.4 に示す理想構成のモデリングを行った結果である。

次に、要求モデル、構造モデルをもとに、パラメトリックモデルの定義を行う。そのため、目的関数として設定した式(4.1)の目的関数を導出するために製品システムを物理的・経済的な観点から捉え､パラメトリックな物理モデル･経済モデルを表現する必要がある。以下では、まず、物理モデルの検討結果を述べ、その後経済モデルの検討結果を述べる。

まず、物理モデルの検討として、式(4.1)の目的関数に含まれる、システム LSI の性能に関する構築結果を述べる。

システム LSI の性能は、三次元集積化のためにチップを分割することによる、メリットとして配線長の短縮により、配線遅延時間の短縮が期待できる。その一方で、微細化を単純なチップ分割のみでは微細化を進展による性能の向上を期待できないというデメリットもある。

システム LSI はロジックチップ内の汎用回路で命令を処理し、必要なデータをロジックチップ内にあるキャッシュメモリに呼び出しに行く。そして、キャッシュメモリにヒットしなかったデータをメインメモリのメモリチップの呼び出しに行く。そのため、システム LSI のパッケージとしての性能は次式であらわされる。

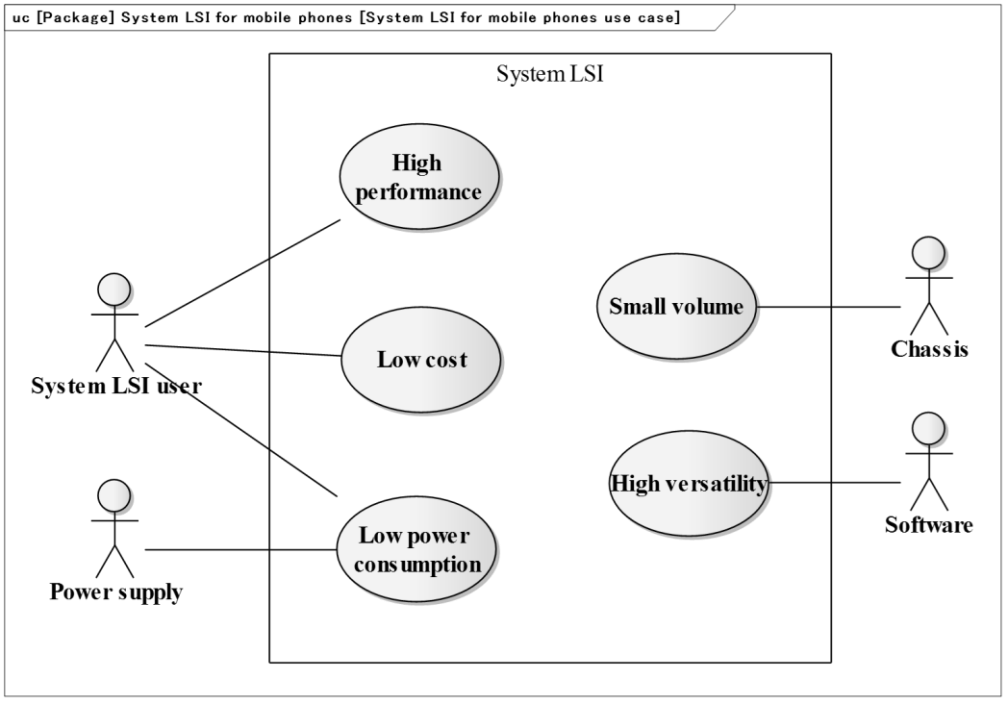


Fig. 4.6 Use case diagram.

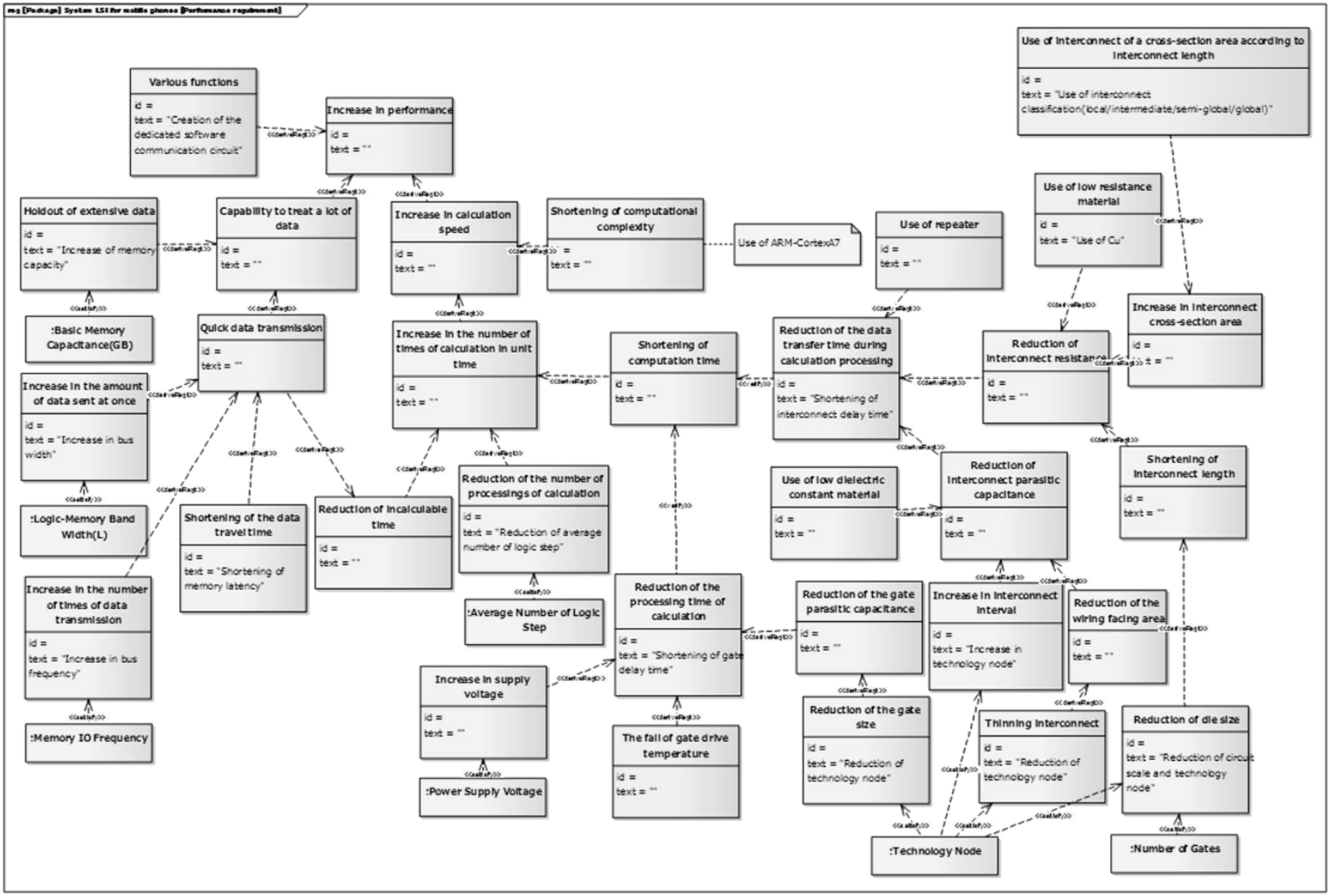
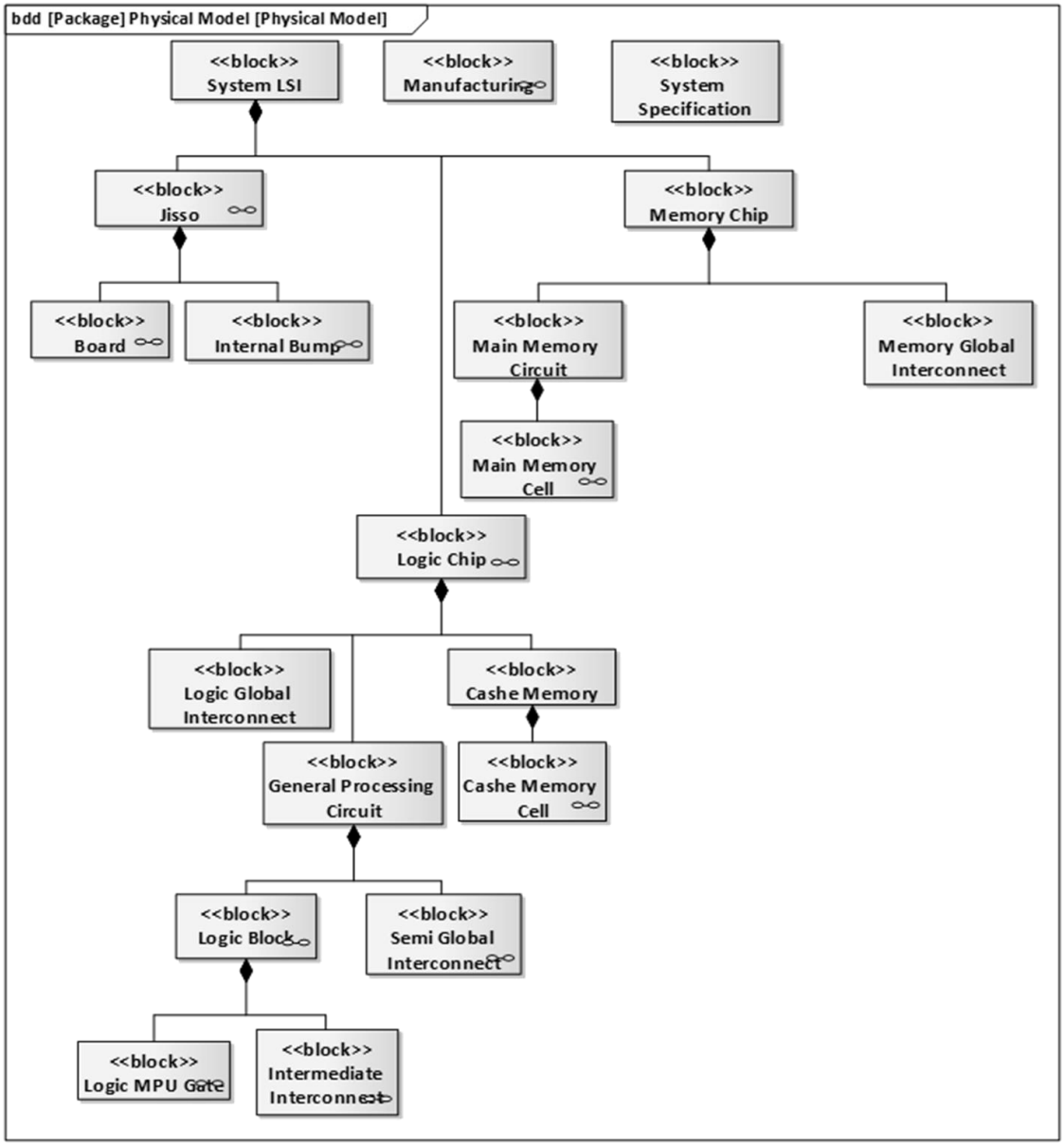


Fig. 4.7 Requirement diagram.



(a) SoC.

Fig. 4.8 Block definition diagram.

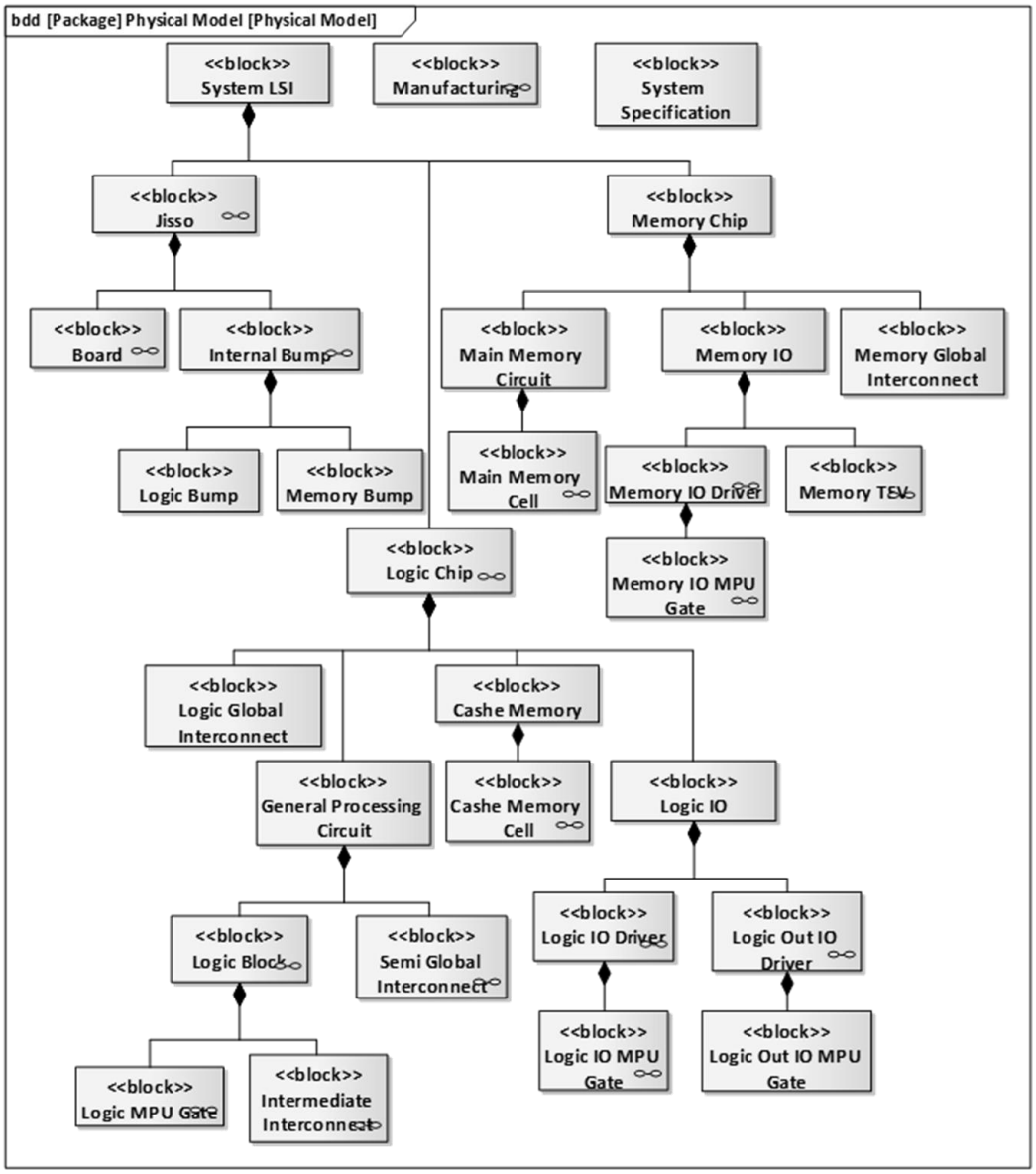
$$P_{System} = P_d \times H_c + P_d \times (1 - H_c)/l_r \tag{4.2}$$

P_d : システム LSI 性能(GIPS)、P_d : 汎用処理回路処理速度(GIPS)、H_c : キャッシュヒット率、l_r : 総レイテンシサイクル(cycle)

ここで汎用回路の処理速度は、1 サイクルで処理可能な平均命令実行数 N_{oave} と駆動周波数 F_f(GHz)により次式で求められる。

$$P_d = F_f \times N_{oave} \tag{4.3}$$

平均命令実行数は汎用演算回路の回路規模に依存し、次式で求められる。



(b) SiP.

Fig. 4.8 Continued.

$$N_{oave} = R_{oave} \times \sqrt{N_g} \tag{4.4}$$

R_{oave}：基準平均命令実行数、N_g：汎用ロジック回路ゲート数(gate)

駆動周波数は信号の伝播に要する最大の遅延時間によって求められる。いま、信号は Fig. 4. 9 に示す配線の各配線層を通過して、チップ内で伝送される。この中で、最も信号の伝送に要する遅延時間よって、駆動周波数 F_f は次式により導出できる。

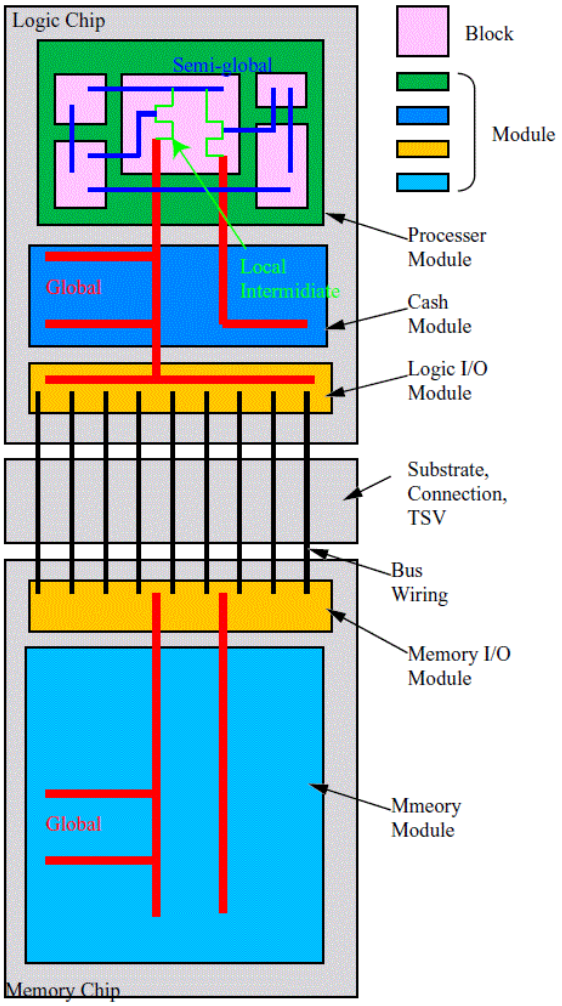


Fig. 4.9 The hierarchic interconnect model[19].

$$F_f = 1/t_{d\max} \tag{4.5}$$

$$t_{d\max} = Max\left(t_{dGlobal}, t_{dSemi}, t_{dInter}\right) \tag{4.6}$$

t_{dmax}：最大命令実行時間(s)、$t_{dGlobal}$：グローバル遅延時間(s)、t_{dSemi}：セミグローバル遅延時間(s)、t_{dInter}：インターミディエイト遅延時間(s)

汎用ロジック回路モジュールとキャッシュメモリモジュールを同期させると仮定する。このとき、グローバル遅延時間、セミグローバル遅延時間、インターミディエイト遅延時間はそれぞれ､ロジックチップ内での遅延時間､汎用演算回路モジュール内での遅延時間、論理ブロック内での遅延時間を意味し、以下の式により導出する。

$$t_{dGlobal} = T_{Global} \times \sqrt{\left(S_{GP} + S_C\right)} \times T_{S,Global} \tag{4.7}$$

T_{Global}：グローバル配線単位長配線遅延時間(s/mm)、$T_{S,Global}$：グローバル配線同期周期、S_{GP}：汎用ロジック回路モジュール面積(mm^2)、S_c：キャッシュメモリモジュール面積(mm^2)

$$t_{dSemi} = T_{Semi} \times \sqrt{S_{GP}} / 2 \times T_{S,Semi} \tag{4.8}$$

T_{Semi}：セミグローバル配線単位長配線遅延時間(s/mm)、$T_{S,Semi}$：セミグローバル配線同期周期

$$t_{dInter} = t_{step} \times N_{s,ave} \times t_{skew} \tag{4.9}$$

t_{step}：段あたりの平均配線遅延時間(s)、$N_{S,ave}$：平均論理段数、t_{skew}：スキュー時間(s)

$$t_{step} = T_{Inter} \times P_{Inter} \times L_{Inter,Block} + t_{gate} \tag{4.10}$$

T_{Inter}：インテーミディエイト配線単位長配線遅延時間(s/mm)、P_{Inter}：インターミディエイト配線ピッチ(mm/pitch)、$L_{Inter,Block}$：ブロック内配線ピッチ数(pitch)、t_{gate}：ゲート遅延時間(s)

キャッシュヒット率はキャッシュ容量と動作するソフトウェアに依存するが、ここでは動作するソフトウェアの中身を一定とし、キャッシュ容量に依存する以下の式より導出する。

$$H_c = R_{Cash} \times \exp(N_C) \tag{4.11}$$

R_{Cash}：キャッシュヒット率係数

キャッシュにヒットしなかったデータは、メインメモリでデータの参照を行う。その際に要する時間をプロセッサ内でのクロック数に換算した値が、レイテンシサイクルである。そのため、レイテンシサイクルはメモリチップ内での遅延時間と、チップ間での遅延時間により以下の式により与えられる。

$$l_r = \frac{t_{Memory} + t_{Jisoo} \times 2}{F_f} \tag{4.12}$$

t_{Memory}：メモリチップ内遅延時間(s)、t_{Jisso}：チップ間遅延時間(s)

$$t_{Memory} = t_{Mem,Cell} + T_{Global} \times \sqrt{S_M} \times 2 \tag{4.13}$$

$t_{Mem,Cell}$：メモリセルでの R/W に要する時間(s)、S_M：メインメモリモジュール面積(mm^2)

SiP の場合には基板と実装部を通過するのに要するチップ間遅延時間が発生し、これは次式により求められる。

$$t_{Jisso} = t_B + N_C \times t_J \tag{4.14}$$

t_B：インターポーザ基板遅延時間(s)、t_J：実装部遅延時間

ロジックチップとメモリチップ間の信号伝送の帯域および周波数は、以下の式により導出する。

$$W_{Bus} = P_d \times (1 - H_c) \times L_o \tag{4.15}$$

W_{Bus}：バス幅(bps)、L_o：命令長(bit)

ここで、バスのピン数は信号: 電源+ グランド=1: 2 を想定しているので、信号ピン数は全体のピン数の三分の一になるので、バス周波数は以下の式で表わされる。

$$F_B = \frac{W_{Bus}}{(N_{I/O,Mem}/3)} \tag{4.16}$$

以上からシステム LSI のパッケージ全体での処理速度を導出する。

次に式(4.1)の目的関数に含まれる、システム LSI の消費電力に関する構築結果を述べる。

消費電力は近年の半導体で問題となっている。その原因は、発熱密度の上昇と静的消費電力の増大である。特に、静的な消費電力の増大の主要な原因となるリーク電流は、デザイルール微細化の進展に伴い大きな問題となっている。これを考慮して、消費電力の算出を行うことが重要である。リーク電流は、Fig. 4.10 に示すように、おもに 3 種類の電流の漏れに分類される。今回の検討ではこの中でも、最も影響の大きいサブスレッショルドリーク電流に着目して検討を行う。

三次元集積化のために、微細化の進展ではなくチップを分割することによって、静的消費電力を抑制できるメリットがある一方で、各チップに搭載される I/O モジュールの影響によって総消費電力が増大するというデメリットがある。システム LSI はメモリチップ、ロジックチップそして、基板、実装部からなっている。この中で、消費電力を考えるのはメモリチップ及びロジックチップであるので、システム LSI の消費電力は以下の式であらわされる。

$$E_{System} = E_{Logic} + E_{Memory} \times N_{Chip} \tag{4.17}$$

E_{Logic}：ロジックチップ消費電力(W)、E_{Memory}：メモリチップ消費電力(W)

いま、ロジックチップの消費電力は汎用演算回路モジュールによる消費電力、キャッシュメモリによる消費電力、I/O モジュールによる消費電力そして、配線による消費電力からなっているので、次式より求まる。

$$E_{Logic} = E_{PG} + E_{Cash} + E_{I/O,L} + E_{Wire} \tag{4.18}$$

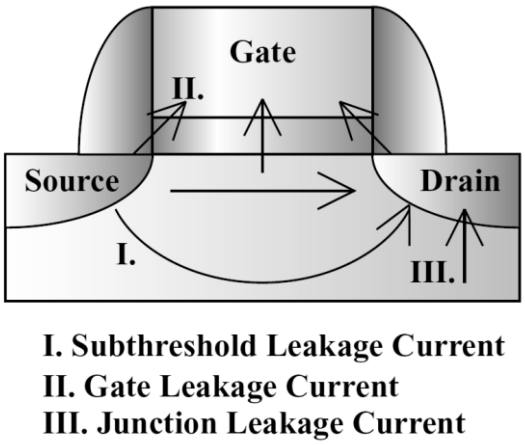


Fig. 4.10 The image of leakage current and its category.

E_{PG}：汎用演算回路モジュール消費電力(W)、E_{Cash}：キャッシュメモリモジュール消費電力(W)、$E_{I/O,L}$：I/O モジュール消費電力(W)、E_{Wire}：配線消費電力(W)

汎用演算回路モジュールの消費電力は静的な消費電力と､動的な消費電力に分けられる。ここで、動的な消費電力はトランジスタのスウィッチングによって、電荷の充放電が繰り返されることによって消費される。また、静的な消費電力はリーク電流、特に閾値電圧以下の電圧で漏れて流れるサブスレッショルドリーク電流に着目すると、汎用演算回路モジュールの消費電力は次式であらわされる。

$$E_{PG} = \frac{1}{2} \times C_g \times {V_{dd}}^2 \times N_g \times F_f \times R_D + I_{leak,sub} \times W_g \times N_g \times V_{dd} \quad (4.19)$$

C_G：ゲート寄生容量(F)、V_{dd}：電源電圧(V)、R_D：駆動率、$I_{leak,sub}$：サブスレッショルドリーク電流密度(A/mm)、W_g：ゲート幅(mm)

同様にキャッシュメモリも静的な消費電力と、動的な消費電力に分割できるので、次式により求められる。

$$E_{Cash} = P_{Dynamic,Cash} \times N_{Cash} \times F_f \times R_a + P_{Static,Cash} \times N_{Cash} \quad (4.20)$$

$P_{Dynamic,Cash}$：キャッシュメモリ 1 セルあたりの動的消費電力(W/Hz-Cell)、R_a：活性化率、$P_{Static,Cash}$：キャッシュメモリ 1 セルあたりの静的消費電力(W/Cell)

I/O ドライバは複数の MPU ゲートによって形成されていることを想定しているので､汎用演算回路モジュールと同様に算出することができ、次式であらわされる。

$$E_{I/O,L} = \frac{1}{2} \times C_g \times {V_{dd}}^2 \times \left(\frac{N_{I/O,L}}{3} \times R_D \right) \times F_B + I_{leak,sub} \times W_g \times \left(\frac{N_{I/O,L}}{3} \times R_D \right) \times V_{dd} + E_{I/O,O} \quad (4.21)$$

$E_{I/O,O}$：外部端子への I/O の消費電力(W)

配線の消費電力は面積と駆動周波数によって求められ、次式であらわされる。

$$E_{Wire} = R_{Wire} \times (S_{PG} + S_{Cash}) \times F_f \tag{4.22}$$

R_{Wire}：配線の消費電力係数(W/mm^2-Hz)

メモリチップの消費電力はメインメモリモジュールによる消費電力と、I/O モジュールによる消費電力に分けられ、次式であらわされる。

$$E_{Memory} = E_M + E_{I/O,M} \tag{4.23}$$

E_M：メインメモリモジュール消費電力(W)、$E_{I/O,M}$：I/O モジュール消費電力(W)

メインメモリもキャッシュメモリと同様に静的な消費電力と、動的な消費電力に分割できるので、次式により求められる。

$$E_{Cash} = P_{Dynamic,Cash} \times N_{Cash} \times F_f \times R_a + P_{Static,Cash} \times N_{Cash} \tag{4.24}$$

$P_{Dynamic,Cash}$：メインメモリ 1 セルあたりの動的消費電力(W/Hz-Cell)、$P_{Static,Cash}$：メインメモリ 1 セルあたりの静的消費電力(W/Cell)

メモリチップの I/O モジュールもロジックチップと同様に、次式により導出する。

$$E_{I/O,M} = \frac{1}{2} \times C_g \times {V_{dd}}^2 \times \left(\frac{N_{I/O,M}}{3} \times R_D \right) \times F_B + I_{leak,sub} \times W_g \times \left(\frac{N_{I/O,M}}{3} \times R_D \right) \times V_{dd} \tag{4.25}$$

以上により、システム LSI のパッケージの消費電力を算出できる。

次に、経済モデルとして、式(4.1)の目的関数に含まれる、システム LSI のコストに関する構築結果を述べる。

三次元集積化のために、Fig. 4.11 に示すように微細化の進展ではなくチップを分割することによって、枯れたプロセスを用いることと、チップ分割によるダイサイズがすることによって歩留まりが向上するというメリットがある。いっぽうで、体積の項で述べたように、分割された各チップに搭載された I/O モジュールにより、総ダイサイズが増大し、ダイコストが増大することと、実装に要するコストの増大というデメリットがある。

$$C_{System} = C_{Die,Logic} + C_{Die,Memory} \times N_{Chip} + C_{Packaging} \tag{4.26}$$

$C_{Die,Logic}$：ロジックチップコスト、$C_{Die,Memory}$：メモリチップコスト、$C_{Packaging}$：パッケージングコスト

ロジックチップコストはチップの製造コストと設計コストに大きく分けられる。製造コストは、チップのダイサイズ、欠陥密度と世代に大きく依存する。LSI を製造する際に

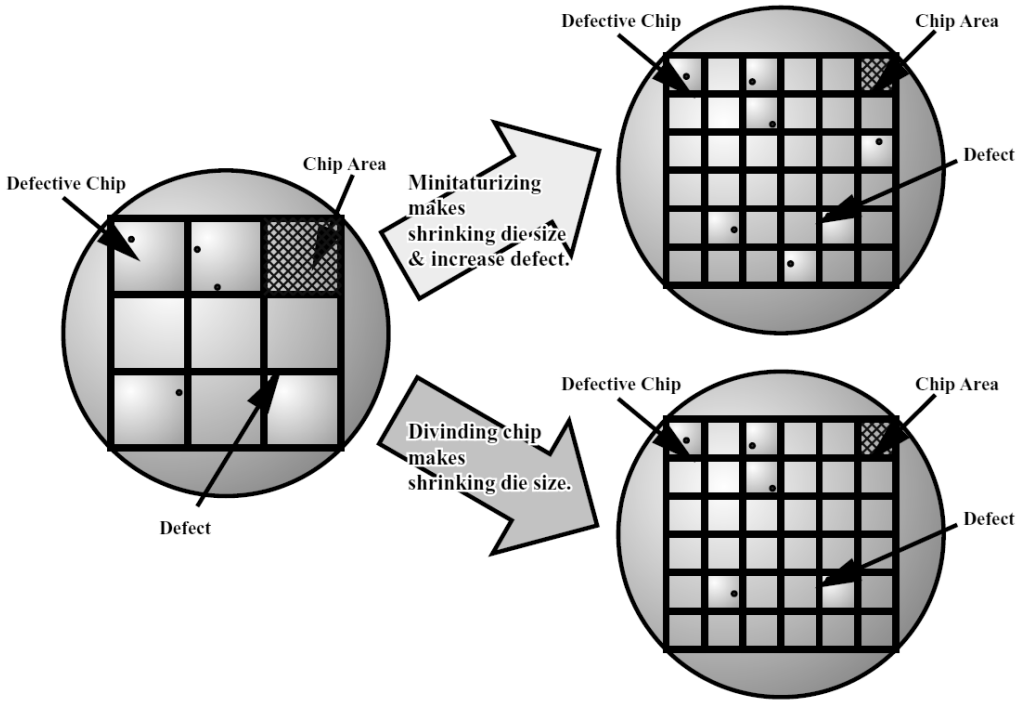


Fig. 4.11 The effect on cost of miniaturization and dividing chips.

は、フォトマスクを用いて露光と現像を行い、LSI を製造している(Fig. 4. 12)。しかし、近年、このマスクコストの増大が大きな問題となっており(Fig. 4. 13) 、専用チップとして製造する場合には、このコストを考慮する必要がある。そこで、設計コストとマスクとコストを考慮したロジックチップのコストは©ユニスリーシステムの TWO COOL®をベースに、次式で求められる。

$$C_{Die,Logic} = CoO\left(k, W_{Cut,Logic}, S_{C,L}, D_{Defect,Logic}\right) \times N_{Photo,L} + C_{Mask}/V_{\mathrm{Pr}oduct} + C_{Design}/V_{\mathrm{Pr}oduct} \quad (4.27)$$

k : テクノロジーノード(nm)、$W_{Cut,Logic}$：ロジックチップ切りシロ(mm)、$S_{C,L}$：ロジックチップ面積(mm^2)、$D_{Defect,Logic}$：ロジックチップ欠陥密度(falts/cm^2)、$N_{Photo,L}$：ロジックチップ露光回数、C_{Mask}：マスクコスト、$V_{Product}$：製造数、C_{Design}：設計コスト

ここで、*CoO* とは Cost of Ownership であり、これは SEMI によって定められた、E35 という基準に基づき算出される。SEMI-E35 は装置ライフサイクルを考えた総合コスト [22)]を定義している。また、式(4.27)は TWO COOL®を使用したデータベース引用を含めたノンパラメトリックな関係であるため、k：テクノロジーノード(nm)、$W_{Cut,Logic}$：ロジックチップ切りシロ(mm)、$S_{C,L}$：ロジックチップ面積(mm^2)、$D_{Defect,Logic}$：ロジックチップ欠陥密度(falts/cm^2)、と *CoO* の TWO COOL®での計算結果と線形補間を用い、応答曲面をあらかじめ用意することとした。

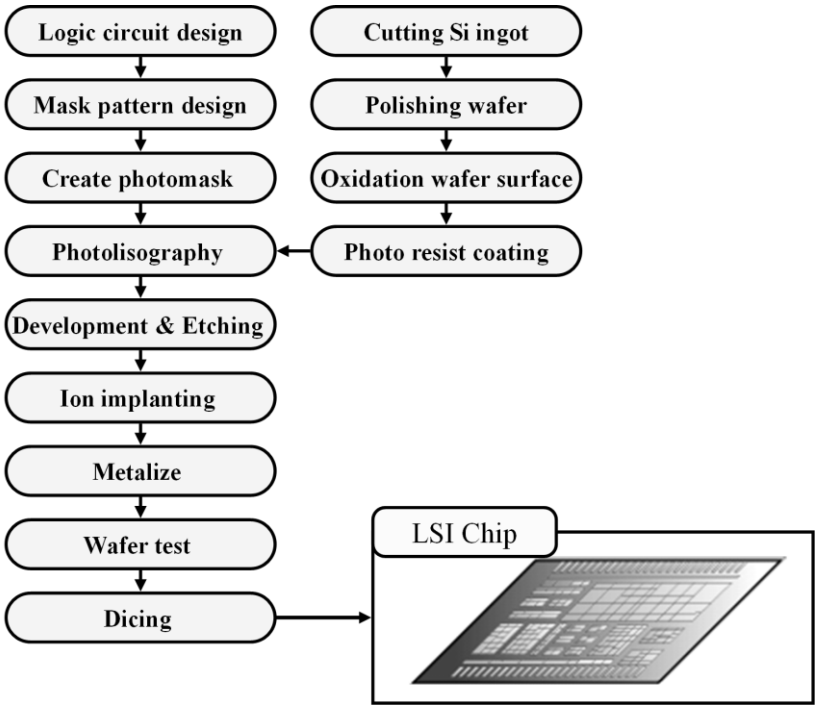


Fig. 4.12 An image of LSI chip manufacturing process (mainly front-end process) [20].

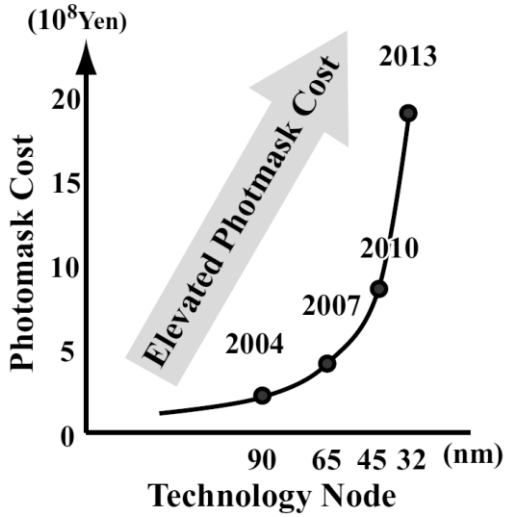


Fig. 4.13 Aspect of elevated photo mask cost by miniaturizing technology node[21].

メモリチップは汎用の製品を使用することを考えると、そのマスクコスト、設計コストは無視できると想定した。そのため、メモリチップコストは次式で表わされる。

$$C_{Die,Memory} = CoO\left(k, W_{Cut,Memory}, S_{C,M}, D_{Defect,Memory}\right) \times N_{Photo,M} \qquad (4.28)$$

k：テクノロジーノード(nm)、$W_{Cut,Memory}$：メモリチップ切りシロ(mm)、$S_{C,M}$：メモリチップ面積(mm^2)、$D_{Defect,Memory}$：メモリチップ欠陥密度(falts/cm^2)、$N_{Photo,L}$：メモリチップ露光回数

メモリチップの CoO に関してもロジックチップと同様ノンパラメトリックな関係であ

るため、上記と同様の方法で応答局面を用意した。

また､パッケージングコストはチップの積層数に依存する実装部を想定したコスト､チップ積層数に依存しないインターポーザ基板部を想定したコストとパッケージングのラインを想定したコストからなっているので、次の式より求められる。

$$C_{Package} = C_{Jisso} \times N_C + C_{Board} + C_{Line} / V_{\mathrm{Pr}oduct} \tag{4.29}$$

C_{Jisso}：実装部想定コスト、C_{Board}：インターポーザ基板部想定コスト、C_{Line}：製造ライン想定コスト

以上よりシステム LSI のパッケージ全体のコストを導出する。

このように、システム LSI のパラメトリックモデルとして、物理モデル・経済モデルを考慮し、検討を行った。

Fig. 4.14-Fig. 4.17 に、上記のパラメトリックモデルを定義した、パラメータの型定義(ブロック定義図)、関係式の型定義(ブロック定義図)、内部ブロック図、パラメトリック図の

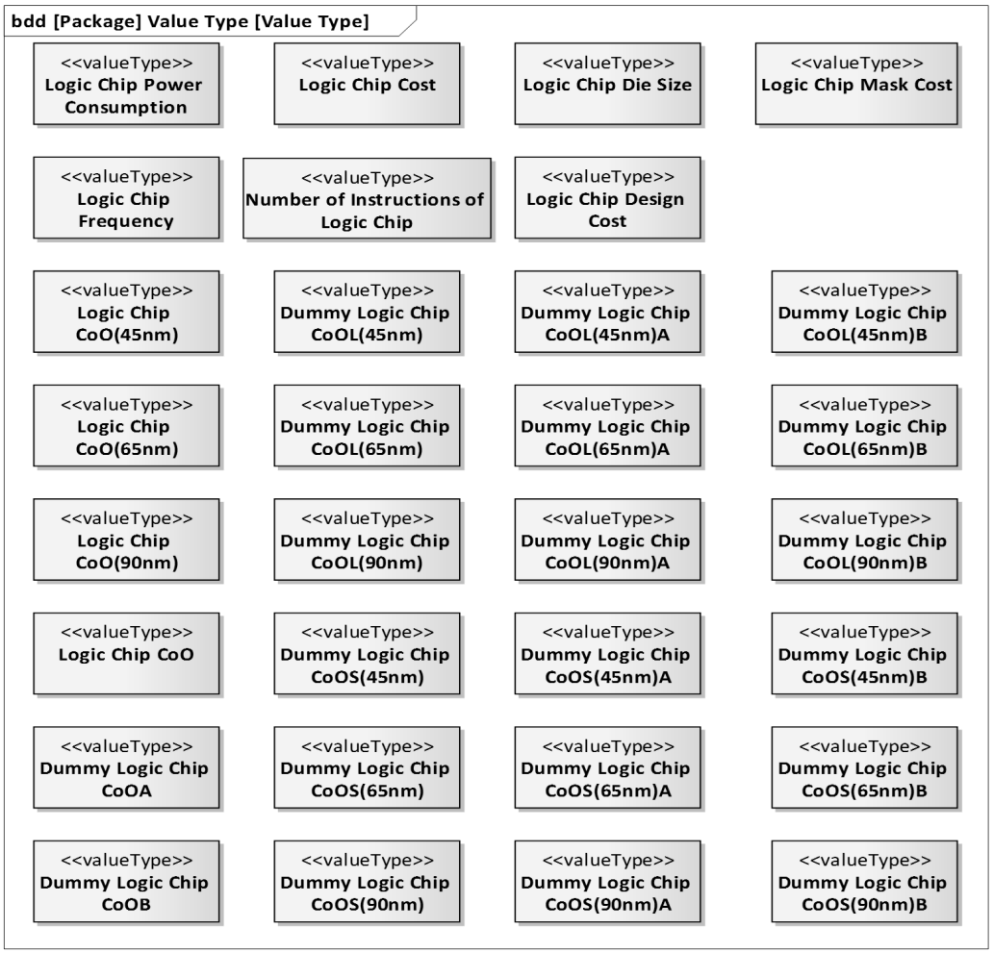


Fig. 4.14 Parameter type definition in block definition diagram.

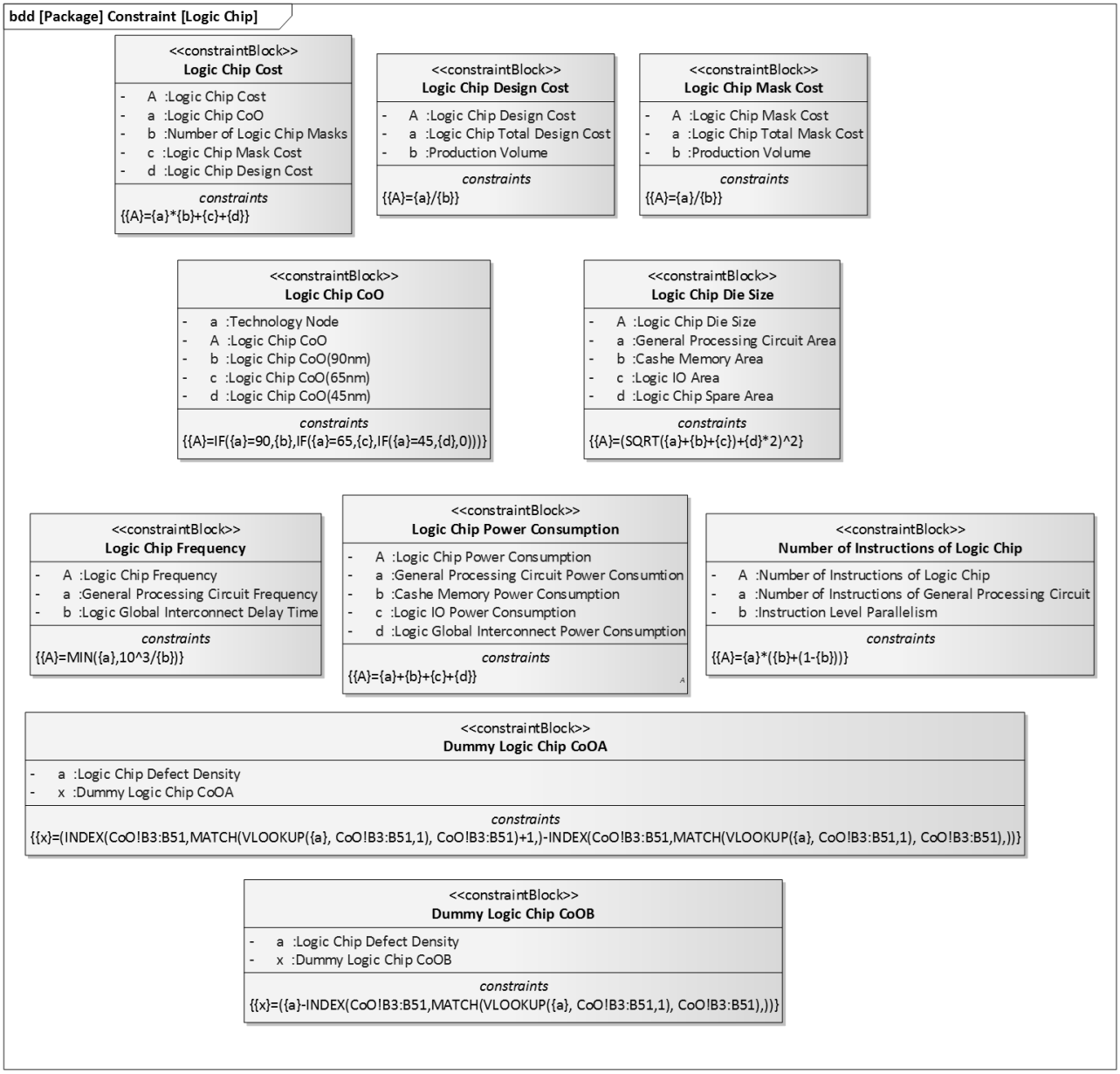


Fig. 4.15 Constraint type definition in block definition diagram.

一部を示す。また、式(4.1)と Table 4.3 をもとに定義した最適化図を Fig. 4.18 に示す。今回の例では制約条件はなかったため、定義していない。ここで載せていない残りのパラメトリックモデルに関しては、システム LSI の SysML によるシステムモデリング結果を付録とさせていただくので、そちらを参照されたい。

このように定義した製品システムのモデルに対し、本システムデザイン手法により自動的に出力されたテキストファイルを Fig. 4.19-Fig. 4.23 に示す。Fig. 4.19 が構造モデルの出力結果、Fig. 4.20 がパラメトリックモデルの内、パラメータ(型定義と内部ブロック図)の出

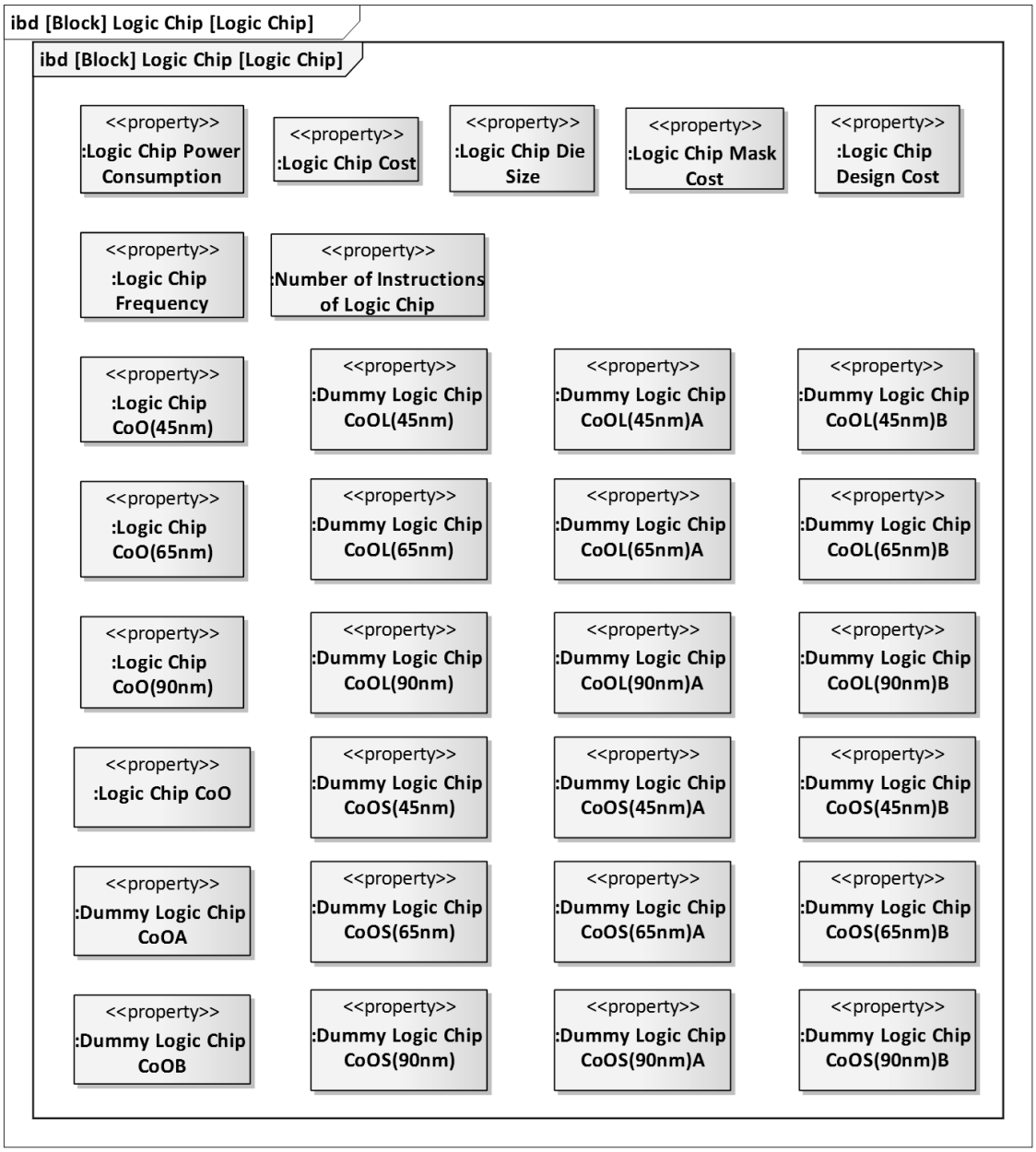


Fig. 4.16 Internal block diagram.

力結果の一部、Fig. 4.21 がパラメトリックモデルの内、関係式(型定義とパラメトリック図)の出力結果の一部、Fig. 4.22 が最適化モデルの内、目的関数の出力結果、Fig. 4.23 が最適化モデルの内、設計変数の出力結果である。図に示すように、システムプロファイルとして定義された情報が、アルゴリズム通りに出力されているのが確認できた。

このシステムプロファイル定義の出力結果の内、構造モデル、パラメトリックモデルの出力ファイルをもとに、本アルゴリズムにより、設計タスク抽出を自動的に実行した。Table 4.4 に実行結果として導出された設計タスクを示す。Table4.4 に示すように、34 個の設計タ

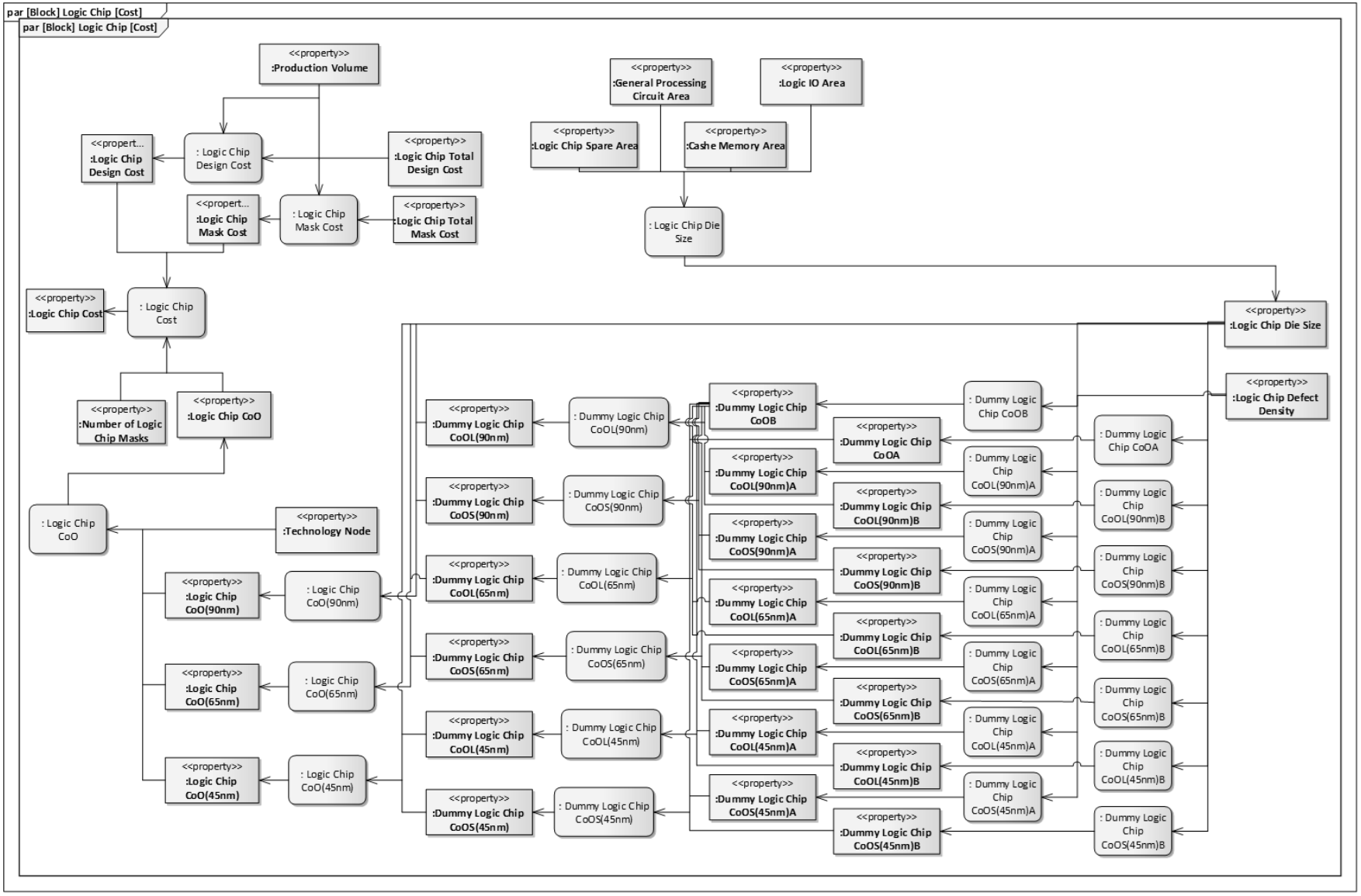


Fig. 4.17 Parametric diagram.

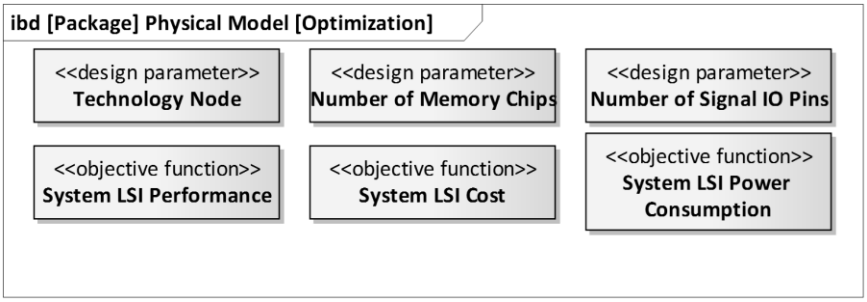


Fig. 4.18 Optimization diagram.

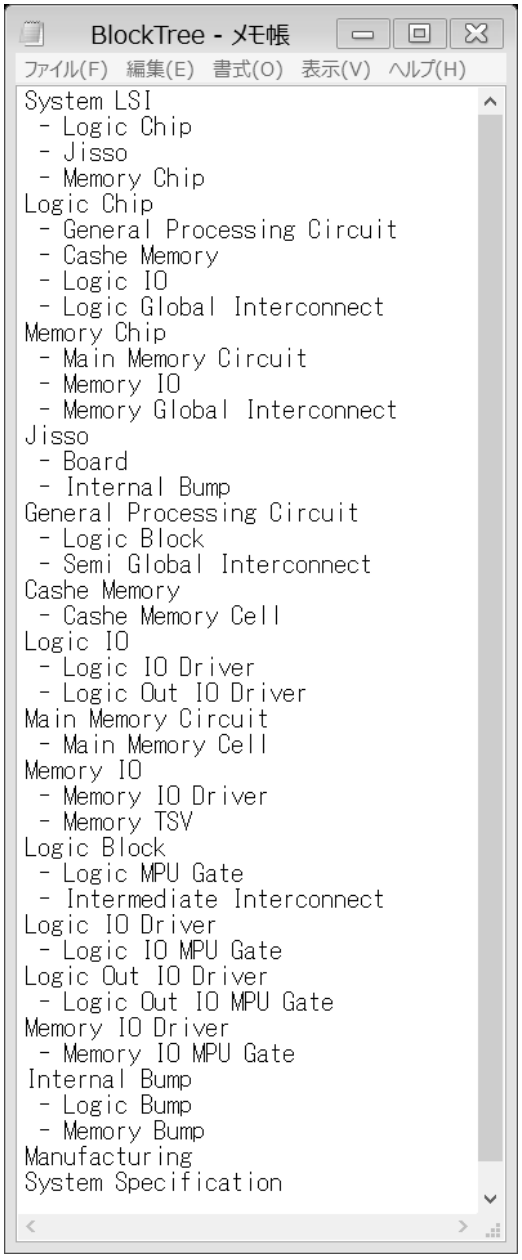
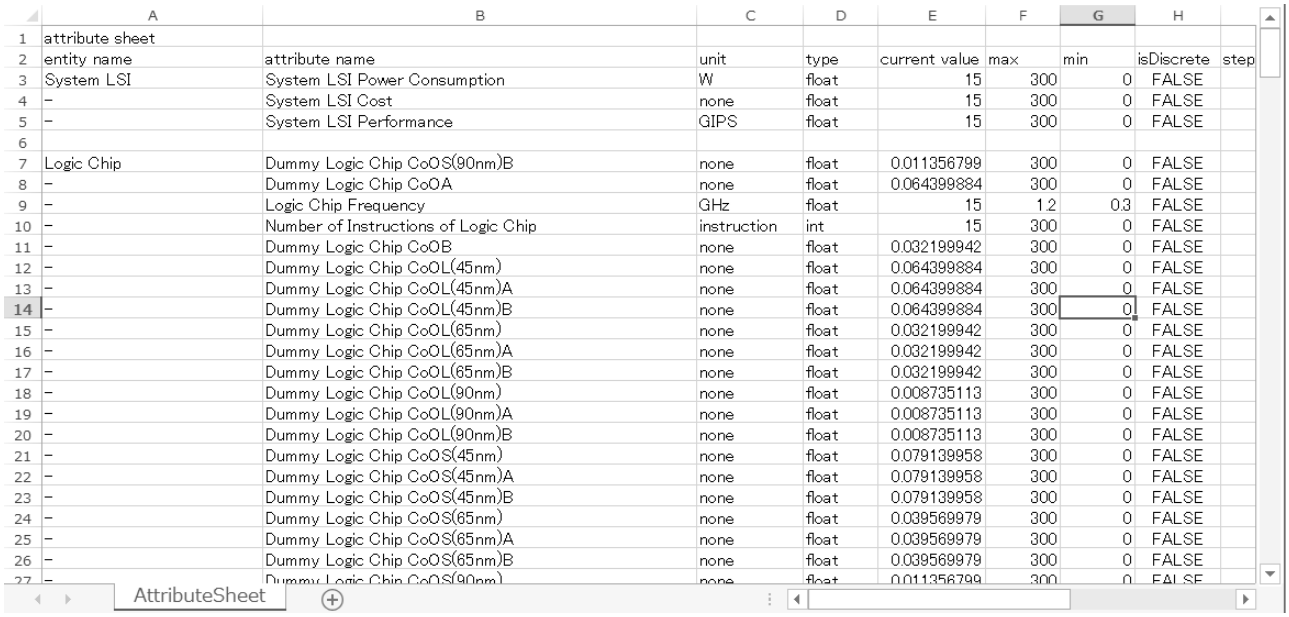


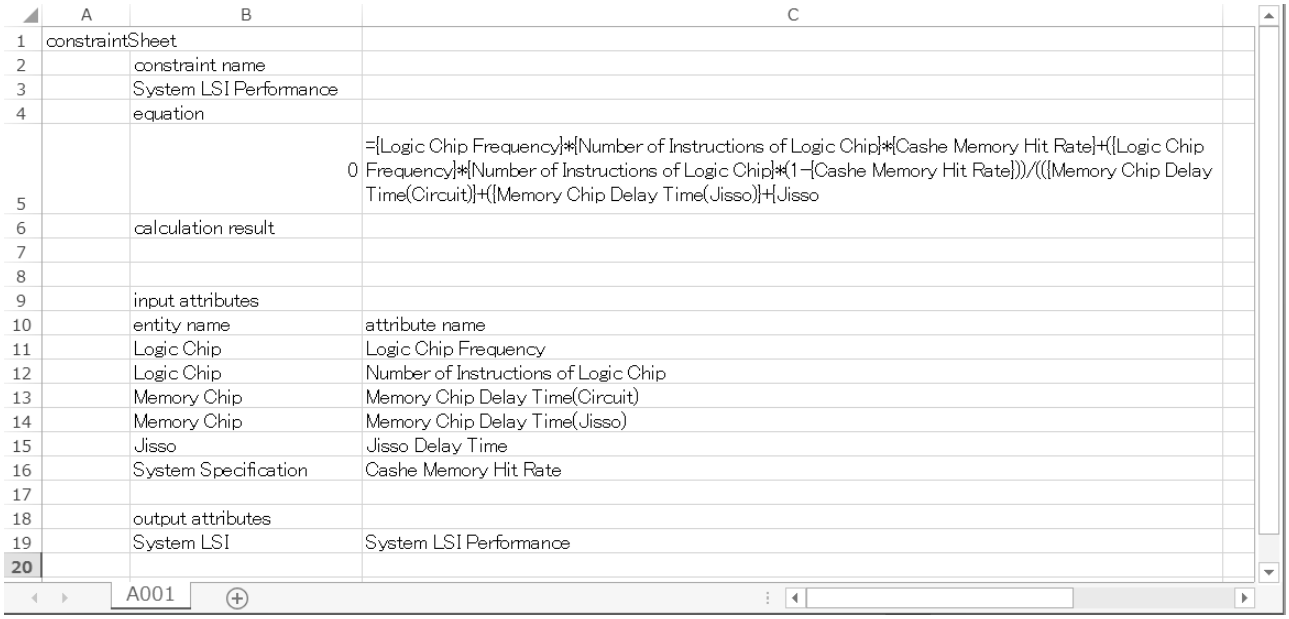
Fig. 4.19 Block definition diagram output .



	A	B	C	D	E	F	G	H	
1	attribute sheet								
2	entity name	attribute name	unit	type	current value	max	min	isDiscrete	step
3	System LSI	System LSI Power Consumption	W	float	15	300	0	FALSE	
4	–	System LSI Cost	none	float	15	300	0	FALSE	
5	–	System LSI Performance	GIPS	float	15	300	0	FALSE	
6									
7	Logic Chip	Dummy Logic Chip CoOS(90nm)B	none	float	0.011356799	300	0	FALSE	
8	–	Dummy Logic Chip CoOA	none	float	0.064399884	300	0	FALSE	
9	–	Logic Chip Frequency	GHz	float	15	1.2	0.3	FALSE	
10	–	Number of Instructions of Logic Chip	instruction	int	15	300	0	FALSE	
11	–	Dummy Logic Chip CoOB	none	float	0.032199942	300	0	FALSE	
12	–	Dummy Logic Chip CoOL(45nm)	none	float	0.064399884	300	0	FALSE	
13	–	Dummy Logic Chip CoOL(45nm)A	none	float	0.064399884	300	0	FALSE	
14	–	Dummy Logic Chip CoOL(45nm)B	none	float	0.064399884	300	0	FALSE	
15	–	Dummy Logic Chip CoOL(65nm)	none	float	0.032199942	300	0	FALSE	
16	–	Dummy Logic Chip CoOL(65nm)A	none	float	0.032199942	300	0	FALSE	
17	–	Dummy Logic Chip CoOL(65nm)B	none	float	0.032199942	300	0	FALSE	
18	–	Dummy Logic Chip CoOL(90nm)	none	float	0.008735113	300	0	FALSE	
19	–	Dummy Logic Chip CoOL(90nm)A	none	float	0.008735113	300	0	FALSE	
20	–	Dummy Logic Chip CoOL(90nm)B	none	float	0.008735113	300	0	FALSE	
21	–	Dummy Logic Chip CoOS(45nm)	none	float	0.079139958	300	0	FALSE	
22	–	Dummy Logic Chip CoOS(45nm)A	none	float	0.079139958	300	0	FALSE	
23	–	Dummy Logic Chip CoOS(45nm)B	none	float	0.079139958	300	0	FALSE	
24	–	Dummy Logic Chip CoOS(65nm)	none	float	0.039569979	300	0	FALSE	
25	–	Dummy Logic Chip CoOS(65nm)A	none	float	0.039569979	300	0	FALSE	
26	–	Dummy Logic Chip CoOS(65nm)B	none	float	0.039569979	300	0	FALSE	
27	–	Dummy Logic Chip CoOS(90nm)	none	float	0.011356799	300	0	FALSE	

AttributeSheet

Fig. 4.20 Part of internal block diagram output.



	A	B	C
1	constraintSheet		
2		constraint name	
3		System LSI Performance	
4		equation	
5		0	=[Logic Chip Frequency]*[Number of Instructions of Logic Chip]*[Cashe Memory Hit Rate]+([Logic Chip Frequency]*[Number of Instructions of Logic Chip]*(1−[Cashe Memory Hit Rate]))/(([Memory Chip Delay Time(Circuit)]+([Memory Chip Delay Time(Jisso)]+[Jisso
6		calculation result	
7			
8			
9		input attributes	
10		entity name	attribute name
11		Logic Chip	Logic Chip Frequency
12		Logic Chip	Number of Instructions of Logic Chip
13		Memory Chip	Memory Chip Delay Time(Circuit)
14		Memory Chip	Memory Chip Delay Time(Jisso)
15		Jisso	Jisso Delay Time
16		System Specification	Cashe Memory Hit Rate
17			
18		output attributes	
19		System LSI	System LSI Performance
20			

A001

Fig. 4.21 Part of parametric diagram output.

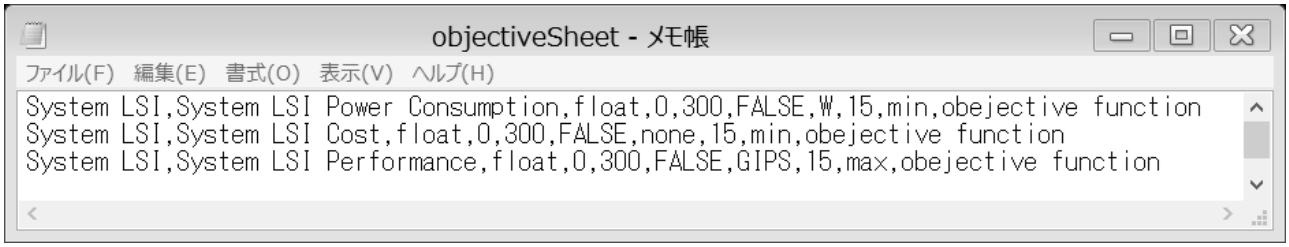


Fig. 4.22 Objective function output.

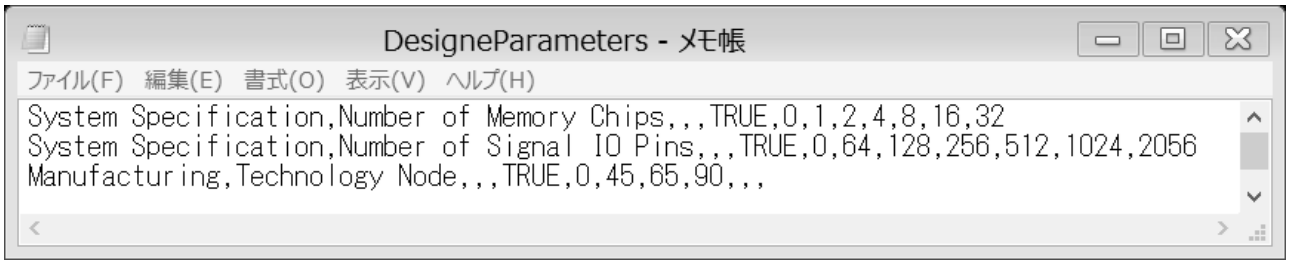


Fig. 4.23 Design variable output

Table 4.4 Extracted design tasks and constructed design variable tasks.

designTask No.	Design task detail
designTask1	System specification input
designTask2	Manufacturing design
designTask3	Memory chip parameter design dependent on technology node
designTask4	Memory chip CoO design
designTask5	Logic chip CoO design
designTask6	Memory IO MPU gate design
designTask7	Memory IO driver design
designTask8	Main memory cell design
designTask9	Memory TSV design
designTask10	Cashe memory parameter design dependent on technology node
designTask11	Logic out IO driver design
designTask12	Logic out IO MPU gate design
designTask13	Logic IO driver design
designTask14	Logic IO MPU gate design
designTask15	Cashe memory cell design
designTask16	Logic bump design
designTask17	Memory bump design
designTask18	Internal bump design
designTask19	Intermediate interconnect design
designTask20	Main memory circuit design
designTask21	Memory global interconnect design
designTask22	Board design
designTask23	Jisso design
designTask24	Logic MPU gate design
designTask25	Logic block design
designTask26	General processing circuit design
designTask27	Semi global interconnect design
designTask28	Logic IO design
designTask29	Cashe memory design
designTask30	Logic global interconnect design
designTask31	Logic chip design
designTask32	System LSI design
designTask33	Memory IO design
designTask34	Memory chip design

スクが抽出され、その内、「designTask3」、「designTask4」、「designTask5」、「designTask10」の 4 つのタスクが主因子分析により抽出された設計タスクであり、それ以外はシステムプロファイルで定義された構造モデルよりクラスタリングされた設計タスク群である。つまり、設計タスクの数は、ブロック定義図で定義された構造モデルの要素 30 個と主因子分析で抽出された 4 個の合計と等しい。Fig. 4.24 に抽出された設計タスクファイルの一例を示す。

この様に出力された製品システムのモデルをもとに、設計タスク抽出がアルゴリズム通りに自動的に行われることが確認できた。

	A	B
1		
2	constraints	
3	A003	
4	A002	
5	A001	
6		
7		
8	inputAttributes	
9	entityName	attributeName
10	Memory Chip	Memory Chip Power Consumption
11	Jisso	Jisso Delay Time
12	Memory Chip	Memory Chip Delay Time(Circuit)
13	System Specification	Number of Memory Chips
14	Memory Chip	Memory Chip Cost
15	Jisso	Jisso Cost
16	Logic Chip	Logic Chip Frequency
17	Memory Chip	Memory Chip Delay Time(Jisso)
18	Logic Chip	Logic Chip Power Consumption
19	Logic Chip	Logic Chip Cost
20	System Specification	Cashe Memory Hit Rate
21	Logic Chip	Number of Instructions of Logic Chip
22		
23		
24	outputAttributes	
25	entityName	attributeName
26	System LSI	System LSI Power Consumption
27	System LSI	System LSI Cost
28	System LSI	System LSI Performance
29		

designTask32

Fig. 4.24 Design task extraction output.

4.2.4　第 3 面(評価手法の構築)の実行結果

第 2 面で出力したパラメトリックモデルと設計タスクの出力ファイルをもとに、本アルゴリズムにより評価手法の構築を自動的に実行した。評価手法の構築では、第 3 章で述べたように表計算ソフトを用いる。また、表計算ソフト以外のソフトウェアと連携することを考慮し、入力ファイルの作成も行われる。Fig. 4.25 に作成された評価手法構築の入力ファイルの一例を、Fig. 4.26 に作成された表計算ソフトへの実装結果の一例を示す。また、このシステム LSI のシステムデザインでは、コストの計算の一部に専用の解析ソフト(TWO COOL)を用いているため、あらかじめ作成した応答曲面のデータベースファイルを用意した。詳細は 3.2.5 節の通りであり、Fig. 3.21 が用意したデータベースファイルの一部である。

Fig. 4.25 Input file for evaluation method.

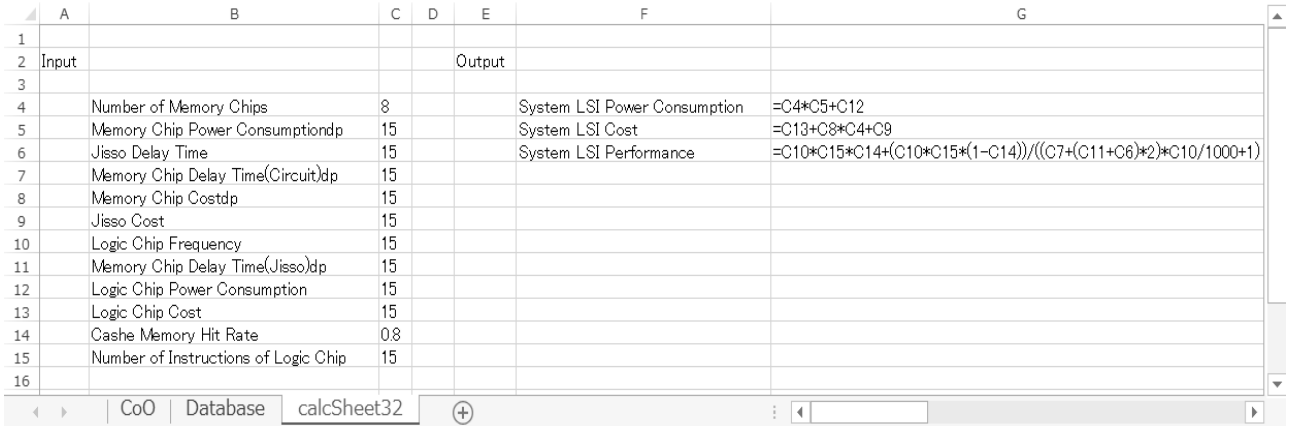


Fig. 4.26 Evaluation method.

このように評価手法の構築もアルゴリズム通り、自動的に行われることが確認できた。

4.2.5　第 4 面(設計ワークフロー構築)の実行結果

この面では、第 2 面で導出された設計タスクの出力ファイルをもとに、本アルゴリズムにより、」設計変数タスクと DSM の定義された入力ファイルの作成と、それらをもとに、DSM 実行ソフトウェアによるパーティショニングを自動実行した。その結果、設計ワークフローのファイル、最適化のファイル、設計変数としたパラメータと、出力されるパラメータが一致しているかの確認を行うタスクのファイルが出力された。Fig. 4.27 に作成された設計変数タスクの一例を、Fig. 4.28 に DSM の定義された入力ファイルを、Fig. 4.29 にパーティショニングの実行結果を示す。また、Fig. 4.30 に出力された設計ワークフローのファイルを、Fig. 4.31 に最適化のファイルを、Fig. 4.32 に設計変数としたパラメータと、出力されるパラメータが一致しているかの確認を行うタスクのファイルを示す。

Fig. 4.29 のパーティショング結果では、本研究で構築したアルゴリズム通り、「designTask00(テクノロジーノード設計)」、「designTask01(メモリチップチップ枚数設計)」、「designTask02(信号ピン数設計)」の設計変数タスクが追加され、システムプロファイル定義で定義された目的関数の含まれている「designTask32(システム LSI 設計)」から依存関係が追加されているのがわかる。また、Fig. 4.29 のパーティショニング結果より、「designTask16(ロジックバンプ設計)」、「designTask17(メモリバンプ設計)」、

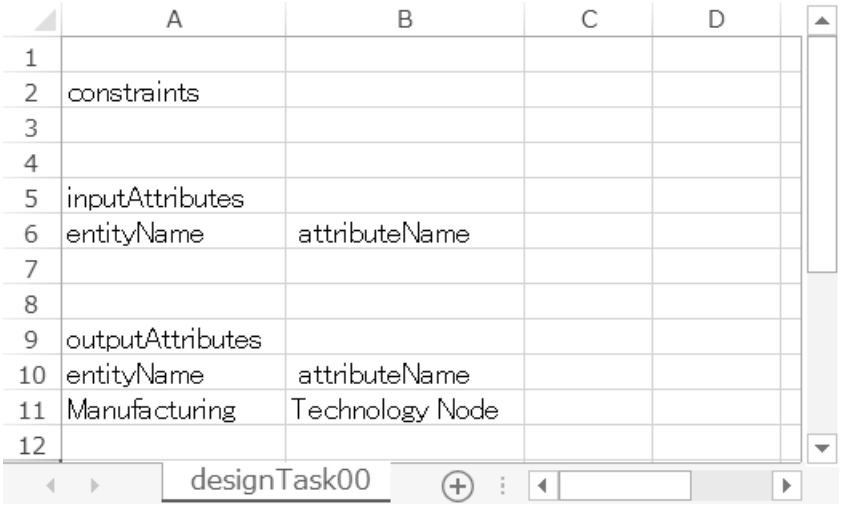


Fig. 4.27 Design variable task file.

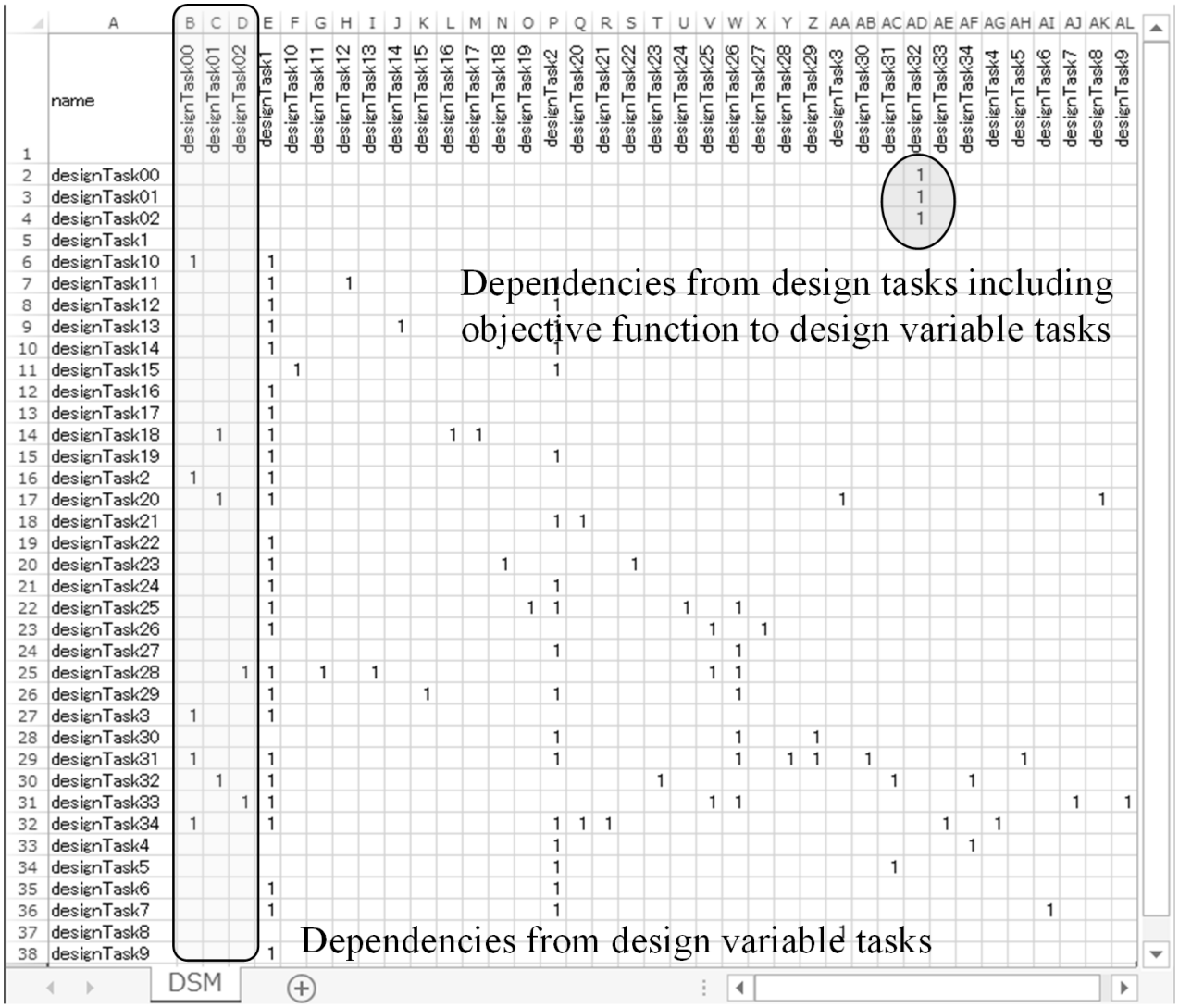


Fig. 4.28 DSM input.

「designTask22(基板設計)」、「designTask9(メモリ TSV 設計)」が、最適化と関係なく、かつ、並行して実行できる設計タスクであることがわかり、それ以外の設計タスクに関しては最適化が必要なことがわかる。このパーティショニング結果に対して、第 3 章で構築したアルゴリズム内の変換ルールを用いることにより、Fig. 4.30 に示す最適化ワークフロー出力ファイル、最適化の出力ファイル、設計変数としたパラメータと、出力されるパラメータが一致しているかの確認を行うタスクの出力ファイルが得られる。また、Fig. 4.31 より、システムプロファイルで定義された最適化モデル内の目的関数、設計変数の他に、自動最適化アルゴリズムにより新規に設計変数となったパラメータが 14 個存在していることがわかる。これらは Fig. 4.32 に示す表計算ソフトに実装された設計変数と出力パラメータの差をとるタスクを用いて、次の最適化実行の際に、制約条件とし

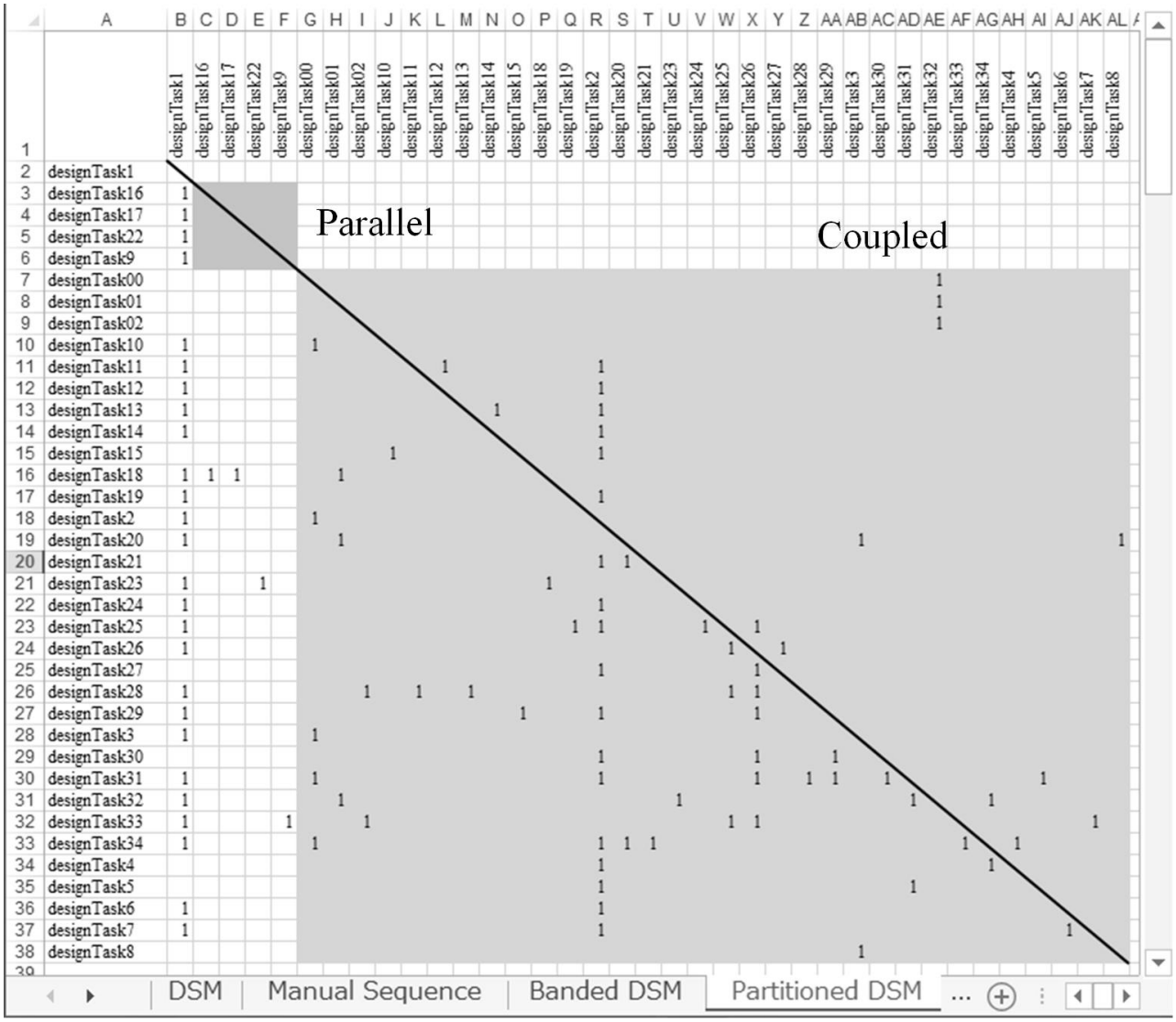


Fig. 4.29 DSM partitioning result.

て追加される。

このように、設計ワークフロー構築でも、構築した自動最適化アルゴリズムにより、規定した入力ファイルから出力ファイルが自動的に生成されることが確認された。

4.2.6 第 5 面(最適化実行)の実行結果

第 4 面の設計ワークフロー構築の出力ファイルと第 3 面の評価手法の構築の出力ファイルから、本アルゴリズムにより、この面で構築された最適化実行ワークフローの最適化に対して、最適化問題を設定したファイルを入力ファイルとして作成し、I-sight 用のカスタ

Fig. 4.30 Design workflow output.

マイズスクリプト実行ファイルを作成することで、自動的に最適化を実行した。Fig. 4.33 に作成された入力ファイルを、Fig. 4.34 に I-sight 上に構築された最適化実行ワークフローを示す。

Fig. 4.33 に示すように、今回のケーススタディでは、最適化の目的関数が複数であったため、自動的に多目的最適化のための最適化手法である「NSGA2」が選択されていることがわかる。今回のシステム LSI のシステムデザインでは、設計変数の数が多いため、最適化手法の詳細をオプションにより変更し、個体数と世代を増加させ、最適化の精度を上昇させている。

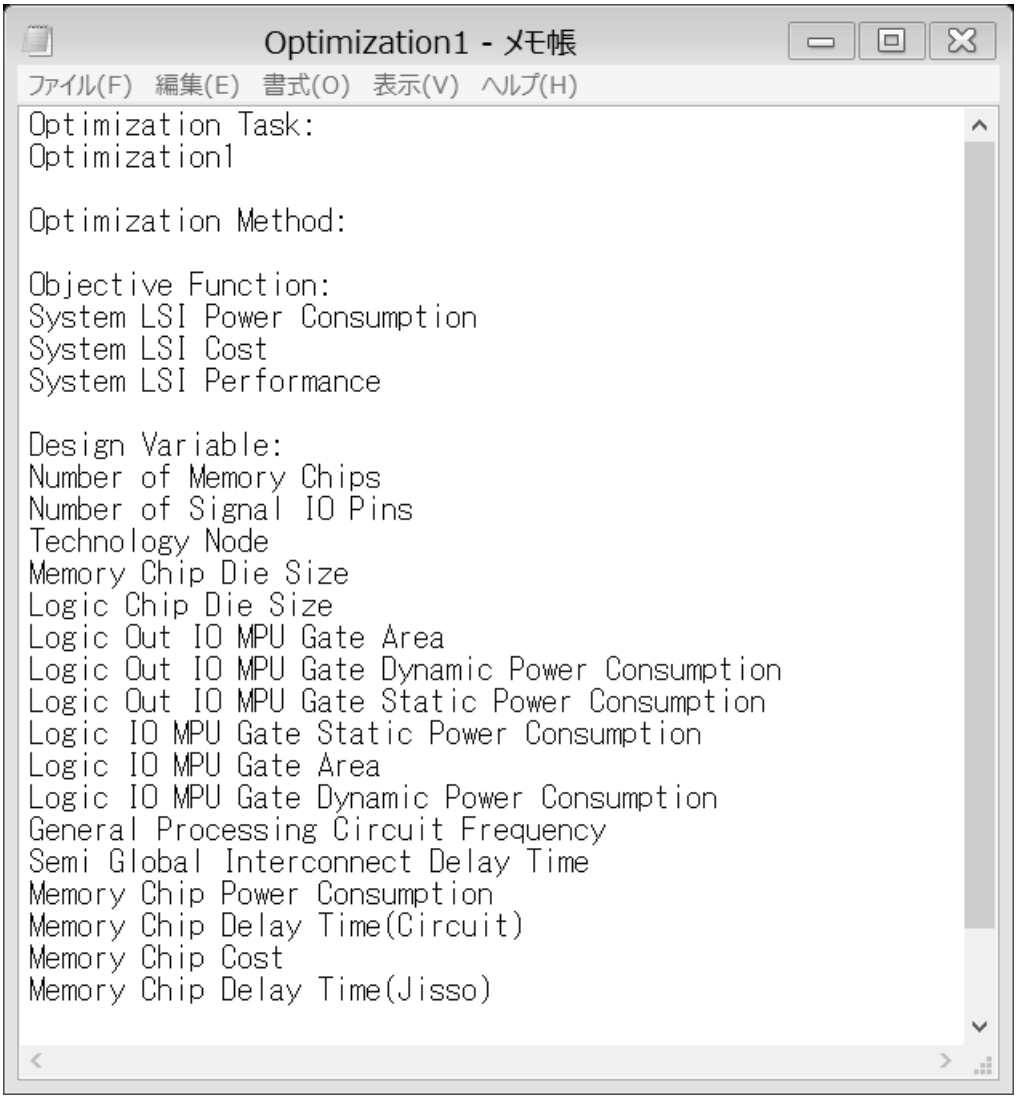


Fig. 4.31 Optimization task output.

	A	B	C	D	E	
1	Memory Chip Die Sizedp		Memory Chip Die Size		Memory Chip Die Sizedf	=ABS(B1-D1)/D1
2	Logic Chip Die Sizedp		Logic Chip Die Size		Logic Chip Die Sizedf	=ABS(B2-D2)/D2
3	Logic Out IO MPU Gate Areadp		Logic Out IO MPU Gate Area		Logic Out IO MPU Gate Areadf	=ABS(B3-D3)/D3
4	Logic Out IO MPU Gate Dynamic Power Consumptiondp		Logic Out IO MPU Gate Dynamic Power Consumption		Logic Out IO MPU Gate Dynamic Power Consumptiondf	=ABS(B4-D4)/D4
5	Logic Out IO MPU Gate Static Power Consumptiondp		Logic Out IO MPU Gate Static Power Consumption		Logic Out IO MPU Gate Static Power Consumptiondf	=ABS(B5-D5)/D5
6	Logic IO MPU Gate Static Power Consumptiondp		Logic IO MPU Gate Static Power Consumption		Logic IO MPU Gate Static Power Consumptiondf	=ABS(B6-D6)/D6
7	Logic IO MPU Gate Areadp		Logic IO MPU Gate Area		Logic IO MPU Gate Areadf	=ABS(B7-D7)/D7
8	Logic IO MPU Gate Dynamic Power Consumptiondp		Logic IO MPU Gate Dynamic Power Consumption		Logic IO MPU Gate Dynamic Power Consumptiondf	=ABS(B8-D8)/D8
9	General Processing Circuit Frequencydp		General Processing Circuit Frequency		General Processing Circuit Frequencydf	=ABS(B9-D9)/D9
10	Semi Global Interconnect Delay Timedp		Semi Global Interconnect Delay Time		Semi Global Interconnect Delay Timedf	=ABS(B10-D10)/D10
11	Memory Chip Power Consumptiondp		Memory Chip Power Consumption		Memory Chip Power Consumptiondf	=ABS(B11-D11)/D11
12	Memory Chip Delay Time(Circuit)dp		Memory Chip Delay Time(Circuit)		Memory Chip Delay Time(Circuit)df	=ABS(B12-D12)/D12
13	Memory Chip Costdp		Memory Chip Cost		Memory Chip Costdf	=ABS(B13-D13)/D13
14	Memory Chip Delay Time(Jisso)dp		Memory Chip Delay Time(Jisso)		Memory Chip Delay Time(Jisso)df	=ABS(B14-D14)/D14
15						

Feedback1

Fig. 4.32 Feedback task output.

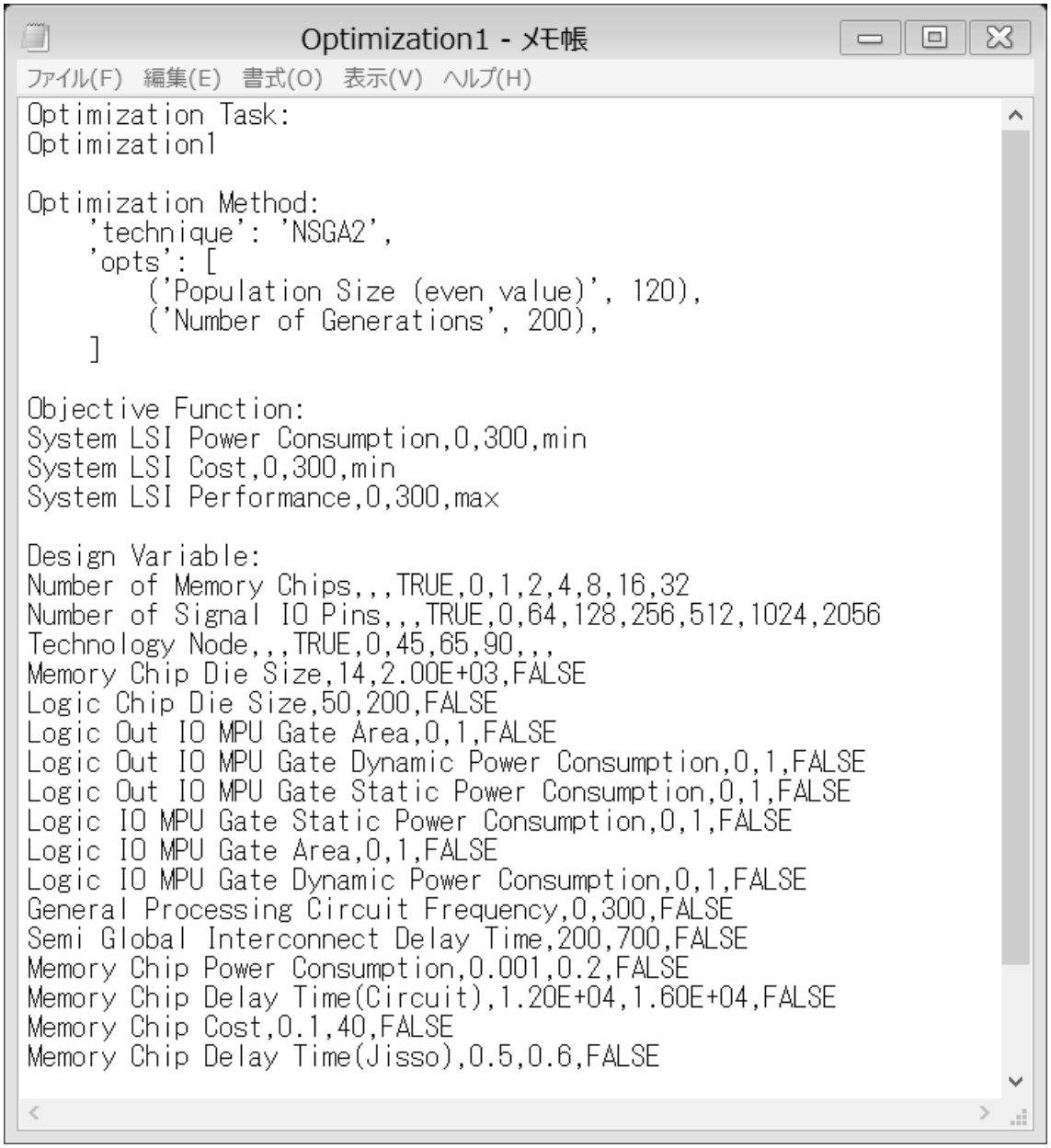


Fig. 4.33 Optimization definition input.

このように、構築した自動最適化アルゴリズムにより、最終的な最適化実行ワークフローが I-sight 上に自動生成され、最適化の実行まで自動的に行われることが確認された。

4.2.7 第 6 面(出力)の実行結果

SDSI-Cubic のケーススタディとして次世代携帯電話用システム LSI を対象とした適正構成の導出を行った。その結果、本研究で構築した自動最適化アルゴリズムを用いて自動的に最適化まで実行できることが確認できた。そのため、次に最適化の結果の考察を行う。

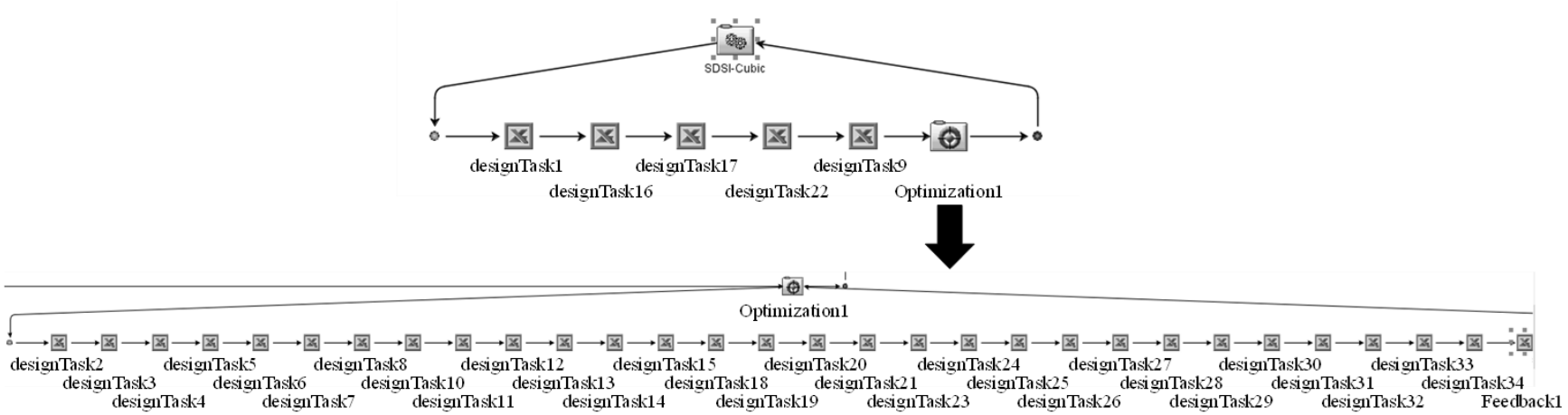


Fig. 4.34 Generated optimization execution flow result about system LSI.

そのために、最適化の出力として、SDSI-C を構成する 3 つの評価因子空間上にプロットしたパレート解とを出力した。Fig. 4.35 にパレート解を示す。

この結果より適正構成は、テクノロジーノード 45 nm、メモリチップ分割数 8、信号ピン数 256 の場合であり、SoC に比較して三次元化した SiP が評価関数(SDSI-C) で 10 倍以上優れたコストパフォーマンスを示している。また、SoC に比べ 3D-SiP がどのテクノロジーノードでも高く、45 nm で最大となっている。これは、65 nm、 90 nm に比べ、高周波数化に伴う消費電力の増大よりも、チップサイズの低減と三次元化による遅延およびコストの低減効果が大きくなったと考えられる。さらに、チップを分割しすぎて三次元化すると、実装コストの増大とチップに占める実装面積の割合が大きくなってコストの増大が効いていると考えられる。また、2D-SoC がこのように高く見積もられた理由は、2D-SoC は、システム LSI として必要なメモリ領域が大きく、全体のチップ面積が大きくなるため、チップ歩留まりが指数関数的に減少するためである。これに対して、3D-SiP は、TSV 技術や

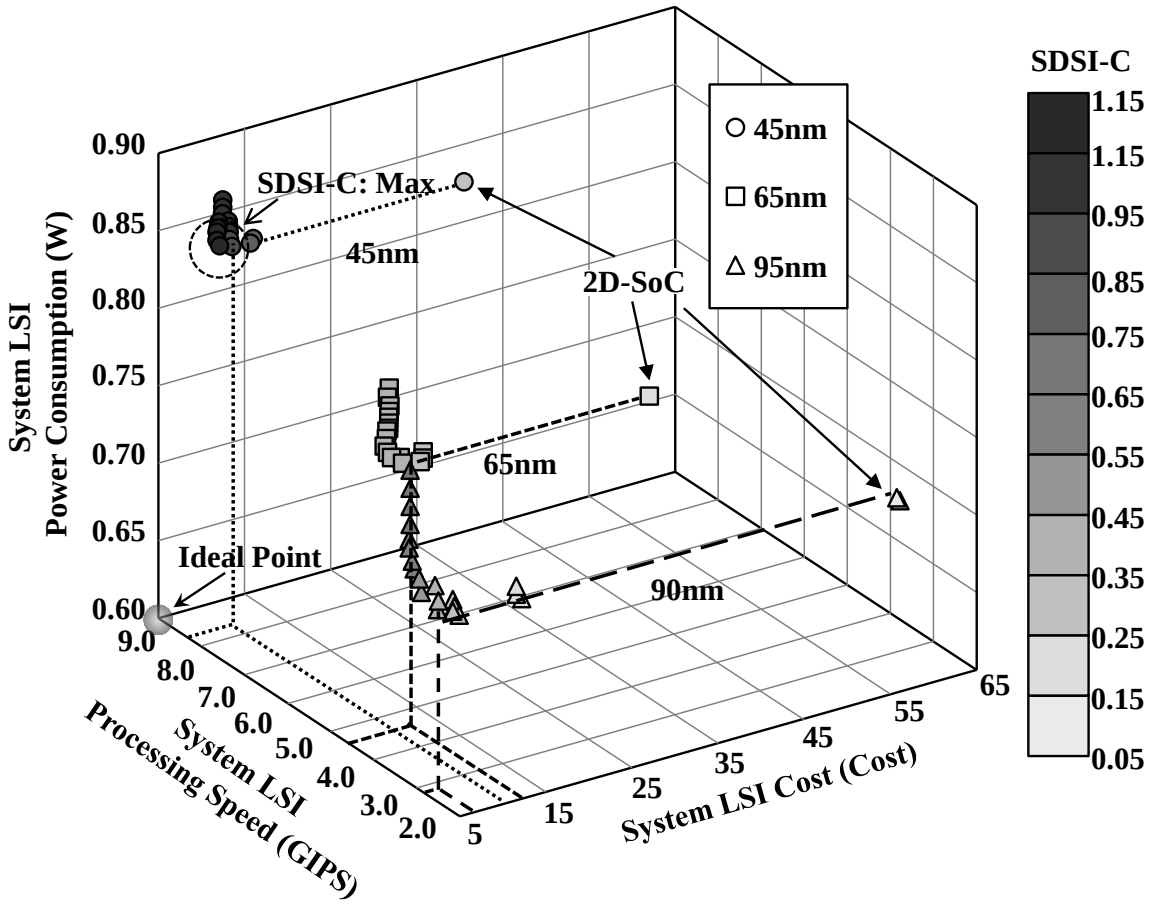


Fig. 4.35 Relation among SDSI-C, Power Dissipation, Performance and Cost of each technology node.

チップ積層技術に大きなコストがかかるにしても、それ以上に 3D 化による極端なチップ面積の低減による歩留まりの向上効果が著しい、ということを示している。

これらは、リーズナブルな結果を示しており、また、適切にトレードオフが評価されていることから、本手法のシステムデザインへの適用可能性が高いことを示していることが明らかになった。さらに、この手法は構想設計段階で、より多くの選択肢の中から非常に短時間で、想定した範囲での全体最適解を得ることが可能であることを示している。

以上の結果から、本研究で提案した新しいシステムデザイン手法 SDSI-Cubic を用いて、次世代携帯電話用システム LSI の適正設計が可能であることを示すとともに、本システムデザイン手法の自動化の検証を行う事が出来た。特に、構想設計段階での 2D-SoC と 3D-SiP の選択と適正設計を可能にしたことは、今後の半導体およびそれを用いた組み込み系の大規模製品システムの適正設計に大きく展開でき、大きな効果が期待できる。

4.3 結　言

構築したシステムデザイン手法の応用展開の対象として、大規模・複雑かつ、要求の多様化が叫ばれている次世代携帯電話用システム LSI の適正設計を行った。その結果、以下の結論が得られた。

(1) 本システムデザイン手法の自動最適化アルゴリズムの適用により、想定した理想構成と、そのモデルであるシステムプロファイル定義をもとに、課題であった多様なトレードオフを適切に評価、最適化し、適正解の導出まで自動的に行えることが明らかになった。

(2) また、本システムデザイン手法により、想定した携帯電話用システム LSI の理想構成(第 4 世代仕様)に対する適正構成はテクノロジーノード 45nm、メモリチップ分割数 8、信号ピン数 256 の場合であることがわかった。

(3) 本システムデザインを実行するために、次世代携帯電話用システム LSI の課題を明らかにし、それを解決するための理想構成の構築と、性能、消費電力、コストと多様なトレードオフを評価可能な物理、経済モデルの構築を行った。

(4) このように、構造段階のシステム LSI の適正設計を自動で行う手法を構築したので、より大規模、複雑なシステム LSI への展開が期待できる。

参考文献

1) 松澤昭, “LSI 技術の課題と今後のあり方”, 電子通信情報学会論文誌, J87-C-11 (2004) 802-809.
2) 風見晴雄, “2004 年度システム設計 LSI 第 1 章: あらゆるところに使われる SoC”, 株式会社半導体理工学研究センター, (2004).
3) 総務省総合通信基盤局, “モバイルビジネス研究会参考資料”, 総務省, (2007).
4) Gordon E. Moore, “Craimming More Components onto Integrated Circuit”, Proceedings of the IEEE, 86-1 (1998) 82-85.
5) Robert H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, A. R. LeBlanc, “Design of ion-implanted MOSFET's with very small physical dimensions”, IEEE Journal of Solid-State Circuits, 9-5 (1974) 256-268.
6) 黒田忠広, “総論: システム LSI の可能性と課題”, 電子情報通信学会誌, (2001) 552-558.
7) Yuan Taur, Tak H. Ning, 芝原健太郎, “最新 VLSI の基礎”, 丸善株式会社, (2002).
8) 松澤明, “LSI 技術の課題と今後のあり方”, 電子情報通信学会論文誌, J87-C-11 (2004) 802-808.
9) 岡本和也, “半導体製造技術の動向とシステムインテグレーションへの展開”, 電子情報通信学会論文誌, J88-C-11 (2005) 839-850.
10) Davis J.A., Venkatesan. R., Kaloyeros Alain, Beylansky M., Souri S.J., Banerjee K., Saraswat, K.C., Rahman Arifur, Reif Rafael, Meindl J.D., “Interconnect Limits on Gigascale Integration (GSI) in the 21st Century”, Proceedings of The IEEE, 89-3 (2001) 305-324.
11) 佐藤了平, “次世代エレクトロニクスイノベーション創出に向けて-SD-SI と 3D-LSI–”, Semi Forum Japan 2007 発表資料集, (2007) 1-8.
12) 赤沢隆, “次世代半導体の鍵を握る SiP の現状と今後の動向”, Mate (Microjoining and Assembly Technology in Electronics) 2009 論文集, (2009) 27-32.
13) International Technology Roadmaps for Semiconductors, “SiP Whitepaper”.
14) Said F. Al-sarami, “A Review of 3-D Packaging Technology”, IEEE. Transaction on Components, Packaging, and Manufacturing Technology, B-21-1 (1998) 2-14.
15) 山田徹, 小椋哲義, “システムインパッケージの設計技術(システム実装、<特集>システム実装を支える設計・シミュレーション技術)”, 電子情報通信学会論文誌, 9-5 (2005)

327-330.

16) Shamik Das, Anantha Chandrakasan, Rafael Reif, “Design Tools for 3-D Integrated Circuit”, Design Automation Conference (ASP-DAC) 2003, Proceedings of the ASP-DAC 2003 Asia and South Pacific, (2003) 53-56.

17) 赤沢隆, “SiP 技術の全て”, 工業調査会, (2005).

18) ITRS (International Technology Roadmaps for Semiconductors), “Assembly and Packaging”, http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_Assembly.pdf, (2007).

19) 半導体技術ロードマップ専門委員会, “第 6 章 配線”, STRJ (Semiconductors technology Roadmaps committee of Japan) 平成 15 年報告, (2003) 121.

20) 馬立稔和, “超精密限界への挑戦”, Mate (Microjoining and Assembly Technology in Electronics) 2008 論文集, (2008) 9-14.

21) 佐々木元, “システム LSI の進化とそれを支える技術”, 電子情報通信学会集積回路研究専門委員会 第 9 回システム LSI ワークショップ資料, (2005).

22) 半導体技術ロードマップ専門委員会, “第 1 章 元気が出る原価管理”, 2005 年度 STRJ 報告書 第 2 部, (2005) 279-281.

第5章　考察

5.1　緒言

本論文では、今日の大規模・複雑化する製品システムを最適にシステムデザインする基本的な手法の構築を目指し、研究を行った。即ち、従来のすり合わせによるシステムデザインではなく、製品システムの全体最適解を導出するためのシステムデザイン手法 SDSI-Cubic とその自動最適化アルゴリズムを構築し、次世代携帯電話用システム LSI の適正設計への応用展開を行うことで、適正解の導出と自動化の検証を行った。しかし、本手法を様々な製品システムに適用していくためには、多くの課題が存在する。本章では、構築した手法の考察と課題に関して述べる。

5.2　構築したシステムデザイン手法に関しての考察

一般的に、大規模システムの設計を行う場合、Fig. 1.1 に示したように、大規模な設計問題を、人間が解けるレベルまで設計問題を分割していき、それらの最適解をすり合わせることで製品システムの解を得ている。システムデザイン手法に関しても同様で、理想的には、大規模な設計空間をコンピュータ上で解精度、解探索時間を考慮し、解を導出可能なレベルまで分割し、それらを階層的に組み合わせ、それぞれの最適解をすり合わせることにより製品システムの最適解は導出できると考えられる。

上記を実現するために、本システムデザイン手法では、コンピュータ上で製品システムを扱うために対象とする製品システムのシステムプロファイルを定義し、設計タスク抽出により、可能な限り独立な設計空間(設計タスク)に分割し、それらを設計ワークフロー構築により階層的に統合することで、最適化ワークフローを構築することが理想だと考えた。Fig. 5.1 に、理想として考えるシステムプロファイル定義から設計タスク抽出、階層型最適化ワークフロー構築までの概略図を示す。

しかし、階層型最適化は、まだ確立された手法はなく、また、本研究の主題ではないため、本研究では、製品システムの全体最適化を行うためのシステムデザイン手法の構築を行い、まず、単一の最適化問題の構築を行うことで、構築したシステムデザイン手法の検証を行った。即ち、最適化問題を一つ構築する基礎的な自動化アルゴリズムを構築・実装し、次世代携帯電話用システム LSI に適用することで、自動的に最適解の導出が行えるかの確認を行った。

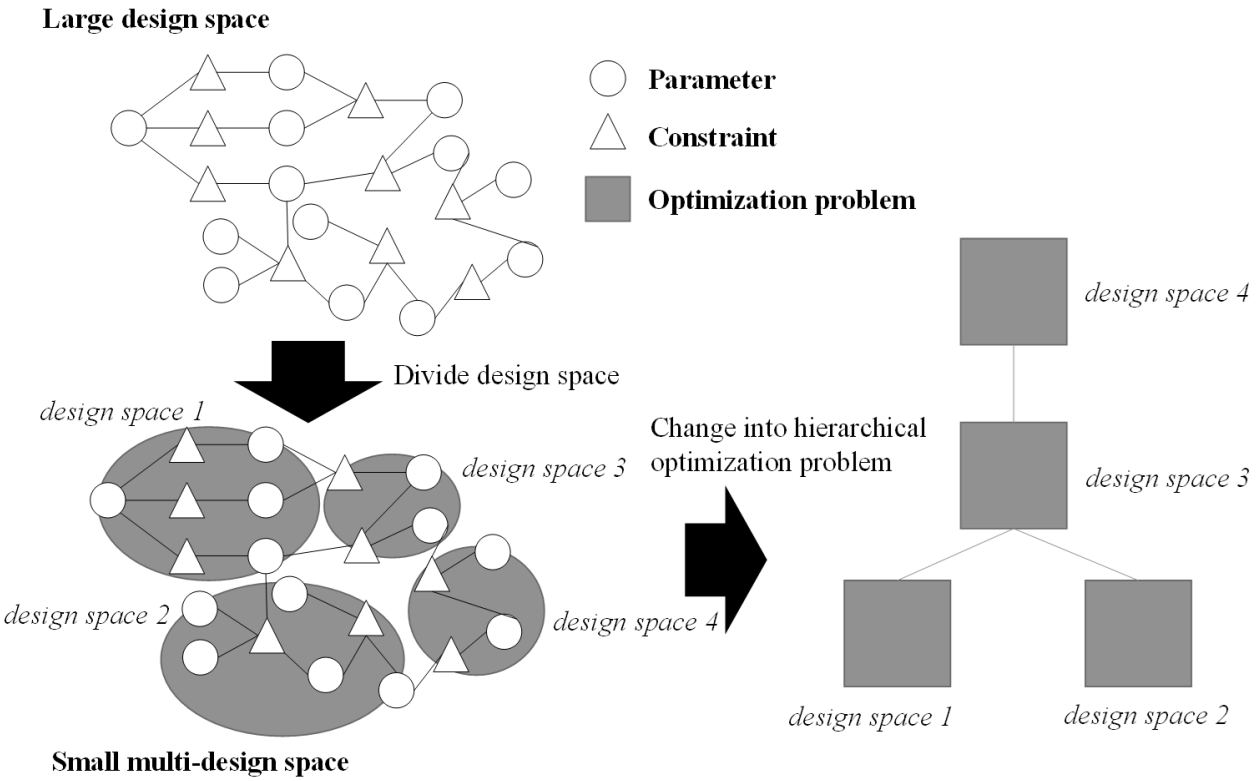


Fig. 5.1 The schematic diagram of constructing hierarchical optimization problem from system profile definition.

本節では、まず、構築したシステムデザイン手法の考察として、システムプロファイル定義と設計タスク抽出に関しての考察を述べる。

(1) システムプロファイル定義

本システムデザイン手法は、構造モデル、パラメトリックモデル、最適化問題を入力とし、それらをシステムプロファイルとして定義することで製品システム全体の最適化を行うための手法である。本手法の入力としては、以下の入力を想定している。

① 構造モデル: 多様な設計ドメインを含む、製品システム全体の構成要素とその関係を定義したもの

② パラメトリックモデル: パラメータとその関係式を定義したもの(但し、関係式に含まれるパラメータの入出力関係は固定されない)

③ 最適化問題: 設計変数、目的関数、制約条件を定義したもの(但し、設計変数としてパラメータ、アーキテクチャを想定)

この中で②、③に関して第 4 章で述べた次世代携帯電話用システム LSI を例として説明する。パラメトリックモデルとしては、パラメータとその関係式は定義されるが、どのパラメータが関係式の入力となるかは、定義する設計者によって異なると考えられる。大規模システムのパラメトリックモデルを作成する場合、複数の設計ドメインの設計者がそれぞれの担当部分のパラメトリックモデルを作成、それらをマージすることで製品システム全体のパラメトリックモデルが作成されると考えられる。その場合、同じ関係式の定義であっても、設計者により、入力となるパラメータ、出力となるパラメータが異なる可能性がある。Fig. 5.2 の例で説明すると、ロジックブロックの設計者がロジックブロック面積を導出するように関係式を定義した場合、ロジックチップ設計者の定義した関係式とそのままつなげることができるが、もし、ロジックブロック面積を入力とする関係式を定義してしまった場合、ロジックチップ設計者の定義した式とつなげることはできない。このように、関係式の入出力パラメータを固定するのではなく、自由に変更できる、あるいは、関係式に関わるパラメータのそれぞれが出力となる式を定義できれば、上記の場合でも、関係式をつなげることは可能となる。

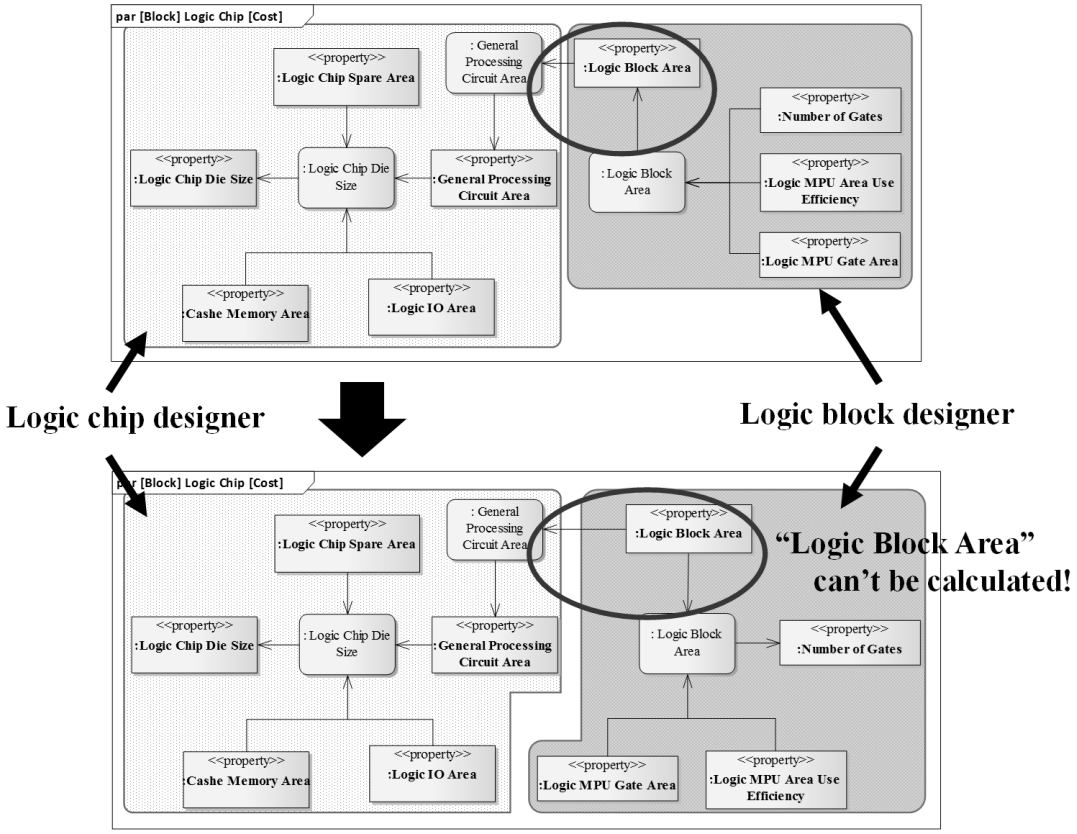


Fig. 5.2 The problem of parametric model.

また、最適化問題の設計変数として、パラメータだけでなく、対象とする製品システムのアーキテクチャの選択も設計変数として入力する可能性があると考えられる。一般的に製品システムの設計では、第 1 章の Fig.1.2 で述べたように、製品システムの概略構成を決定した後、アーキテクチャの設計を行い、各設計ドメインの設計に移る。この中のアーキテクチャ設計の際に複数のアーキテクチャが想定される場合、それらを設計変数として想定することは十分に考えられる。

本システムデザイン手法では、このように与えられた入力を SysML でシステムプロファイル定義を行うが、構築したシステムプロファイル定義手法では、上記の②パラメトリックモデル、③最適化問題に関して、入力として与えられると予想されるものを、現状、システムプロファイルとして定義することができないという課題が存在する。即ち、②に関しては、現状の SysML のパラメトリック図を用いたパラメトリックモデルの表記方法では、関係式の入力パラメータと出力パラメータを明確に定義する必要があるため、入出力の方向を固定せずに関係式を定義することができないという手法上の制約が存在している。そのため、今後の課題としては、パラメータとその関係式を、パラメータの入出力関係を固定せずに SysML 上で定義する手法を構築することが挙げられる。また、③に関して、構築した手法では、設計変数、目的関数、設計変数を、SysML の拡張定義手法を用いて定義しているが、その際に、設計変数として、パラメータしか定義できないという課題が存在する。設計変数となるのは、信号ピン数は何本か、テクノロジーノードは何 nm のものを利用するか、などのパラメータだけではなく、特定の機能をどの部品で実現するのか、というアーキテクチャレベルでの設計変数も考えられる。しかし、現状、アーキテクチャの選択を設計変数として定義するためには、それぞれのアーキテクチャのモデルを定義する必要がある。Fig. 5.3 を例として説明すると、(1)のようにハードウェアで全ての機能を実現する場合、(2)のようにハードウェアとソフトウェアの両方で全ての機能を実現する場合、(3)のようにソフトウェアで全ての機能を実現する場合、それぞれで構造モデル、パラメトリックモデルは異なると予想され、その場合、これら 3 つのパターンのモデルをすべて定義しておき、それらをアーキテクチャの設計変数の値により切り替える必要がある。しかし、アーキテクチャがより多く想定される場合には、すべてのモデルを定義することは現実的ではない。そのため、今後、アーキテクチャを設計変数とできるようなシステムプロファイルの定義方法を構築する必要がある。

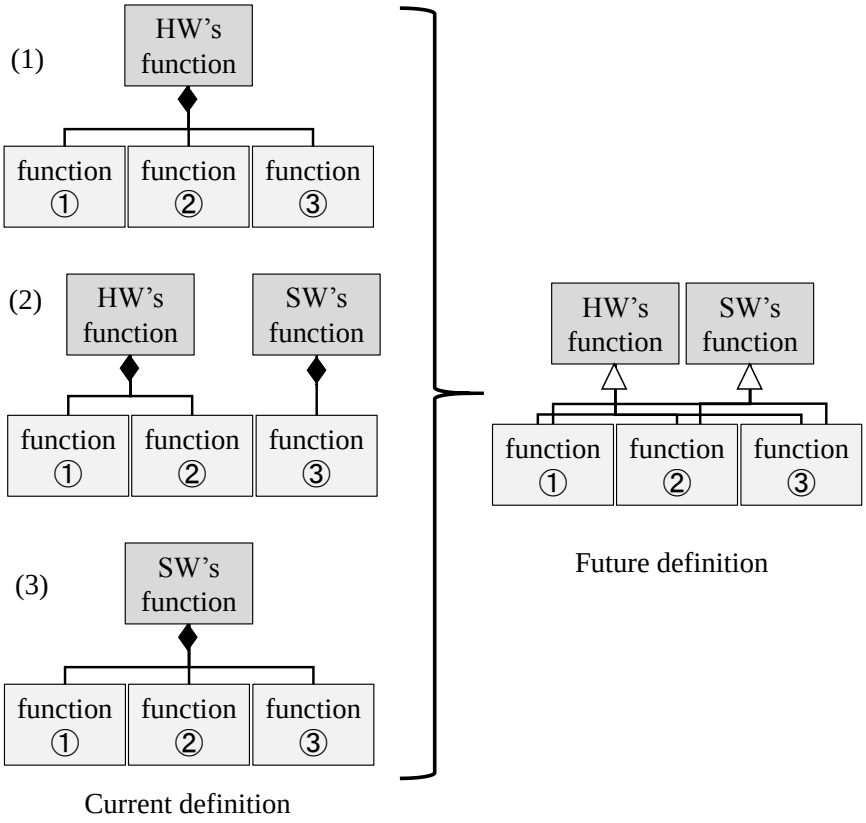


Fig. 5.3 Current architecture design model and future model.

(2) 設計タスク抽出

先に述べたように、設計タスク抽出の目的は、大規模、複雑な製品システムの最適化を行うために、解精度、解探索時間を考慮し、適切な設計空間(設計タスク)の分割を行うことである。その際に考慮すべき事柄は、分割された設計タスクのサイズと、設計タスク間の依存関係、特に相互依存関係を減らし、可能であれば全て独立に、それが難しい場合は、可能な限り独立な設計タスクの分割を行うことである。しかし、全てを独立になるように設計タスクを抽出する場合、製品システムには多様な依存関係が含まれているため、設計タスク一つ一つのサイズと莫大になり、かつ、最悪の場合、一つしか抽出されない可能性がある。この場合、設計タスク一つ一つの設計空間が大きくなるため、一つ一つの設計にかかる時間が増加してしまい、全体として設計に大きな時間がかかることが予想される。そのため、設計タスク間の依存関係、特に相互依存関係を可能な限り減らした設計タスクの抽出を行う必要がある。

そのために、本研究では、①設計タスク間の相互依存となる可能性のあるパラメータ、②多くの設計タスクに関わる可能性のあるパラメータ、の 2 つを主成分分析とクラスタ

リングを合わせた「機能によるクラスタリング」により抽出し、構造モデルをベースとした「構造によるクラスタリング」とマージすることで適切なサイズの設計タスクとして抽出できるようにした。本研究では、まず、単階層の最適化問題の構築を行うために、最適化問題に含まれる関係式と含まれない関係式を分離する必要があり、設計変数に関わる関係式群、それ以外の関係式群を分離できるように本手法を利用した。即ち、構築した設計タスク抽出手法では、主成分分析により、特定のパラメータに強く依存する関係式群が設計タスクとして抽出されるため、それらの内、設計変数に関わる設計タスク群を、後の自動化アルゴリズムと連携させ、一つの最適化問題としてまとめられるようにし、設計変数に関わらない設計タスク群は最適化問題とは関係なく実行することで、可能な限り、最適化問題に含まれる関係式が少なくなるようにした。

しかし、本研究で対象とした次世代携帯電話用システム LSI の事例では、今回構築した設計タスク抽出を用いた単階層最適化で最適化を行うことができたが、多階層の最適化にはまだ適用できていない。そのため、今後、構築した設計タスク抽出手法を階層型最適化に利用し、検証を行っていく必要がある。

5.3 構築したシステムデザイン手法の自動実行アルゴリズムに関する考察

ここでは、構築したシステムデザイン手法の自動実行アルゴリズムに関する考察として、構築されたシステムプロファイル定義から最適化ワークフロー構築アルゴリズムの考察を行う。

本アルゴリズムの目的は、定義されたシステムプロファイルから、設計タスク抽出により、可能な限り独立な設計空間(設計タスク)に分割し、それらを設計ワークフロー構築により階層的に統合することで、最適化ワークフローを構築、階層型最適化を自動的に実行できるようにすることである。しかし、前述したように、階層型最適化を自動的に行うことは難しいため、本研究では、まず、抽出された設計タスクをもとに、最適化ワークフロー内に単一の階層の最適化問題の構築を行い、最適化を自動的に実行するためのアルゴリズムの構築を行った。その際に、本アルゴリズムでは、単階層の最適化ワークフローしか構築できず、最適化も単階層でしか実行できていない。そのため、今後、多階層の最適化ワークフローの構築を行う必要があるが、その際の課題として、Fig. 5.4 に示すように 3 つ挙げられる。一つは、最適化問題に含まれる相互依存関係を考慮した最適化問題の分割の課題、

もう一つは、それぞれの最適化問題の定義、即ち、分割した最適化問題の設計変数、目的関数、制約条件、最適化手法をどのように定義するのかという課題、最後は、分割した最適化問題の解をどのように統合するのかである。この内、最適化問題の分割に関しては、設計タスク抽出手法として構築したがまだ、多階層での検証は行えていない。また、上記の課題に対し、現状、階層型最適化手法[1,2)]が提案されているが、それらは、それぞれの最適化問題の設定は個々に設計者が行う前提である、あるいは、同階層の最適化問題間には依存関係は存在していないという前提の下で使用できる手法であり、使用条件が非常に限定されているため、解決できる手法は存在していない。そのため、今後の課題としては、上記の課題を解決するため、解精度、解探索時間を考慮した適切な設計タスク抽出手法の構築と、それを元にした階層型最適化ワークフローの構築アルゴリズムの構築、階層型最適化手法を新たに構築し、本手法に実装することが挙げられる。

5.4 結言

本章では、構築したシステムデザイン手法に対し、考察を行い、課題の明確化と今後の展望のまとめを行った。その結果、構築した手法は基礎的な手法であり、今後、大規模・複雑な多様な製品システムに、汎用的に用いるためには、本手法の入力となる項目すべて

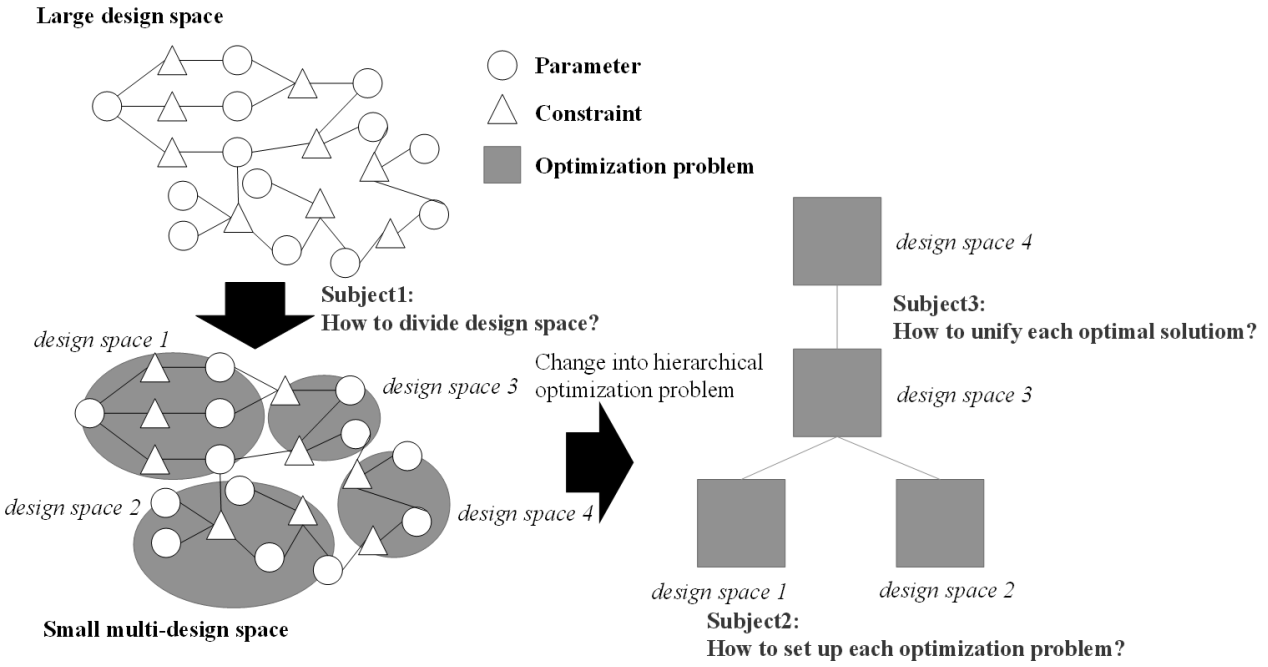


Fig. 5.4 The problem of hierarchical optimization workflow.

をシステムプロファイルとして定義する手法を構築し、階層型最適化を考慮し、解精度、解探索時間を考慮した適切な設計タスク抽出手法の構築と、それを元にした階層型最適化ワークフローの構築アルゴリズムの構築、階層型最適化手法を新たに構築し、本手法に実装することが課題である。

参考文献

1) 小森茂明, 泉井一浩, 吉村允孝, “最適設計問題の多階層表現による定式化と最適化戦略の構築”, 日本機械学会　第 12 回設計工学・システム部門講演会講演論文集, (2002) 42-45.

2) H.M.Kim, N.F.Michelena, P.Y.Papalambros, Tao Jiang, “Target Cascading in Optimal System Design”、Transactions of the ASME, 125 (2003) 474-480.

第6章　結言

本研究では、従来のすり合わせ設計における、設計ドメイン毎の部分最適解のすり合わせによるシステム適正解の導出という課題を解決するため、すり合わせの原因である製品システム内に含まれる依存関係の把握、分析、整理、解決するための手法のシステム化・デジタル化を行い、自動的に最適解を導出する新たなシステムデザイン手法を考案、その自動最適化アルゴリズムの構築と次世代携帯電話用システム LSI(Large Scale Integration)への応用展開と自動化の検証を行った。本章では、結言として、各章のまとめを行い、本論文の最終的な結論を述べる。

まず第一章では、製品システムの大規模、複雑化に対する従来のシステムデザイン手法の課題を明らかにした。従来手法としては、設計者の能力に依存したすり合わせ設計であり、製品システム内の依存関係を扱う手法のシステム化、デジタル化が不十分であるため、すり合わせによるシステム適正解しか導出できないことが課題であった。それに対し、様々なシステム化、デジタル化手法が提案されつつあるが、一部に関してのシステム化、デジタル化しか出来ておらず、すり合わせを根本的に解決するためには、これらを適切に組み合わせ、製品システム全体の最適解を導出する新たなシステムデザイン手法の構築が必要であるという結論を得た。

第二章では、第一章の課題を受け、製品システム全体の最適解を導出可能なシステムデザイン手法を新たに構築、提案した。具体的には、多様な設計ドメイン間の依存関係を把握するための製品システムのモデリング手法、依存関係を考慮し可能な限り独立な設計タスクを抽出する設計タスク抽出手法を新たに開発し、これらを用いて製品システム全体を評価、最適化することで、設計者の能力に依存したすり合わせという課題を解決できる新たなシステムデザイン手法を開発した。本システムデザイン手法は、製品システム全体の依存関係の定義、その中から可能な限り独立な設計タスクの抽出と、設計タスク間の依存関係を考慮した最適化ワークフローを作成して、最適解を求めることにより、適正設計を行うものである。

第三章では、第二章で構築したシステムデザイン手法に対し、自動最適化を実行するためのアルゴリズムを組込み、ソフトウェアとしての実装を行った。即ち、従来手法を組み合わせるだけでは、自動最適化を行うことは不可能であったため、各面を連携させるための製品システムのモデリング手法として、SysML を用いた自動化のための製品システムの

構造モデル、パラメトリックモデルの定義方法を構築し、最適化問題を定義するために、目的関数、設計変数、制約条件からなる最適化のメタモデルを構築し、SysML の拡張プロファイルを用いて SysML での最適化モデルの定義手法を構築した。また、設計ワークフロー構築で用いる DSM(Design Structure Matrix)に対し、最適化の依存関係を考慮した新たな設計ワークフロー導出アルゴリズムを構築した。これに加え、本システムデザイン手法の各面の手法を連携させるための手法とファイル群の規定を行うことで、実際にシステムプロファイル定義から、最適解の導出までを自動的に行うアルゴリズムを構築、Eclipse 上にソフトウェアを実装した。

第四章では、構築したシステムデザイン手法を用いて、次世代携帯電話用システム LSI を対象としたシステムデザインを行い､本手法を用いた適正設計と自動化の検証を行った。その結果、本手法の自動最適化アルゴリズムにより、想定した次世代携帯電話用システム LSI の理想像に対して適正解を導出することが可能であることが確認できた。

以上より、本研究の最終的な結論を以下に述べる。

(1) 従来のシステムデザイン手法の課題は、要求、仕様を考慮した製品システムのモデリング、設計ドメインの分割と、各設計ドメインへの設計目標の設定、各設計ドメインの個別最適解のすり合わせの 3 点が設計者の能力に大きく依存しており、製品システムに含まれる多様な依存関係が適切に扱われていないことであり、これらのシステム化、デジタル化が製品システムの最適解の導出には必要であることを明らかにした。

(2) (2)(1)の課題を解決すべく、製品システム全体の依存関係のモデリングが可能な手法として提案されている SysML(Systems Modeling Language)に着目、可能な限り独立な設計タスクを抽出するための手法を新たに開発し、これらの手法と、評価を行うための表計算ソフト、手戻りの最小となる設計ワークフローを導出するための手法である DSM を連携させることにより、想定した理想構成から製品システム全体の最適解を導出可能なシステムデザイン手法を開発した。

(3) 本手法は、性能やコストといった要求を反映した製品システム全体のプロファイル定義、すなわちパラメトリックモデルを含むモデリングを行い、その中から重要設計項目の抽出と相反問題を考慮した最適化ワークフローを作成して、最適解を自動で求めることにより、適正設計を行うものである。

(4) (4)(2)で提案した手法に対し、実際に自動化を行うためのアルゴリズムを作成、ソフト

ウェアへの実装を行った。その際に、従来手法では自動最適化の課題であった、設計ワークフロー導出のための手法である DSM への最適化の依存関係の組込みに対し、設計変数のタスクを追加し、目的関数からのフィードバックを定義するアルゴリズムと、パーティショニング結果から、最適化のタスク群を決定するアルゴリズムを新たに開発し、組込むことにより自動最適化が可能なアルゴリズムを構築した。

(5) (5)(2)(4)で構築したシステムデザイン手法とその自動最適化アルゴリズムを用いて、次世代携帯電話用システム LSI のシステムデザインと自動化の検証を行った。その結果、本手法により自動的に解を求めることが可能であり、かつ、想定した次世代携帯電話用システム LSI の理想構成(第 4 世代仕様)に対する適正構成はテクノロジーノード 45nm、メモリチップ分割数 8、信号ピン数 256 の場合であることがわかった。

このように、本研究では、従来のすり合わせでは困難であった製品システムの最適解の導出という問題に対し、製品システムの最適解を導出可能な手法の構築と、その自動化アルゴリズムを構築し、次世代携帯電話用システム LSI への応用展開と自動化の確認を行うことができた。本論文は、今後、より大規模、複雑化している製品システムへ本手法を適用、展開し、システムデザインを行うことにより、事前に効率の良い製品システム設計ができるとともに、スマートな社会システムの形成に寄与できるものと考えている。

謝辞

本研究は、大阪大学大学院工学研究科　岩田剛治准教授の懇切丁寧な御指導、御助言により遂行し得たものである。ここに厚く御礼申し上げます。岩田剛治准教授には、様々な会議、プロジェクト、学会など、最も長い時間ご指導頂きました。また、本論文を含め、研究討論の場では、何時間も丁寧に粘り強く討論して頂き、その討論の姿勢は私の中で、討論の際の基本姿勢として刻まれております。今後は、私自身、後輩への指導等、討論の際には、岩田剛治准教授の姿勢を理想とし、本質的理解が得られるまで粘り強く行うべく精進させていただきたく思います。御指導頂き、本当にありがとうございました。

東京大学大学院工学系研究科　青山和浩教授には、副査をお引き受け頂き、本論文への貴重なご意見、ご指導を頂きました。ここに厚く御礼申し上げます。また、本論文を完成させる上で、東京大学での討論、学会の場でシステムデザインやモデリングに関する貴重なご意見を頂きましたこと、深く感謝致します。

大阪大学工学研究科　荒井栄司教授には、副査をお引き受け頂き、本論文への貴重なご意見、ご指導を頂きました。ここに厚く御礼申し上げます。また、本論文に関する討論の際に、荒井栄司教授の思想には深く感銘を受けるとともに、自分自身の思想の未熟さを認識致しました。今後は、設計に対する本質的理解を深め、自分自身の思想をもとに設計に対する理想像を構築できるように精進いたします。ご指導いただき、ありがとうございました。

大阪大学工学研究科　望月正人教授には、副査をお引き受け頂き、本論文への貴重なご意見、ご指導を頂きました。ここに厚く御礼申し上げます。また、本論文に関する討論の際にご教授頂きました、研究に対する姿勢は私の心に深く刻まれております。ご指導いただき、ありがとうございました。

大阪大学工学研究科　佐藤了平教授(現　大阪大学産学連携本部　佐藤了平特任教授)の懇切丁寧な御指導、御教示を頂き、本研究を遂行することができました。本研究を進めるにあたり非常に多くの有益な御意見を賜り、ここに厚く御礼申し上げます。また、研究室に配属されてから現在に至るまで､様々な会議､プロジェクトをともに行わせていただき、研究だけでなく、取り組む姿勢、考え方、リーダーとしての心構えや振る舞いなど、数えきれない事柄を学ばせて頂きました。特に、常に自ら率先して動き、皆を引っ張っていく姿には、大変感動し、私の中の理想像として深く心に刻まれております。御指導頂き、本当にありがとうございました。

大阪大学大学院工学研究科　森永英二助教には、研究を遂行する上で様々な御助力を賜りました。ここに厚く御礼申し上げます。森永英二助教には、同研究室の先輩として、様々なアドバイスを頂き、また、研究の取り組み方、物事への丁寧な対応など、研究だけでなく、研究者としての生き方に対し、多大な事柄を学ばせて頂きました。色々とご相談、アドバイスを頂き、ありがとうございました。

大阪大学産学連携本部　渥美幸一郎特任教授には、懇切丁寧なご指導、御助言を賜り、本研究を行うことができました。ここに深く御礼申し上げます。また、研究など快く相談に乗っていただき、人生の先輩としてアドバイスを頂きましたことを心より感謝いたします。

大阪大学産学連携本部　岡本和也特任教授には、懇切丁寧なご指導、御助言を賜り、本研究を行うことができました。ここに深く御礼申し上げます。また、研究に関しての様々な技術的情報の提供や、研究や論文に対する客観的な視点でのご意見を頂き、大変ありがとうございました。

山口大学大学院理工学系研究科　古賀毅准教授には、設計タスク抽出手法の実装を始めとして、様々な技術的な支援を賜りました。ここに厚く御礼申し上げます。また、討論や学会の場などで、研究に関して様々なご助言を頂き、大変ありがとうございました。

Dassault　Systèmes　工藤啓治様には、本研究を遂行する上で、技術的支援を始め、様々なご指導、御助言を賜りました。ここに厚く御礼申し上げます。

株式会社グローバルアシスト　坂本武志様には、本システムデザイン手法のソフトウェアへの実装を始めとして、様々な技術的な支援を賜りました。ここに厚く御礼申し上げます。

大阪大学　産学連携本部　白水佐和子様には、日々の研究生活において多大なるご援助を頂きましたこと、ここに厚く御礼申し上げます。

多屋淳志氏(現　株式会社デンソー)には、研究を行う上で多大なご助言を頂き、研究に対する姿勢、指導に対する姿勢など多くのことを学ばせて頂きました。ここに厚く御礼申し上げます。

宮川春彦氏(現　株式会社 東芝)には、研究室のリーダーとして大変多くのことを学ばせて頂きました。また、日々の研究生活を送る上でも多大なるご援助を頂きましたこと、ここに厚く御礼申し上げます。

また、日々の研究生活を送る上で、卒業された研究室の御先輩、同輩、後輩諸氏を始め

とする方々に多大な御援助を賜りましたことを、心より感謝いたします。

また、本研究を遂行するに当たり、有益な御助言を頂戴いたしました日本学術振興会第177委員の皆様にも、多くの御助力賜りました。ここに深く感謝いたします。

最後に、高校、大学の選択、博士後期課程の進学に際し、私自身の決断を尊重し無償で支援し、見守ってくれた両親に最大限の敬意と謝意と愛情を表し、謝辞とさせていただきます。

付録

A1. 次世代携帯電話用システム LSI のシステムプロファイル定義結果

SysML のブロック定義図、内部ブロック図、パラメトリック図と本研究で提案する最適化図を用いて、第 4 章で述べた次世代携帯電話用システム LSI を対象にシステムプロファイル定義を行った結果を以下に示す。

Fig. A.1 から Fig. A.82 に次世代携帯電話用システム LSI を、要求、想定した理想構成をもとに SysML で定義した結果を示す。まず、Fig. A.1 にシステム LSI の構造を定義したブロック図を示す。図に示すように、本研究では、システム LSI の構造をツリー上にサブシステムに分解したブロック群とシステム LSI の製造工程、システム仕様を定義するブロックを作成した。その理由としては、すべての属性がシステム LSI という構造内のみで決定されているわけではなく、工程情報や外部からのシステム要求としての仕様からいくつかの属性が決定されると考えたためである。

次に Fig. A.2 にシステム LSI の内部ブロック図、Fig. A.3(a)にシステム LSI の消費電力に関するパラメトリック図、Fig. A.3(b)に性能に関するパラメトリック図、Fig. A.3(c)にコストに関するパラメトリック図を示す。また、Fig. A.4 に制約式の詳細を示したブロック図を示す。

以下、同様に、Fig. A.5 にロジックチップの内部ブロック図、Fig. A.6(a)にロジックチップの消費電力に関するパラメトリック図、Fig. A.6(b)に性能に関するパラメトリック図、Fig. A.6(c)にコストに関するパラメトリック図を示す。また、Fig. A.7 に制約式の詳細を示したブロック図を示す。

Fig. A.8 に汎用処理回路の内部ブロック図、Fig. A.9(a)に汎用処理回路の消費電力に関するパラメトリック図、Fig. A.9(b)に性能に関するパラメトリック図、Fig. A.9(c)にコストに関するパラメトリック図を示す。また、Fig. A.10 に制約式の詳細を示したブロック図を示す。

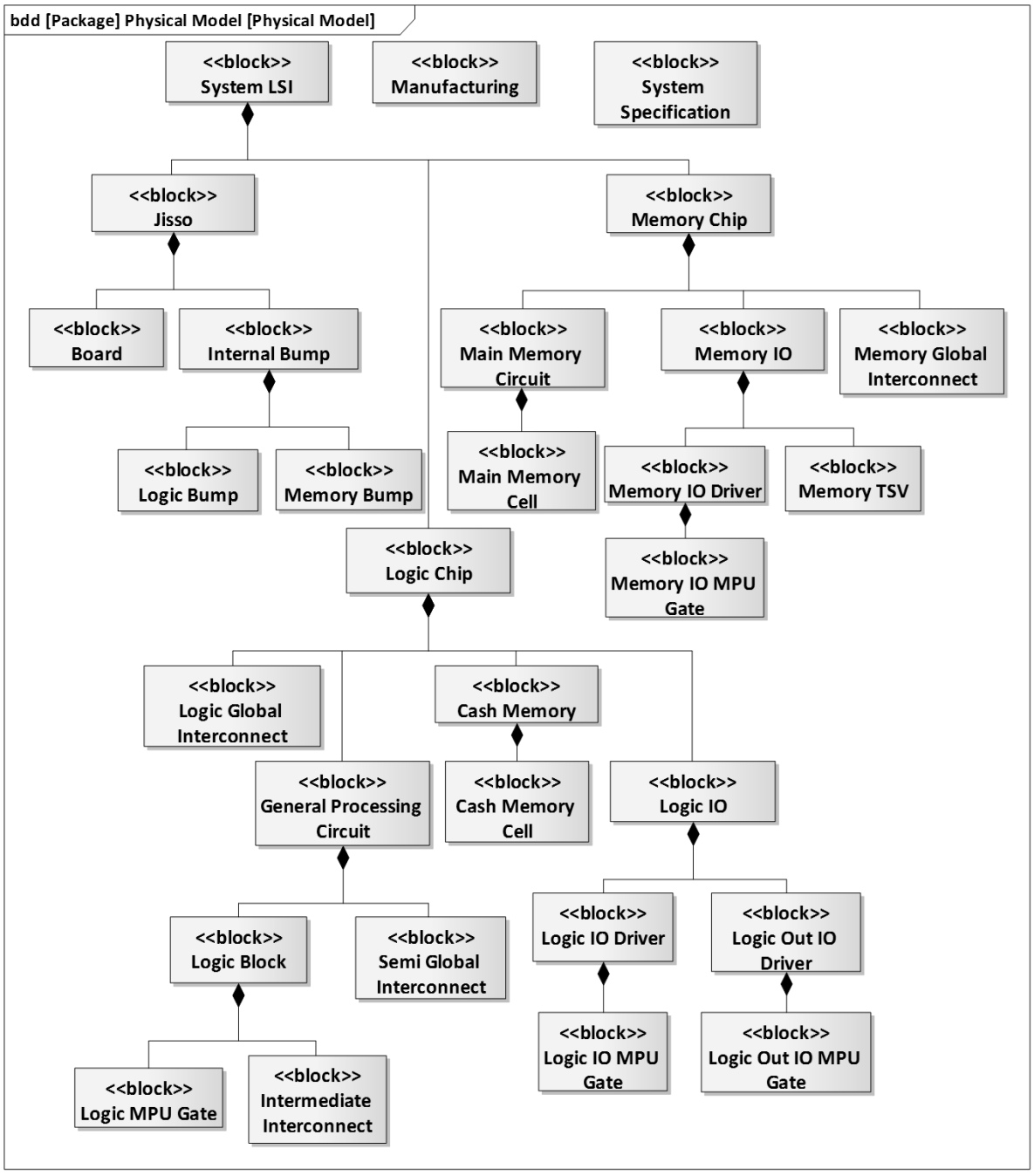


Fig. A.1 System LSI block definition diagram.

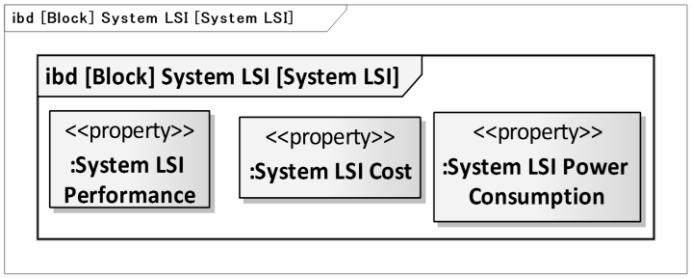
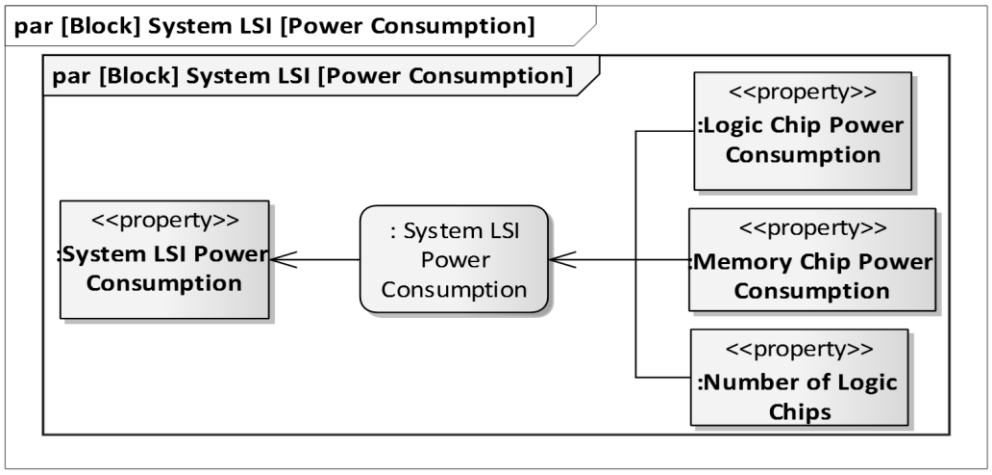
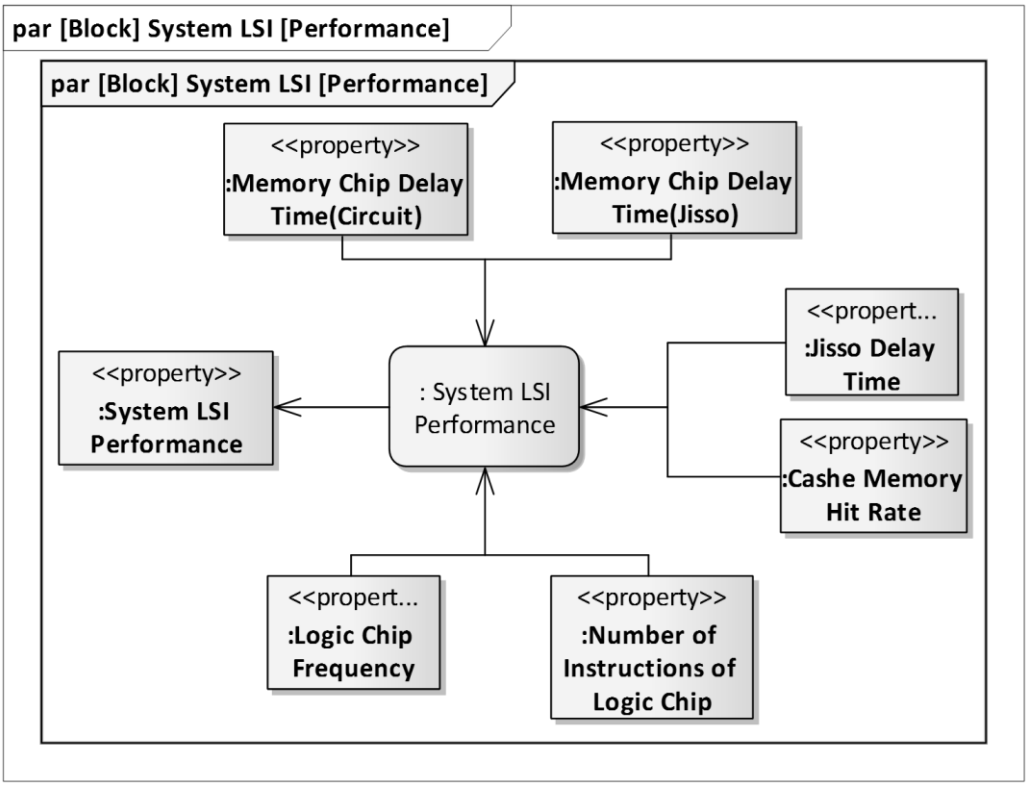


Fig. A.2 System LSI internal block diagram.

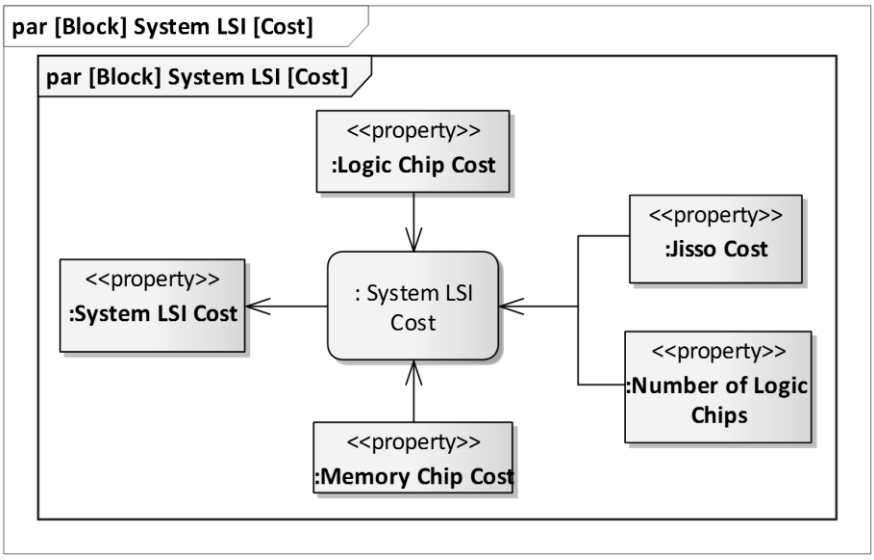


(a) Power consumption.



(b) Performance.

Fig. A.3 System LSI parametric diagram.



(c) Cost.

Fig. A.3 Continued.

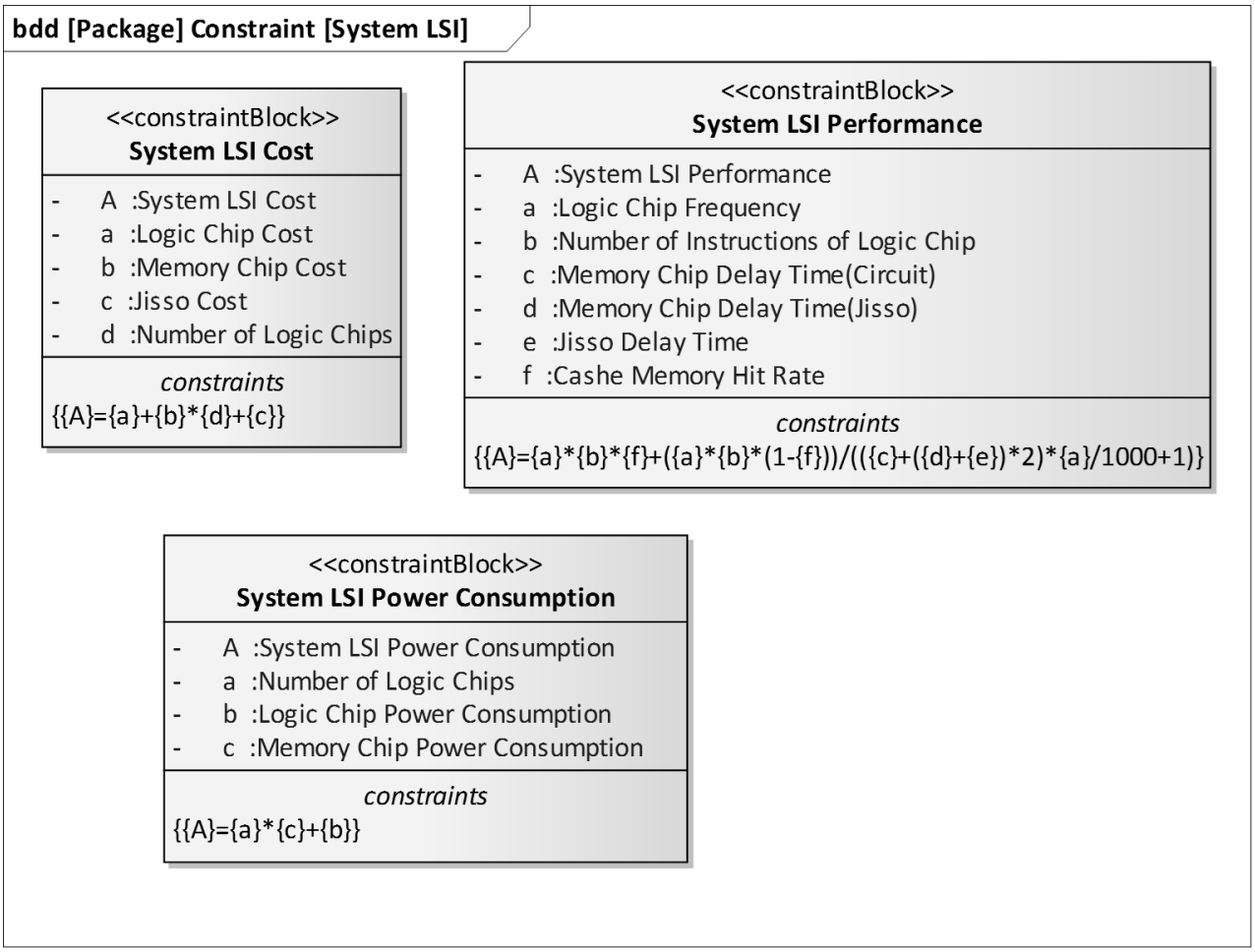


Fig. A.4 Detail of System LSI constraint block.

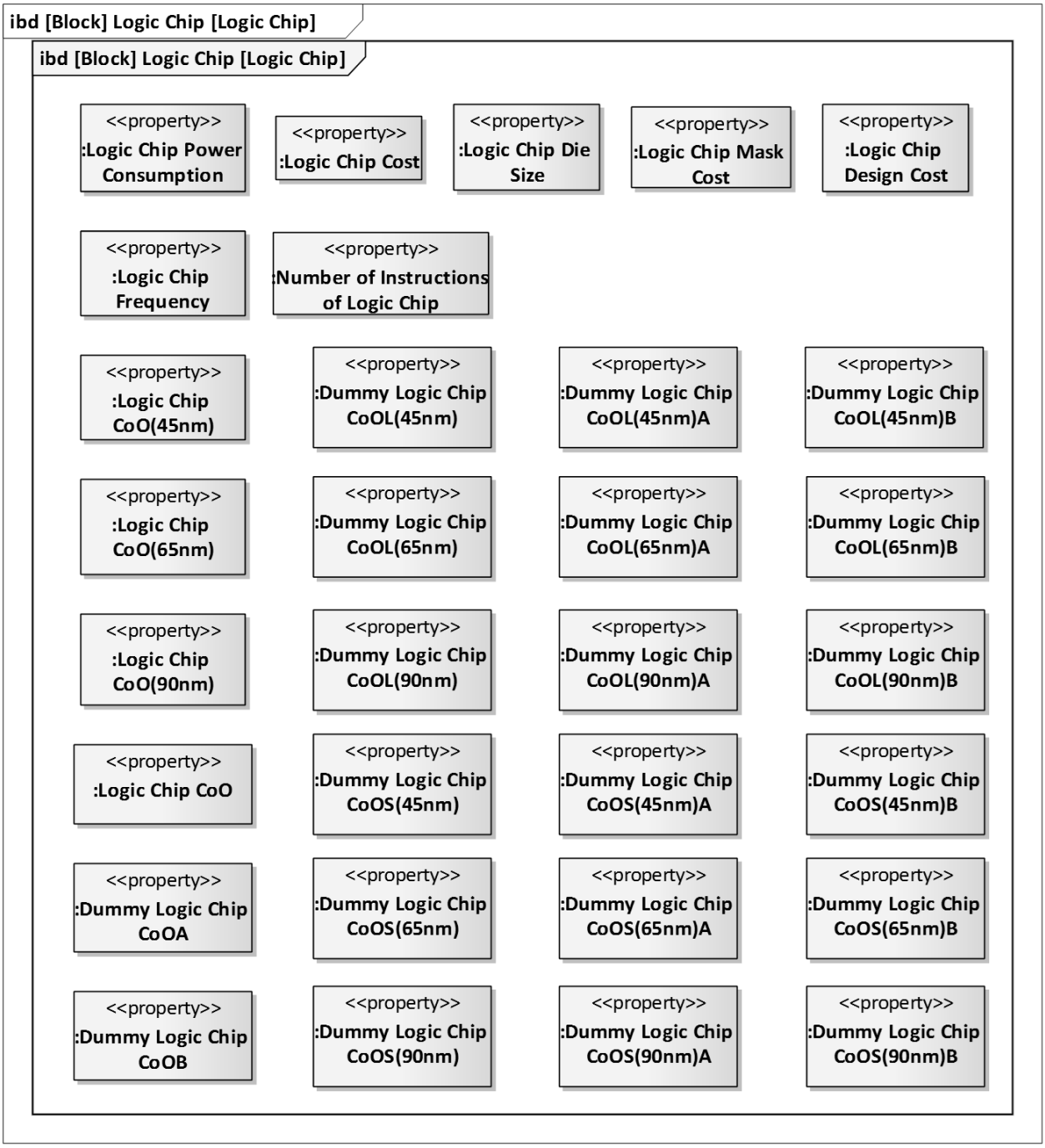
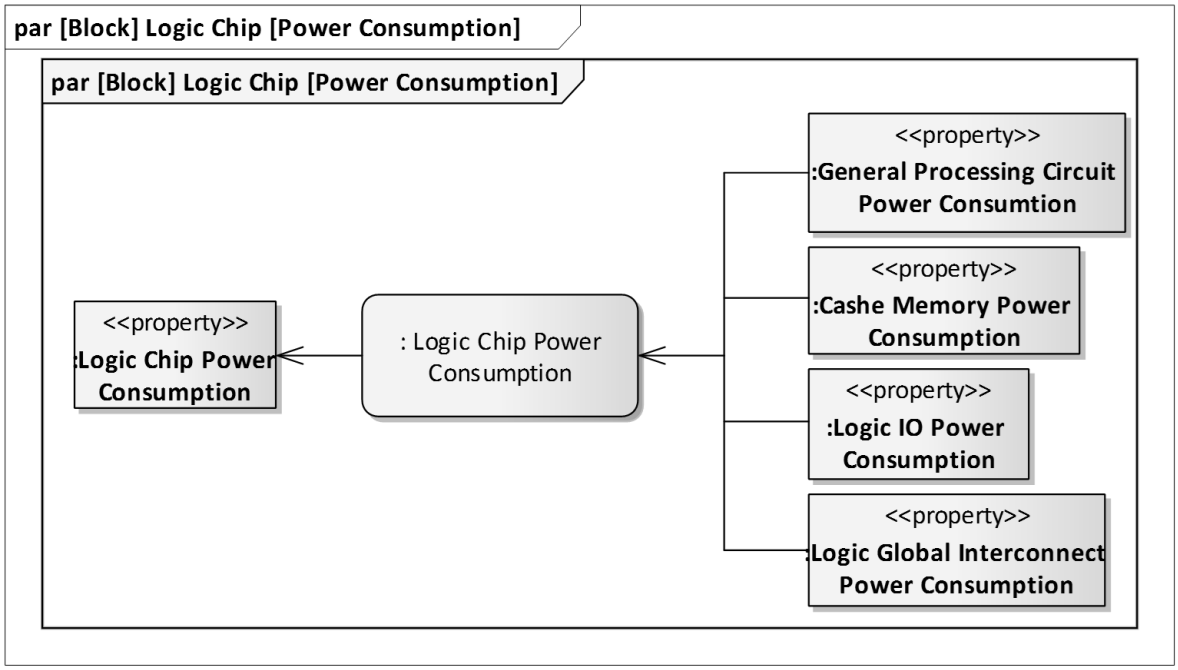
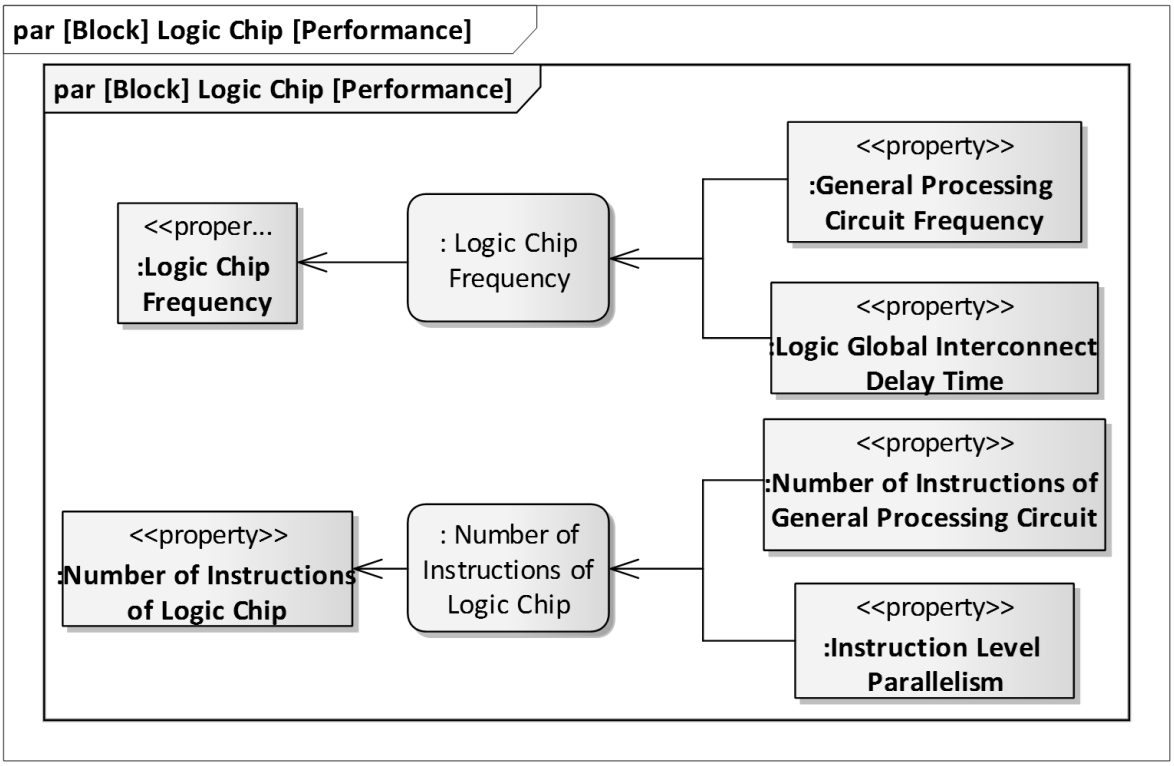


Fig. A.5 Logic chip internal block diagram.

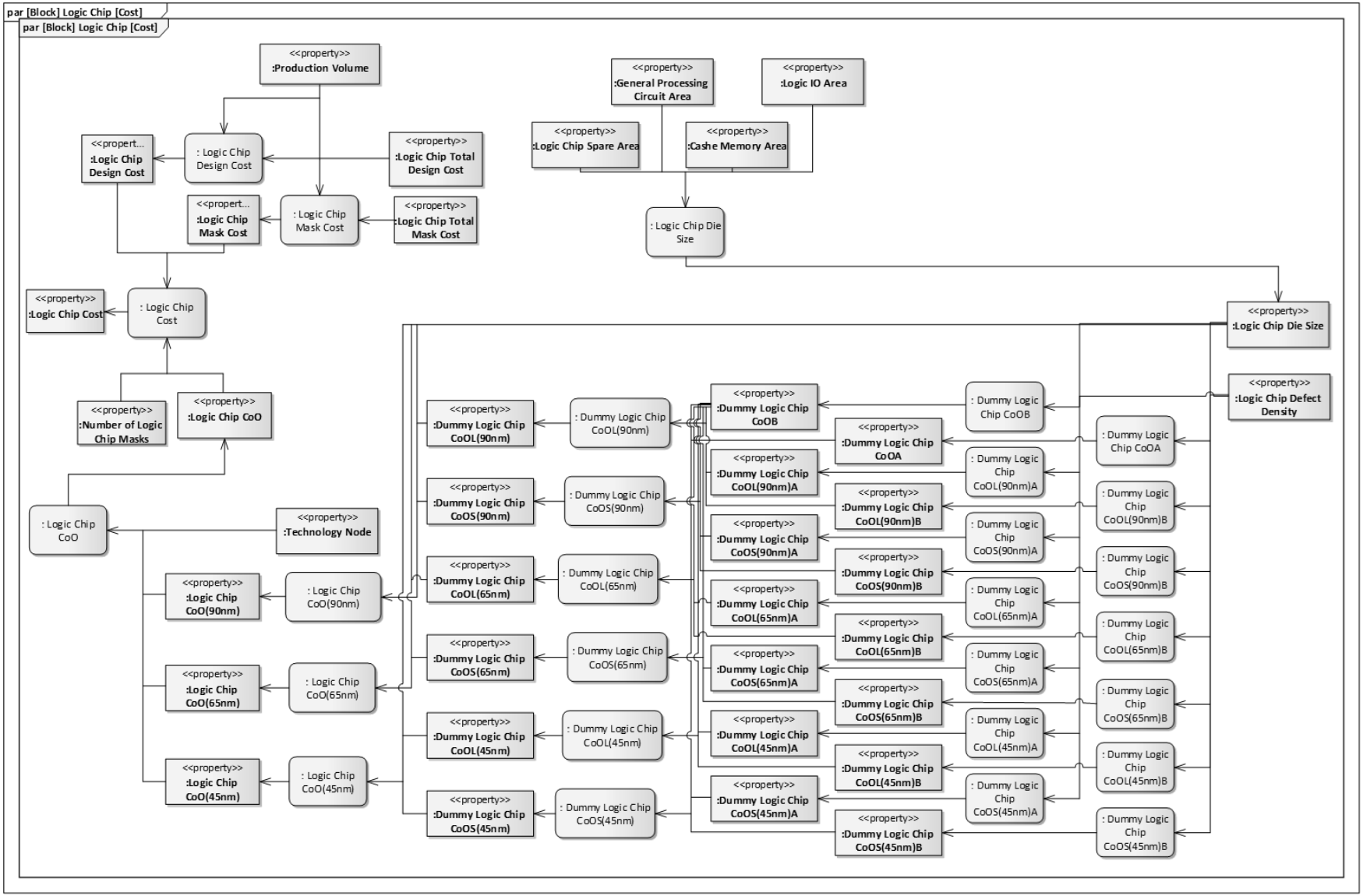


(a) Power consumption.



(b) Performance.

Fig. A.6 Logic chip parametric diagram.



(c) Cost.

Fig. A.6 Continued.

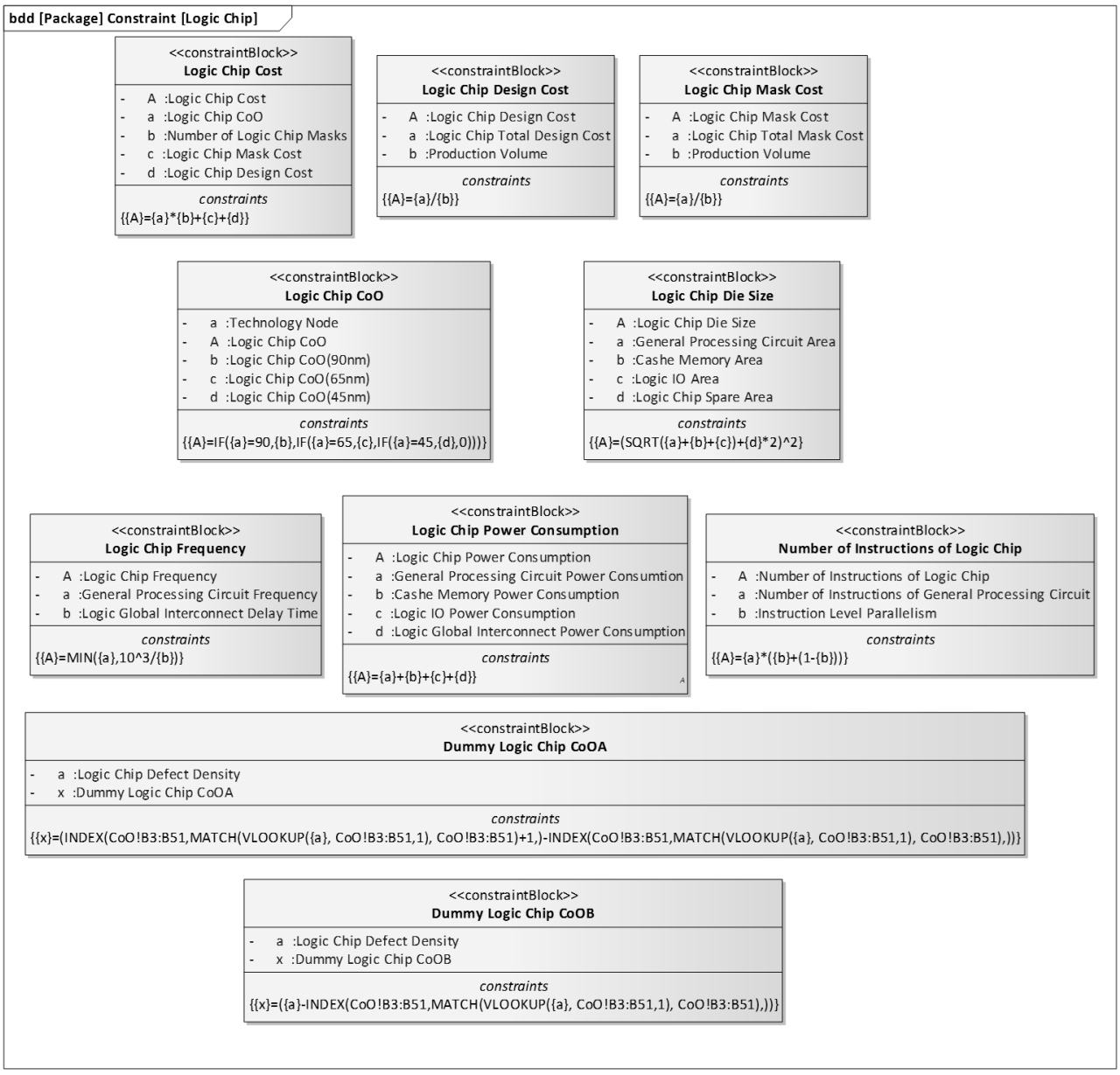


Fig. A.7 Detail of logic chip constraint block.

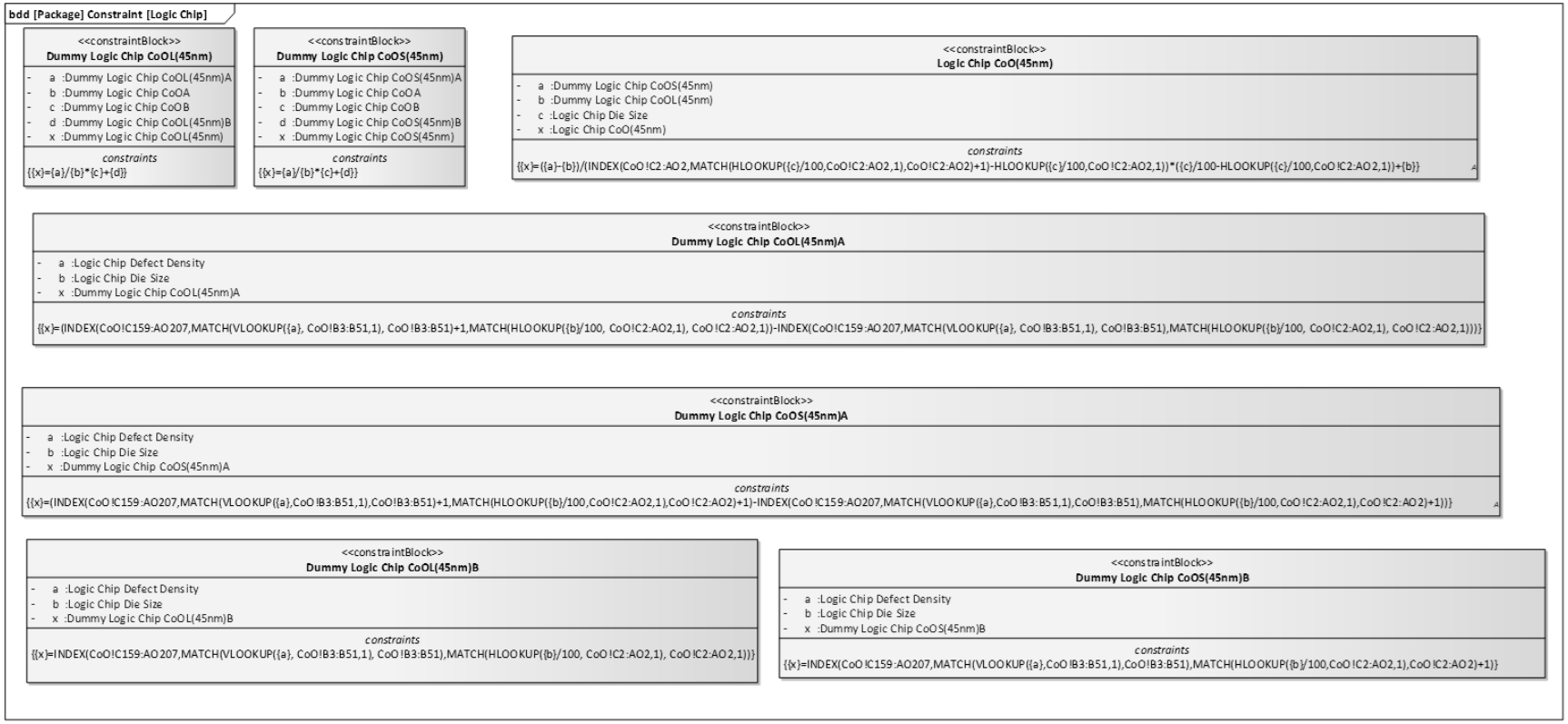


Fig. A.7 Continued.

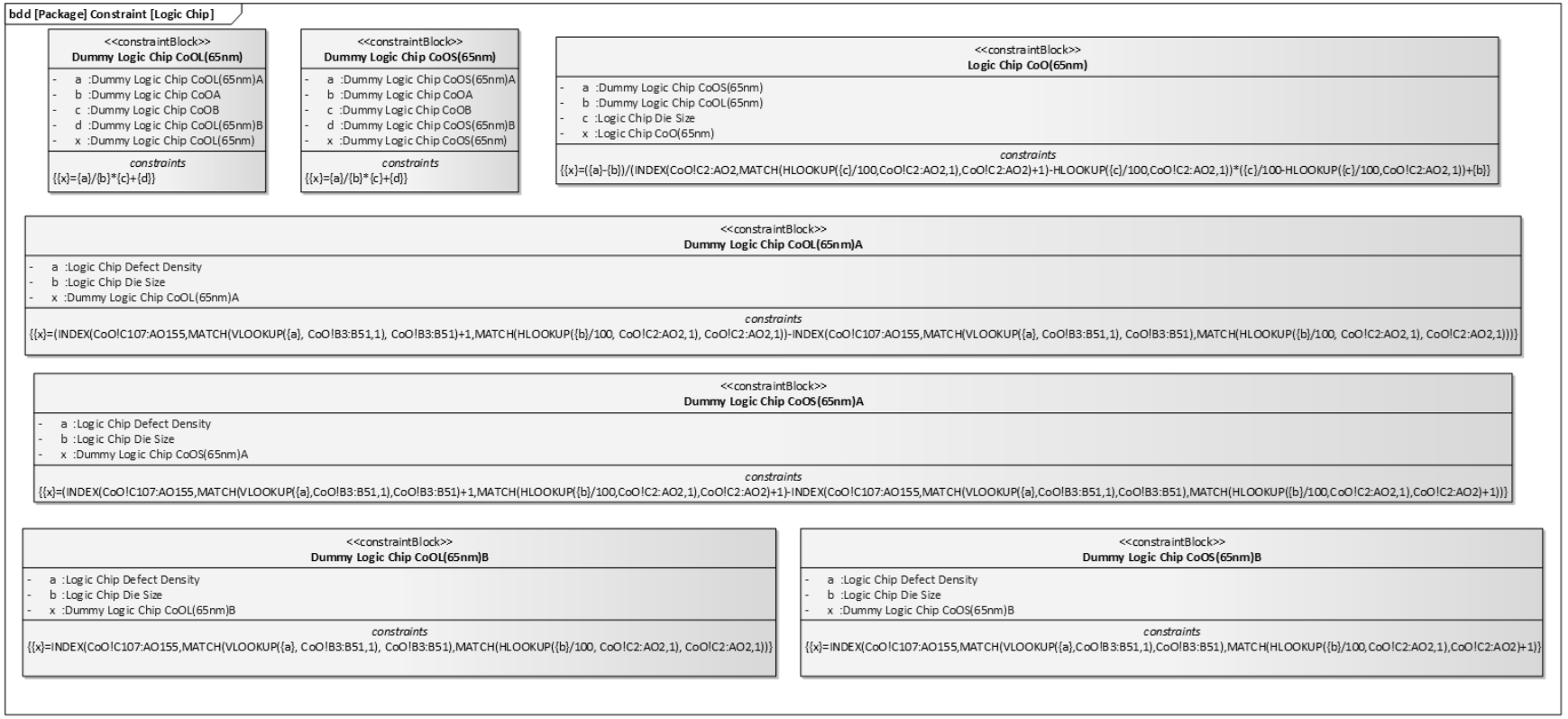


Fig. A.7 Continued.

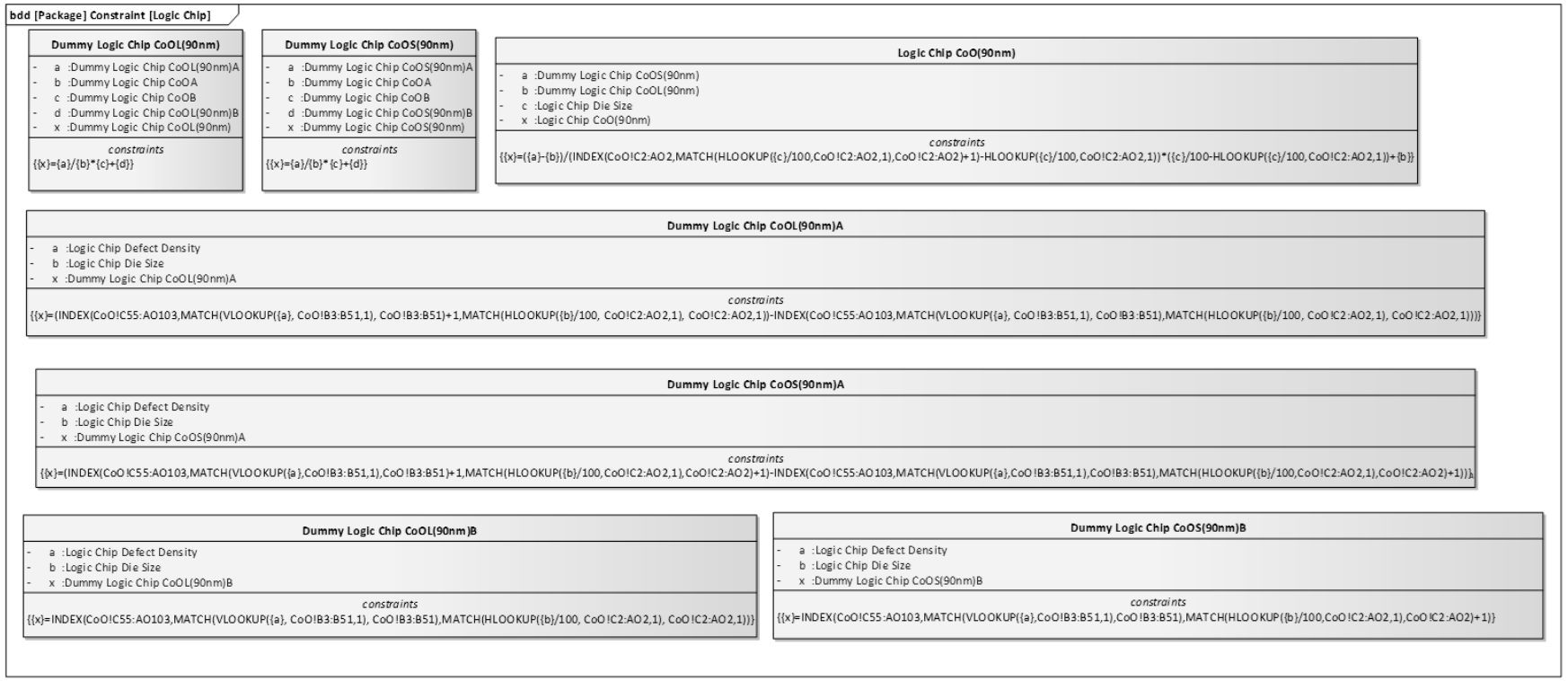


Fig. A.7 Continued.

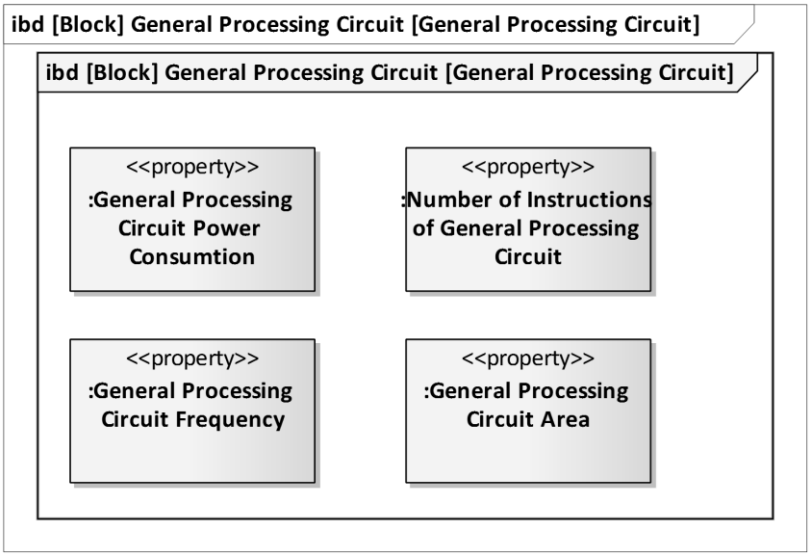
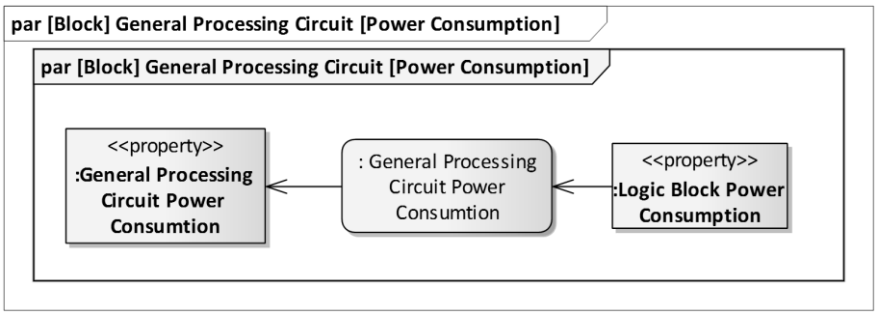
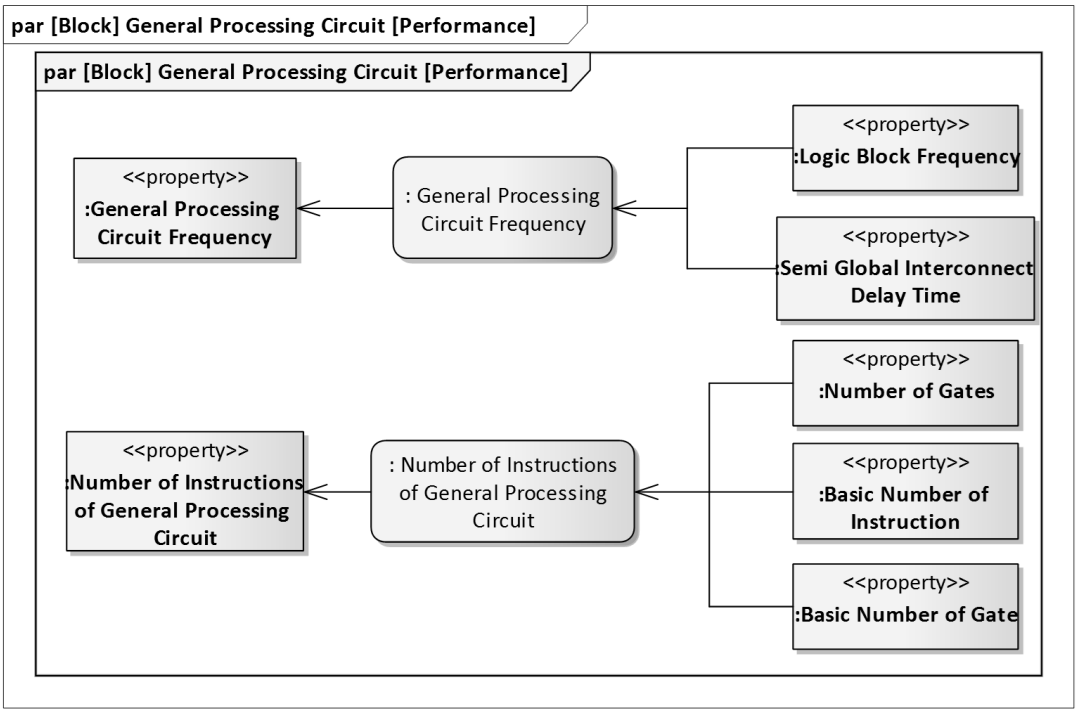


Fig. A.8 General processing circuit internal block diagram.

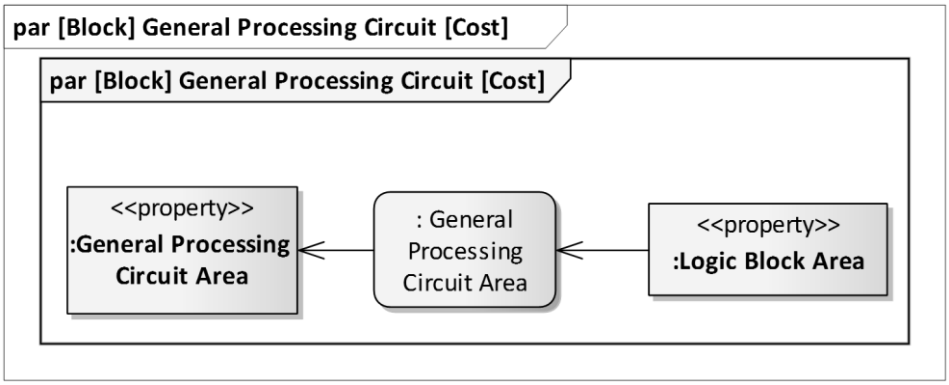


(a) Power consumption.



(b) Performance.

Fig. A.9 General processing parametric diagram.



(c) Cost.

Fig. A.9 Continued.

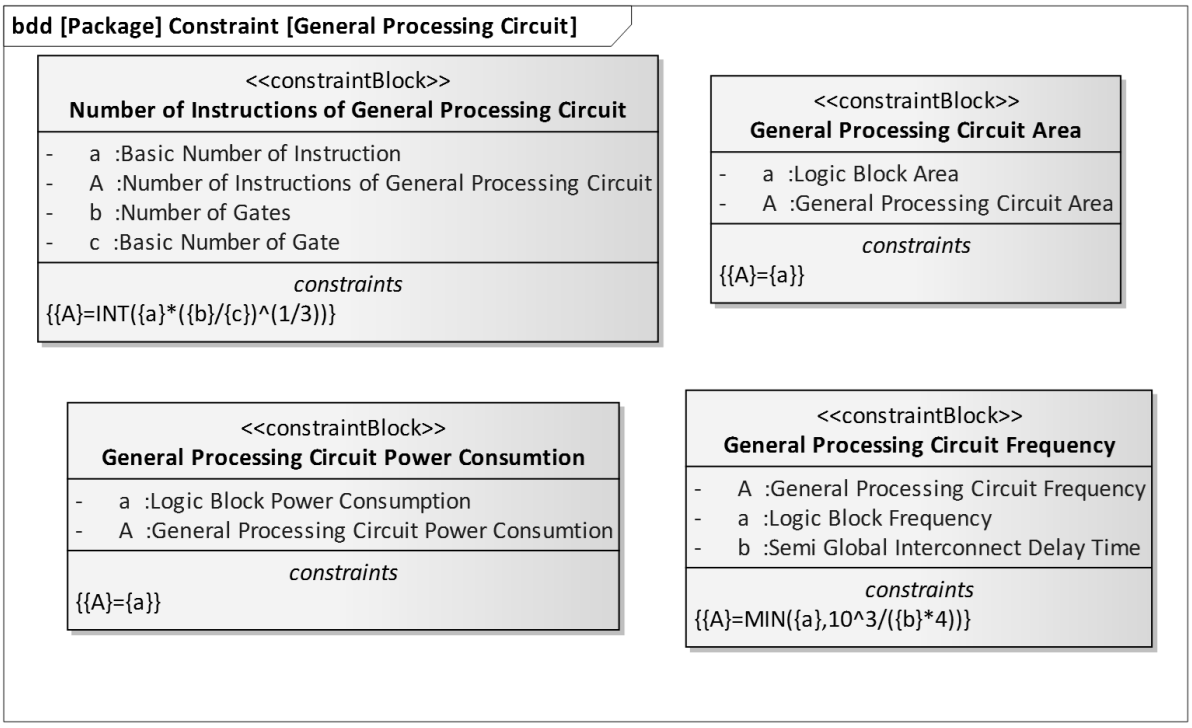


Fig. A.10 Detail of general processing circuit constraint block.

Fig. A.11 にロジックブロックの内部ブロック図、Fig. A.12(a)にロジックブロックの消費電力に関するパラメトリック図、Fig. A.12(b)に性能に関するパラメトリック図、Fig. A.12(c)にコストに関するパラメトリック図を示す。また、Fig. A.13 に制約式の詳細を示したブロック図を示す。

Fig. A.14 にロジック MPU ゲートの内部ブロック図、Fig. A.15(a)にロジック MPU ゲートの消費電力に関するパラメトリック図、Fig. A.15(b)にコストに関するパラメトリック図を示す。また、Fig. A.16 に制約式の詳細を示したブロック図を示す。

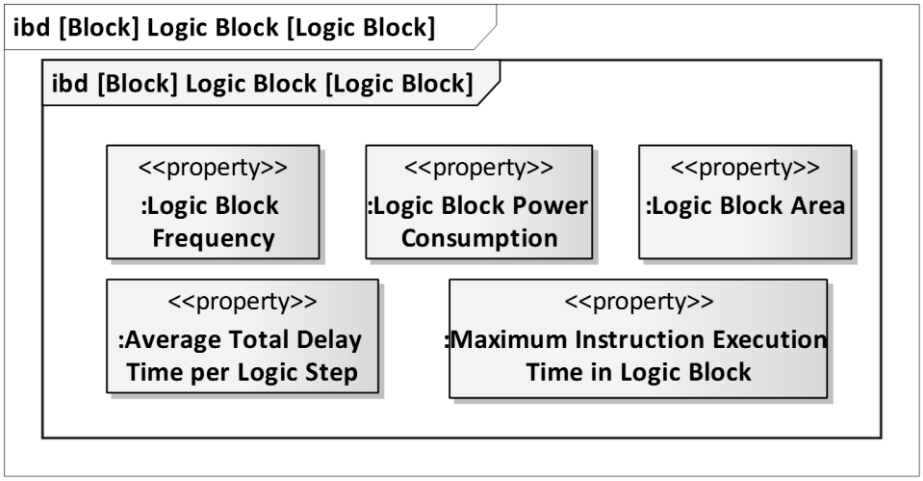
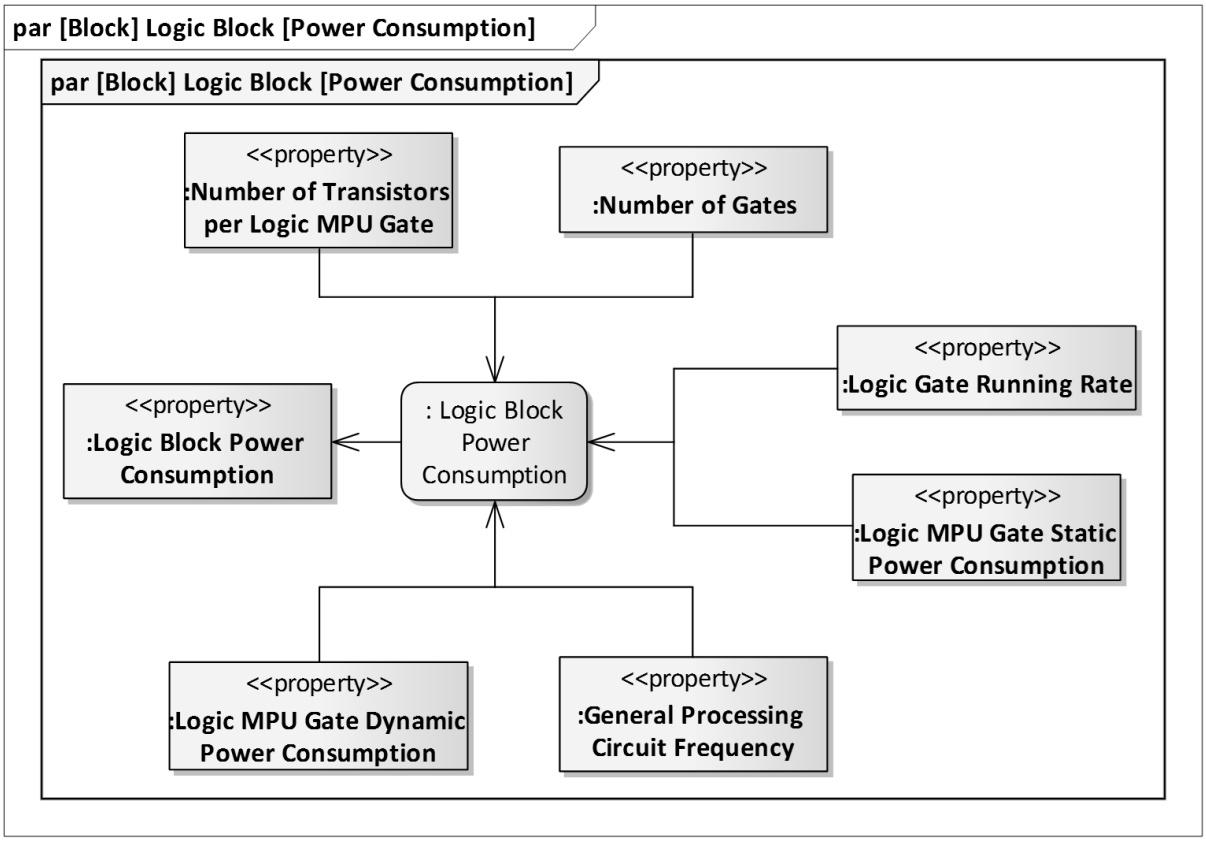
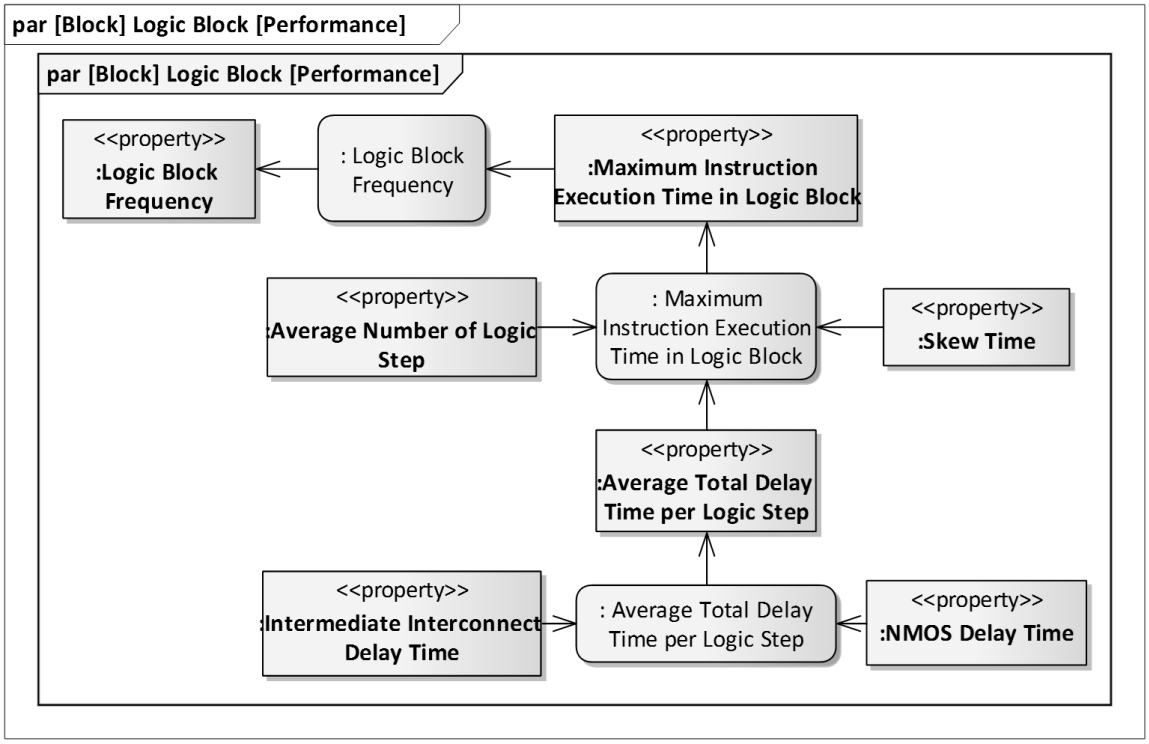


Fig. A.11 Logic block circuit internal block diagram.

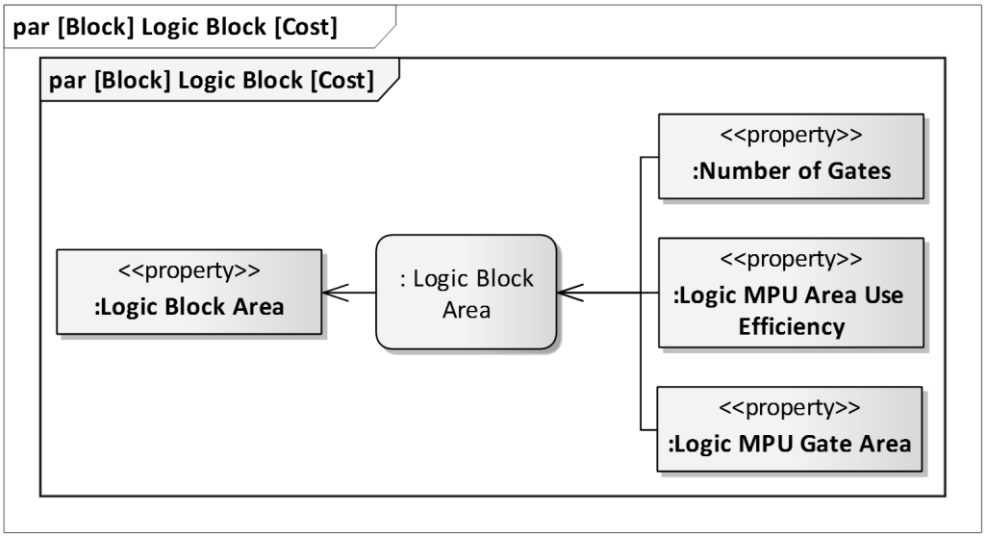


(a) Power consumption.

Fig. A.12 Logic block parametric diagram.



(b) Performance.



(c) Cost.

Fig. A.12 Continued.

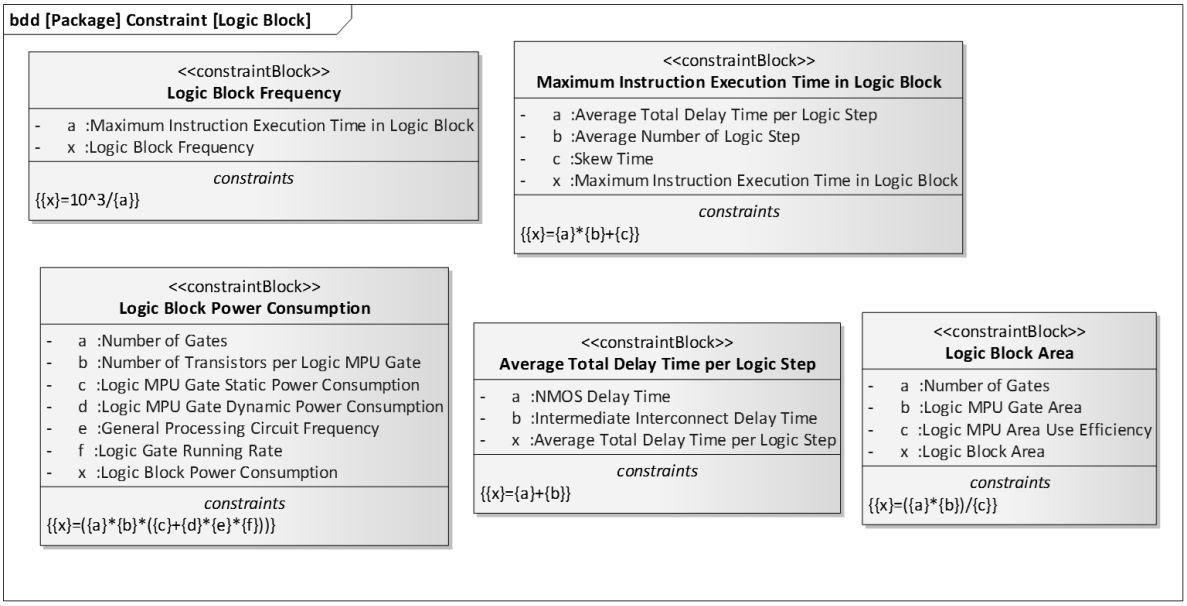


Fig. A.13 Detail of logic block constraint block.

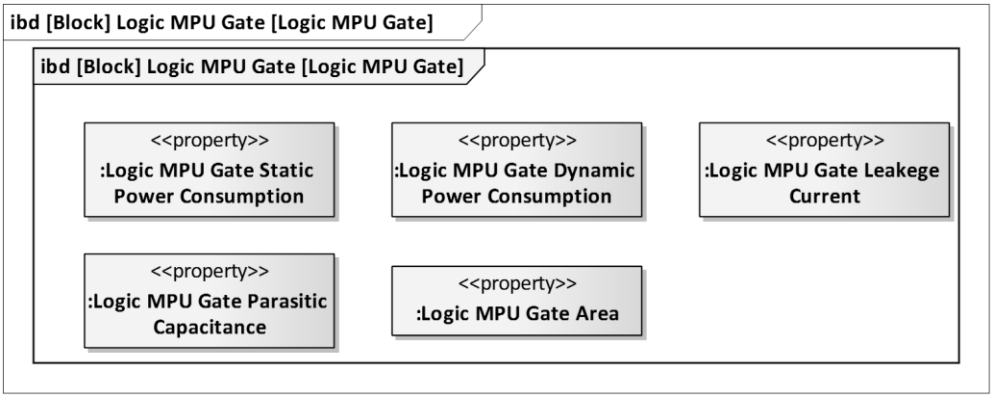
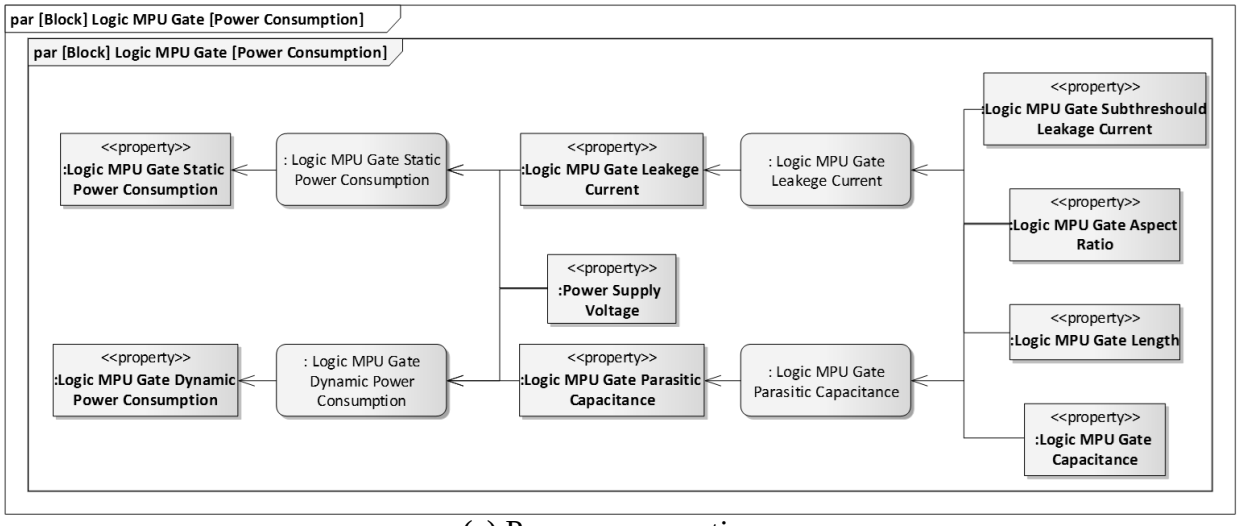
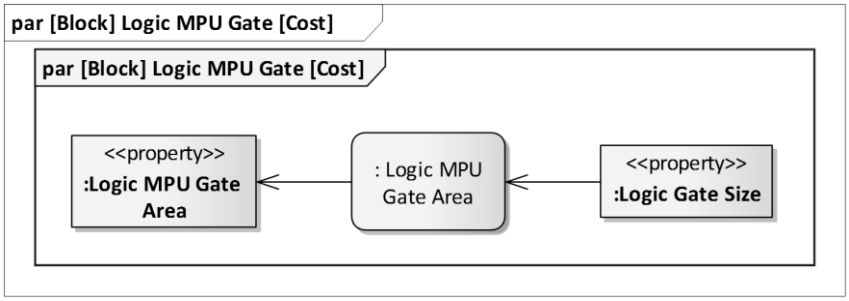


Fig. A.14 Logic MPU Gate internal block diagram.



(a) Power consumption.

Fig. A.15 Logic MPU Gate parametric diagram.



(b) Cost.
Fig. A.15 Continued.

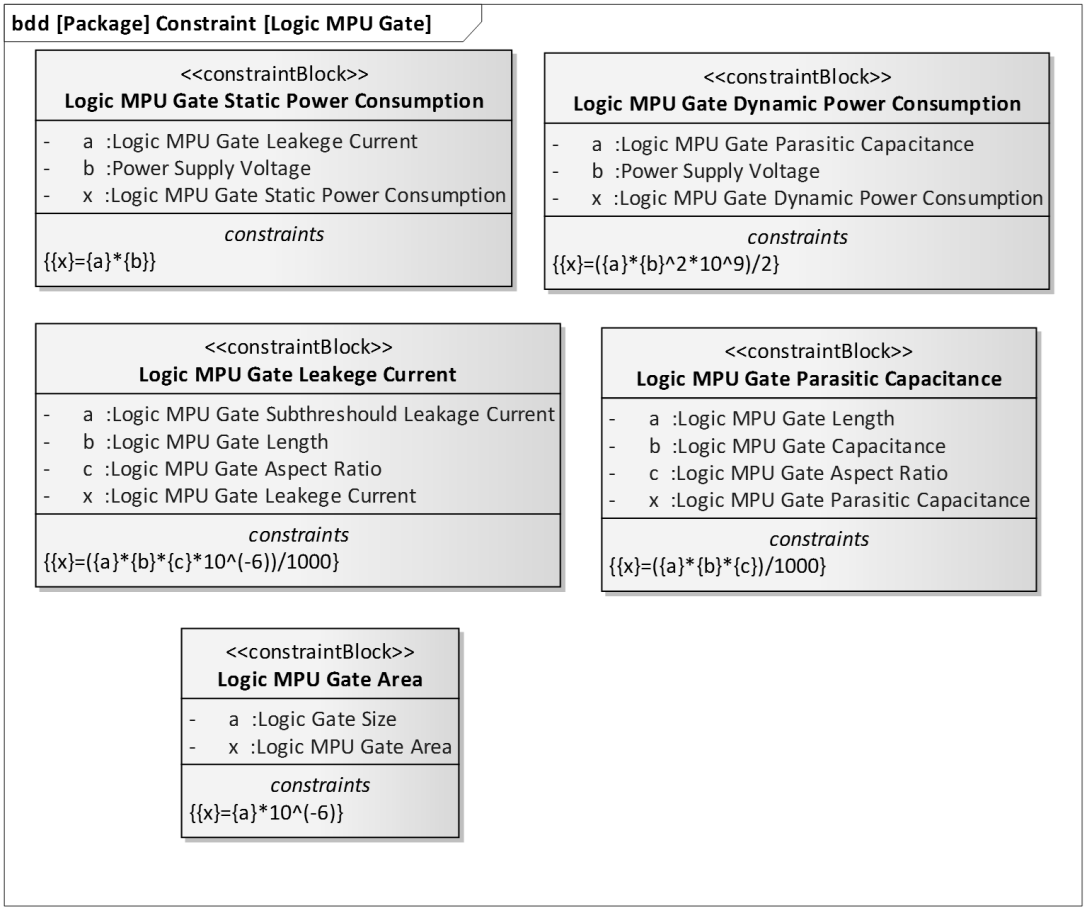


Fig. A.16 Detail of logic MPU Gate constraint block.

Fig. A.17 にインターミディエイト配線の内部ブロック図、Fig. A.18 にインターミディエイト配線の性能に関するパラメトリック図を示す。また、Fig. A.19 に制約式の詳細を示したブロック図を示す。

Fig. A.20 にセミグローバル配線の内部ブロック図、Fig. A.21 にセミグローバル配線の性能に関するパラメトリック図を示す。また、Fig. A.22 に制約式の詳細を示したブロック図を示す。

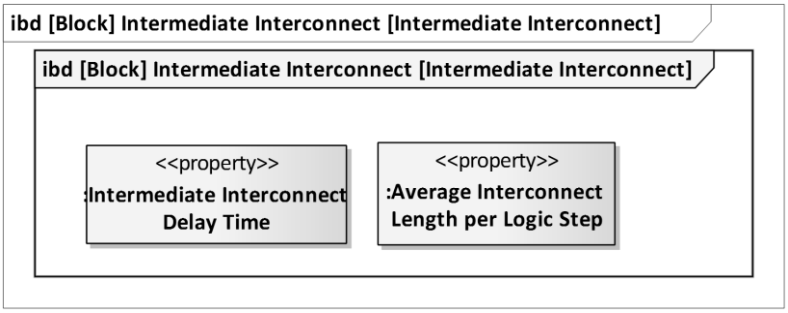


Fig. A.17 Intermediate interconnect internal block diagram.

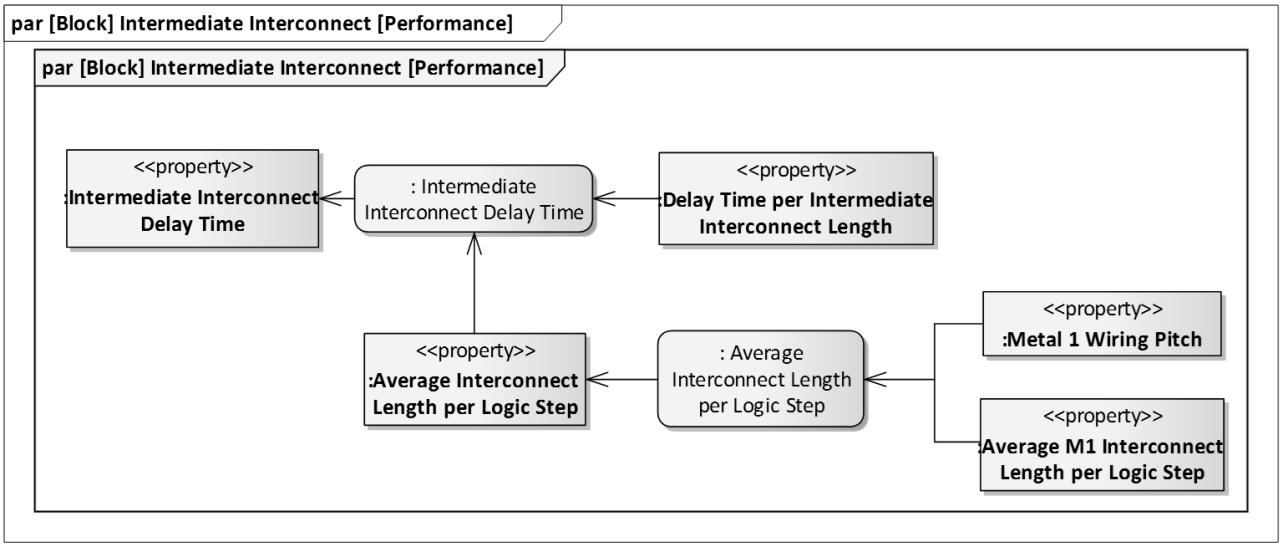


Fig. A.18 Intermediate interconnect performance parametric diagram.

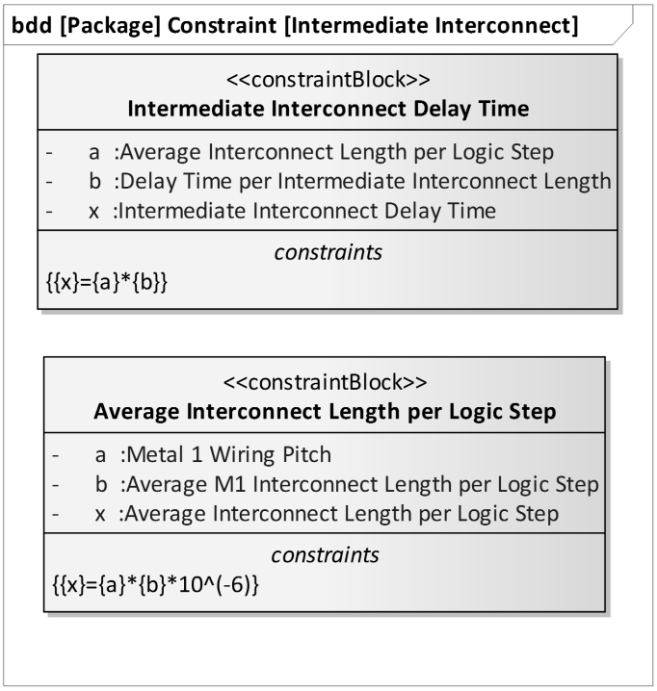


Fig. A.19 Detail of intermediate interconnect constraint block.

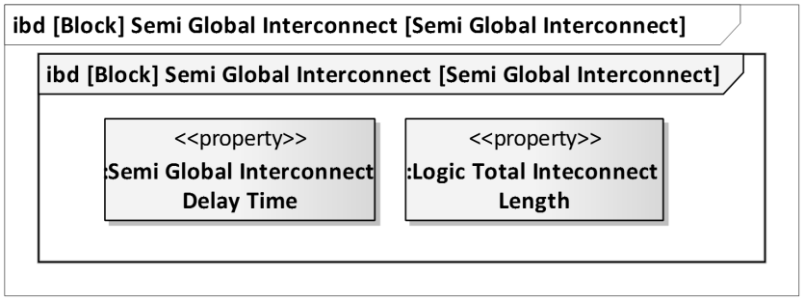


Fig. A.20 Semi global interconnect internal block diagram.

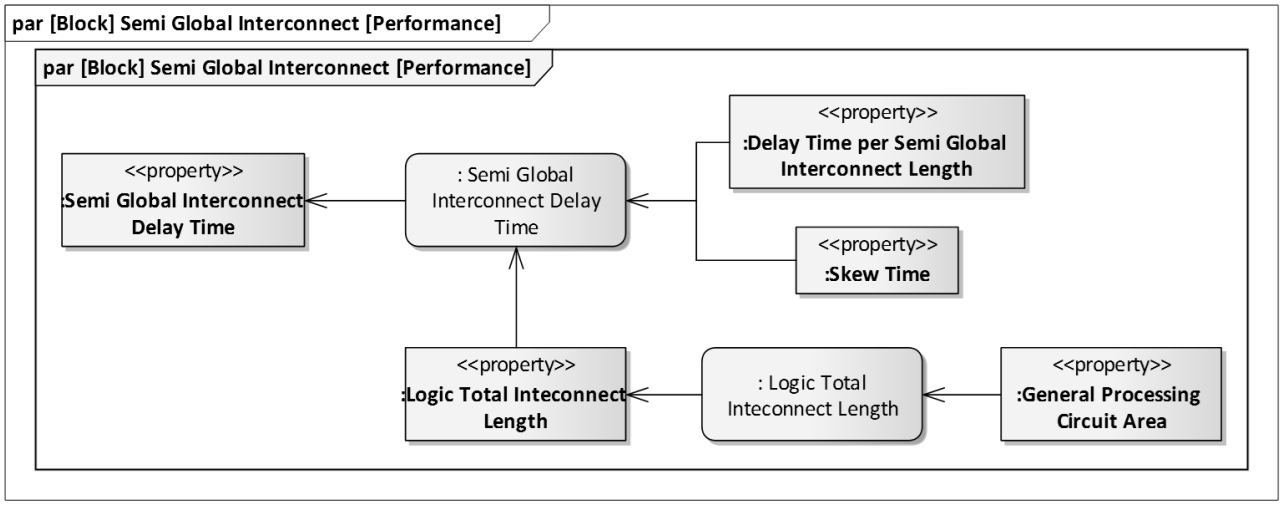


Fig. A.21 Semi global interconnect performance parametric diagram.

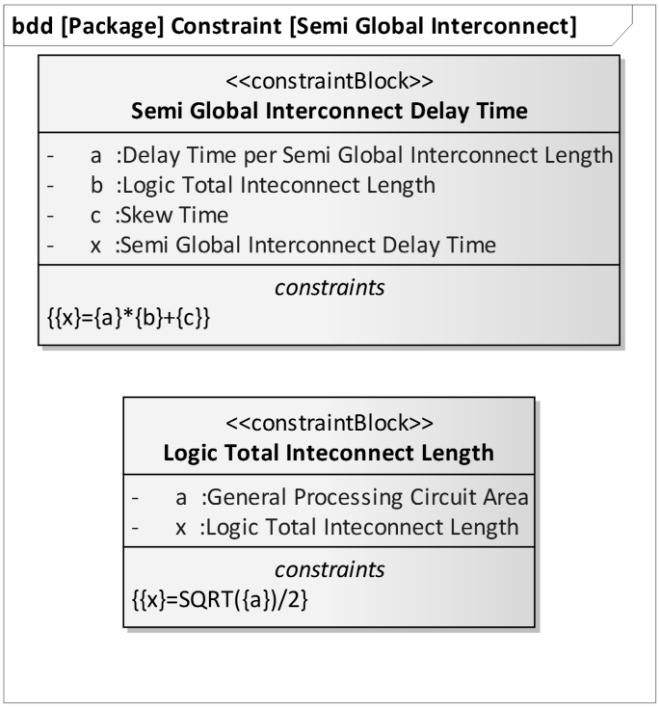


Fig. A.22 Detail of semi global interconnect constraint block.

Fig. A.23 にキャッシュメモリの内部ブロック図、Fig. A.24(a)にキャッシュメモリの消費電力に関するパラメトリック図、Fig. A.24(b)にキャッシュメモリのコストに関するパラメトリック図を示す。また、Fig. A.25 に制約式の詳細を示したブロック図を示す。

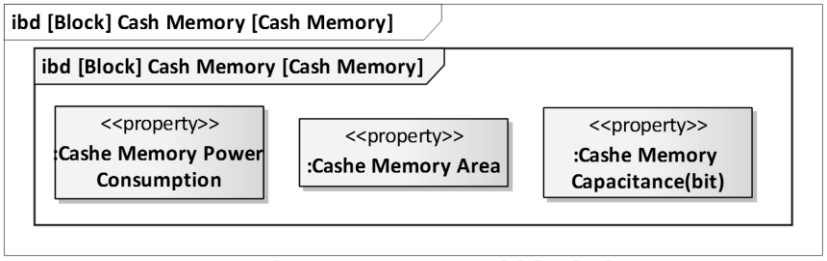
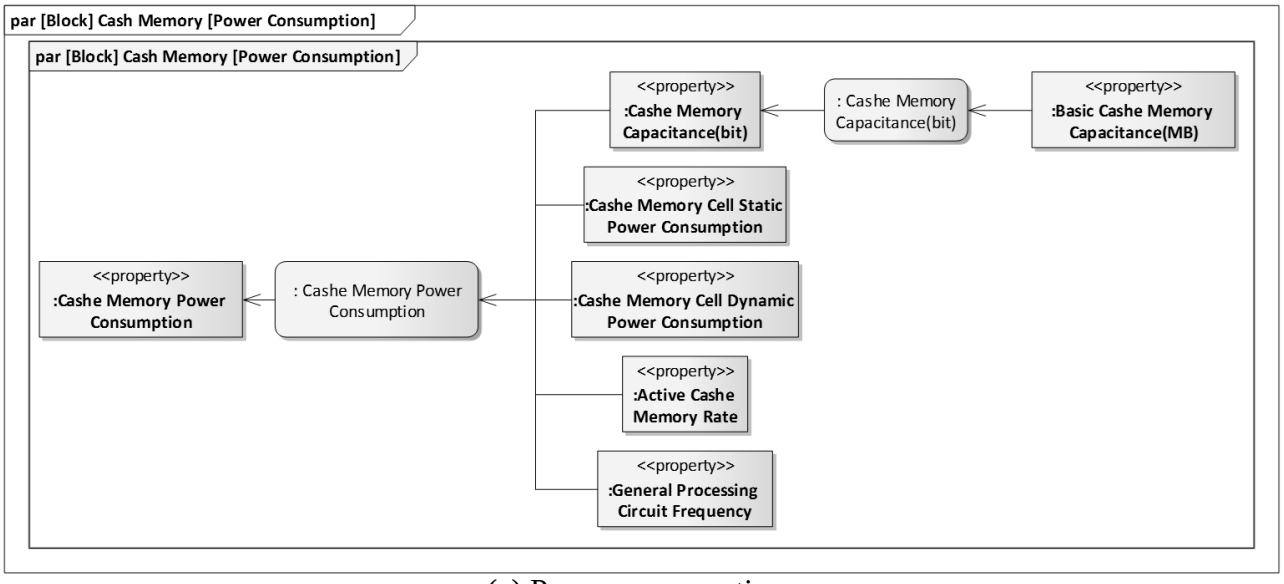
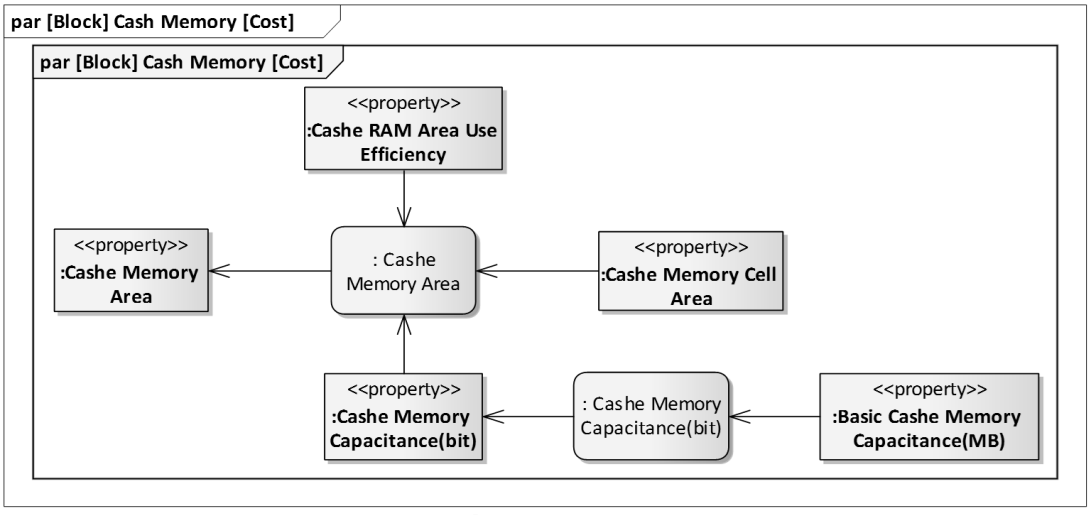


Fig. A.23 Cash memory internal block diagram.



(a) Power consumption.



(b) Cost.

Fig. A.24 Cash memory parametric diagram.

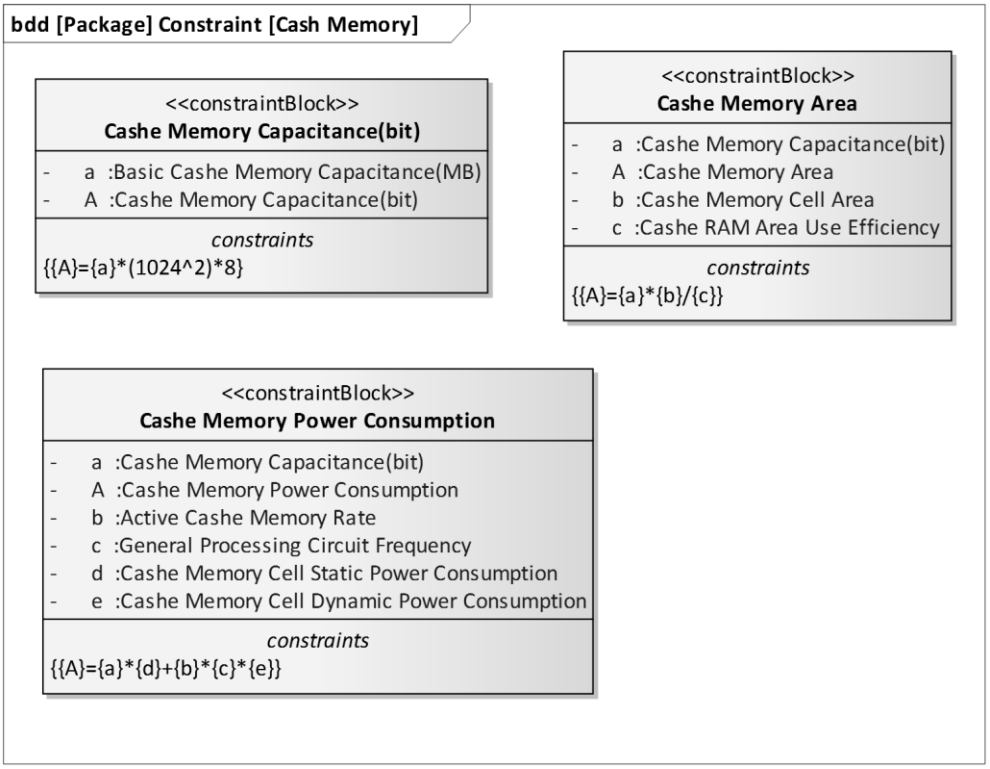


Fig. A.25 Detail of cash memory constraint block.

Fig. A.26 にロジック IO の内部ブロック図、Fig. A.27(a)にロジック IO の消費電力に関するパラメトリック図、Fig. A.27(b)にロジック IO のコストに関するパラメトリック図を示す。また、Fig. A.28 に制約式の詳細を示したブロック図を示す。

Fig. A.29 にロジック IO ドライバの内部ブロック図、Fig. A.30(a)にロジック IO ドライバの消費電力に関するパラメトリック図、Fig. A.30(b)にロジック IO ドライバのコストに関するパラメトリック図を示す。また、Fig. A.31 に制約式の詳細を示したブロック図を示す。

Fig. A.32 にロジック IOMPU ゲートの内部ブロック図、Fig. A.33(a)にロジック IOMPU ゲートの消費電力に関するパラメトリック図、Fig. A.33(b)にロジック IOMPU ゲートのコストに関するパラメトリック図を示す。また、Fig. A.34 に制約式の詳細を示したブロック図を示す。

Fig. A.35 にロジック外部 IO ドライバの内部ブロック図、Fig. A.36(a)にロジック外部 IO ドライバの消費電力に関するパラメトリック図、Fig. A.36(b)にロジック外部 IO ドライバのコストに関するパラメトリック図を示す。また、Fig. A.37 に制約式の詳細を示したブロック図を示す。

Fig. A.38 にロジック外部 IO MPU ゲートの内部ブロック図、Fig. A.39(a)にロジック外部 IO MPU ゲートの消費電力に関するパラメトリック図、Fig. A.39(b)にロジック外部 IO MPU ゲートのコストに関するパラメトリック図を示す。また、Fig. A.40 に制約式の詳細を示したブロック図を示す。

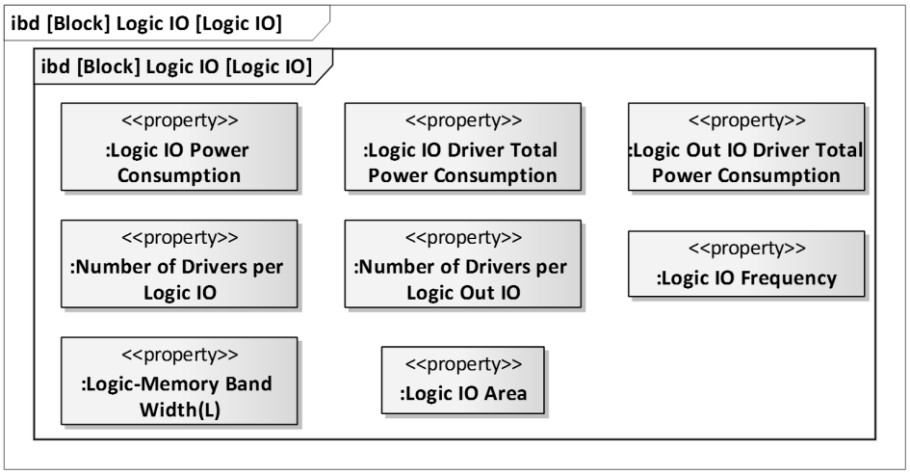
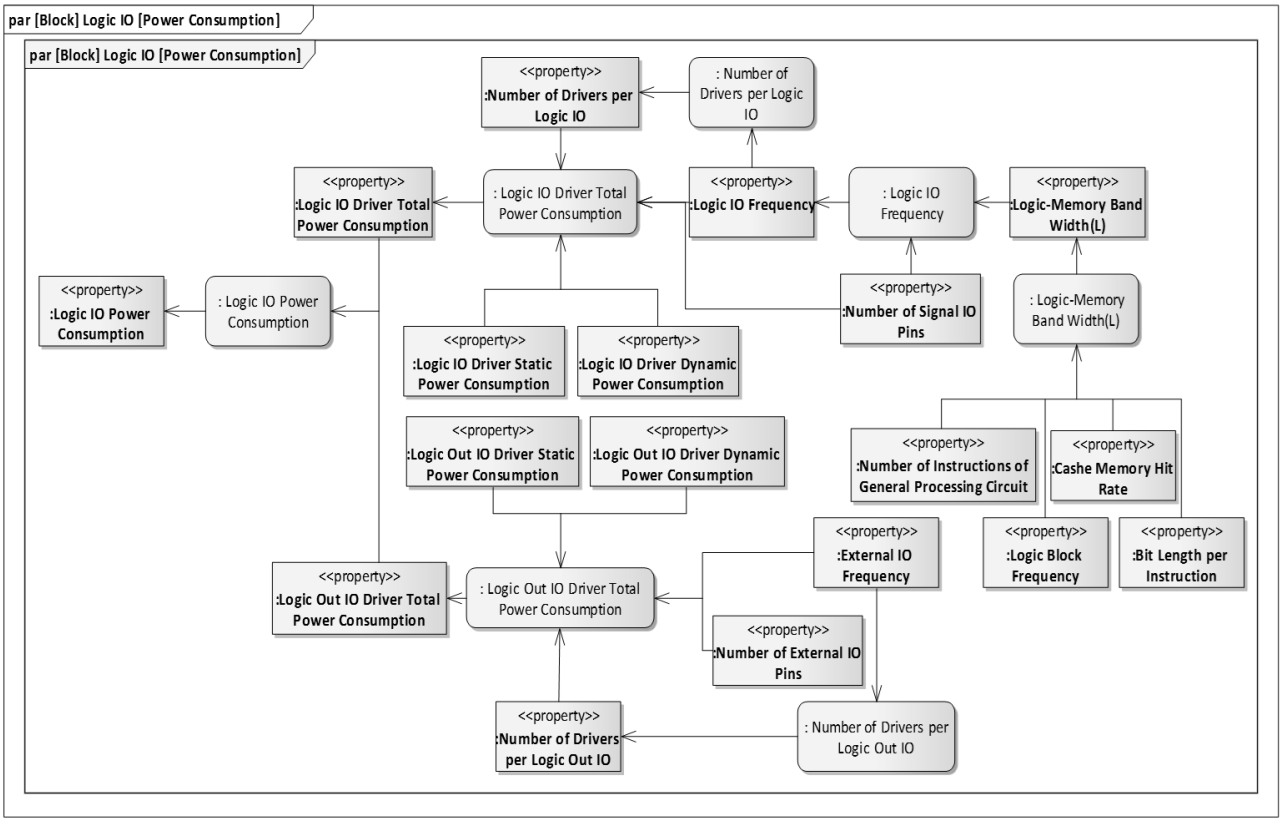
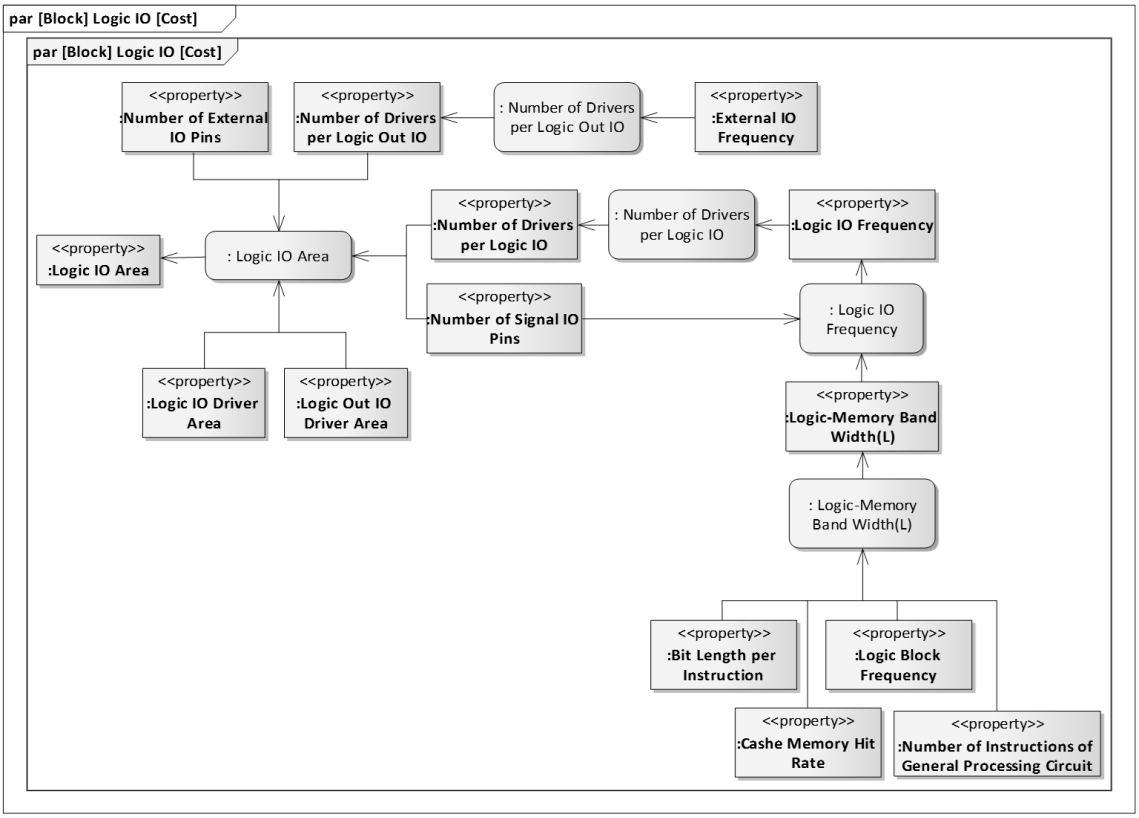


Fig. A.26 Logic IO internal block diagram.



(a) Power consumption.

Fig. A.27 Logic IO parametric diagram.



(b) Cost.
Fig. A.27 Continued.

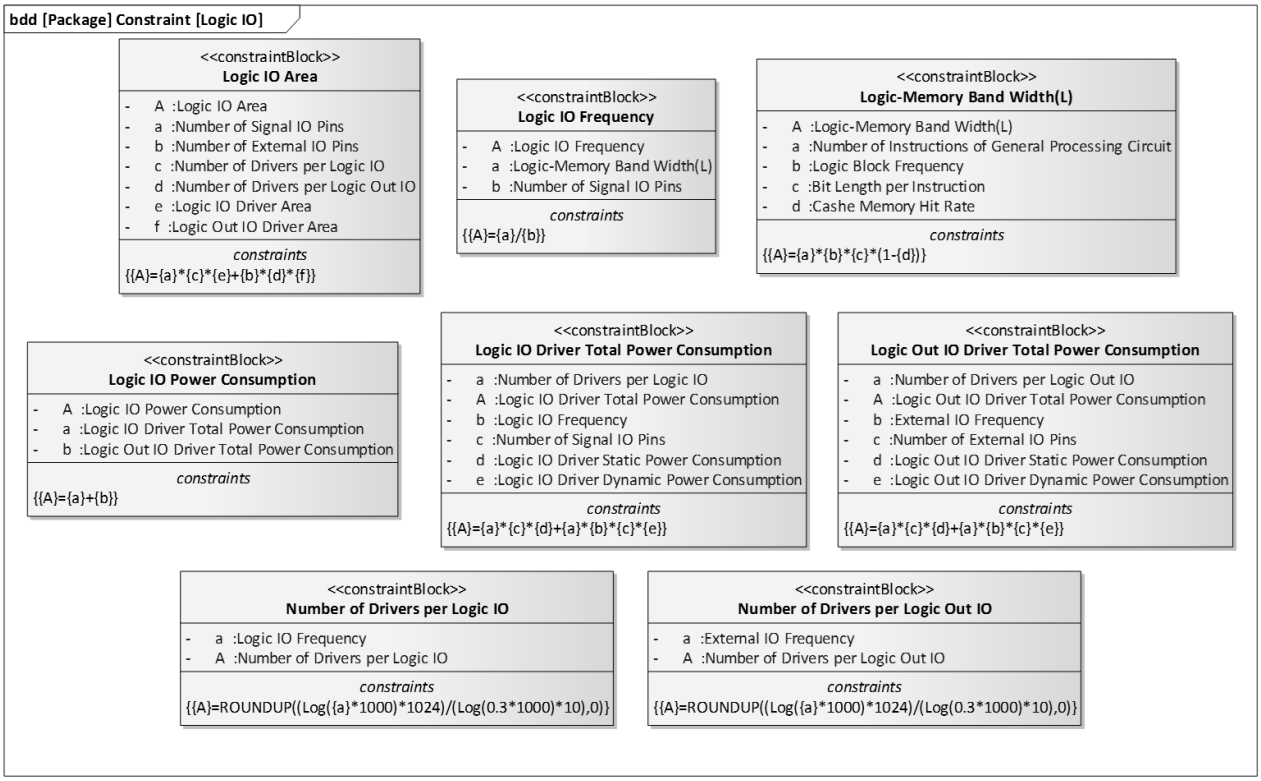


Fig. A.28 Detail of logic IO constraint block.

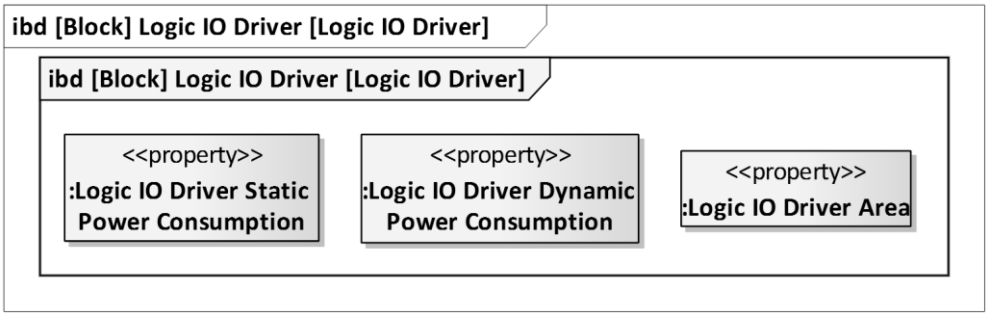
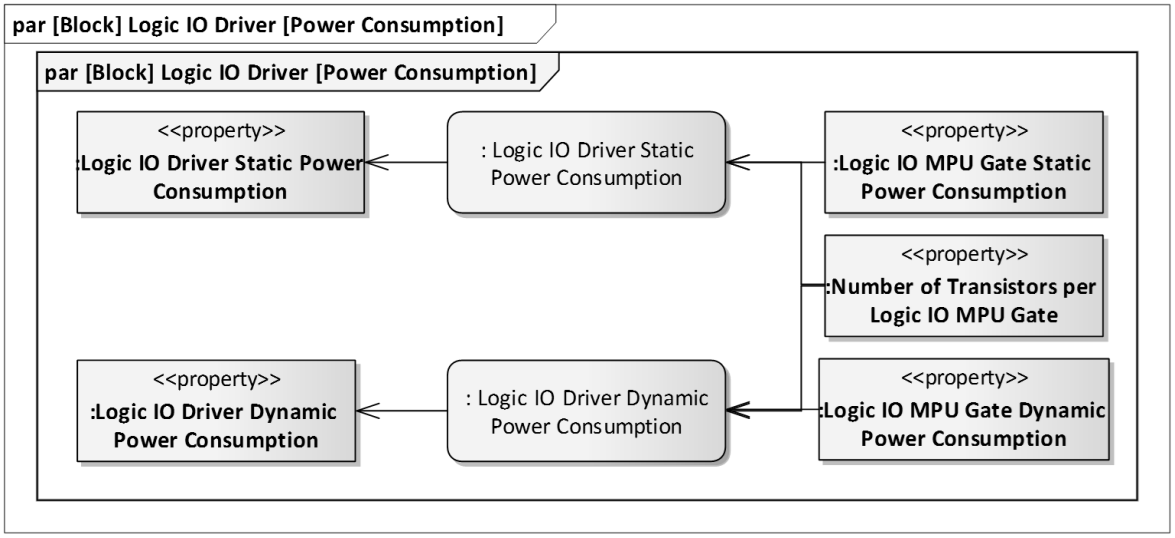
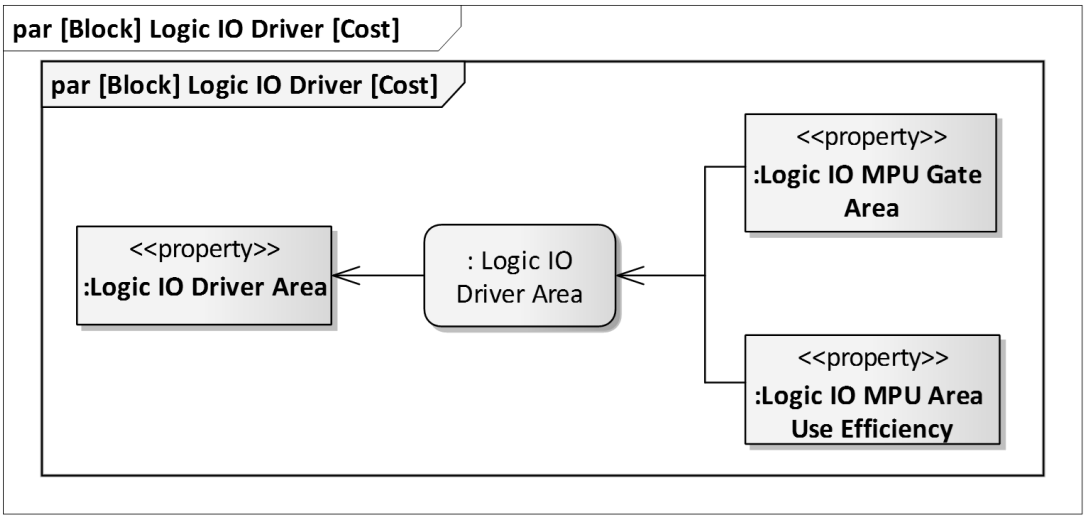


Fig. A.29 Logic IO driver internal block diagram.



(a) Power consumption.



(b) Cost.

Fig. A.30 Logic IO driver parametric diagram.

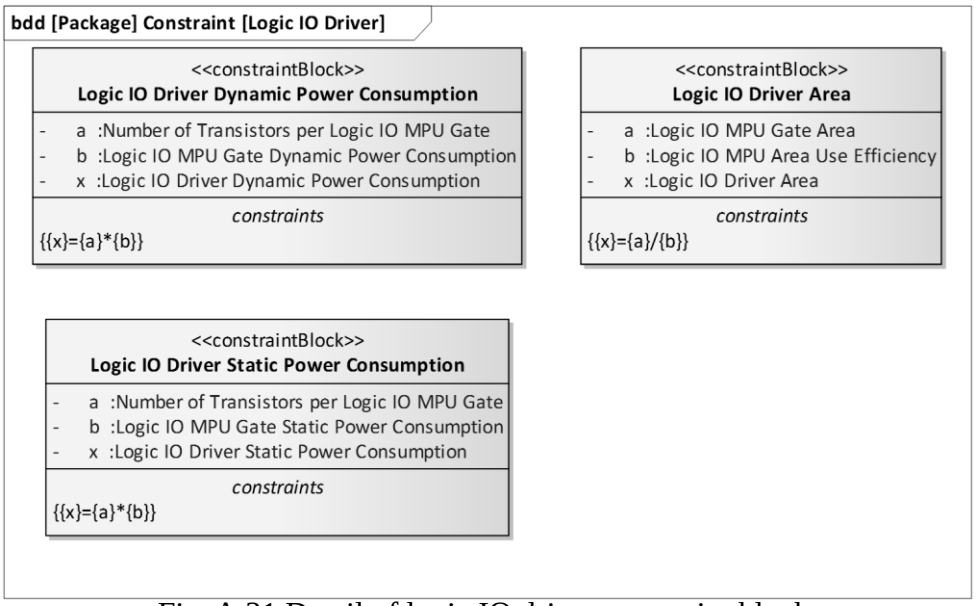


Fig. A.31 Detail of logic IO driver constraint block.

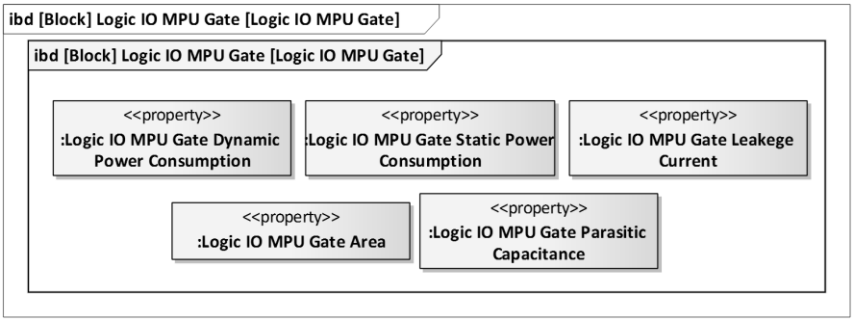
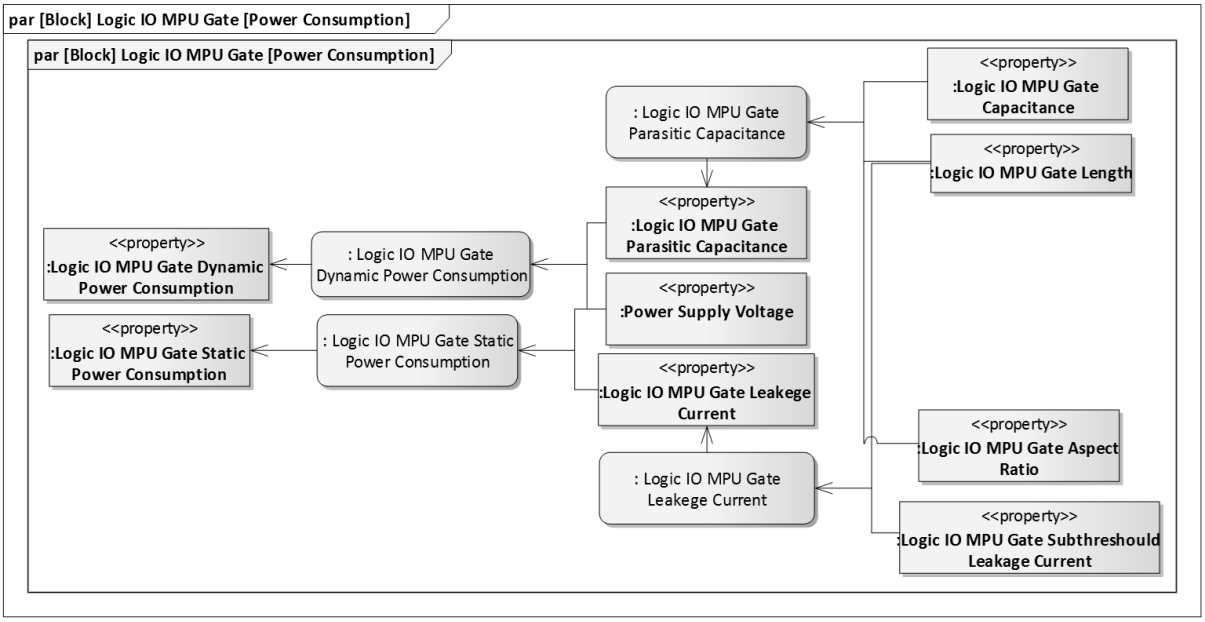
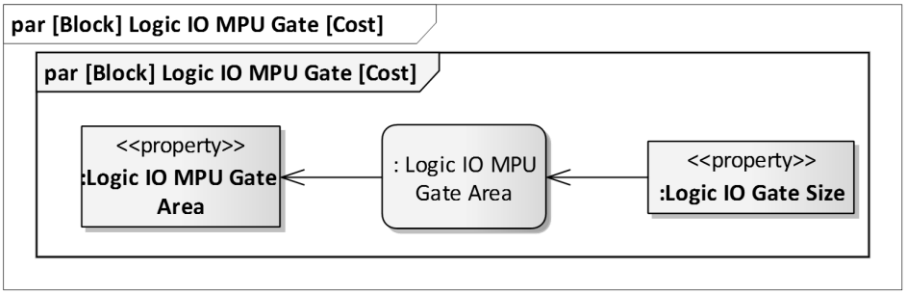


Fig. A.32 Logic IO MPU gate internal block diagram.



(a) Power consumption.

Fig. A.33 Logic IO MPU gate parametric diagram.



(b) Cost.

Fig. A.33 Continued.

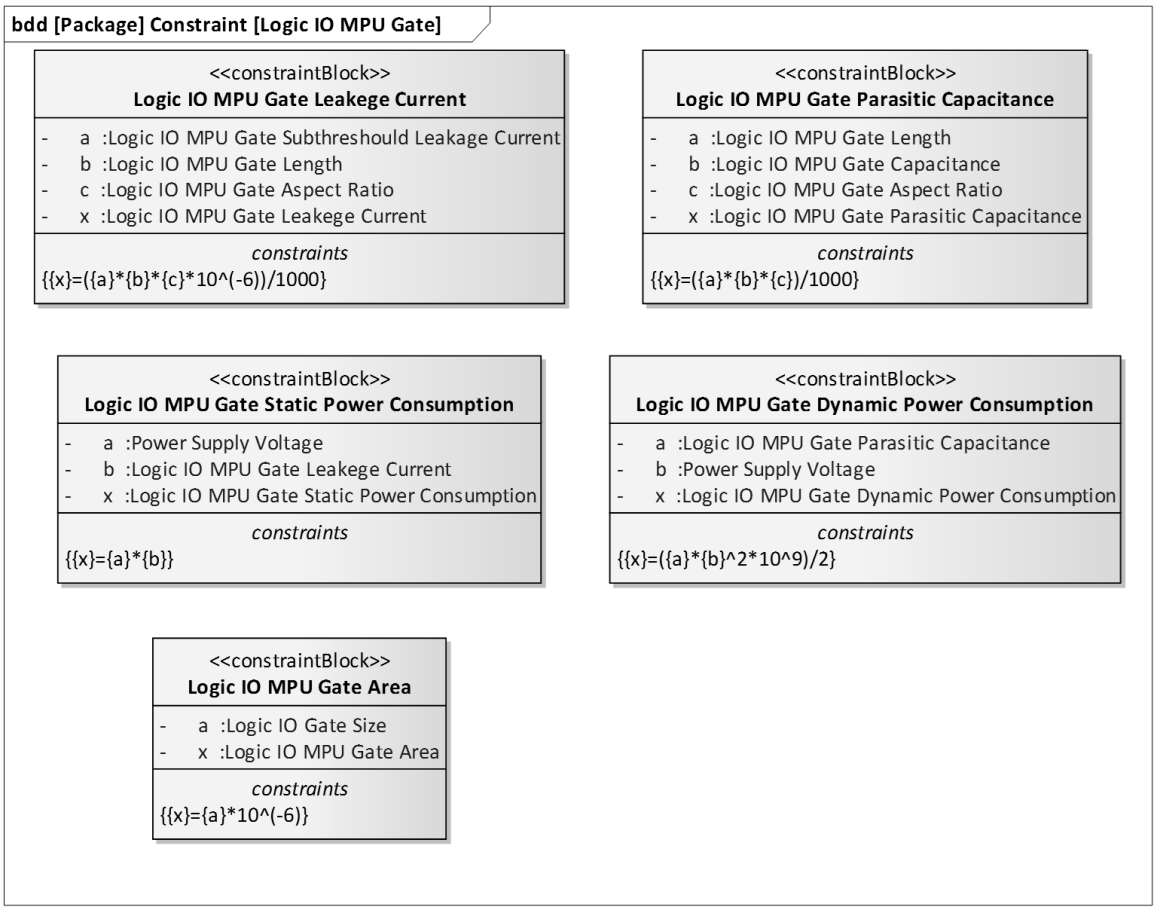


Fig. A.34 Detail of logic IO MPU gate constraint block.

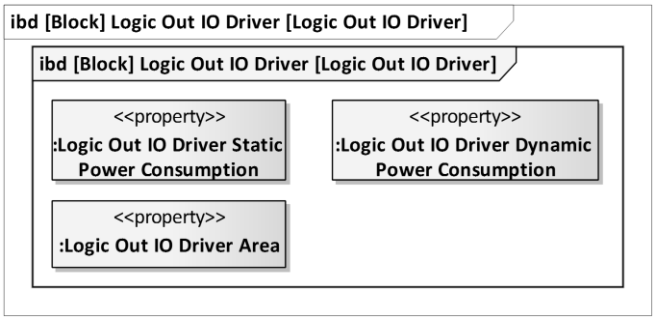
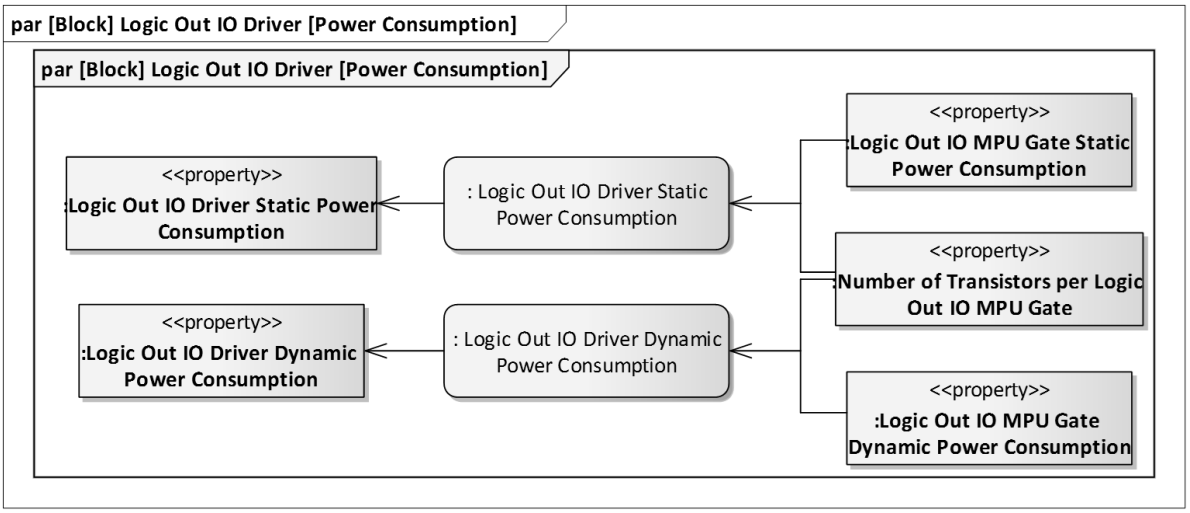
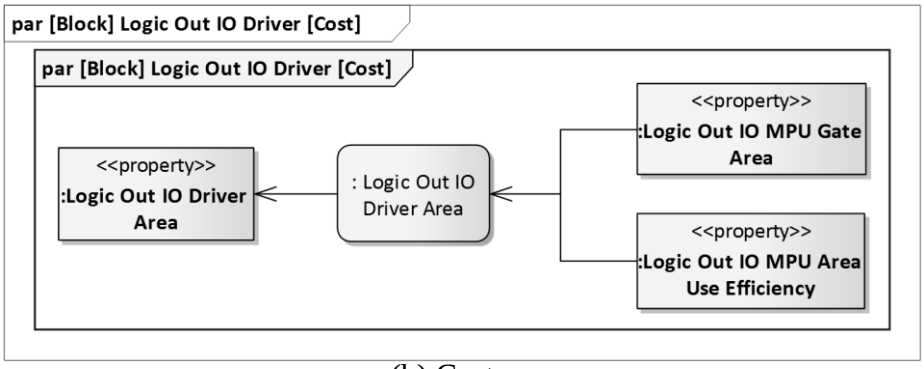


Fig. A.35 Logic out IO driver internal block diagram.



(a) Power consumption.



(b) Cost.

Fig. A.36 Logic out IO driver parametric diagram.

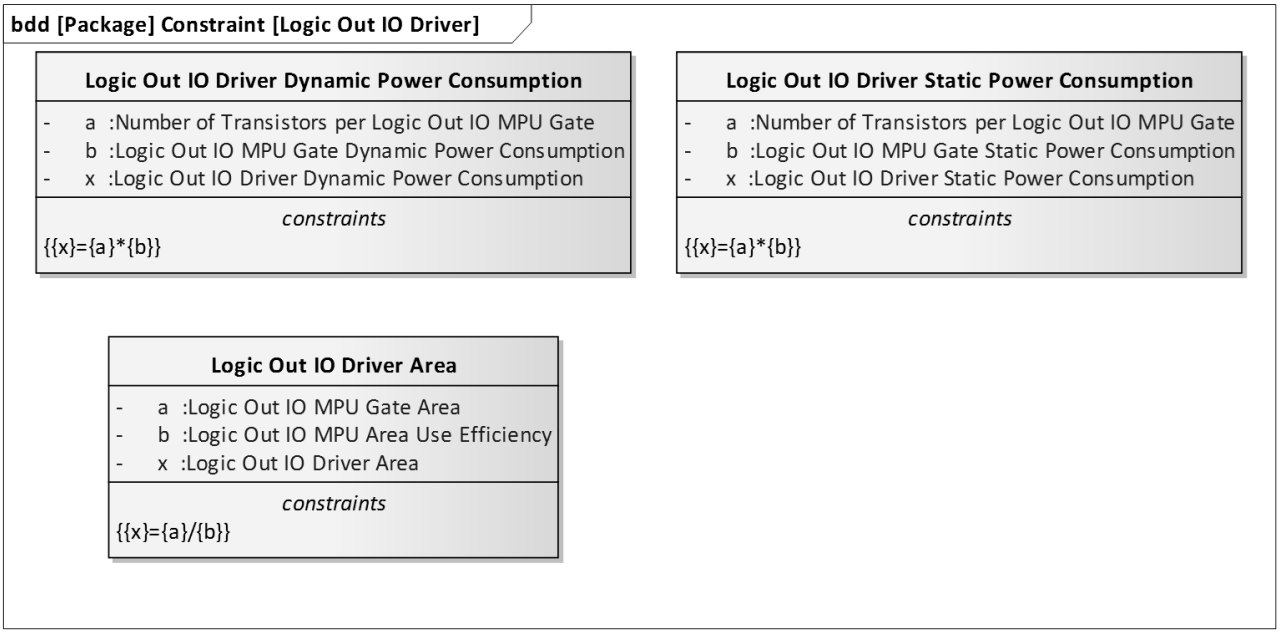


Fig. A.37 Detail of logic out IO driver constraint block.

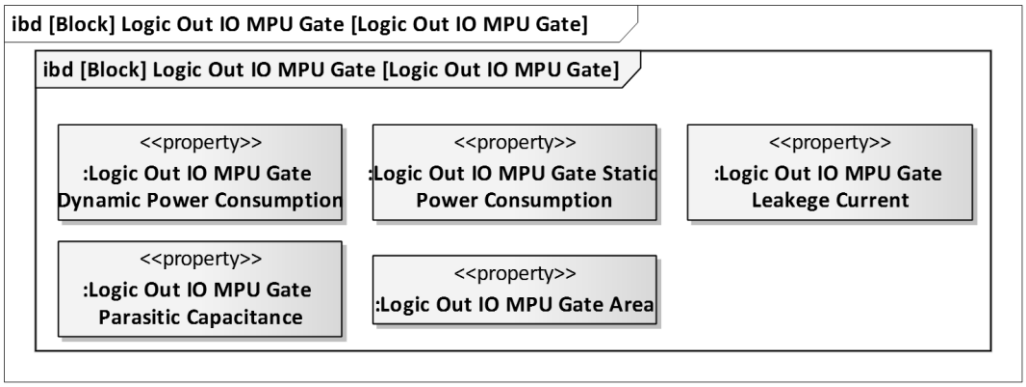
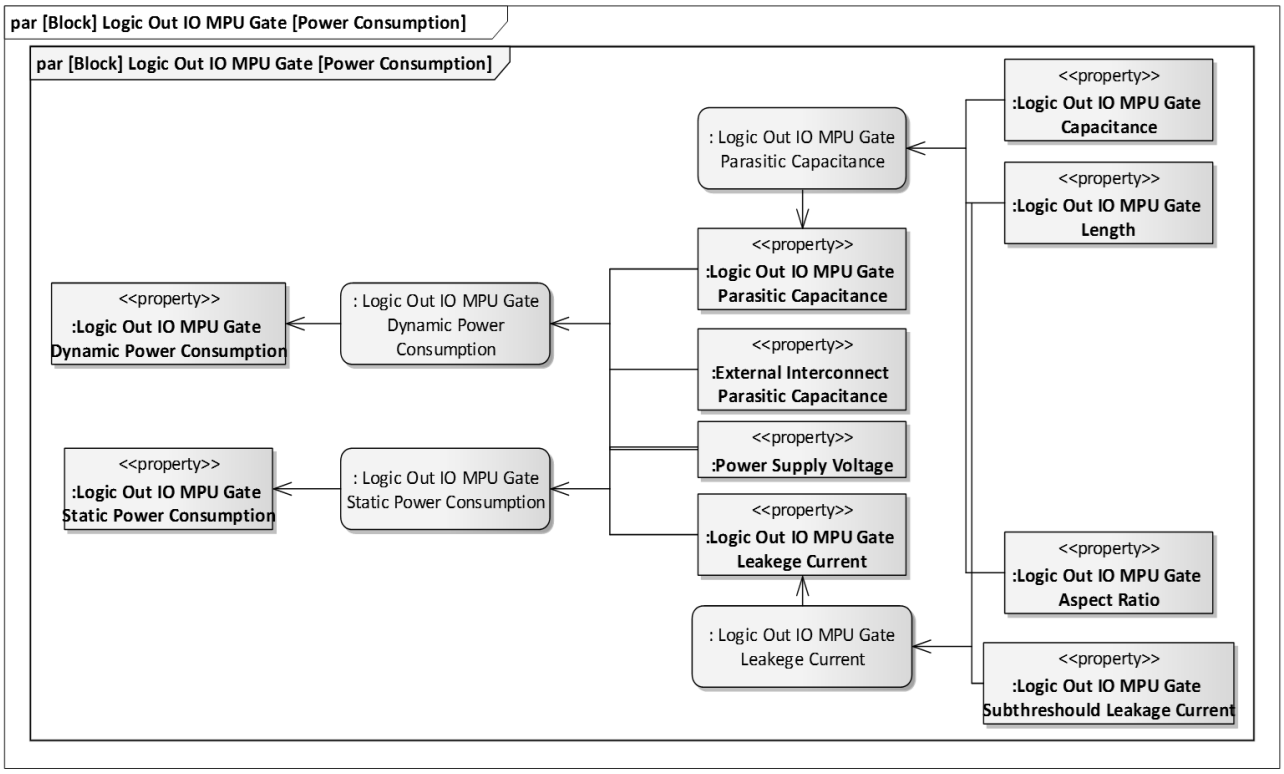
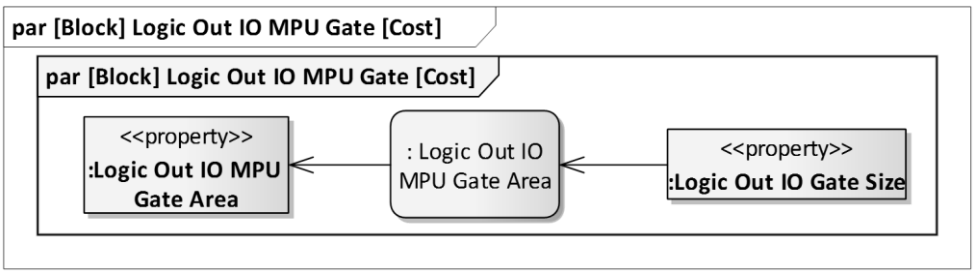


Fig. A.38 Logic out IO MPU gate internal block diagram.



(a) Power consumption.



(b) Cost.

Fig. A.39 Logic out IO MPU gate parametric diagram.

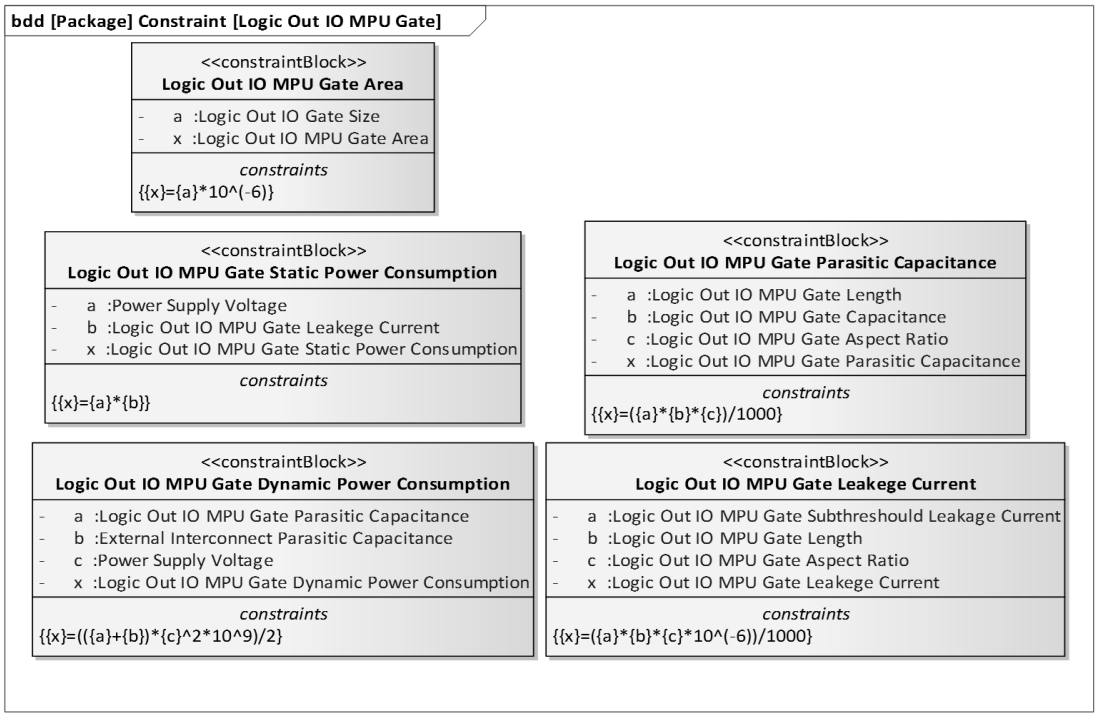


Fig. A.40 Detail of logic out IO MPU gate constraint block.

Fig. A.41 にロジックグローバル配線の内部ブロック図、Fig. A.42(a)にロジックグローバル配線の消費電力に関するパラメトリック図、Fig. A.42(b)にロジックグローバル配線の性能に関するパラメトリック図を示す。また、Fig. A.43 に制約式の詳細を示したブロック図を示す。

Fig. A.44 にメモリチップの内部ブロック図、Fig. A.45(a)にメモリチップの消費電力に関するパラメトリック図、Fig. A.45(b)にメモリチップの性能に関するパラメトリック図、Fig. A.45(c)にメモリチップのコストに関するパラメトリック図を示す。また、Fig. A. 46 に制約式の詳細を示したブロック図を示す。

Fig. A.47 に主記憶回路の内部ブロック図、Fig. A.48(a)に主記憶回路の消費電力に関するパラメトリック図、Fig. A.48(b)に主記憶回路の性能に関するパラメトリック図、Fig. A.48(c)に主記憶回路のコストに関するパラメトリック図を示す。また、Fig. A.49 に制約式の詳細を示したブロック図を示す。

Fig. A.50 にメインメモリセルの内部ブロック図、Fig. A.51(a)にメインメモリセルの消費電力に関するパラメトリック図、Fig. A.51(b)にメインメモリセルの性能に関するパラメトリック図、Fig. A.51(c)にメインメモリセルのコストに関するパラメトリック図を示す。また、Fig. A.52 に制約式の詳細を示したブロック図を示す。

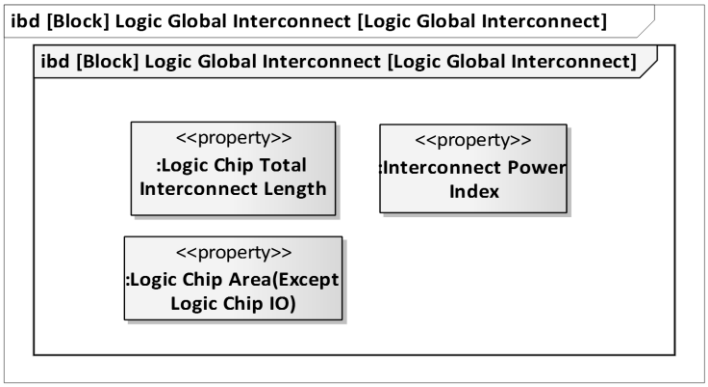
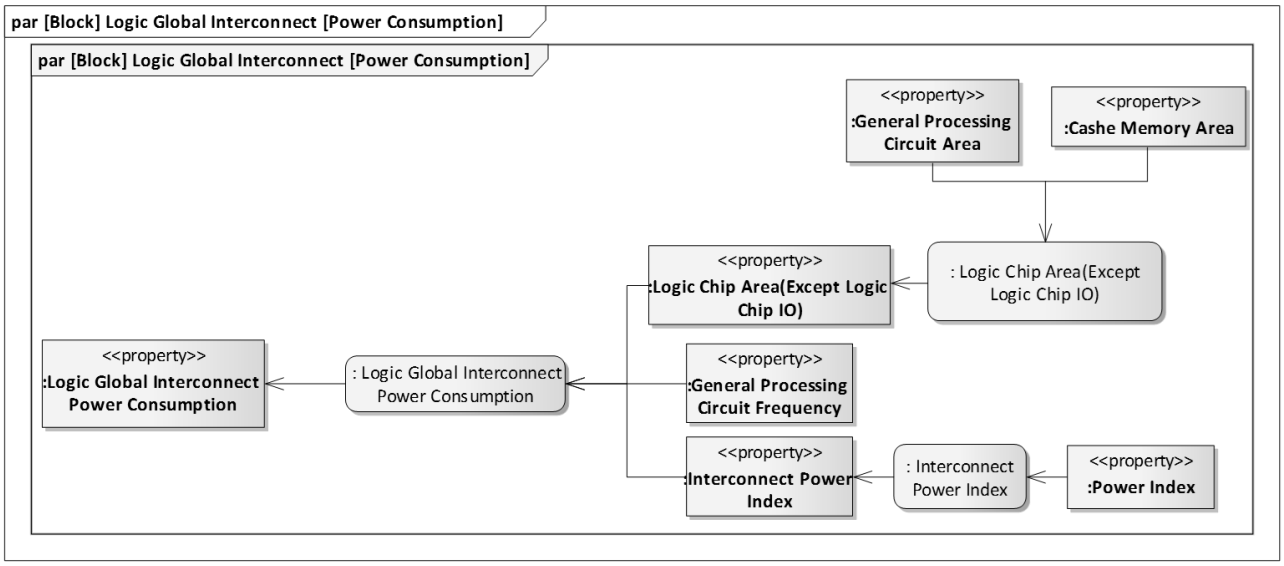
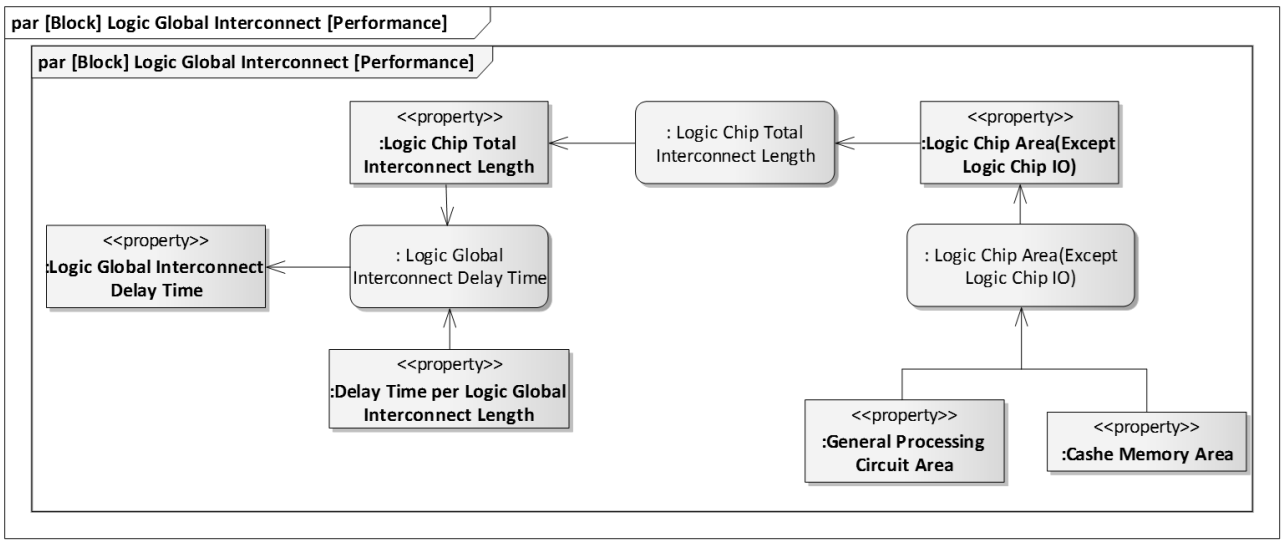


Fig. A.41 Logic global interconnect internal block diagram.



(a) Power consumption.



(b) Performance.

Fig. A.42 Logic global interconnect parametric diagram.

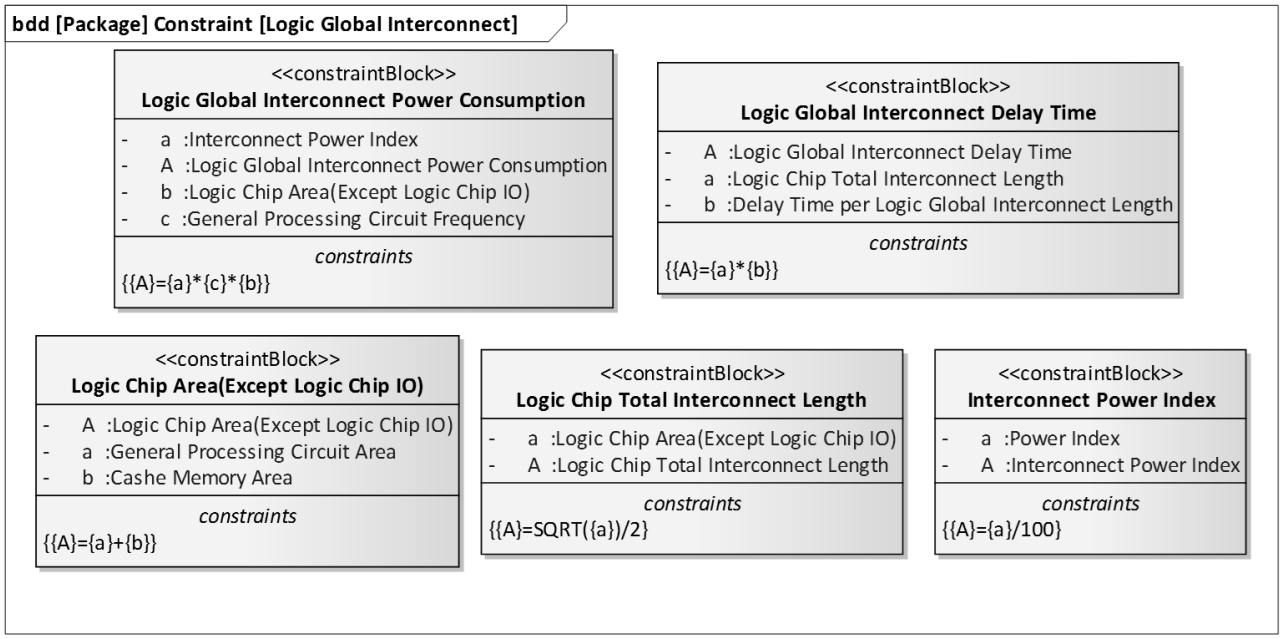


Fig. A.43 Detail of logic global interconnect constraint block.

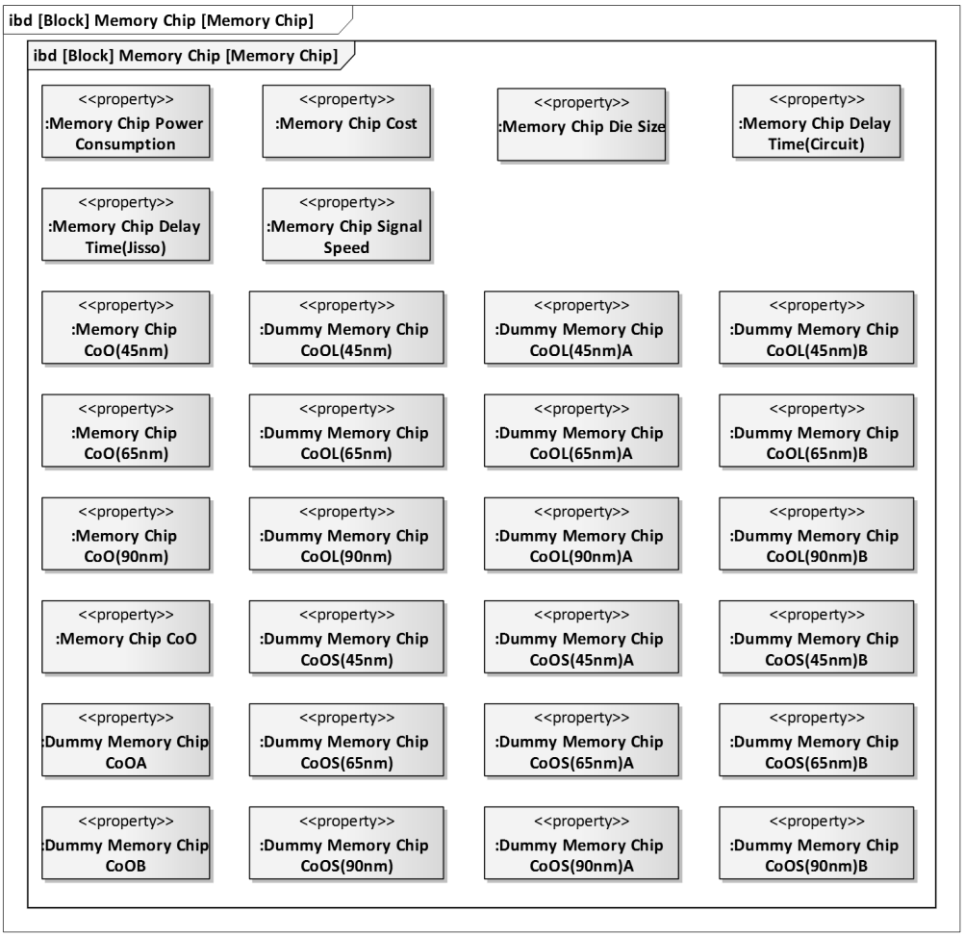
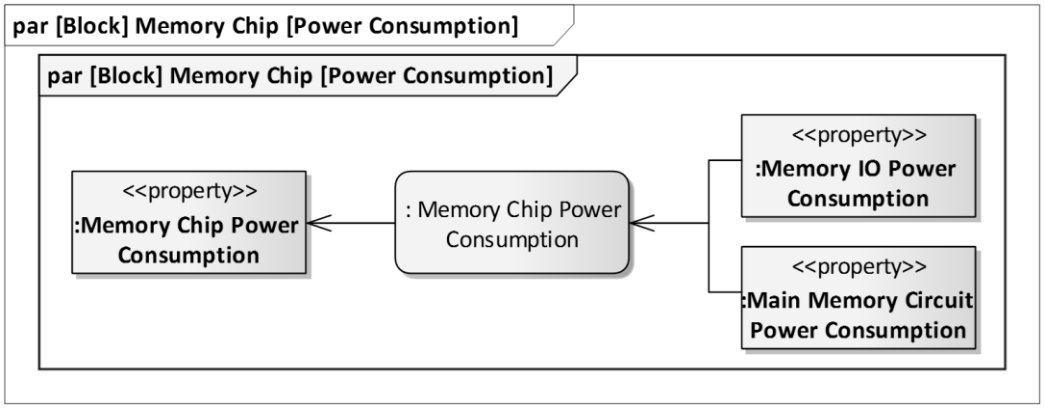
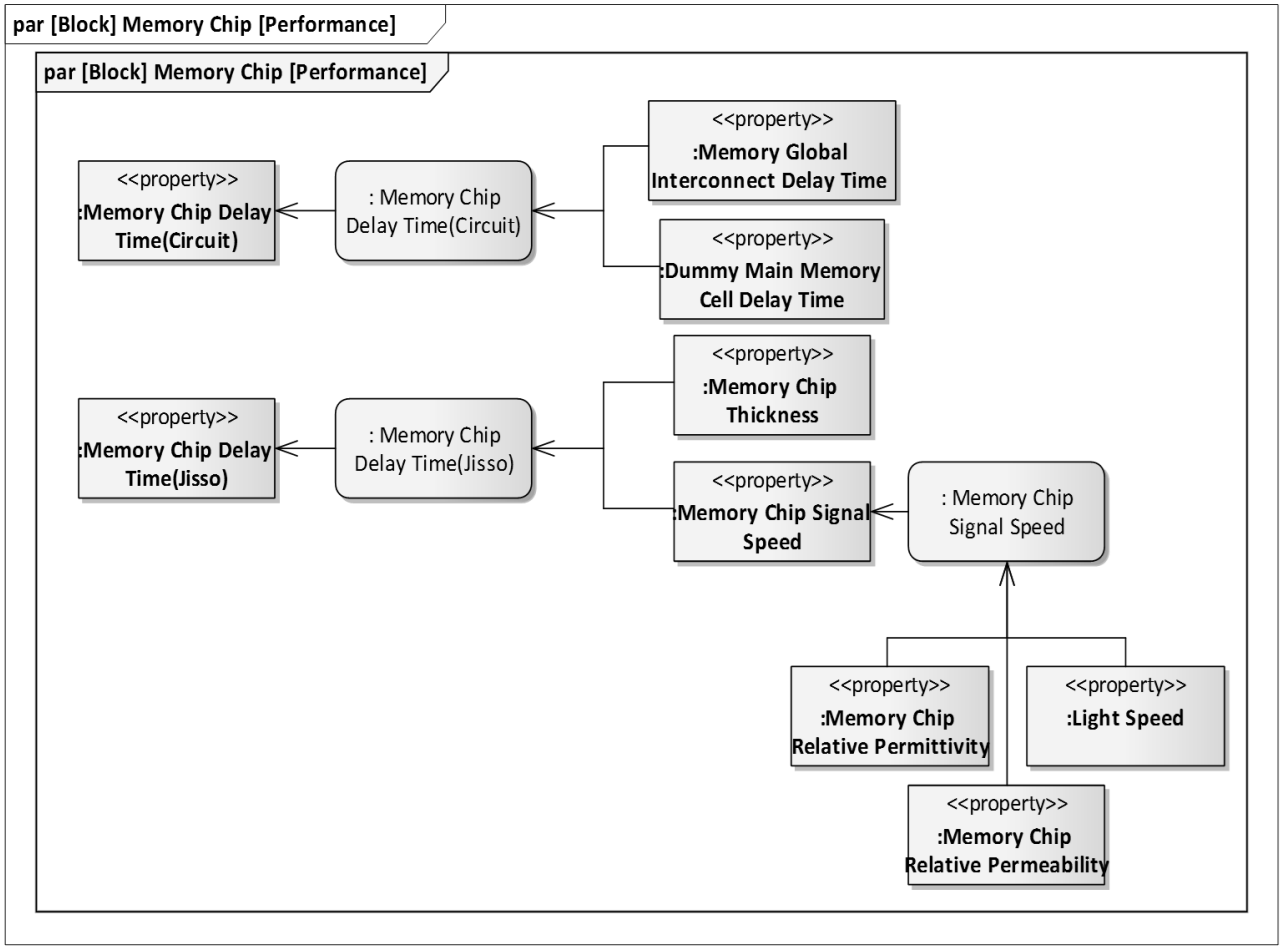


Fig. A.44 Memory chip internal block diagram.



(a) Power consumption.



(b) Performance.

Fig. A.45 Memory chip parametric diagram.

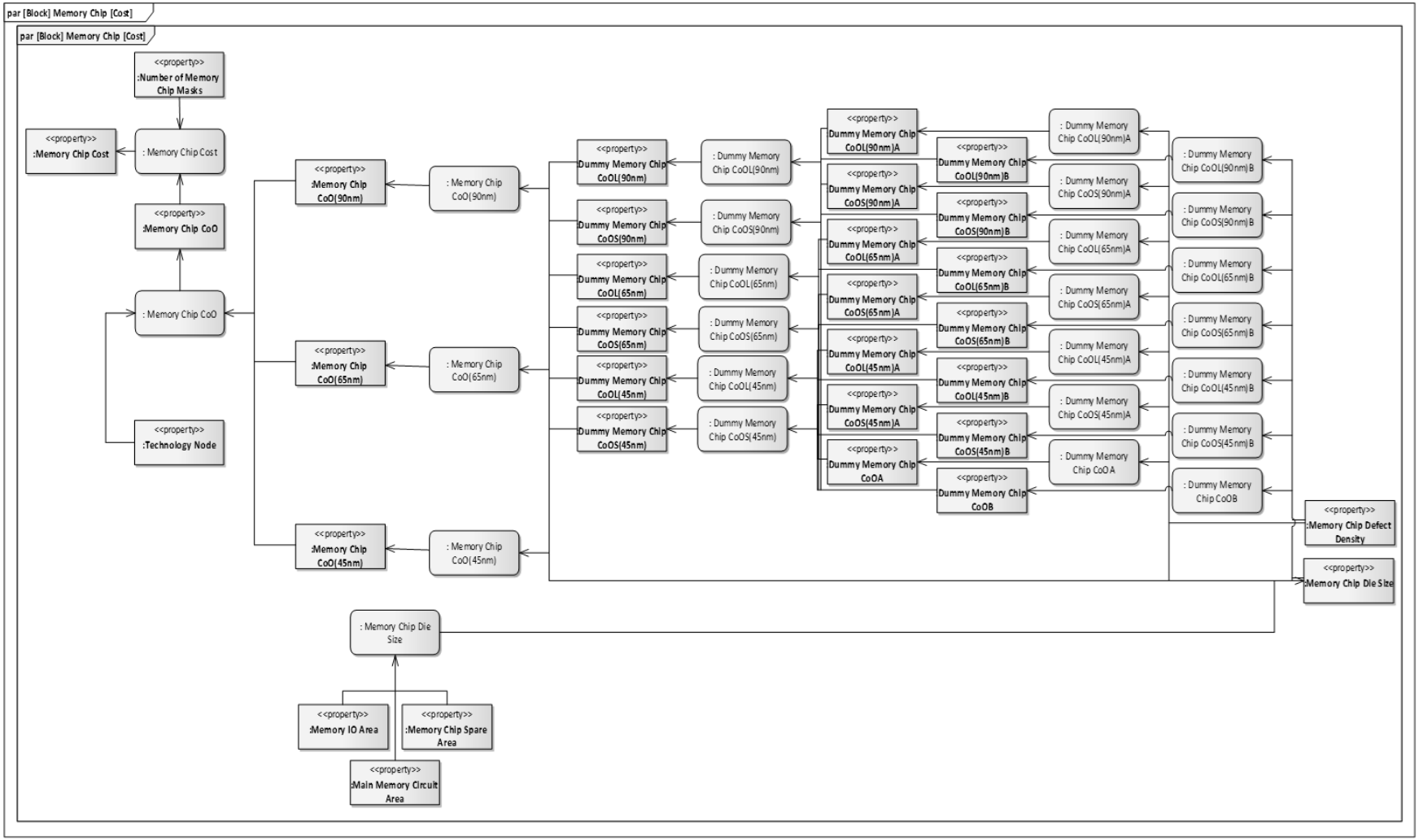


Fig. A.45 Continued.

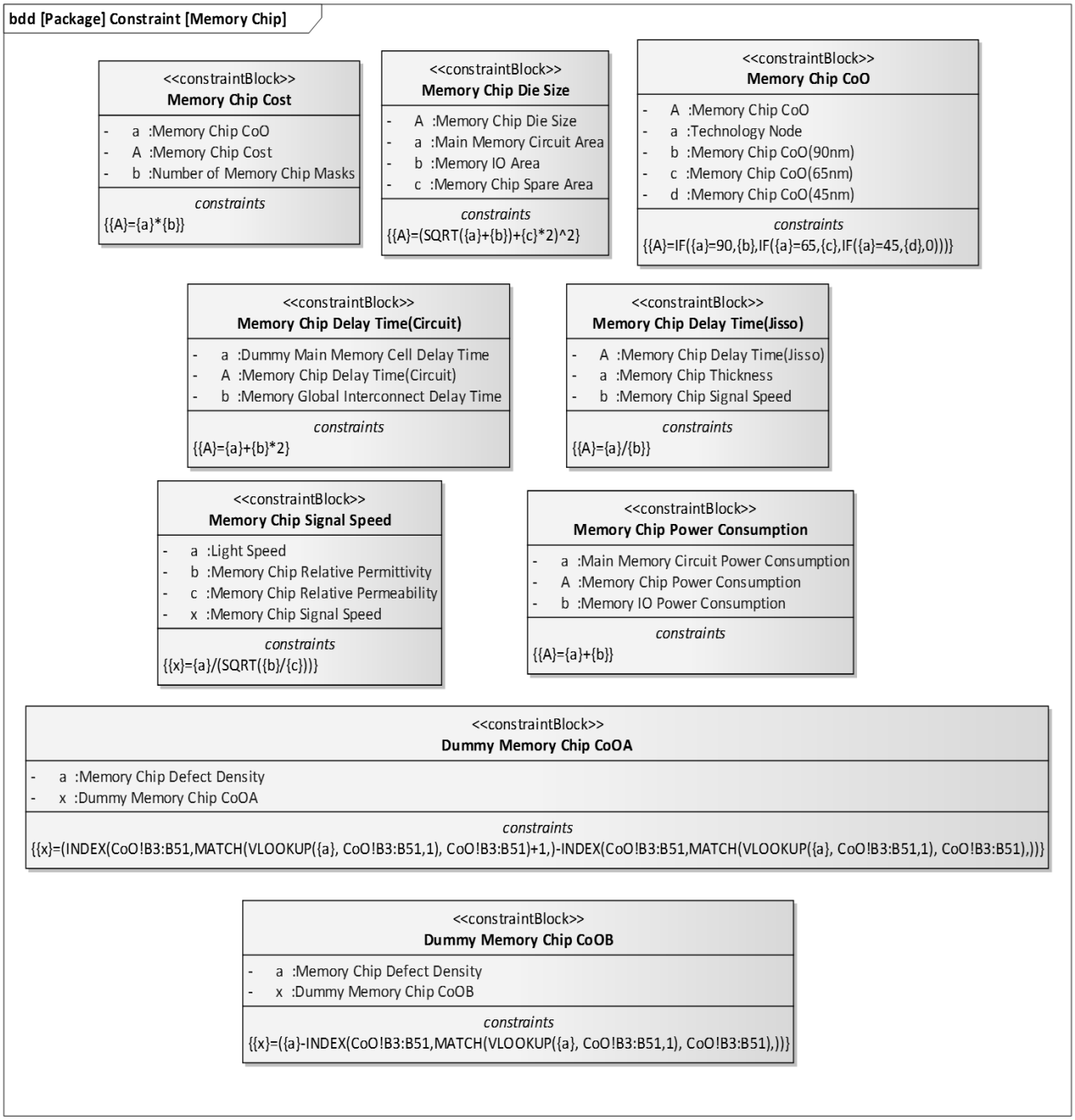


Fig. A.46 Detail of memory chip constraint block.

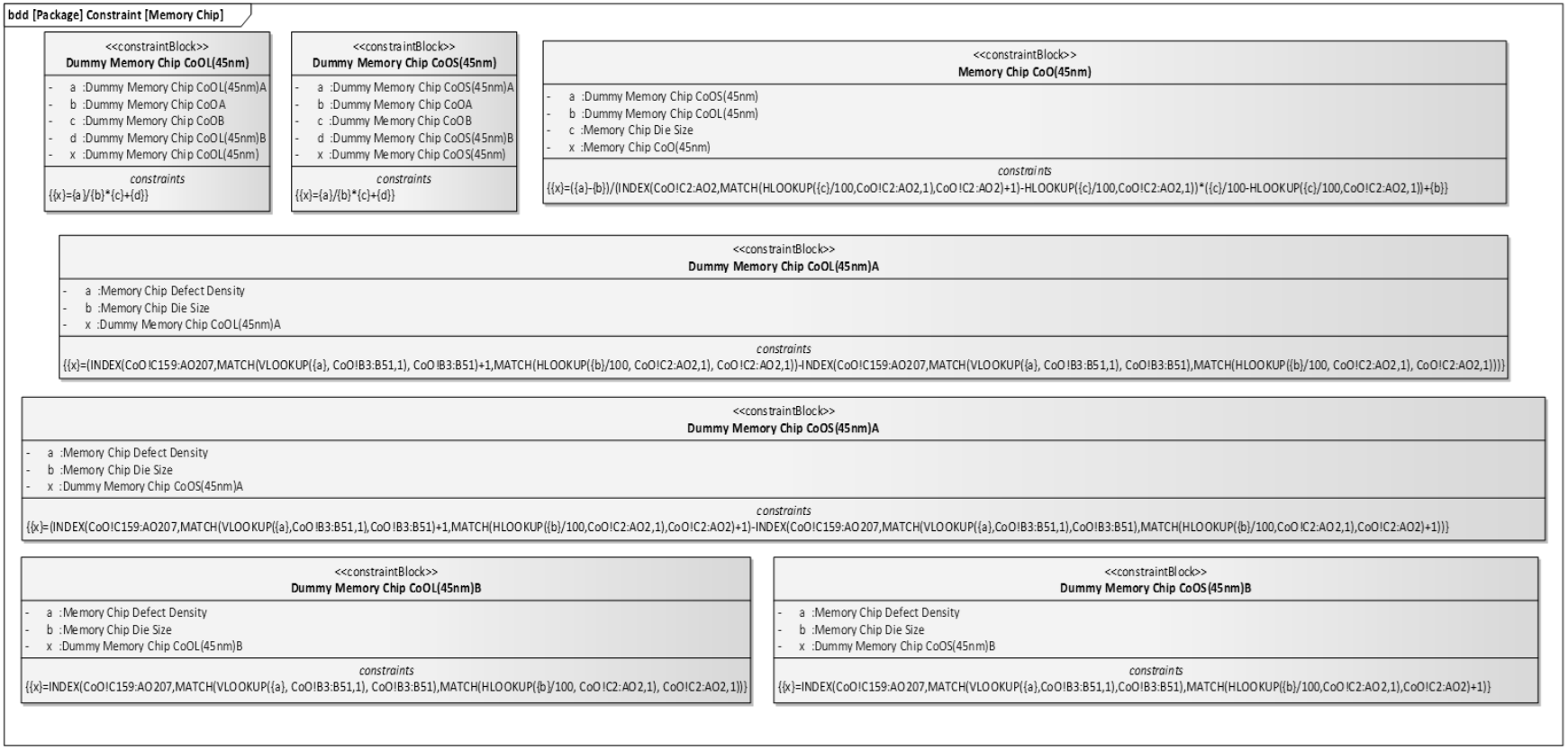


Fig. A.46 Continued.

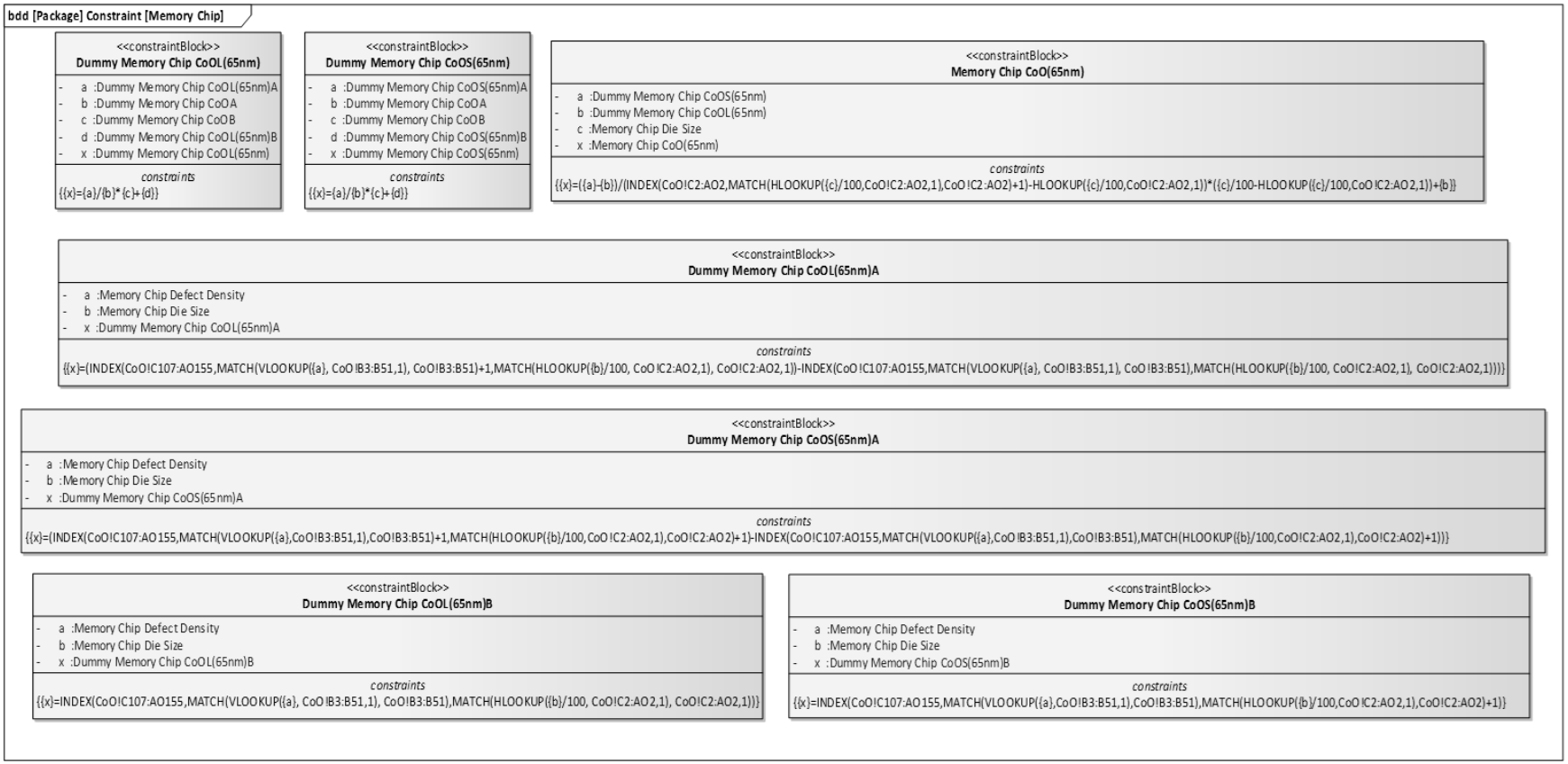


Fig. A.46 Continued.

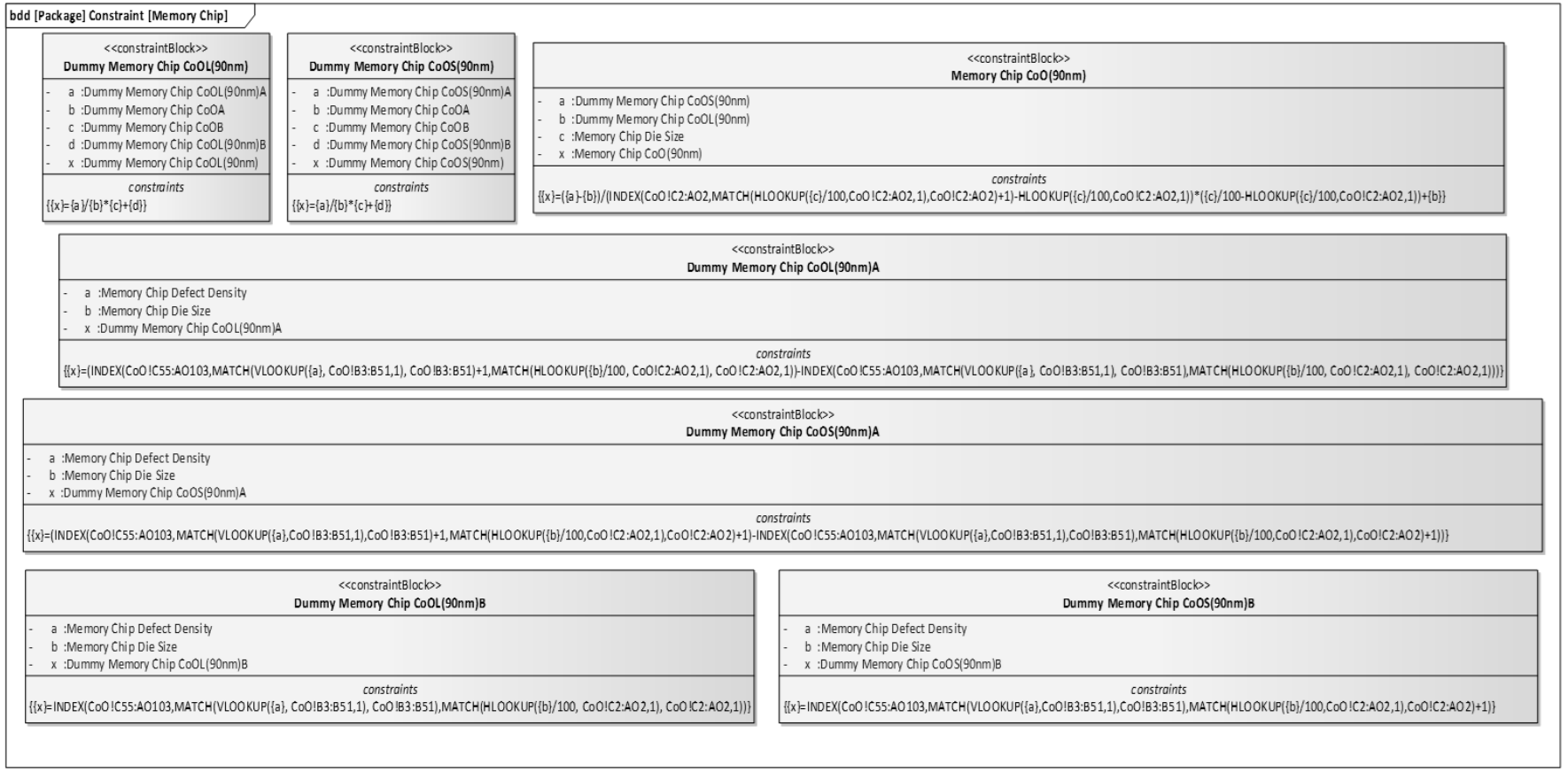


Fig. A.46 Continued.

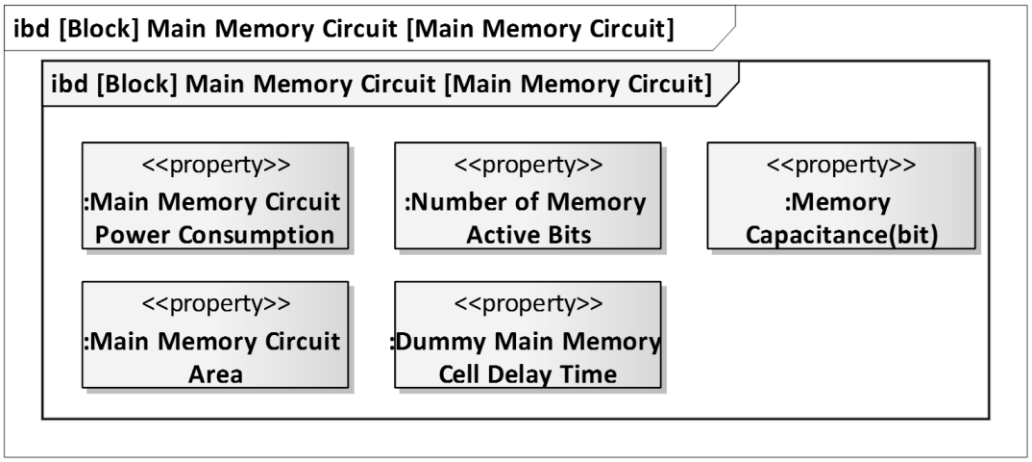
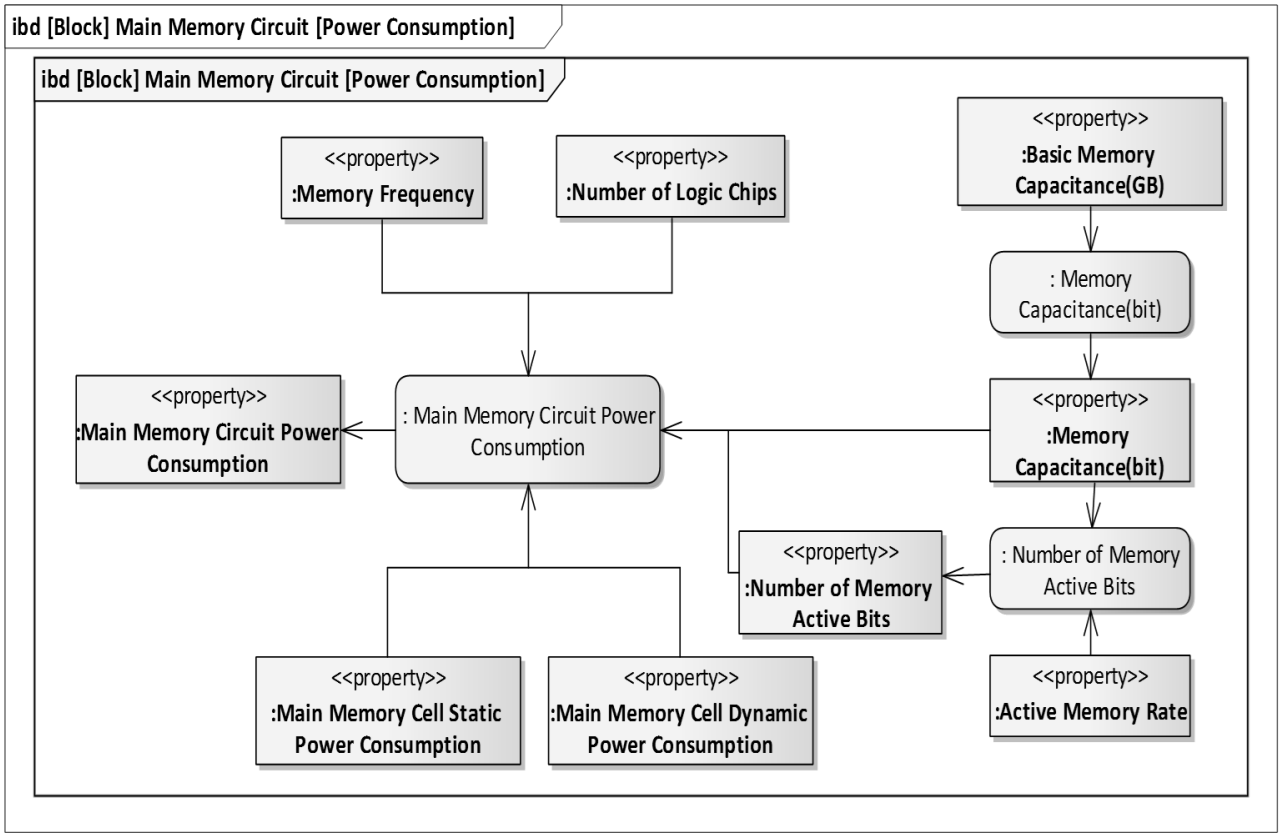
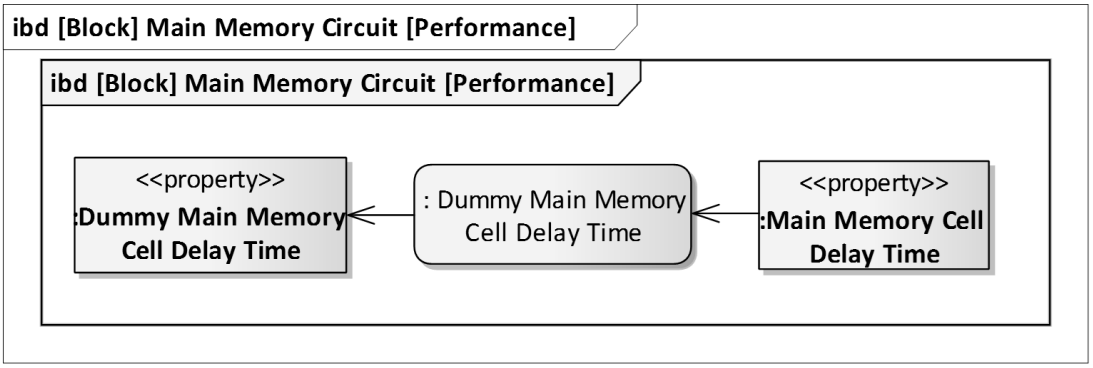


Fig. A.47 Main memory circuit internal block diagram.

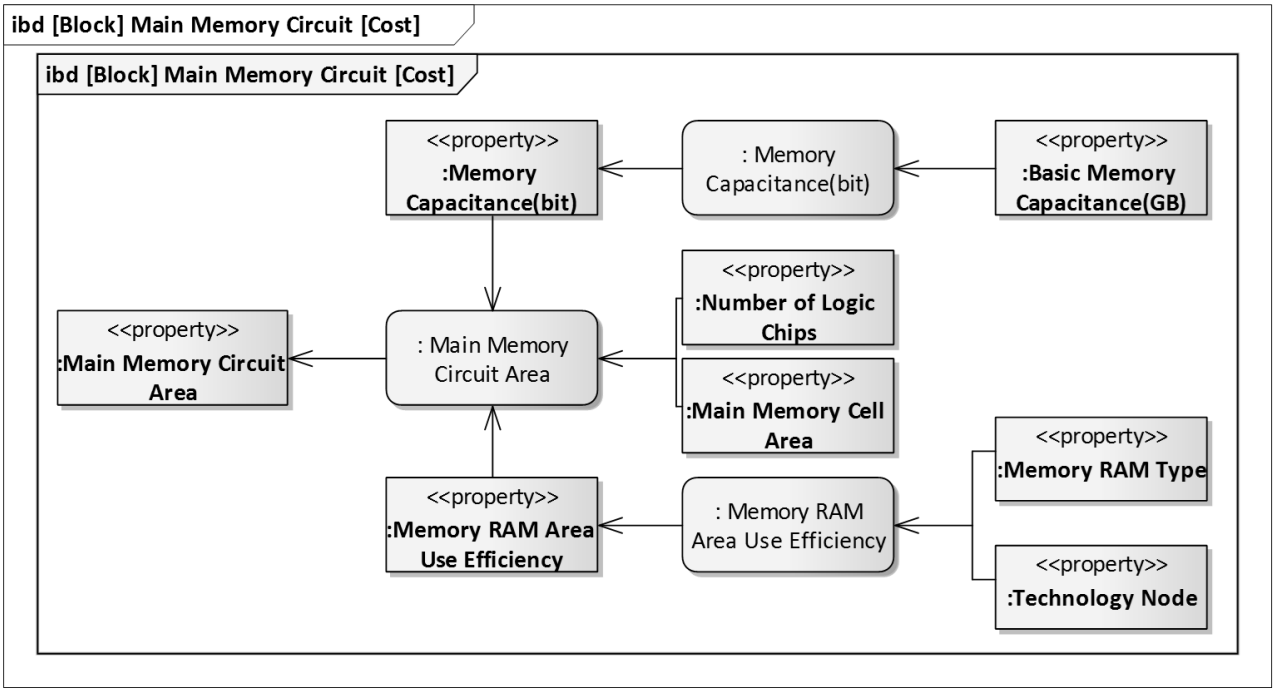


(a) Power consumption.

Fig. A.48 Main memory circuit parametric diagram.



(b) Performance.



(c) Cost.

Fig. A.48 Continued.

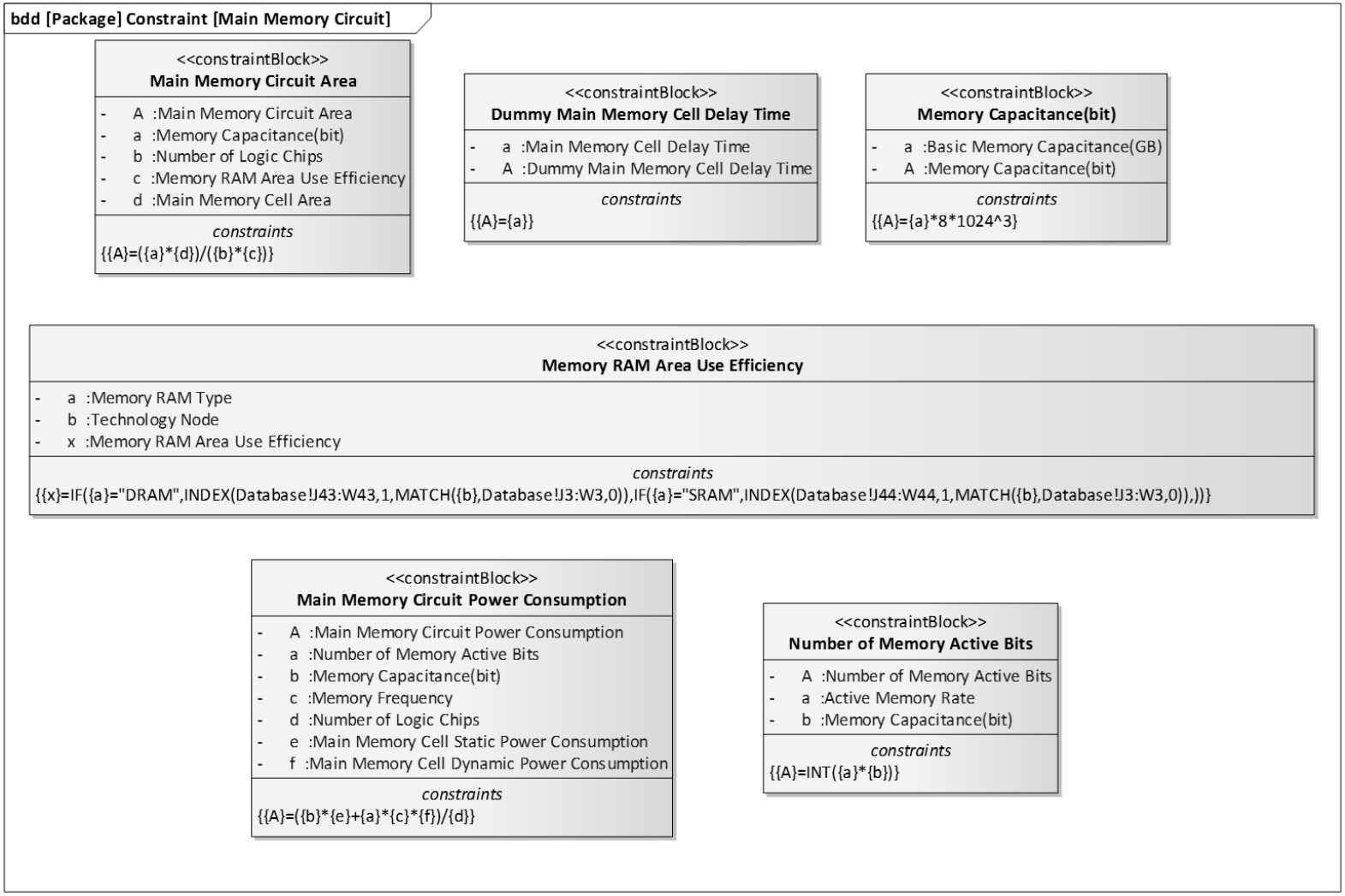


Fig. A.49 Detail of main memory circuit constraint block.

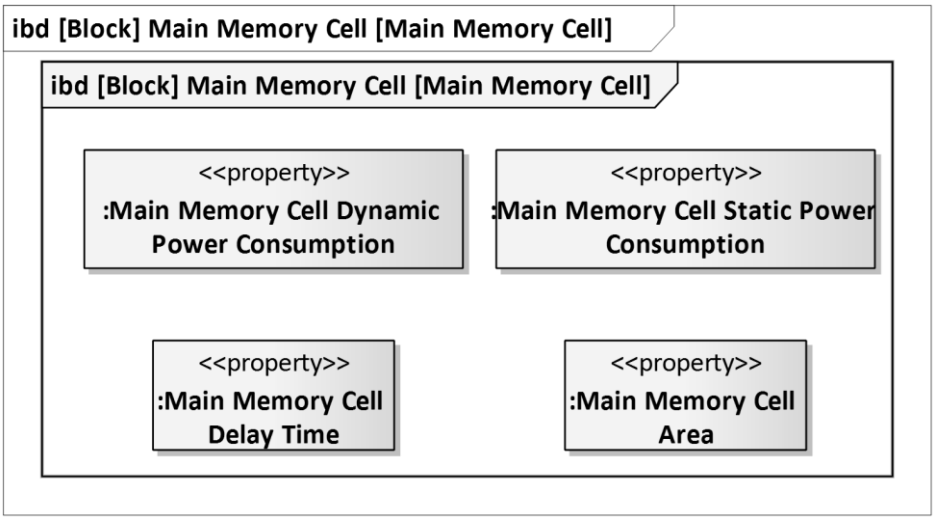
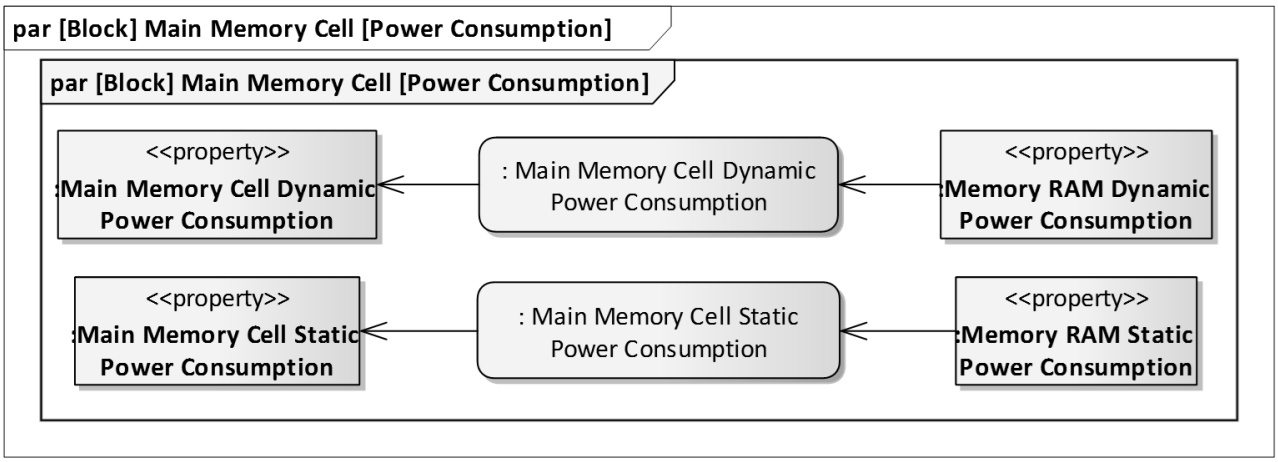
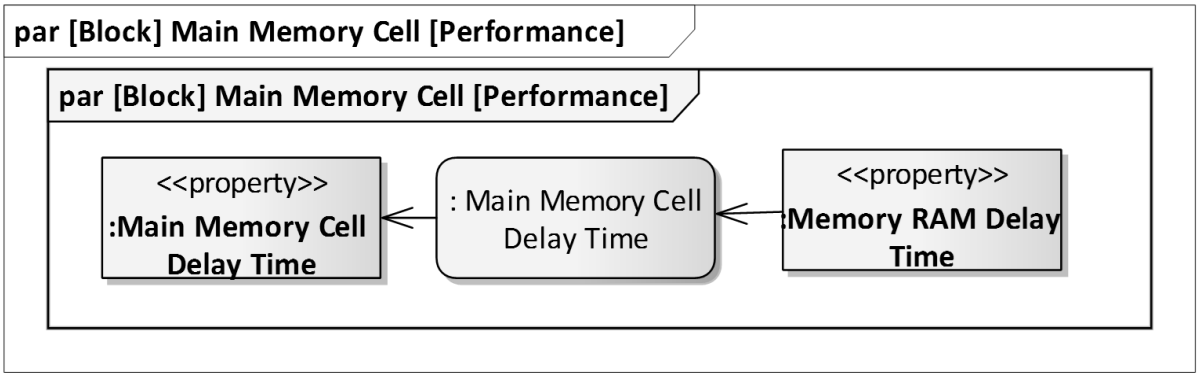


Fig. A.50 Main memory cell internal block diagram.

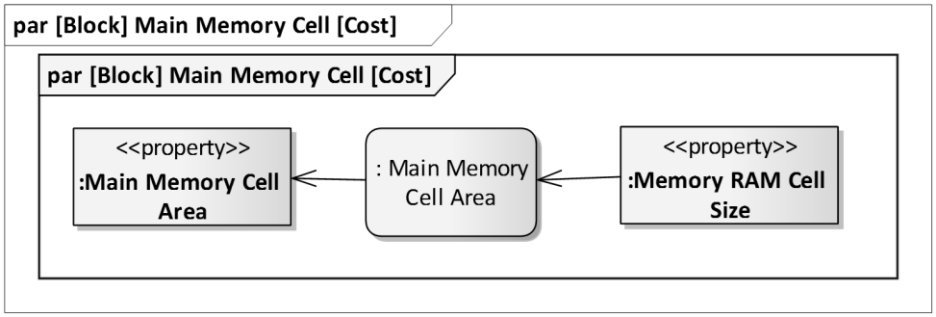


(a) Power consumption.



(b) Performance.

Fig. A.51 Main memory circuit parametric diagram.



(c) Cost.

Fig. A.51 Continued.

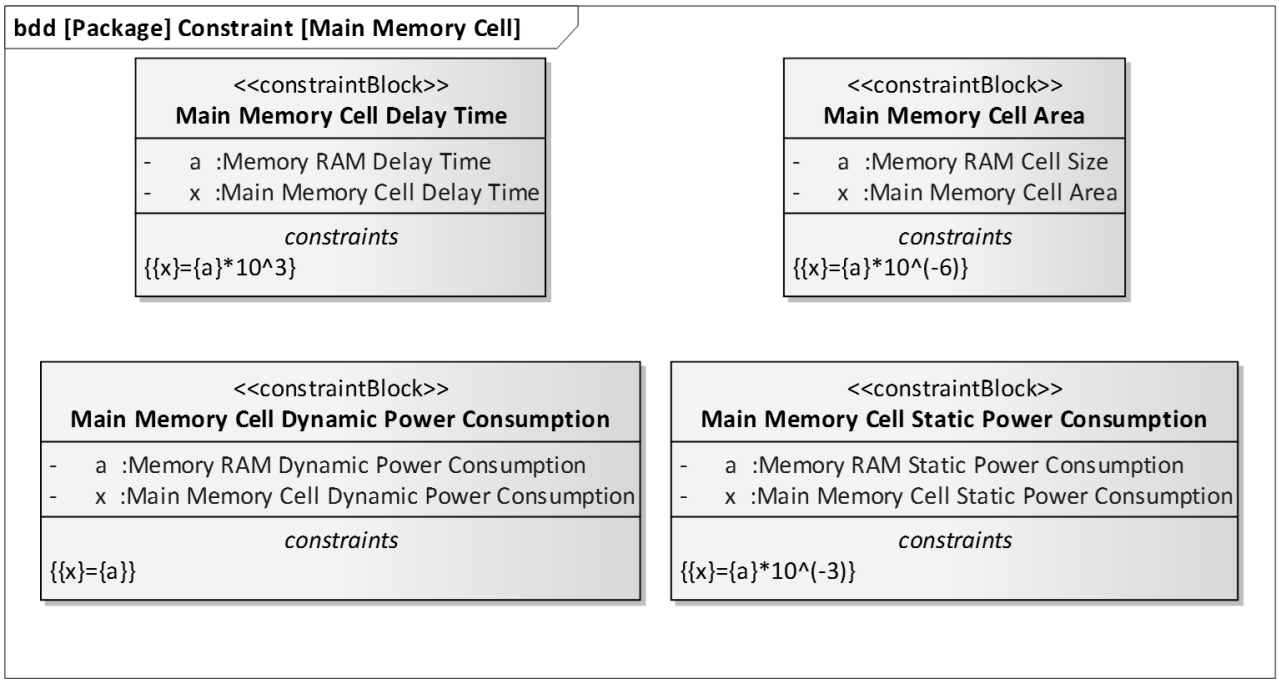


Fig. A.52 Detail of main memory circuit constraint block.

Fig. A.53 にメモリ IO の内部ブロック図、Fig. A.54(a)にメモリ IO の消費電力に関するパラメトリック図、Fig. A.54(b)にメモリ IO のコストに関するパラメトリック図を示す。また、Fig. A.55 に制約式の詳細を示したブロック図を示す。

Fig. A.56 にメモリ IO ドライバの内部ブロック図、Fig. A.57(a)にメモリ IO ドライバの消費電力に関するパラメトリック図、Fig. A.57(b)にメモリ IO ドライバのコストに関するパラメトリック図を示す。また、Fig. A.58 に制約式の詳細を示したブロック図を示す。

Fig. A.59 にメモリ IOMPU ゲートの内部ブロック図、Fig. A.60(a)にメモリ IOMPU ゲートの消費電力に関するパラメトリック図、Fig. A.60(b)にメモリ IOMPU ゲートのコストに関するパラメトリック図を示す。また、Fig. A.61 に制約式の詳細を示したブロック図を示す。

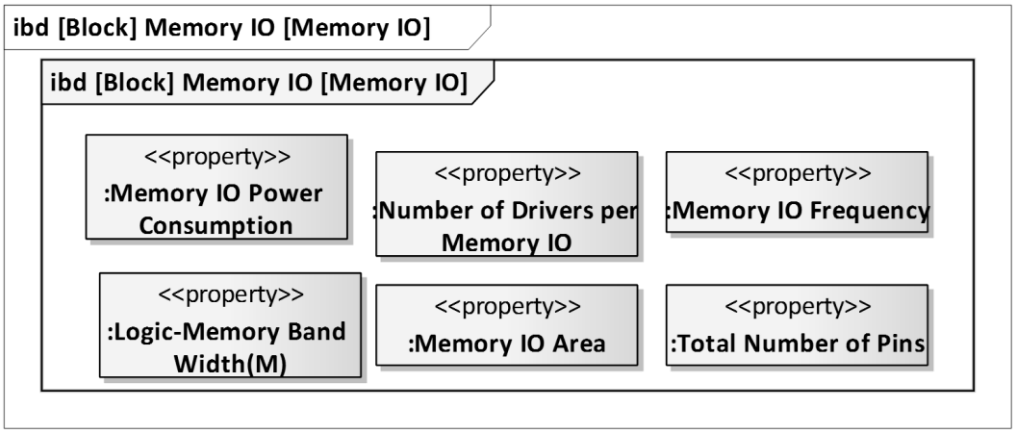
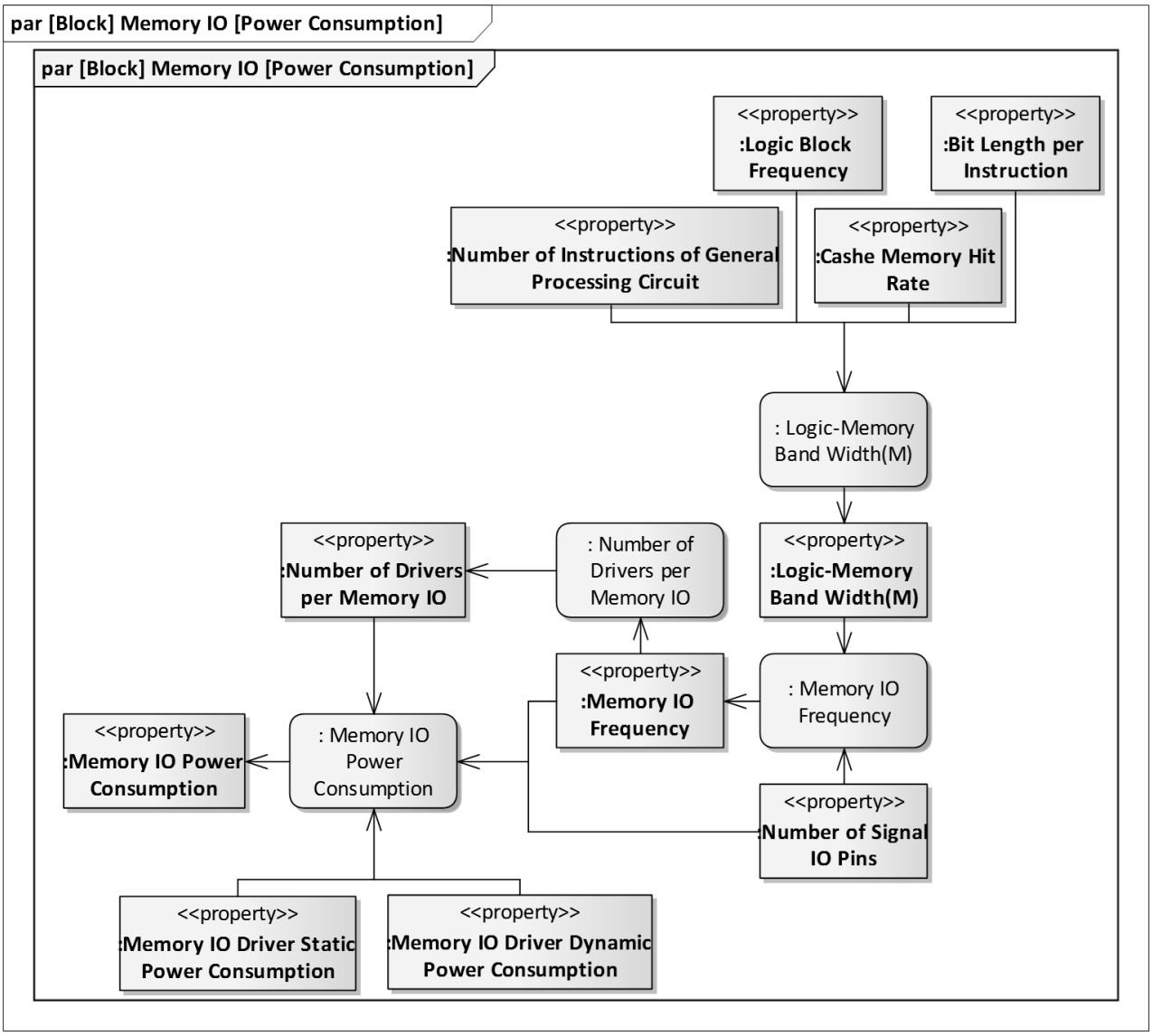
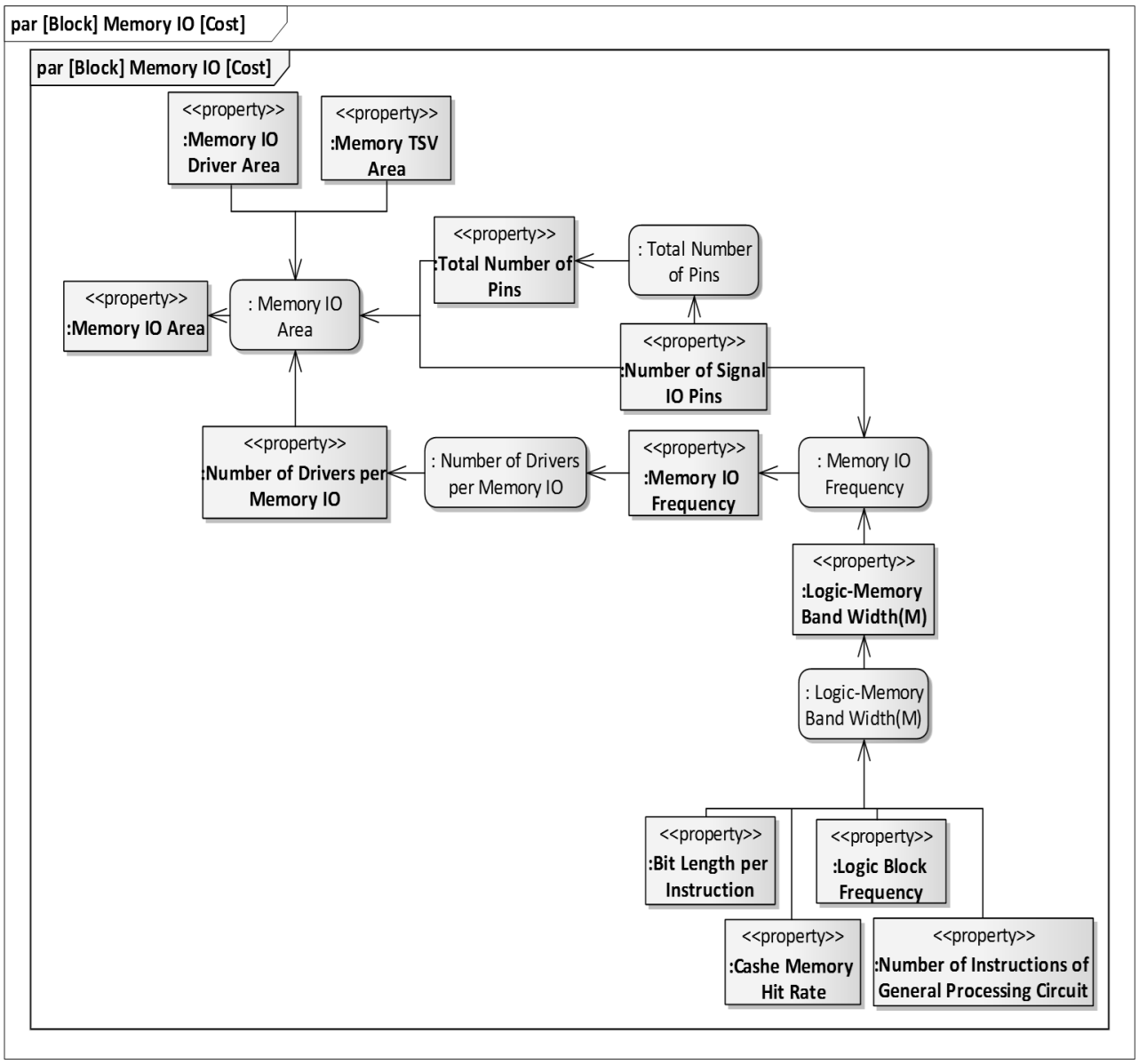


Fig. A.53 Memory IO internal block diagram.



(a) Power consumption.

Fig. A.54 Memory IO parametric diagram.



(b) Cost.

Fig. A.54 Continued.

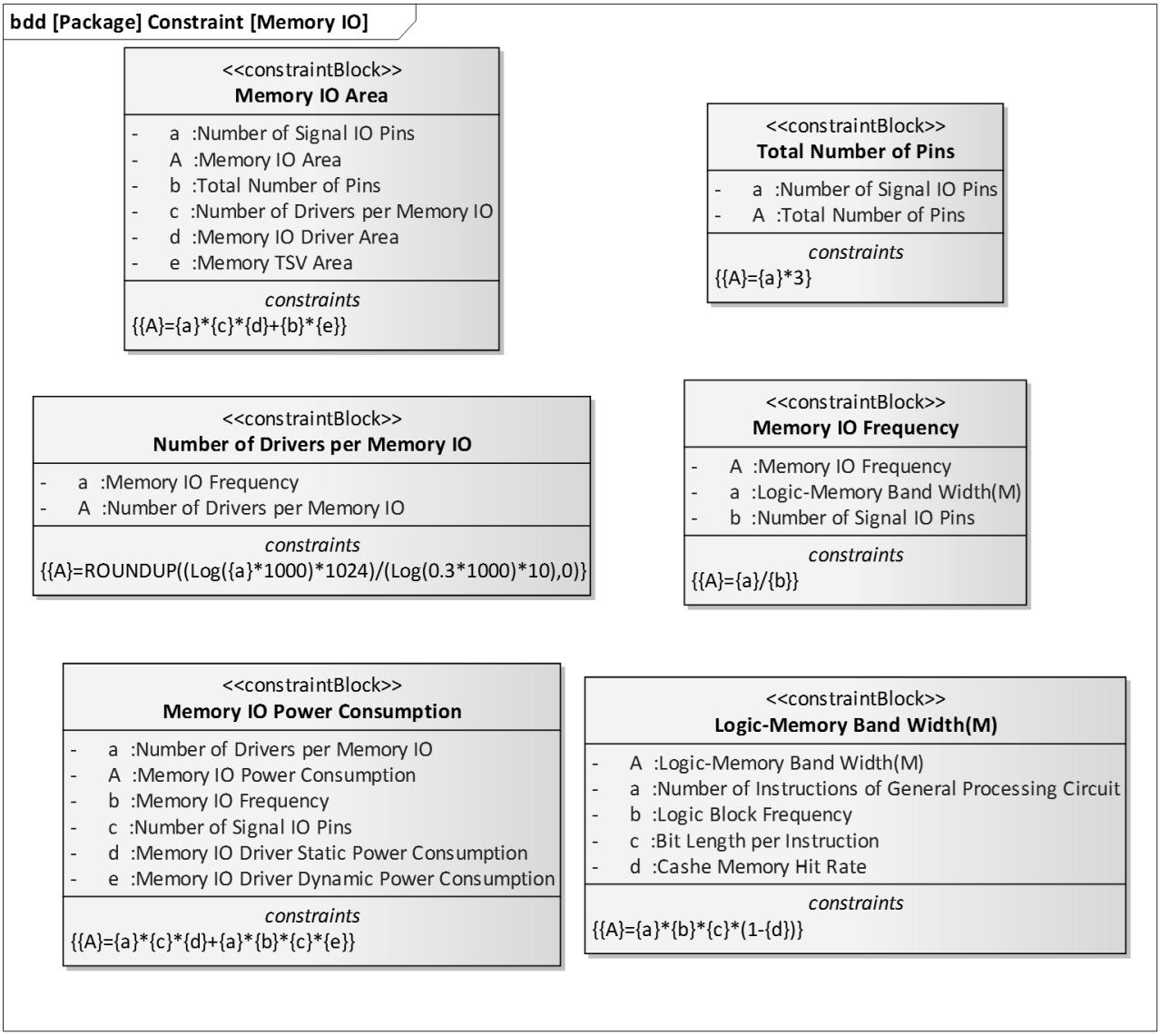


Fig. A.55 Detail of memory IO constraint block.

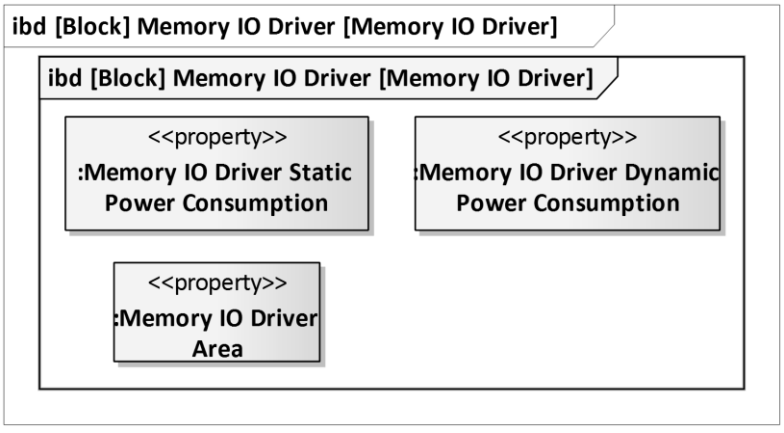
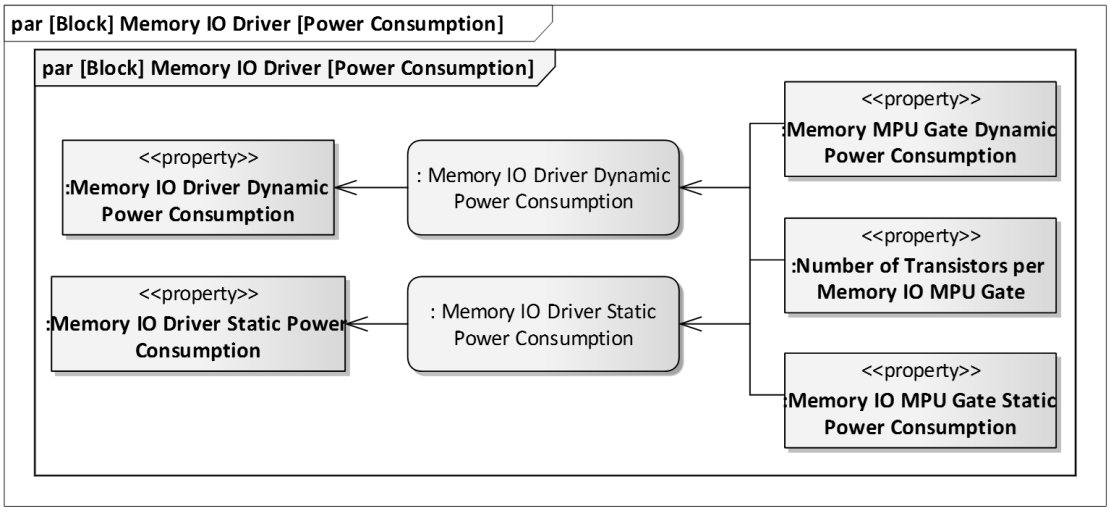
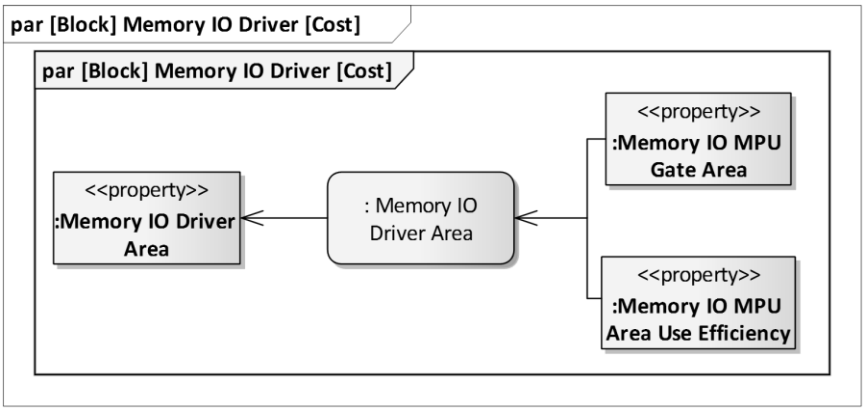


Fig. A.56 Memory IO driver internal block diagram.



(a) Power consumption.



(b) Cost.

Fig. A.57 Memory IO driver parametric diagram.

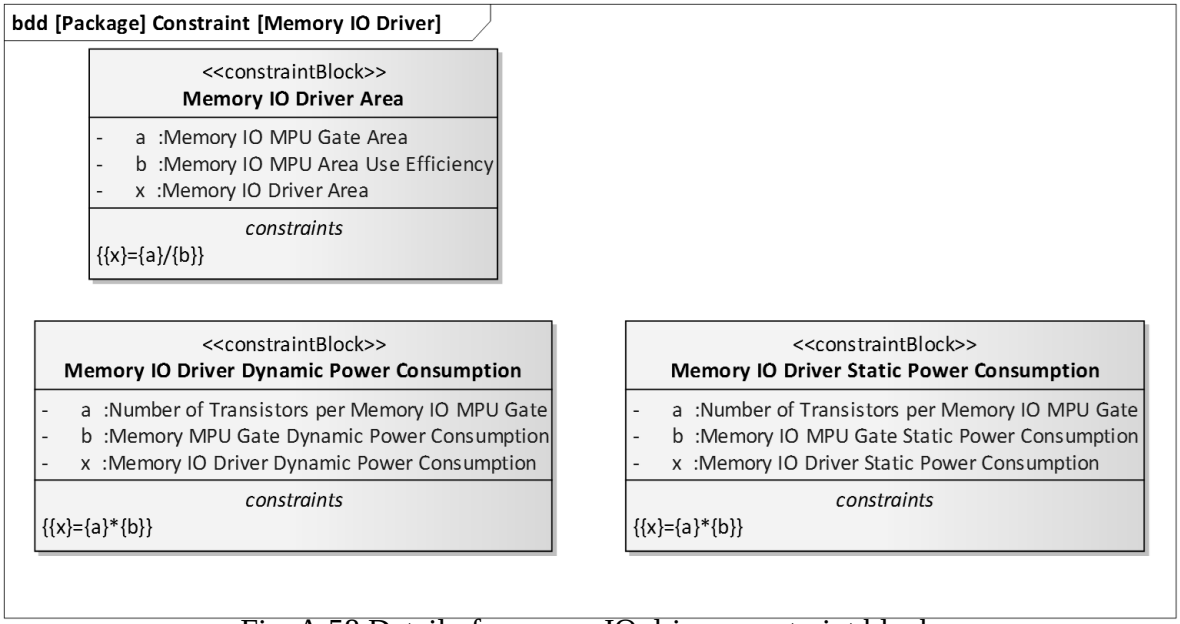


Fig. A.58 Detail of memory IO driver constraint block.

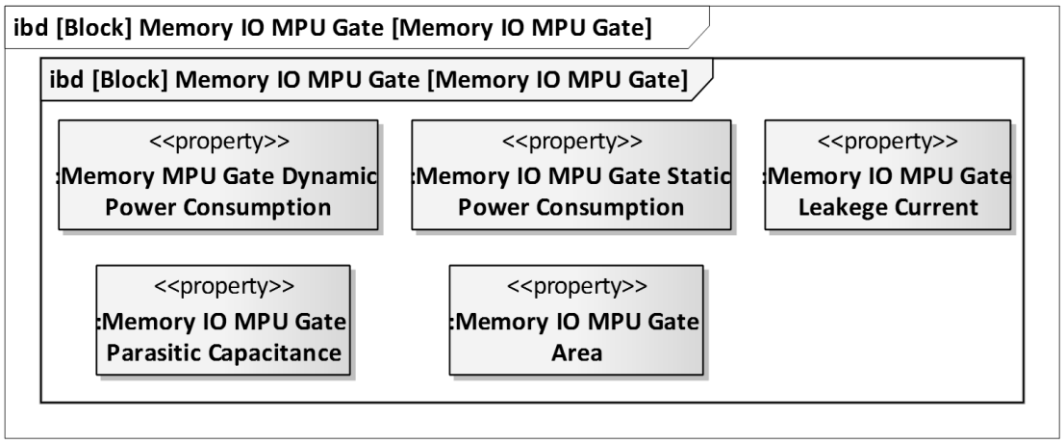
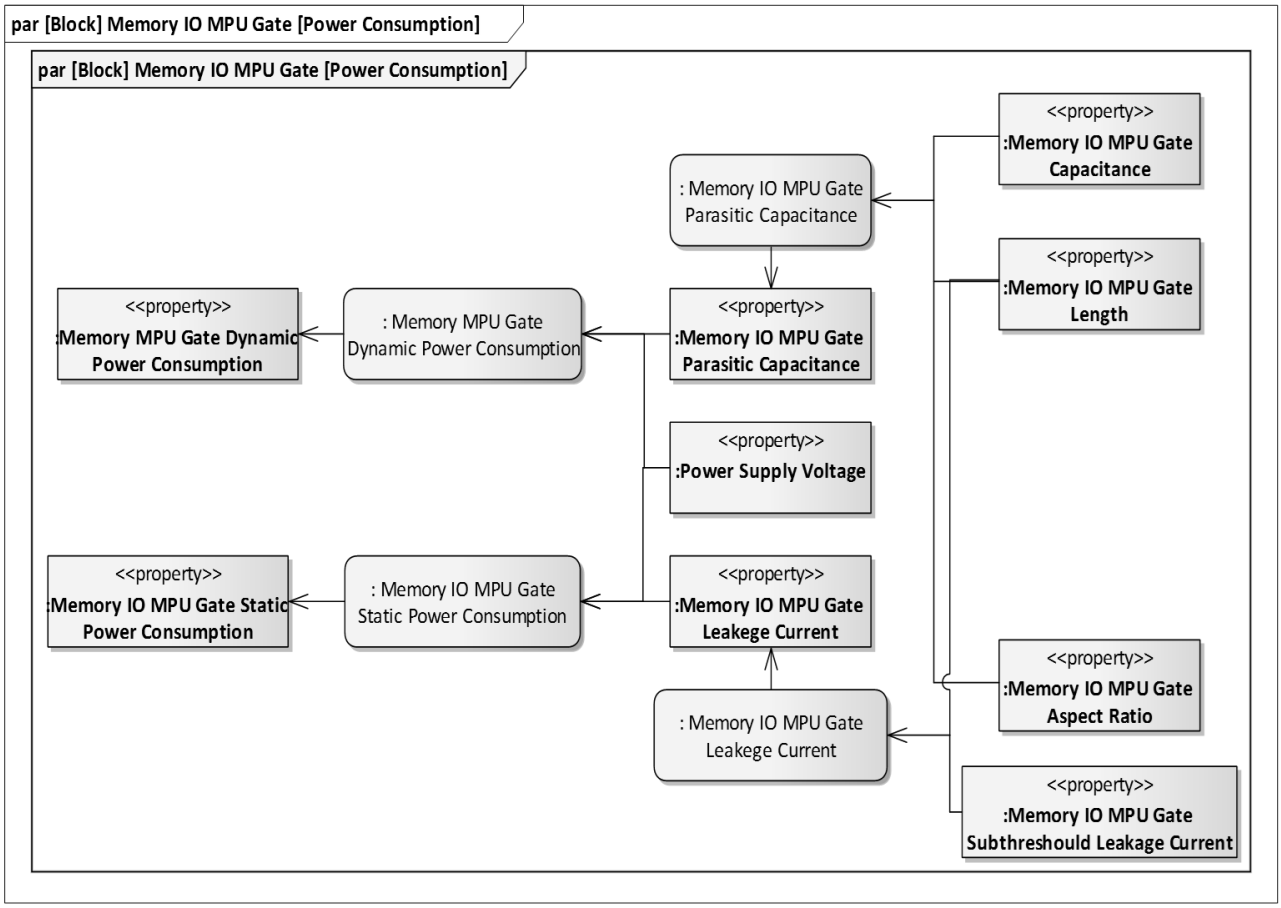
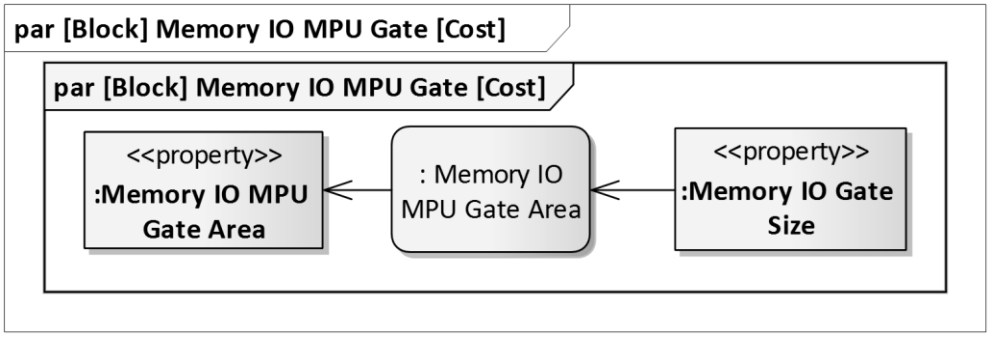


Fig. A.59 Memory IO MPU gate internal block diagram.



(a) Power consumption.

Fig. A.60 Memory IO MPU gate parametric diagram.



(b) Cost.

Fig. A.60 Memory IO MPU gate parametric diagram.

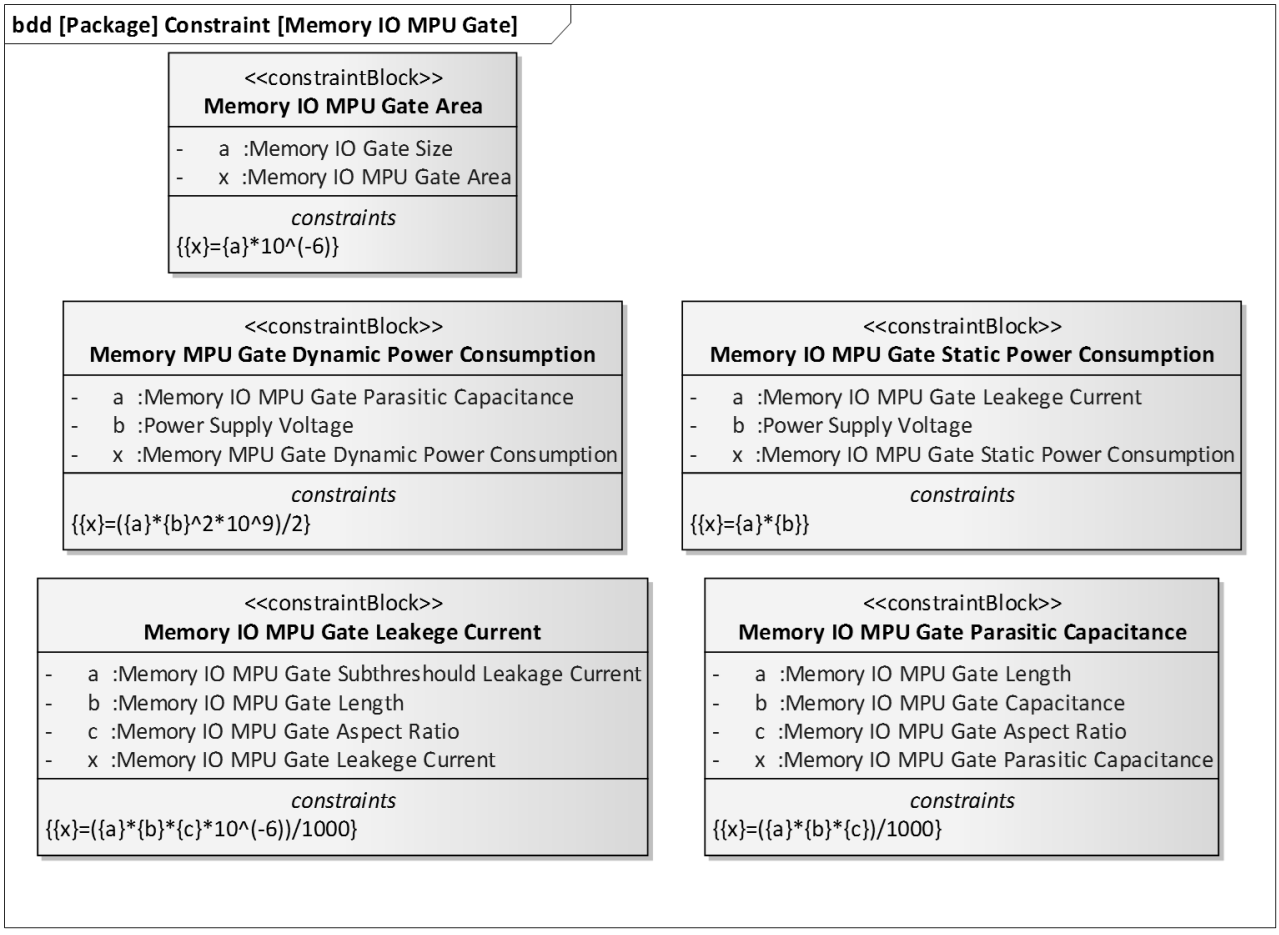


Fig. A.61 Detail of memory IO MPU gate constraint block.

Fig. A.62 にメモリ TSV の内部ブロック図、Fig. A.63 にメモリ TSV のコストに関するパラメトリック図を示す。また、Fig. A.64 に制約式の詳細を示したブロック図を示す。

Fig. A.65 にメモリグローバル配線の内部ブロック図、Fig. A.66 にメモリグローバル配線の性能に関するパラメトリック図を示す。また、Fig. A.67 に制約式の詳細を示したブロッ

ク図を示す。

Fig. A.68 に実装部の内部ブロック図、Fig. A.69(a)に実装部の性能に関するパラメトリック図、Fig. A.69(b)に実装部のコストに関するパラメトリック図を示す。また、Fig. A.70 に制約式の詳細を示したブロック図を示す。

Fig. A.71 に基板の内部ブロック図、Fig. A.72 に基板の性能に関するパラメトリック図を示す。また、Fig. A.73 に制約式の詳細を示したブロック図を示す。

Fig. A.74 にバンプ部の内部ブロック図、Fig. A.75(a)にバンプ部の性能に関するパラメトリック図、Fig. A.75(b)に実装部のコストに関するパラメトリック図を示す。また、Fig. A.76 に制約式の詳細を示したブロック図を示す。

Fig. A.77 にロジック接続バンプの内部ブロック図、Fig. A.78 にロジック接続バンプの性能に関するパラメトリック図を示す。また、Fig. A.79 に制約式の詳細を示したブロック図を示す。

Fig. A.80 にメモリ接続バンプの内部ブロック図、Fig. A.81 にメモリ接続バンプの性能に関するパラメトリック図を示す。また、Fig. A.82 に制約式の詳細を示したブロック図を示す。

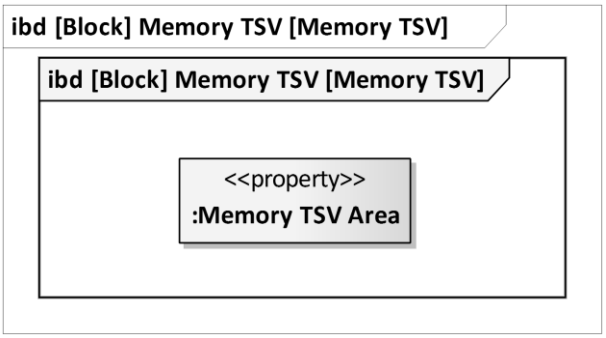


Fig. A.62 Memory TSV internal block diagram.

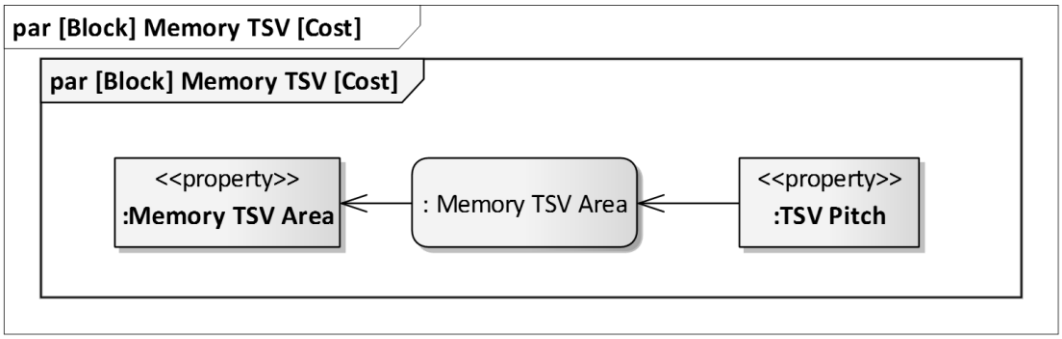


Fig. A.63 Memory TSV cost parametric diagram.

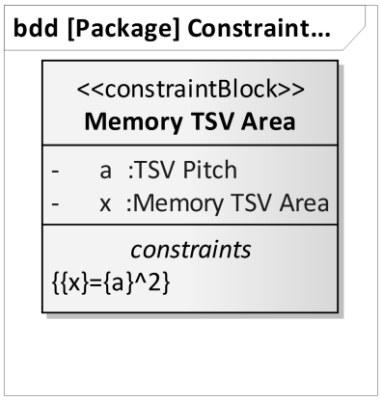


Fig. A.64 Detail of memory TSV constraint block.

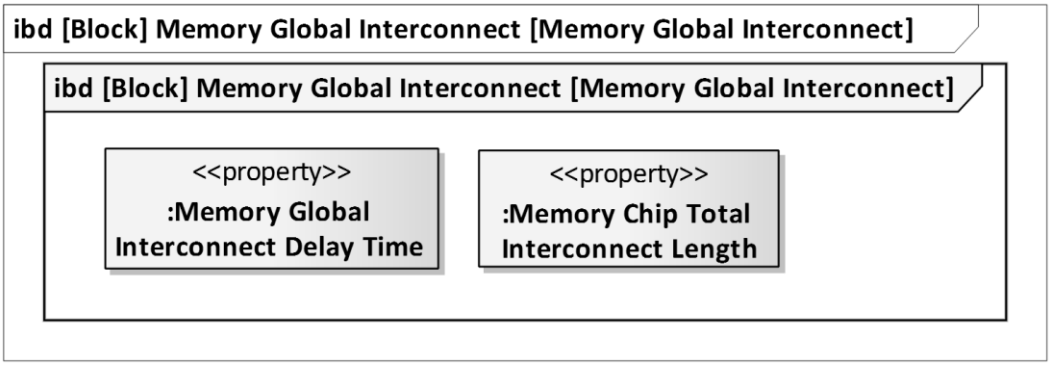


Fig. A.65 Memory global interconnect internal block diagram.

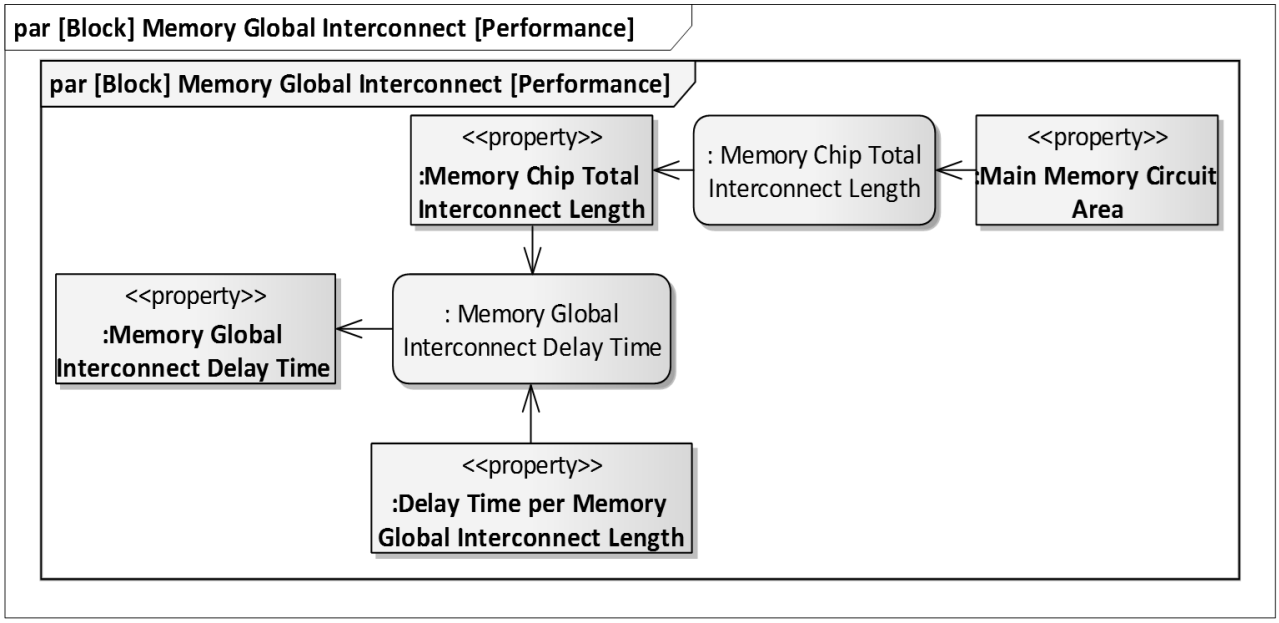


Fig. A.66 Memory global interconnect performance parametric diagram.

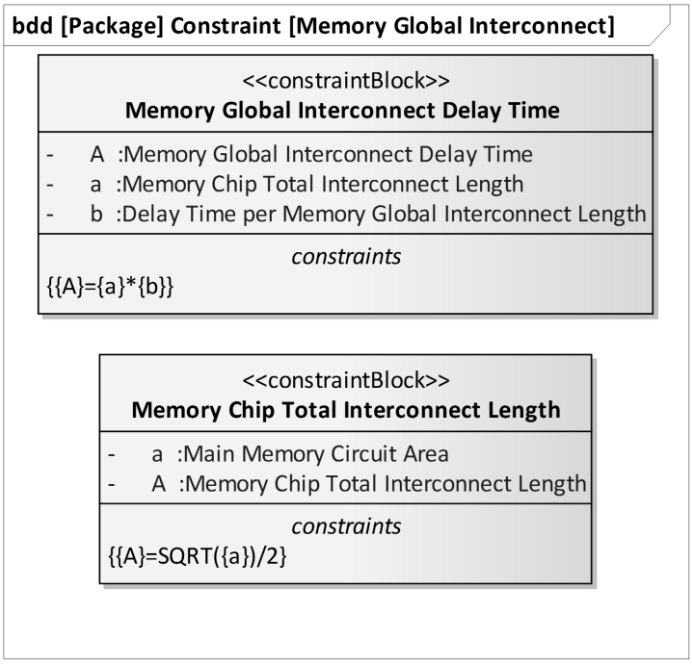


Fig. A.67 Detail of memory global interconnect constraint block.

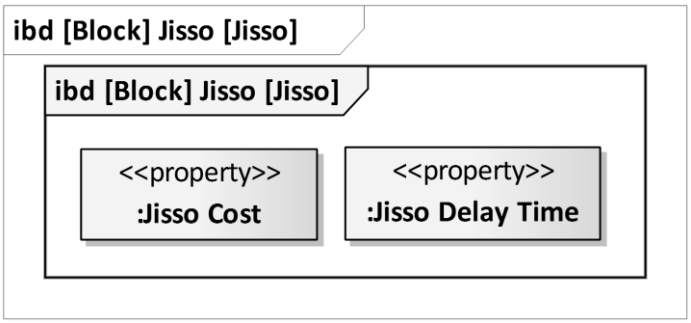
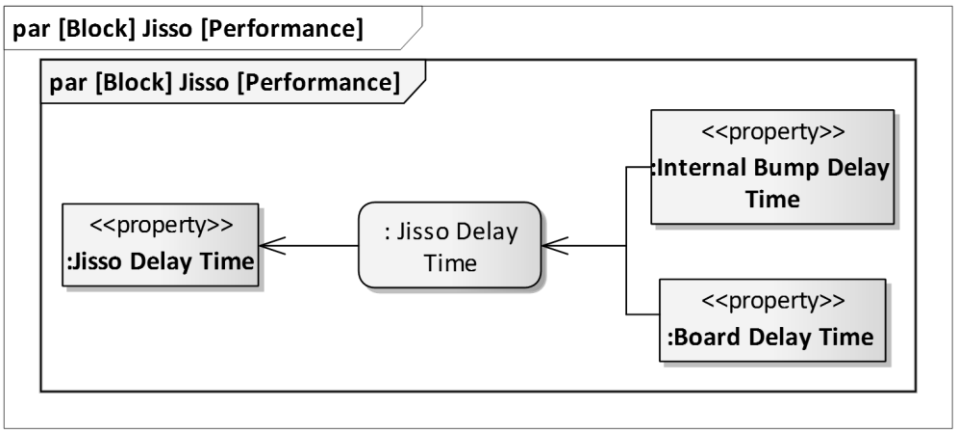
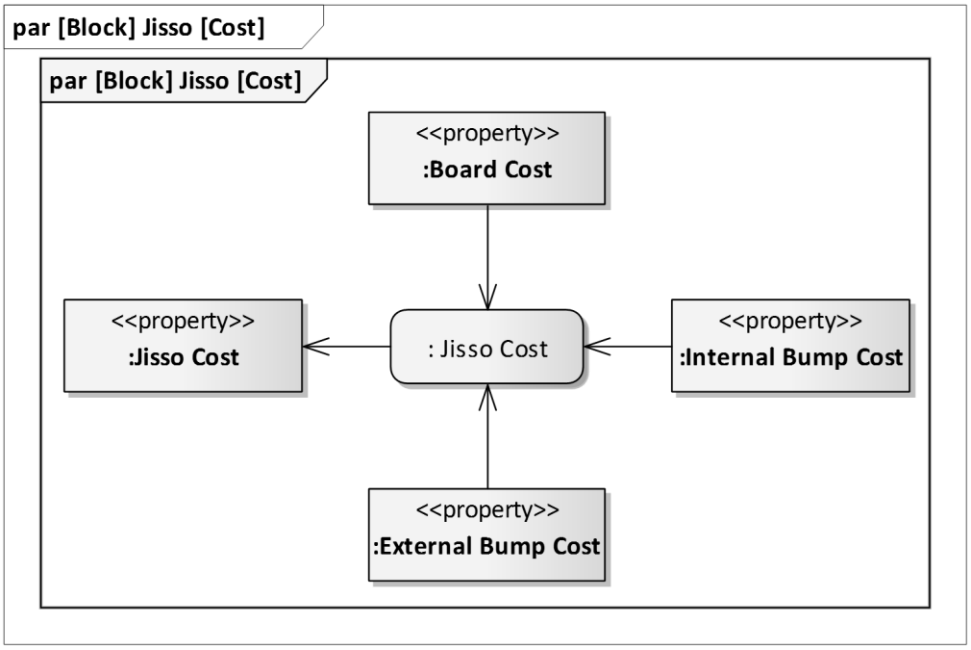


Fig. A.68 Jisso internal block diagram.



(a) Performance.

Fig. A.69 Jisso parametric diagram.



(b) Cost.

Fig. A.69 Continued.

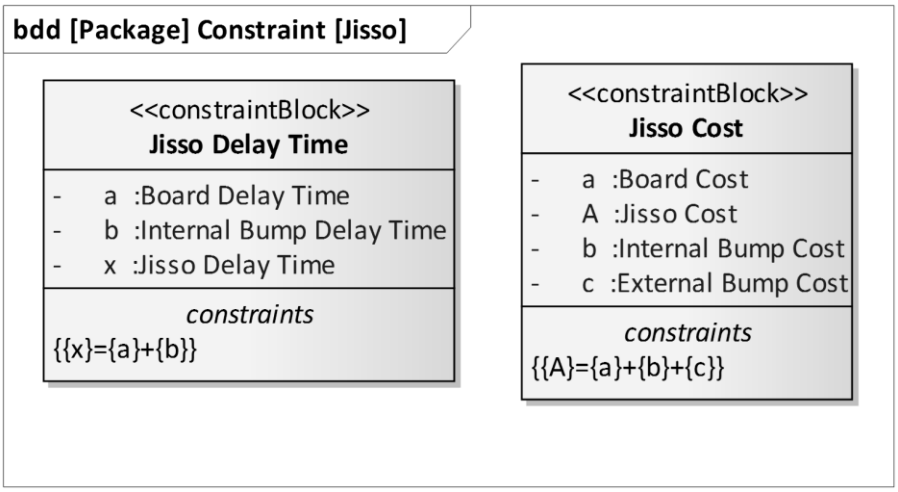


Fig. A.70 Detail of jisso constraint block.

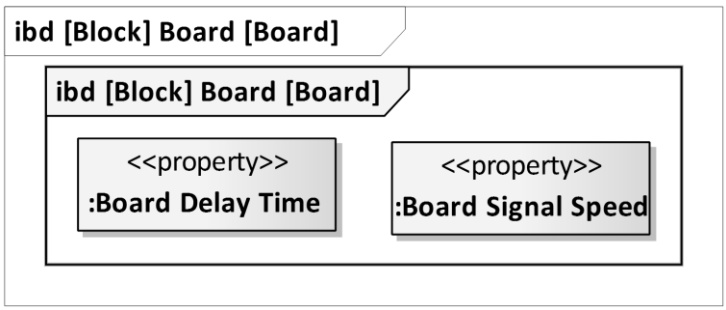


Fig. A.71 Board internal block diagram.

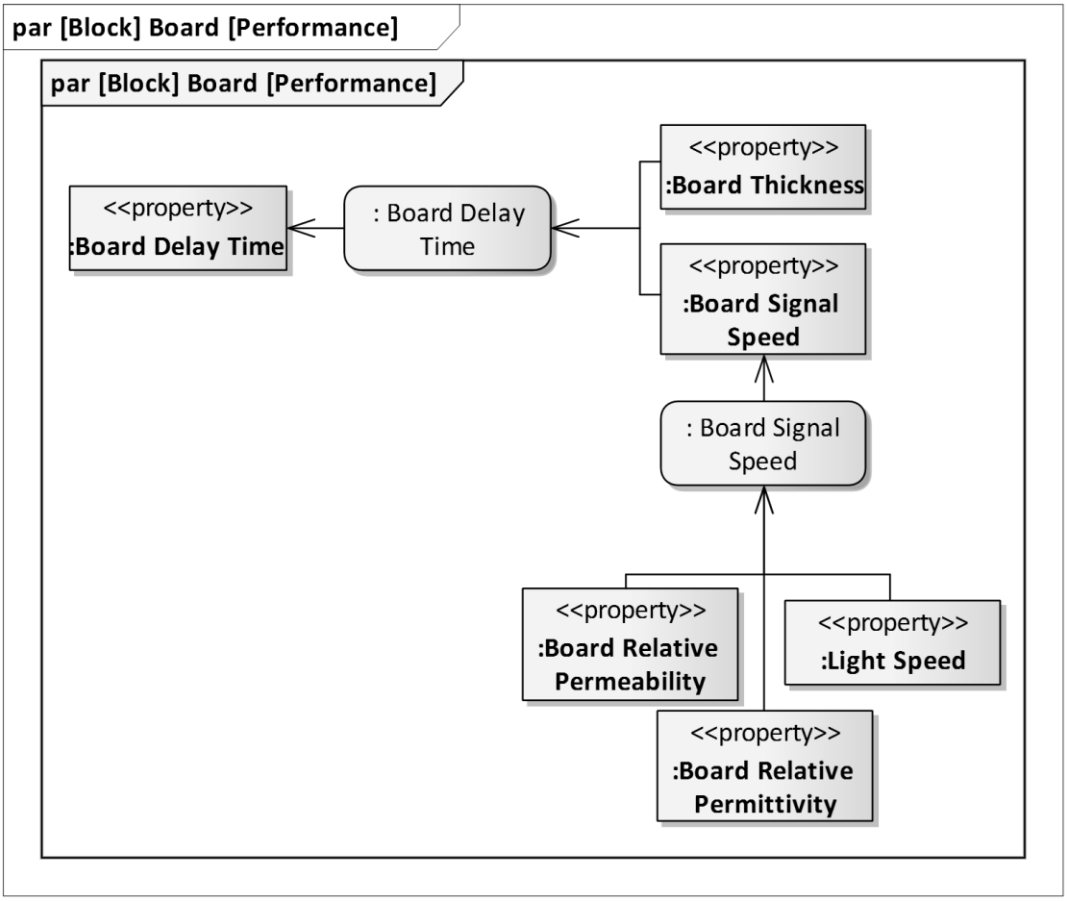


Fig. A.72 Board performance parametric diagram.

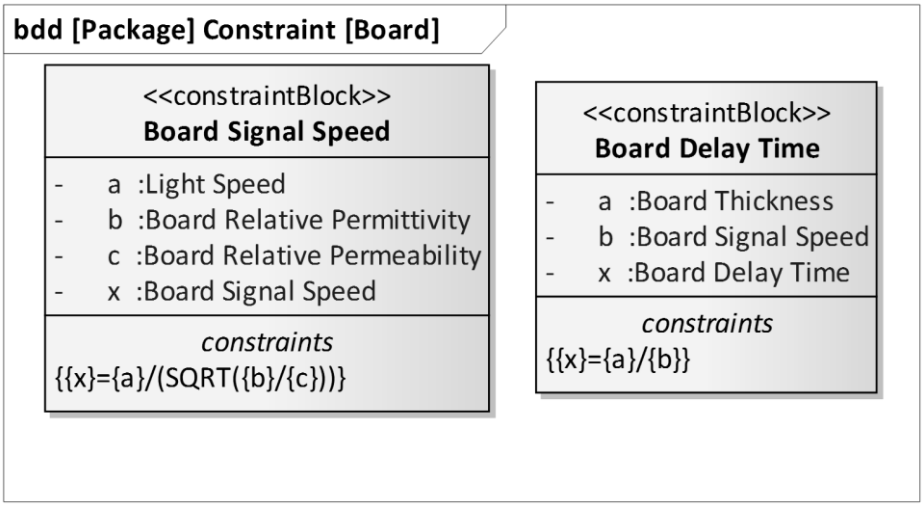


Fig. A.73 Detail of board constraint block.

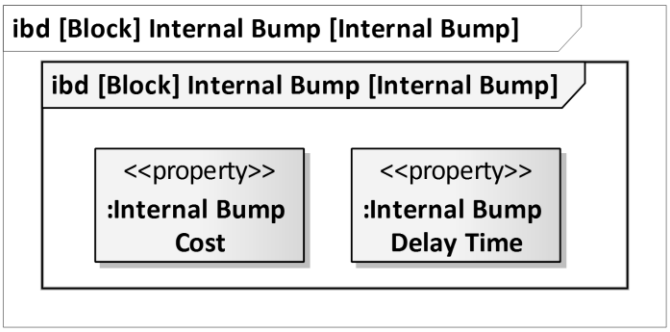
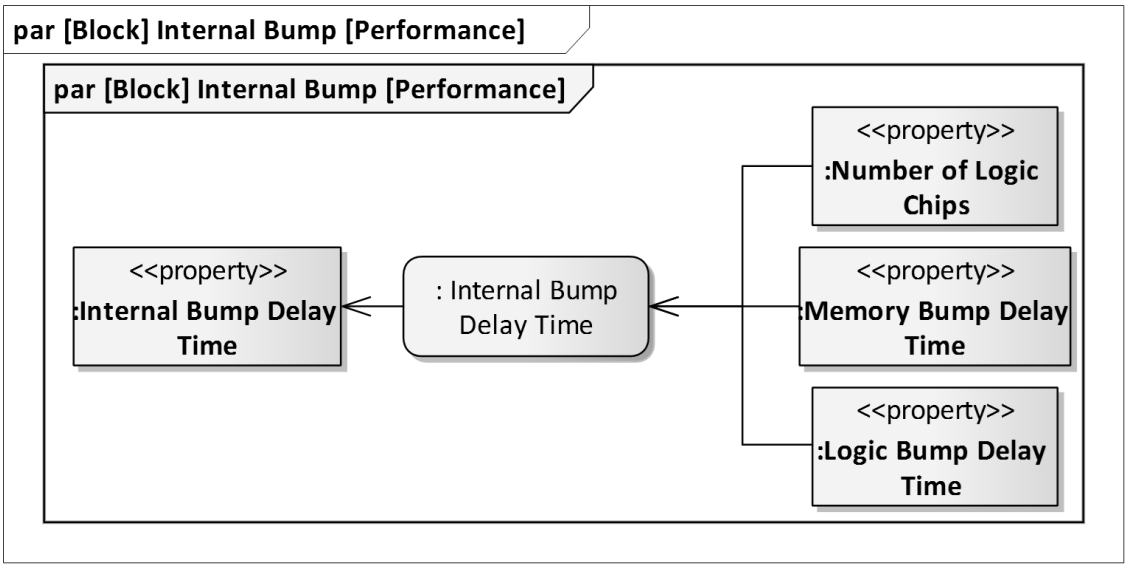
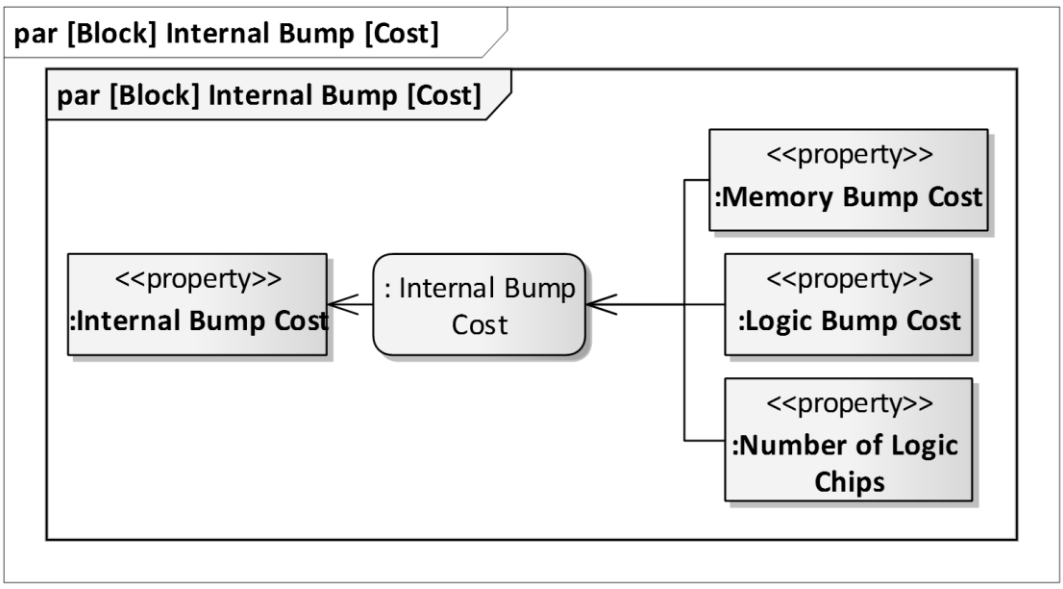


Fig. A.74 Internal bump internal block diagram.



(a) Performance.



(b) Cost.

Fig. A.75 Internal bump performance parametric diagram.

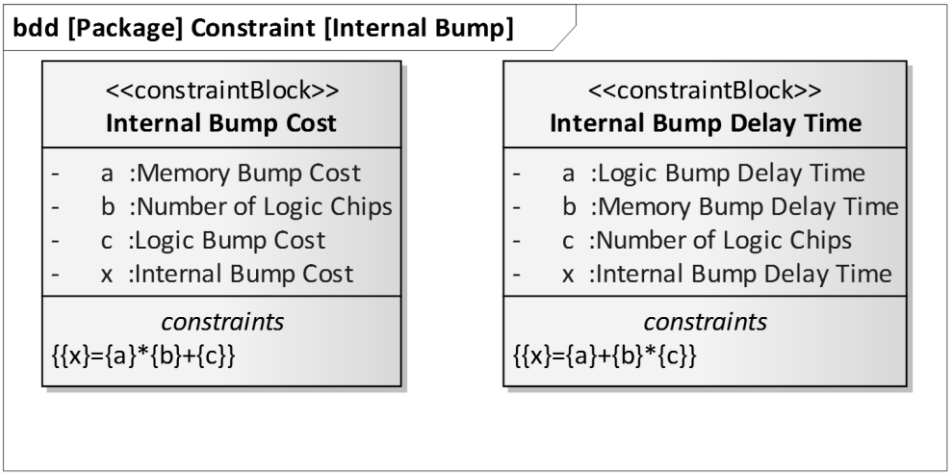


Fig. A.76 Detail of internal bump constraint block.

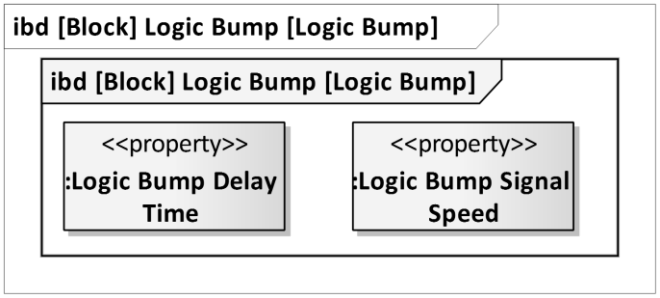


Fig. A.77 Logic bump internal block diagram.

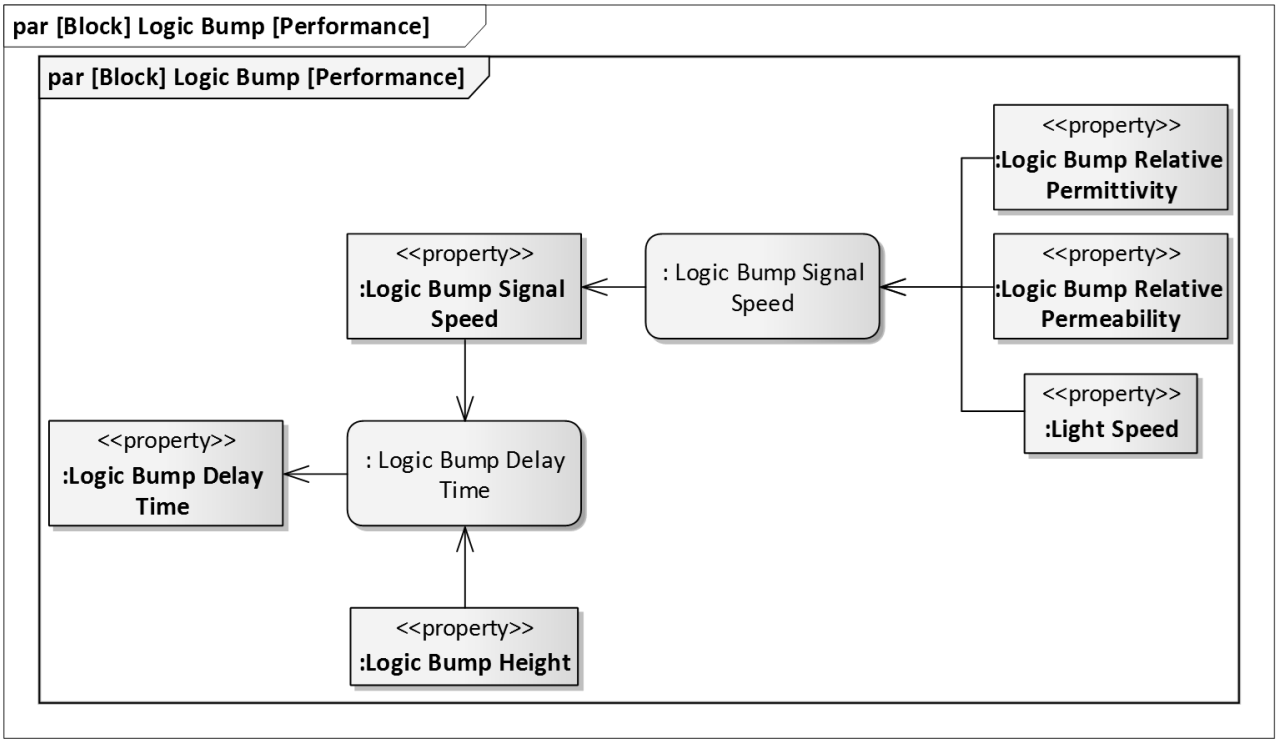


Fig. A.78 Logic bump performance parametric diagram.

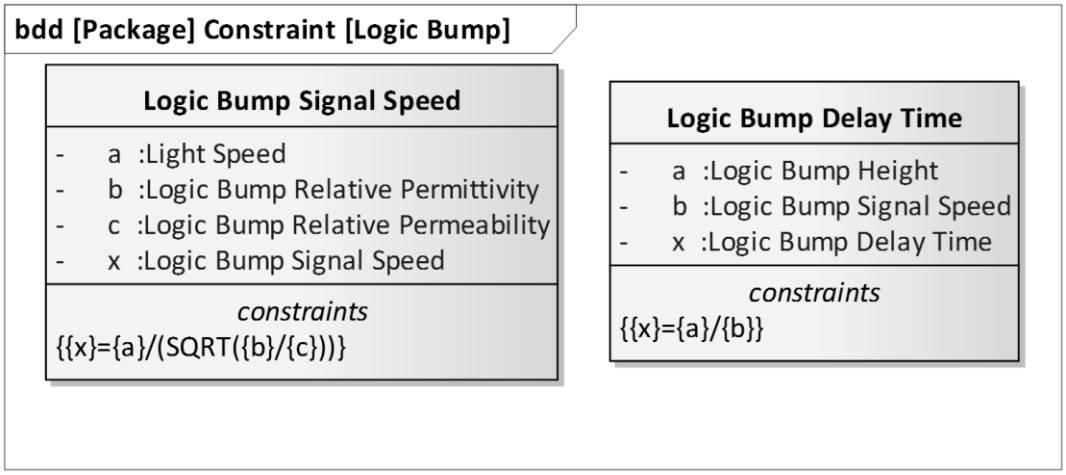


Fig. A.79 Detail of logic bump constraint block.

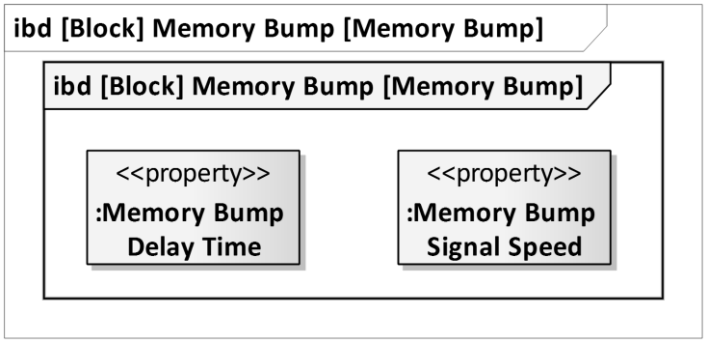


Fig. A.80 Memory bump internal block diagram.

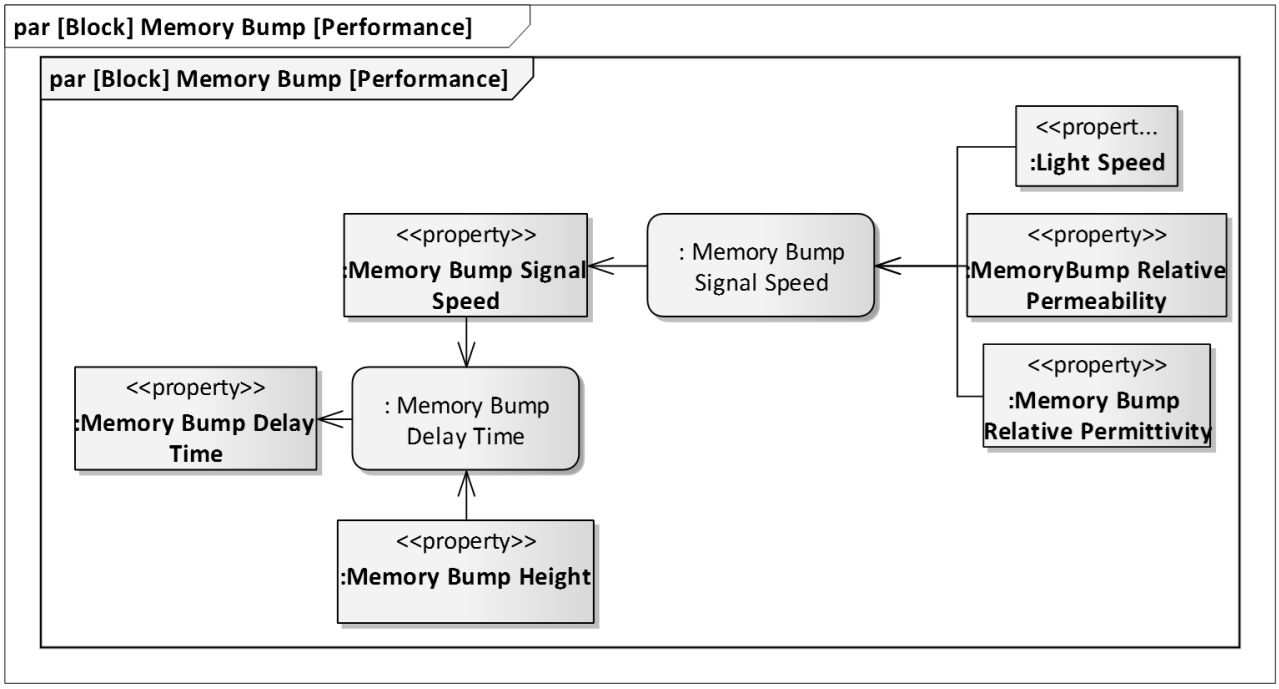


Fig. A.81 Memory bump performance parametric diagram.

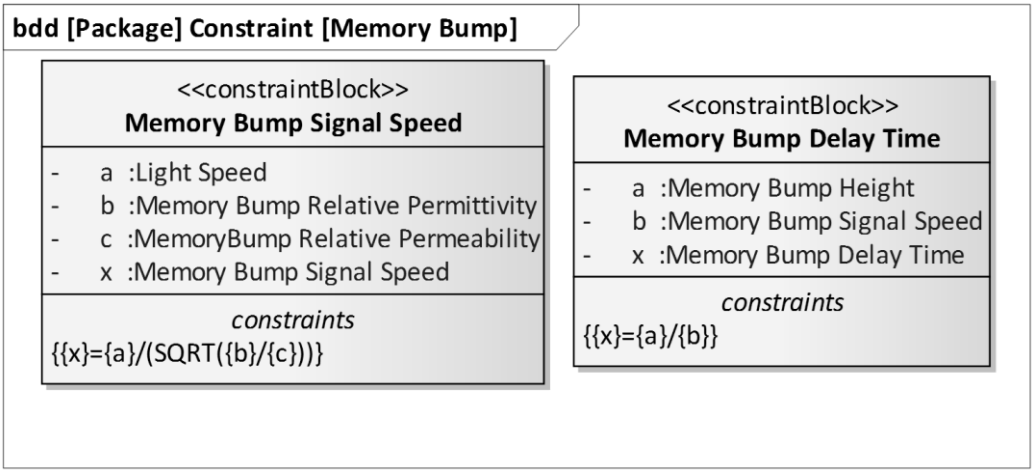


Fig. A.82 Detail of memory bump constraint block.

このようにブロック図、内部ブロック図、パラメトリック図、最適化図を用いることで、第 3 章で示した情報と等価な情報を定義でき、なおかつ、設計目標である、設計変数、目的関数も定義可能であることが確認できた。

I. 本論文に関する公表論文

(1) 村田秀則, 岩田剛治, 多屋淳志, 佐藤了平, 森永英二, 岡本和也, 工藤啓治, 青山和浩, 古賀毅, “システムデザイン手法 SDSI-Cubic を用いたシステム LSI の適正設計”, エレクトロニクス実装学会誌, 17-7 (2014)529-541.

(2) 村田秀則, 岩田剛治, 佐藤了平, 森永英二, 岡本和也, 青山和浩, 古賀毅, “システム定義手法を用いたシステムデザインにおける自動最適化アルゴリズムに関する研究”, スマートプロセス学会誌, 3-1 (2014)67-75.

(3) Hidenori Murata, Yoshiharu Iwata, Ryohei Satoh, Eiji Morinaga, Kazuhiro Aoyama, Tsuyoshi Koga, “Automated Construction of System Design Work-flow in SDSI-Cubic”, The 7th Asia-Pacific Council on Systems Engineering Conference, (2013)(CD-ROM).

II. 本論文に関する学会発表

(1) 村田秀則, 岩田剛治, 多屋淳志, 佐藤了平, 森永英二, 岡本和也, 工藤啓治, 青山和浩, 古賀毅, “次世代システム LSI のシステムデザイン・インテグレーション手法に関する研究”, Proceedings of the 16th Symposium on "Microjoining and Assembly Technology in Electronics", 16 (2010)299-304.

(2) Hidenori Murata, Yoshiharu Iwata, Atsushi Taya, Ryohei Satoh, Eiji Morinaga, Kazuya Okamoto, Keiji Kudo, Kazuhiro Aoyama, Tsuyoshi Koga, “Study on system design system integration method for system LSI”, Proceedings of International Conference on Electronics Packaging 2010, (2010)128-133.

(3) 村田秀則, 岩田剛治, 多屋淳志, 安村孝成, 佐藤了平, 森永英二, 岡本和也, 工藤啓治, 青山和浩, 古賀毅, “次世代システム LSI の階層型最適化手法に関する研究”, Proceedings of the 17th Symposium on "Microjoining and Assembly Technology in Electronics", 17(2011)121-126.

(4) Hidenori Murata, Yoshiharu Iwata, Atsushi Taya, Ryohei Satoh, “System Model Visualization Method for 3D-SiP”, Proceedings of International Conference on Electronics Packaging, (2011)781-786.

(5) 村田秀則, 岩田剛治, 多屋 淳志, 佐藤了平, "システムデザインのためのモデリング手法に関する研究", 日本機械学会第 21 回設計工学・システム部門講演会講演論文集,

(2011)260-264.

(6) 村田秀則, 岩田剛治, 佐藤了平, 森永英二, "システムデザイン手法フレームワーク SDSI-Cubic に関する研究―システムデザインワークフローに応じた最適化問題の自動構築―", 日本機械学会第 22 回設計工学・システム部門講演会講演論文集, (2012) 783-787.

(7) 村田秀則, 岩田剛治, 佐藤了平, 森永英二, 青山和浩, 古賀毅, "システムデザイン手法フレームワーク SDSI-Cubic に関する研究―システムデザインワークフローの自動構築―", 日本機械学会第 22 回設計工学・システム部門講演会講演論文集, (2012)778-782.

(8) 村田秀則, 菊池 昌恵, 岩田 剛治, 佐藤 了平, 森永 英二, "SDSI-Cubic を用いた組込み機器用ハード・ソフト最適化の自動実行を考慮したモデリング手法に関する研究", 日本機械学会第 23 回設計工学・システム部門講演会講演論文集, (2013)(USB メモリ).