設計自動化 64－10
(1992. 10. 23)

ペトリネットに基づく非同期式回路の検証について

本間 一樹
東京工業大学
情報工学科
ihonma@cs.titech.ac.jp

米田 友洋
東京工業大学
情報工学科
yoneda@cs.titech.ac.jp

東京工業大学 工学部 情報工学科 〒152 東京都目黒区大岡山 2–12–1

あ ら ま し 本稿では、ペトリネットに基づく非同期式回路の自動検証方法について考察する。非同期式回路は、配線遅延は存在せず、素子遅延は有界であるがその上限値は未知であるとしてモデル化し、これをペトリネットを用いて表現する。また、仕様もペトリネットを用いて表現し、これらのペトリネットから導出された決定性有限オートマトンを比較することにより、検証を行なう。この様な検証方式はDillらによって提案されているが、livenessの性質が容易には検証できないという問題点がある。そこで、この問題点についての解決方法を示し、非同期式回路のより厳密な検証方法について提案する。

和文キーワード 検証, 非同期式回路, ペトリネット, オートマトン

Automatic Verification Method of Asynchronous Circuits using Petri nets

Ichiki Homma
Department of Computer Science
Tokyo Institute of Technology
ihonma@cs.titech.ac.jp

Tomohiro Yoneda
Department of Computer Science
Tokyo Institute of Technology
yoneda@cs.titech.ac.jp

Department of Computer Science, Tokyo Institute of Technology 2–12–1 Ookayama Meguro-ku Tokyo 152, Japan

Abstract This paper presents an automatic verification method of asynchronous circuits using Petri nets. We assume that asynchronous circuits have arbitrary gate delays and no wire delays, and describe them using Petri nets. Petri nets are also used to express the specification. Then, the verification is performed by comparing the deterministic finite automata that are constructed from those Petri nets. This method was first proposed by David L. Dill.His method, however, cannot verify "liveness property" easily. In this work, we extend Dill's method such that it can handle liveness properties.

英文 key words Verification, asynchronous circuit, Petri nets, automata.

1 はじめに

近年半導体技術の進歩により、素子のスイッチング速度は年々高速化している。一方、集積回路の集積度も高集積化の一途をたどっている。この二つの集積回路に関する技術進歩により、新たに『配線遅延』の問題が持ち上がってきた。つまり、同期式回路は『クロック信号は回路全体で同時に変化する』という仮定の下で設計されているが、先に述べた様な『ピコ秒素子』が実現されれば、その様な仮定ができる領域が非常に小さくなり、この領域内に回路を収めることができなければ、素子速度に見合う程十分に高速化されたプロセッサを実現することが困難になると考えられる。[1]

この問題の解決方法の一つとして、『非同期式回路設計法に基づき、プロセッサを非同期式回路で実現する』という手法がある。非同期式回路は古くから研究がなされているが、その中でも代表的な回路モデルとしてSpeed-Independent 回路（以下 SI 回路）がある。このモデルにおける遅延仮定は『配線遅延は存在せず、素子遅延には上界がない』というもので、この仮定の下で回路の動作を因果関係で保証するようにプロセッサを設計することが考えられる[1]。

一方で現在用いられている同期式回路の検証システムでは、素子遅延を全く無視するか、単純な遅延を扱うものが多い。しかしこれらの検証システムをSI回路の仮定に従った非同期式回路の検証に使用するのは、現実的には有効なものと考えられるが、厳密な検証とは言い難い。

非同期式回路のより厳密な検証法の一つに、Dill らの考案した、仕様、回路構成をペトリネットを用いて表現し、それらから変換された決定性有限オートマトンの入出力信号の遷移を調べて検証を行なう方法[3]が挙げられる。この方法の利点として、検証を階層的に行なうことが可能であることが挙げられるが、この検証システムにおいても safety（入出力信号線の遷移に関して、仕様の起こす以外の遷移を、回路が起こさないこと）に関する検証は可能であるが、liveness（入出力信号線の遷移に関して、仕様の起こす遷移を全て回路が起こすこと）に関してはなんら保証をしておらず、完全な検証とは言い難い。

そこで本稿では先ず、Dill らの考案した方法とその問題点について述べ、次にその問題点の解決方法について考察して、より厳密に非同期式回路の検証を行なう方法を示す。

2 非同期式回路

本節では非同期式回路について説明する。

2.1 遅延モデル

非同期式回路とは、遅延のばらつきに関わらず、設計者が期待する動作をする回路である。回路実現の際に問題となるのは、『一定の仕方で抽象化された回路モデル上で誤った動作をするかどうか』である。これまで数多く研究されてきた回路モデルの一つとして SI（Speed-Independent）回路が挙げられる。

[1]長い配線に関してはそれらを buffer 素子として考えることで取り扱うことができる。ただし、実用的な回路を実現するために、モジュール内のローカルな配線（fanout branch 等）に関してはその遅延を 0 と仮定する。

			x	$\overline{x}$
データ	有効データ	$X=0$	0	1
		$X=1$	1	0
	無効データ		0	0
未使用			1	1

表 1: 2 線 2 相方式で用いる符号

SI（Speed-Independent）回路

配線遅延は存在せず、素子遅延の大きさは有限であるがその上限値は未知である。

本稿で扱う非同期式回路は、全てこの SI 回路に基づくものとする。

2.2 Muller の C 素子

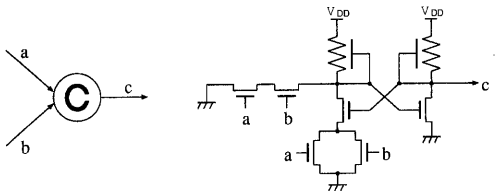

図 1: Muller の C 素子

Muller の C 素子（以下 C 素子）は、待ち合わせ素子（rendezvous）、合流（join）、あるいは最終入力応答（last of）回路とも言われる。これは相安定素子で、全ての入力が 1 になった後だけ出力が 1 になり、全ての入力が 0 になった後だけ出力が 0 になる。図 1に、論理回路記号と nMOS 回路図とで示してある。

2.3 2 線 2 相方式

非同期式回路はクロック信号を持たないため、データに時間情報を含めた形でデータの送受信を行なう。具体的にデータの到着を確認する方法として、以下の様に定義する。1 ビットのデータにゼロ（0）、イチ（1）、無効データ（–）の 3 状態を与える。この 3 値信号は 2 本の信号線を用いて表現し、00 を無効データ、01 をゼロ、10 をイチと割り当てる（表1）。全てのビットが無効データである時、このデータを spacer と呼ぶ。全てのビットがイチ、もしくはゼロの時、このデータを code と呼ぶ。各ビットにおいて、ゼロと無効データ、イチと無効データ間の遷移は認められるが、ゼロとイチの間の遷移は認められない。

2 線 2 相方式とは上で定義した符号を用いて、データの転送方法として spacer → code → spacer →... の様に信号線を変化させて、データの到着を確認する方式である。

2.4 2 線 2 相式レジスタ間データ転送

プロセッサにおける基本動作の一つは、レジスタ間データ転送である。レジスタ間データ転送の方式として 2 線 2 相方式を用いる。データ転送の方式は組合せ回路の有る

無しに関わらず、2相制御モジュールとレジスタを図2に示される様に接続する。このデータ転送方式は以下のと通りである。なお、初期状態では全てのデータ信号線はspacerを保持し、全ての制御信号線は値0を持つとする。

(1) 2相制御モジュールは転送元レジスタへの要求を1に変化させる。

(2) 転送元レジスタは、出力をspacerからcodeに変化させる。

(3) 組合せ回路は、入力がspacerからcodeに変わったため、出力をspacerからcodeに変化させる。

(4) 転送先のレジスタは組合せ回路のcode出力を読み込んで、2相制御モジュールへの応答を1に変化させる。

(5) 2相制御モジュールは応答が1に上がったのを見て、1であった要求を0に変化させる。

(6) 転送元レジスタは、出力をcodeからspacerに変化させる。

(7) 組合せ回路は、入力がcodeからspacerに変わったため、出力をcodeからspacerに変化させる。

(8) 転送先レジスタは、組合せ回路のspacer出力を受け取ると、1であった応答を0に変化させる。

(9) 2相制御モジュールは応答が0に下がったのを見て、再び最初に戻り要求を1に変化させる。

この様に、一つの論理動作の終了を示す動作完了信号を発生させ、それを次の動作の開始信号として一連の動作を進行させる、**要求・応答プロトコル**を用いることにより、非同期式回路でのデータ転送を実現する。

2.5 2線2相式レジスタの回路構成

ここでは非同期式回路の例として、先のレジスタ間転送モデルに沿った2線2相式レジスタの回路構成を挙げる。2線2相式レジスタは、以下の性質を必要とする。

(1) 入出力線は、要求、応答、データ入力線、データ出力線の4つを持つ。

(2) 初期状態では、応答は0であり、データ出力線はspacerである。

(3) 要求が0の時、データ出力線はspacerとなる。また、要求が1の時、データ出力線はcodeとなる。

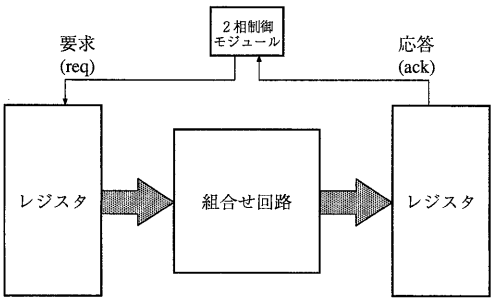

図 2: 2線2相式レジスタ間データ転送

(4) データ入力線にcodeが到着して、データが記憶部に書き込まれて初めて、応答が1となる。また、データ入力線にspacerが到着して初めて、応答が0となる。ただし、

(5) 要求が1になっている時、入力線にcodeが到着しても記憶部への書き込みは行なわれず、応答は1にならない。

また、2線2相式レジスタを駆動するための条件として、以下の事柄が挙げられる。

- codeが到着して応答が0の時は、記憶部への書き込みが行なわれている途中であるため、要求は1にしてはならない。

この条件は、レジスタにデータが書き込まれる間に要求を1にすると、記憶部の値の変化がそのままレジスタの出力に現れてしまう可能性があるため、加える必要のある制限である。

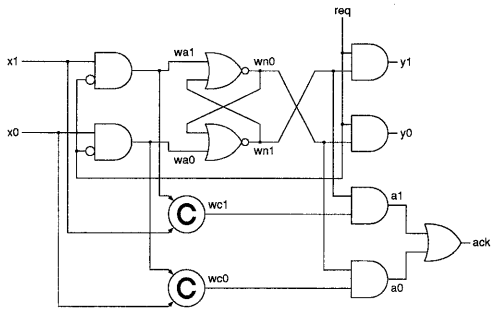

図 3: 非同期式回路によるレジスタの構成

以上の条件を元に、1ビットの記憶部を一つ持つレジスタ回路を構成したものが、図3である。

図3において、NORフリップフロップが1ビットの記憶を保持する部分、出力につながっているANDゲートがspacer生成回路、入力につながっているANDゲートは入力遮断回路、その他は応答生成回路となっている。

2.6 マスタスレーブ・レジスタ

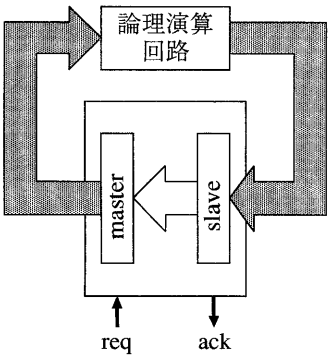

図 4: マスタスレーブ・レジスタ

図2を見ればわかるように、転送の際には必ず転送元と転送先の二つのレジスタが必要になってくる。そしてそ

の二つのレジスタが同一のレジスタ名であることが多い（例えばプログラム・カウンタのインクリメント等）。そこで、非同期式回路では特にこの二つのレジスタを一つの組として考え、**マスタスレーブ・レジスタ**と呼んでいる（図4）。これは同期式回路におけるものとは、異なることに注意を要する。またここでは転送元をマスタ（master）、転送先をスレーブ（slave）と呼ぶことにする。

一般には、転送元と転送先のレジスタが異なってもよいが、本稿では簡単化のため、転送元と転送先が同一のマスタスレーブ・レジスタについて扱うものとする。この様なマスタスレーブ・レジスタは以下の性質を必要とする。

(1) 入出力線は、要求、応答、データ入力線、データ出力線の4つを持つ。

(2) 初期状態では、応答は0であり、データ出力線はspacerである。

(3) 要求が0の時、データ出力線はspacerとなる。また、要求が1の時、データ出力線はcodeとなる。

(4) 要求が1の時、入力線に有効データが到着し、マスタ側の記憶部に無効データが到着して初めて、応答が1になる。

(5) 要求が0の時、入力線に無効データが到着し、マスタ側の記憶部にスレーブ側の記憶部の内容が書き込まれて初めて、応答が0になる。

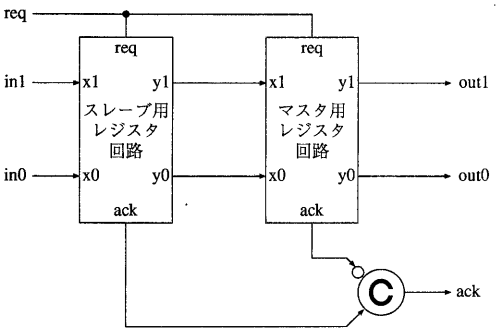

図5: マスタスレーブ・レジスタの回路構成

以上の性質を元に、マスタ側のレジスタとして図3を、またスレーブ側のレジスタとして図3において、出力に接続されているANDと入力に接続されているANDを入れ換えた回路を使用して、マスタスレーブ・レジスタを構成すると、図5の様になる。

3 ペトリネット（Petri net）

本節ではペトリネットについて説明する。

3.1 ペトリネットの形式的定義

ペトリネットNとは5項組$N=(P,T,I,O,\mu_0)$である。但し、

(1) $P=\{p_1,p_2,\ldots,p_n\}$はプレース（place）の有限集合であり、$n\geq 0$である。

(2) $T=\{t_1,t_2,\ldots,t_m\}$はトランジション（transition）の有限集合であり、$m\geq 0$である。

(3) $I\subseteq P\times T$は入力関数

(4) $O\subseteq T\times P$は出力関数

(5) $\mu_0\subseteq P$は初期マーキング

また、$\bullet t=\{p|(p,t)\in I\}, t\bullet=\{p|(t,p)\in O\}$をそれぞれ、tの入力集合、出力集合という。

3.2 トランジションの発火規則

(1) あるトランジションtとマーキングμにおいて、tの入力プレース全てにトークンが存在する場合、tは**発火可能**（**enable**）であるといい、そうでない時は**発火可能ではない**（**disable**）という。

(2) 発火可能なトランジションはいつ発火しても良いし、しなくても良い。

(3) あるマーキングにおいて、発火可能なトランジションtが発火した時、全ての$p\in\bullet t$からトークンを取り去り、全ての$p\in t\bullet$にトークンを書き込む。

3.3 ペトリネットの状態

ペトリネットの状態sは、その時のマーキングμによって決まる。また、ペトリネットNにおいて、ある状態sからトランジションの発火の繰り返しにより、状態s'が得られる時、s'はsから**可達である**（**reachable**）という。また、sからs'へ到達するまでのマーキングとトランジションの交互の系列を**発火系列**と呼ぶ。

またペトリネットの性質として、ある限られた組のマーキングのみ初期マーキングから可達であるため、マーキングの仕方は有限であることが挙げられる。このことは、ペトリネットの状態数が有限であることと同値である。

4 検証方法

ここでは先ず回路構成（以下実現と呼ぶ）および仕様を、ペトリネットを用いて表現する方法について説明し、次にそれを用いて検証を行なう方法について説明する。

4.1 実現のペトリネットによる表現方法

実現をペトリネットを用いて表現するため、次の様な方法を考える。1本の信号線に対して1本以上のトランジションを用意し、その信号線及びそれに対応する各トランジションに対して、同一の信号線名を割り当てる。その信号線に対応する各トランジションの発火について、現在信号線が0の状態であったら0→1の遷移と解釈し、信号線が1の状態であったら1→0の遷移と解釈して表現する。これは、ある回路に対してペトリネットを割り当てた時、ペトリネットの状態を回路の状態と対応させ、トランジションの発火を信号線の遷移に対応させるものである。例として2入力ANDゲートのペトリネットによる表現を、図6に示す[2]。この表現では、入力信号が素子に印加されたため、素子の出力信号が遷移できる条件が揃ったが、出力遷移が起こる以前に入力信号が再び印加

[2]図において、例えば二つのaというトランジションはそれぞれ別のものであるが、発火系列においては同一のものとして扱う

されたため、最初に印加した入力に対する出力遷移はその後永久に生じないという、いわゆる**慣性遅延**（図7）をモデル化している。

この方法においては、ペトリネットの発火系列が、回路における信号変化の到達可能な軌跡を表現することになる。

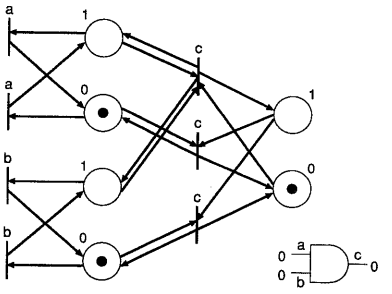

図 6: 2 入力 AND ゲートの Petri net による表現

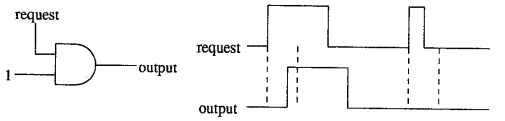

図 7: 慣性遅延

一般に実現を表現するためには、複数の回路及び素子を接続して行なうのが普通である。ここでは信号線とトランジションが対応しているため、回路の接続を表現するには、共通結線に対して同一の信号線名を割り当てればよいことになる。図8に、2 入力 AND の出力を 2 入力 OR の一つの入力に接続した回路のペトリネットによる表現を示す。

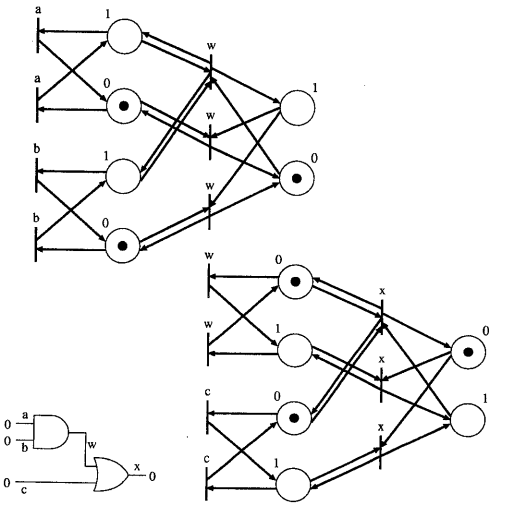

図 8: 回路の接続例

4.2 仕様のペトリネットによる表現方法

検証を行なう場合、実現に対する回路の仕様を表現し、その仕様に対して、実現が仕様を満足するかどうかを比較検討する。ここで言う仕様とは、『ある一定の手順に従って回路を動作させる時、その回路が満たすべき、入力及び出力信号線の遷移系列を表現するもの』である。つまり、注目している実現に対して、入力及び出力のみを取り出して、仕様から生成されるべき入出力遷移系列を表現するものであれば良い。ここでは仕様についてもペトリネットを用いて表現する方法をとる。

例として、先のレジスタについての仕様を構成してみる。レジスタの満たすべき条件及び駆動条件については先に述べた通りである。この条件を大まかに表現すると、図9の様になる。

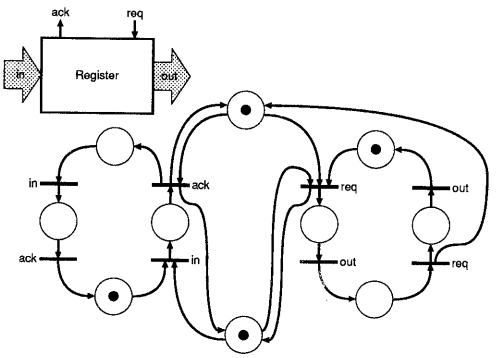

図 9: レジスタの仕様

図9において、右側のループは request と出力データに関しての関係を示し、左側のループは acknowledge と入力データの関係を示しており、それぞれの動作は互いにほぼ独立して起こることを表現している。また中央上部のプレースは、request が 1 の時は入力データが到着しても acknowledge が返されない（記憶部への書き込みが行なわれない）ことを表現したものであり、中央下部のプレースは、入力データが到着して acknowledge が 1 に変化していない場合は、request は 1 にしてはならないと言う、駆動条件を表現したものである。この仕様の表現方法においては、実現が満たすべき動作条件と、実現を動作させる際の駆動条件が同一のペトリネット上に表現できることが特徴である。

図9を元に、レジスタの仕様を表す完全なペトリネットを構成したものを図16に示す。

4.3 Dill の検証方法

検証を実際に行なう時は、仕様・実現ともペトリネットを決定性有限オートマトン（以下 DFA）に変換する。仕様についてはペトリネットは一つのみ与えられるため、これに対応する DFA を生成する。一方実現に対しては、複数のペトリネットが一般には与えられることになる。各ペトリネットを DFA に変換した後、実現を表す一つの DFA を求めることは、実現に含まれる全ての DFA の直積を求めることになる。しかしこれは、仕様との一致性を見る

上で不必要な状態まで生成することになり、効率が低下する。そこで実現に関しては、各ペトリネットをDFAに変換したものを個別に保持しておき、必要に応じて状態を生成していくことにする。

Dillらの考案した方法[3]ではさらに、仕様のDFAに対して元の入力信号を出力信号、元の出力信号を入力信号とするDFA（Dillらはこれを仕様の鏡像、仕様^{M}と呼んだ）を生成し、実現のDFAと接続する操作を行なっている（図10）。

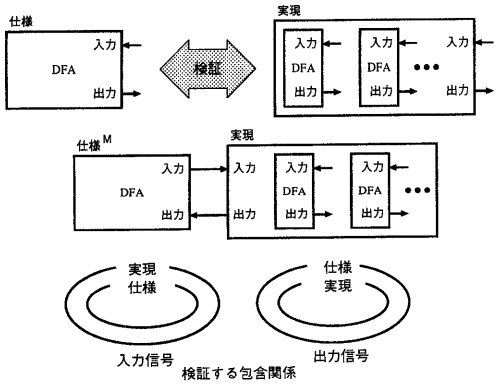

図10: Dillの検証方法

この方法では、仕様^{M}のDFAと、実現を表すDFAの集合の和集合において、各DFAの出力の中で発火可能（遷移可能）な信号線に対し、それを入力として持つ全てのDFAがその信号の遷移を受理できるかどうかということを、深さ優先探索（Depth-First Search、以下DFS）により調べている。この様にすることで、(1)入力信号線の遷移について、仕様で受理できる遷移を実現で全て受理すること、(2)出力信号線の遷移について、仕様が出力する以外の遷移を実現が出力しないこと、の2点を調べることによって検証を行なうのである。

このオートマトンの比較による方法の利点は、もし実現が仕様の条件を満たすことが検証されれば、その回路については実現を仕様のペトリネットで置換しても良いことになり、検証を階層的に行なうことが可能になってくる（図11）。このことは検証を行なう際に、状態数を減少させるという大きな効果がある。

しかしDillの方法においては、入出力信号の遷移について、図10の包含関係のみの検証しか行なっていないため、safetyについてのみの検証であり、出力信号に関するlivenessについては何の保証もしていない。これは極端な場合、実現が出力遷移を一つも起こさないような回路であっても、Dillの方法では検証器は『誤動作無し』の答を返すことになる。

4.4 検証方法の改良

出力信号線の遷移に関して、『現在のDFAの状態において実現が仕様を満たしている』ということについて考察してみる。safetyについては先にも示した通り、『実現の出力信号線の遷移が仕様でも存在していること』である。一方livenessについては『仕様の出力する遷移が実現にも存在していること』である[3]。つまり仕様の出力信号線の遷移に関して、実現の出力信号線の遷移が完全に一致していればsafetyかつlivenessが検証できたことになる。例えば、図12において仕様にxとyの二つの出力信号線の遷移が存在している時、実現においてもxとyの遷移のみが発火可能な時、仕様と実現のDFAの状態は対応したものとし、この状態において、出力信号の遷移に関して実現は仕様を満たしていると考える。

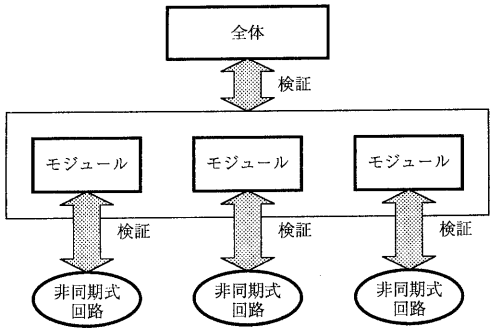

図11: 階層的な検証

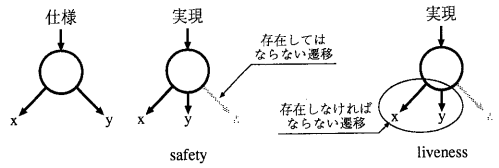

図12: 出力に関する検証事項

図13の様に、実現が内部信号遷移を含む場合、非同期式回路においては『回路内部の発火可能な信号線の遷移が全て終了して、回路内部が安定した時点（出力信号線の遷移はまだ発火していないものとする）で、次に発火可能な出力信号線の遷移が仕様において発火可能な出力信号線の遷移に等しい』なら『実現は仕様を満たす』と考える。つまり『実現のDFAにおいて、現在の状態から、入出力信号ではない（つまり内部の）信号線の遷移を辿って到達できる全ての状態に対して、(1)内部信号線の遷移が可能な場合は、仕様の出力信号線の内で発火可能なもの以外の遷移を起こさず、(2)発火可能であるのは入出力信号の遷移のみである場合は、仕様の出力信号線の内で遷移可能なものと、完全に同一の出力信号線が発火可能である場合』に、仕様と実現の状態は対応しているものとみなし、この状態において、出力信号の遷移に関して実現は仕様を満たしているとする。ただし、内部信号遷移のみからなるループが存在してはならない。

以上の事を元に、DFSによって検証を行なう、検証器全体のフローチャートを図14に示す。

[3]これは一般のlivenessよりも強い制約を持っているが、ここではlivenessをこの様な意味で用いるものとする

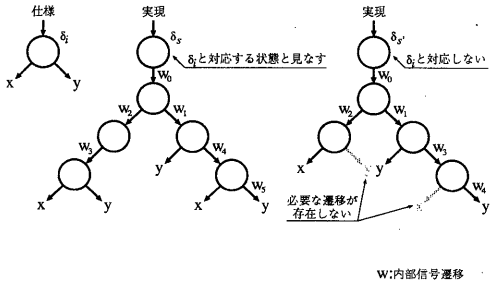

図 13: 出力信号線の遷移に関しての状態の対応

5 検証例

ここでは、livenessを扱える様に上記の改良を加えた検証器を用いて、レジスタを検証した例を示す。

5.1 レジスタ回路の検証

図3のレジスタを表すペトリネット、および仕様を表す図16のペトリネットを二つの検証器に入力して結果を得た。二つの検証器とも『誤動作は発生しない』という結果を返し、生成した状態数も共に148状態であった。これは実現の回路構成がlivenessを完全に満たしていたためである。次に、レジスタのacknowledge生成回路において、acknowledge出力に接続されている2入力ORゲートを2入力ANDゲートに置き換えて、acknowledgeが1にならないような回路を構成し、これを二つの検証器に入力してその結果を得た。結果は、Dillの検証器は『誤動作は発生しない』という結果を返し、生成した状態数は32状態であった。これに対して、今回の検証器は結果として誤動作を発生するまでのパスを返し、誤動作の可能性があることを示した。

	生成状態数	検証結果
Dill's system	148	OK
New system	148	OK

表 2: 検証結果：正しいレジスタ回路

	生成状態数	検証結果
Dill's system	32	OK
New system	–	error path

表 3: 検証結果：誤ったレジスタ回路

5.2 階層的検証例

階層的な検証例として、先に挙げたマスタスレーブ・レジスタの検証を行なった。マスタスレーブ・レジスタにおいて、マスタ側の回路は先のレジスタの回路をそのまま用いて回路を構成しているため、先に行なった検証結果より、回路の代わりに図16のペトリネットを用いた。スレーブ側についてもほぼ同様である。

マスタスレーブ・レジスタの大まかな仕様を図15に示す。これは先に挙げた条件をペトリネットに変換したものである。

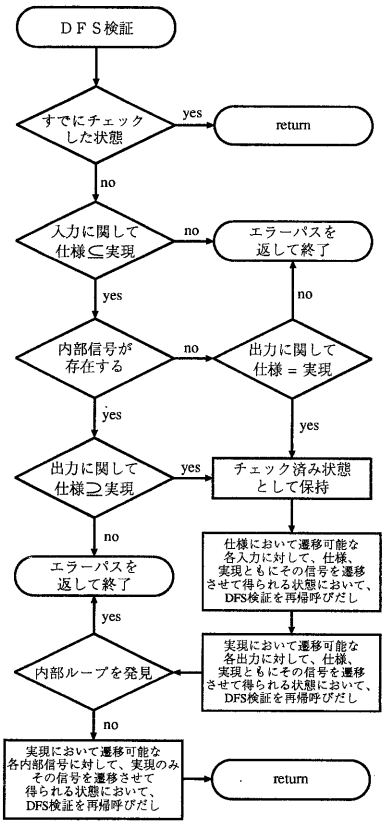

図 14: 検証器のアルゴリズム

検証の結果、両検証器とも実現は仕様を満足するという結果を返し、生成状態数は84であった。

	生成状態数	検証結果
Dill's system	84	OK
New system	84	OK

表 4: 検証結果：マスタスレーブ・レジスタ

6 結論

6.1 本稿のまとめ

本稿では先ず、非同期式回路の検証方法の一つとしてDillらの考案した方法について、出力信号線遷移のlivenessに関しての問題点を指摘した。次にその解決方法として、出力信号線については遷移可能な内部信号線の有無を考慮して、検証を行なう方法を提案し、より厳密な非同期式回路の検証を可能にした。

また、仕様と実現をペトリネットで与えるため、回路の動作が把握しやすく仕様を構成することが容易である。さらに、検証の結果正しい回路と判定されれば、実現のペトリネットを仕様のそれと置き換えることが可能であり、階層的な検証を実現している。

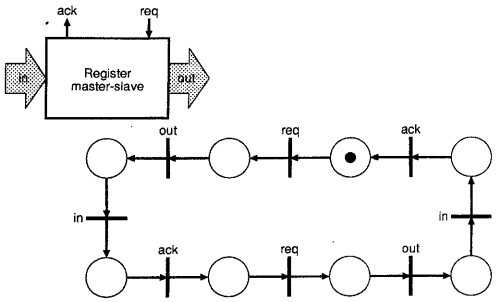

図 15: マスタスレーブ・レジスタの仕様

6.2 今後の課題

実用的なシステムの検証を行なう場合、検証時間・生成状態数をいかに削減するかが非常に重要な問題である。今回の方法では、階層的に検証を行なうことで、生成状態数の削減を行なうことが可能であるが、階層的な構造を何段も生成すると、上位の層において検証が失敗した際に一番最初の層から再び検証を実行しなければならず、効率の低下を招く。そこで階層構造は 4、5 層程度が限度と考えられる。しかし、モジュールが大きくなると、それを表現する DFA の状態数が急激に増加する。そこで、実現を表す各ペトリネットを一つに合成し、[4] に示される Partial order に基づく解析を行なうことにより、状態数の少ない DFA を求めることができ、これを用いて検証を行なえば、検証時間、生成状態数の削減が可能と思われる。今後はこの Partial order に基づく解析を今回の検証システムに組入れ、検証を高速化することが課題となる。

謝辞

本研究を行なうにあたり、適切な助言を頂きました、当麻喜弘教授、南谷崇教授、そして検証器のプログラムを提供して下さった、David L. Dill に深く感謝します。

なお、本研究の一部は (財) 大川情報通信基金 1992 年度研究助成によるものである。

参考文献

[1] 南谷 崇 : 同期式プロセッサの限界と非同期式プロセッサの課題, 信学技報,FTS90-45,December 1990.

[2] D. E. Muller and W. S. Bartky : A theory of asynchronous circuits,Proc.Int.Symp.Theory of switching,pages 204 – 343,Harvard Univ. Press Massachusetts 1959.

[3] David L. Dill,Steven M. Nowick,Robert F. Sproull : Specification and automatic verification of self-timed queues, Technical Report CSL-TR-88-387,Stanford University,California 1989.

[4] Tomohiro Yoneda and Holger Schlingloff : On Model Checking for Petri Nets and a Linear-Time Temporal Logic, FTS92-1,1992.

[5] 上野 洋一郎 : 非同期式プロセッサのアーキテクチャに関する研究, 修士論文, 東京工業大学,Feburuary 1990.

[6] 本間 一樹 : 非同期式プロセッサのアーキテクチャに関する研究, 卒業論文, 東京工業大学,Feburuary 1991.

[7] 川下 達也 : 非同期式プロセッサの検証に関する研究, 卒業論文, 東京工業大学,Feburuary 1991.

A 付録

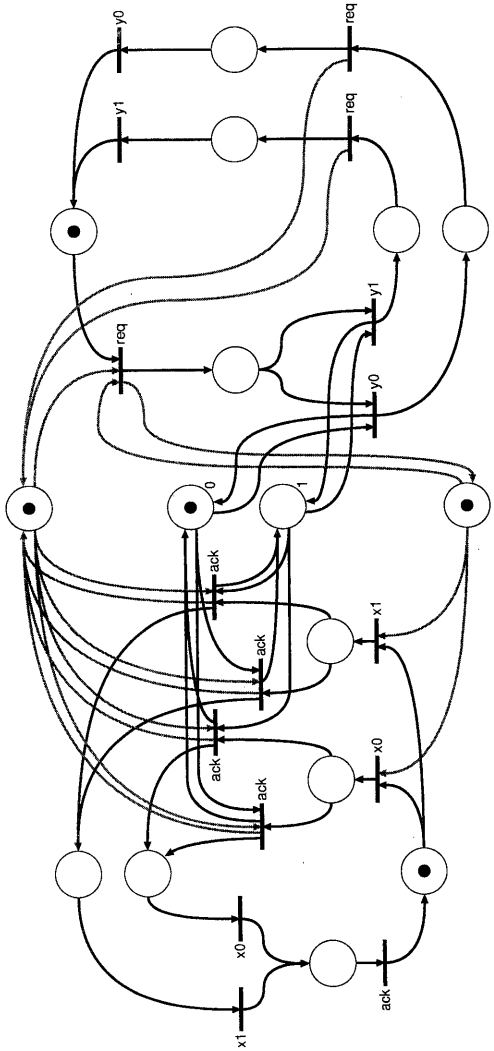

図 16: レジスタの仕様