# 非同期式プロセッサ制御回路合成の一手法

1J−7　　籠谷裕人　土居仁士　南谷崇

東京工業大学 工学部

## 1 はじめに

近年の素子技術はスイッチング遅延が1ピコ秒にせまる高速なデバイスを実現しつつある。しかし従来のプロセッサ回路はチップ全体へのクロック分配が必要であり、配線遅延が相対的に大きくなるためこうした素子を活用できるような高速のクロックを用いることができない[1]。

素子の高速性を有効に活用する一つの方法は、プロセッサを非同期式に構成することである。非同期式回路は、同期式回路の設計にあるような論理設計とチップ設計の相互依存性を排除でき、また、回路を拡張する場合のタイミング設計のやり直しも不要となり拡張性に富むといった利点を持つ。しかしながら、これまでのところ非同期式プロセッサの設計方法は充分には研究されていない。

最近、非同期式プロセッサ設計手法として、プロセッサの各機能モジュールを他のモジュールと通信する独立したプロセスとして記述し、このプロセス記述からMullerのC素子を含むゲートレベルの制御回路を合成する手法が提案されている[3]。しかしこの方法には、生成される回路の自由度が大きすぎ合成やその最適化にコストがかかる、任意の関数が素子として実現できないとMullerモデルの遅延仮定の下では正しく動作しないなどの問題点がある。また合成のためのアルゴリズムが定式化されていない。

本稿では、あらかじめ正しく動作することが保証されている基本回路を結合することで、非同期式プロセッサの各機能モジュール制御回路を簡単なアルゴリズムにより合成する一手法を述べる。

## 2 制御回路合成

設計対象となるプロセッサはいくつかの機能モジュールからなり、モジュール間の通信はrequest-acknowledgeによる4相ハンドシェークで実現される。モジュール自身も、あらかじめ定められたハンドシェークを行うよう設計された基本回路から構成することで、合成の手順が大幅に簡素化できる。この基本回路に対応したプロセスを基本プロセスと呼び、本手法の中心は、元のプロセス記述をこの基本プロセスの集合に分割することである。回路合成の流れは、図1のように表される。ただし、本稿ではレジスタやデータパスは扱わず、モジュールの制御回路のみを対象とする。

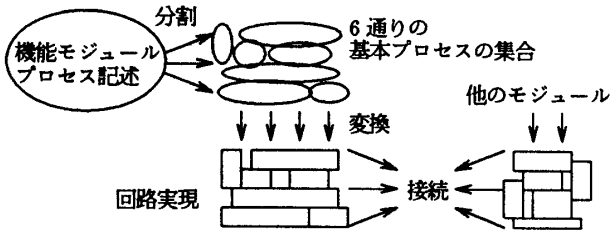


図1: 合成手順

## 3 プロセス記述

### 3.1 記述方式

ここでは、HoareのCSP[2]をもとにして文献[3]で提案されたプロセスの記述言語をそのまま利用する。簡単に紹介すると、基本的なコマンドはポートXでの通信$X$と、変数mの値の送受$X!m$, $X?m$、変数の演算$v := f(x,y)$である。また$S1$, $S2$をそれぞれ基本コマンドとした時、$S1;S2$は直列動作を意味し、$S1$, $S2$をそれぞれ通信コマンドとした時、$S1 \bullet S2$は二つの同時動作を意味する(•の方が優先順位が高い)。さらに$G_i$を論理式、$S_i$をコマンド列とした時、選択コマンド$[G_1 \rightarrow S_1 \| \ldots \| G_n \rightarrow S_n]$はガード$G_i$が真の時に対応する$S_i$が実行される。この時$G_i$は最大で一つしか真とならない。$[G_1 \rightarrow S_1 | \ldots | G_n \rightarrow S_n]$は上と同様であるが$G_i$はいくつでも真となり得、そのうちの一つが選択される。プローブ$\bar{X}$はポートXの相手プロセスがその通信のためにpendingされていることを示す論理変数であり、ガードの一部に用いられる。$*[\ldots]$は無限の繰り返しを表す。

### 3.2 記述の制約

合成対象とする機能モジュールのプロセス記述は以下のような制約を満たすものとする。

- 無限ループである。
- プローブ$\bar{X}_i$をガードに含む分岐$B$がもしあるならば、
  - その分岐はプロセスのループの先頭にあり、
  - すべてのガードに何らかのプローブが一つだけ含まれ、
  - 分岐後のさらなる分岐によるシーケンスは唯一の$X_i$の通信によって前後に分離でき、
  - プローブ$\bar{X}_i$が真の時、$\bar{X}_i$を含む$B$のガードの集合$G_{\bar{X}_i}(B)$の要素のうち、正確に一つだけが真となり、
  - $G_{\bar{X}_i}(B)$の各要素は、次の$X_i$での通信の完了まで値を変えない。
- プローブをガードに含まない分岐$B$は、
  - 分岐時に、$B$のガードの集合$G(B)$の要素のうち、正確に一つだけが真となり、
  - $G(B)$の各要素は、次の$X_i$での通信の完了まで値を変えない。

### 3.3 記述例

例として、あるマイクロプロセッサで使われるALUモジュールの記述を以下に挙げる。

$$
\begin{aligned}
M \equiv *[&[\bar{A} \rightarrow A?op; \\
&\quad [unary(op) \rightarrow X?x; \langle z,f\rangle := uop(x,op,f); Z!z \\
&\quad \|binary(op) \rightarrow X?x \bullet Y?y; \langle z,f\rangle := bop(x,y,op,f); Z!z] \\
&\quad |\bar{F} \rightarrow F!f]].
\end{aligned}
$$

## 4 基本プロセス

以下では、プロセス分割のターゲットとなる基本プロセスを定義する。記述の中で、$A$, $B$などは単一の通信コマンドを表す。

**能動プロセス**　$*[A]$.

外部からの要求によらず、単一の通信を起動する。

**空プロセス**　$*[[\bar{A} \rightarrow A]]$.

要求を常に受け付け、即完了させる。

**同時実行プロセス**　$*[[\bar{A} \rightarrow A_1 \bullet A_2 \bullet \ldots \bullet A_n \bullet A]]$.

要求によって複数の通信$A_1 \sim A_n$を同時に実行する。

**直列実行プロセス**　$*[[\bar{A} \rightarrow A_1; A_2; \ldots; A_n \bullet A]]$.

要求によって複数の通信$A_1 \sim A_{n-1}$を順次実行し、$A_n$と$A$を同時実行する。

**調停プロセス**　$*[[\bar{A}_1 \rightarrow B_1 \bullet A_1; C_1 | \ldots | \bar{A}_n \rightarrow B_n \bullet A_n; C_n]]$.

複数の要求を調停し、衝突を防ぐ。またパイプライン機能を持つ。

**条件分岐プロセス**　$*[[\bar{A} \rightarrow [exp_1 \rightarrow A_1 \bullet A \| \ldots \| exp_n \rightarrow A_n \bullet A]]]$.

単一の要求によって起動され、直後に論理式$exp_1 \sim exp_n$により分岐し、対応する通信を実行する。$exp_1 \sim exp_n$は分岐時に正確に一つだけが真であり、通信の実行中は変化してはならない。

---

A Method for Automatic Synthesis of Control Circuits in Asynchronous Processors

Hiroto Kagotani, Hitoshi Doi, Takashi Nanya

Faculty of Engineering, Tokyo Institute of Technology

## 5 プロセス分割の手順

以上の制約を満たすプロセス記述は次の手順により基本プロセスに分割することができる。ただし、以下で$\Sigma \equiv A_1 \bullet A_2 \bullet \ldots \bullet A_n$, $\sigma \equiv \Sigma_1; \Sigma_2; \ldots; \Sigma_n$である。

1. 先頭の分岐のプローブを含むガードが他の論理式を持つ場合、同種のプローブでグループ分けし、論理式による分岐をその直後に移す。
2. 基本プロセス以外のすべてのプロセスに、以下のうち適用可能なものを任意の順序で適用して、右辺のものに置き換える。

(a) $*[\ldots;\sigma;\Sigma;\ldots]. \Rightarrow *[\ldots;X;\ldots]., *[[\bar{X} \rightarrow \sigma;\Sigma \bullet X]]$.

(b) $*[[\ldots|\bar{A} \rightarrow \ldots;\sigma;\Sigma \bullet A;\ldots|\ldots]]$.
$\Rightarrow *[[\ldots|\bar{A} \rightarrow \ldots;X \bullet A;\ldots|\ldots]]., *[[\bar{X} \rightarrow \sigma;\Sigma \bullet X]]$.

(c) $*[\ldots;[a \rightarrow A[]\ldots[]b \rightarrow B];\ldots]$.
$\Rightarrow *[\ldots;X;\ldots]., *[[\bar{X} \rightarrow [a \rightarrow A \bullet X[]\ldots[]b \rightarrow B \bullet X]]]$.

(d) $*[[\ldots|\bar{Z} \rightarrow \ldots;[a \rightarrow A_1 \bullet Z;A_2[]\ldots[]b \rightarrow B_1 \bullet Z;B_2];\ldots|\ldots]]$.
$\Rightarrow *[[\ldots|\bar{Z} \rightarrow \ldots;X_1 \bullet Z;X_2;\ldots|\ldots]]$.,
$*[[\bar{X_1} \rightarrow [a \rightarrow A_1 \bullet X_1[]\ldots[]b \rightarrow B_1 \bullet X_1]]]$.,
$*[[\bar{X_2} \rightarrow [a \rightarrow A_2 \bullet X_2[]\ldots[]b \rightarrow B_2 \bullet X_2]]]$.

(e) $*[\ldots;[\ldots[]a \rightarrow []\ldots];\ldots]$.
$\Rightarrow *[\ldots;[\ldots[]a \rightarrow X[]\ldots];\ldots]., *[[\bar{X} \rightarrow X]]$.

(f) $*[[\ldots|\bar{A} \rightarrow \ldots;[\ldots[]a \rightarrow A[]\ldots];\ldots|\ldots]]$.
$\Rightarrow *[[\ldots|\bar{A} \rightarrow \ldots;[\ldots[]a \rightarrow X \bullet A[]\ldots];\ldots|\ldots]]., *[[\bar{X} \rightarrow X]]$.

(g) $*[[\bar{Z} \rightarrow [a \rightarrow Z[]\ldots[]b \rightarrow Z]]]. \Rightarrow *[[\bar{Z} \rightarrow Z]]$.

分割された基本プロセスのいくつかは、プローブが異なる以外は全く同じ動作をするものがあり得る。この場合、その基本プロセスを共有させることでハードウェア量を削減できる。

## 6 基本回路の実現例

6通りの基本プロセスを実際の回路で実現する方法は、必ずしも一つではないが、一例として図2のように構成することができる。

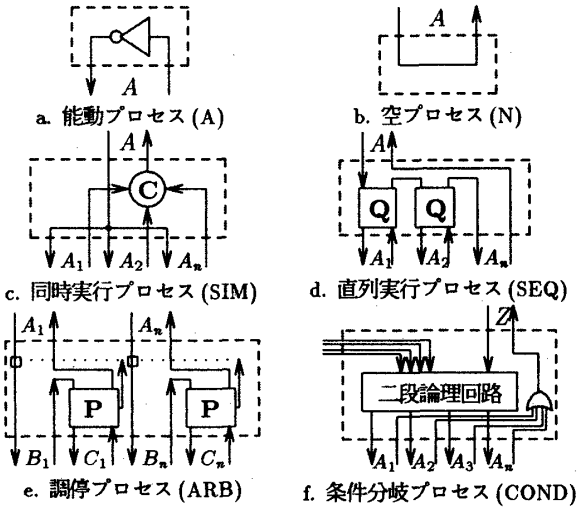


図 2: 回路モジュールの実現

図2中、Qは$*[[\bar{A} \rightarrow B;A]]$.を実現する二相制御要素(図3)で、Pは$*[[\bar{A} \rightarrow A;B]]$.を実現するパイプライン要素(図4)を表す。また図2.e中の点線はアービタを示し、パイプライン要素はその下の実行中に新たな要求が入らないようにアービタに指示する。図2.fの左側は安定したレジスタからの入力である。

## 7 基本回路の結合とモジュールの結合

基本回路間の通信路の結合は、通常単純に2本の信号線を接続すればよい。しかし、プロセス内の複数の場所で同一の通信路に対する通信を行っている場合、分割された複数の基本プロセス間で通信

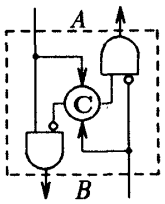


図 3: 二相制御要素(Q)
$*[[\bar{A} \rightarrow B;A]]$.

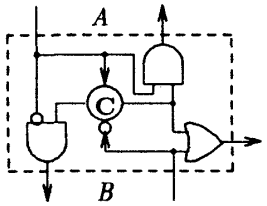


図 4: パイプライン要素(P)
$*[[\bar{A} \rightarrow A;B]]$.

を共有する必要がある。また、基本プロセスの共有をする場合にも通信路の共有が起こる。この場合は、共有しないものとして変換した各通信路を合流要素(図5)によって結合する。

また、モジュール間でプローブされていないポート同士を結合する場合、どちらもアクティブモード(すなわち4相ハンドシェークでのrequestを出す側)で実現するため直接には結合できない。この場合は同期要素(図6)を介して接続し、通信の同期をとる。

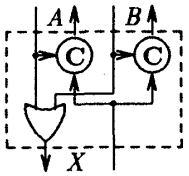


図 5: 合流要素(M)
$*[[\bar{A} \rightarrow X \bullet A[]\bar{B} \rightarrow X \bullet B]]$.

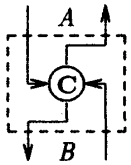


図 6: 同期要素(SYN)
$*[[\bar{A} \wedge \bar{B} \rightarrow A \bullet B]]$.

## 8 合成例

プロセス記述例として挙げたALUモジュールの制御回路合成結果を図7に示す。

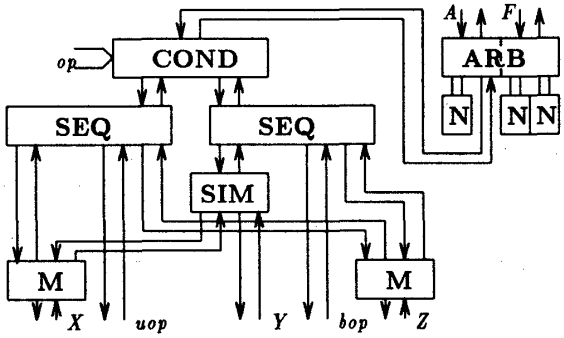


図 7: ALU 制御回路の合成

## 9 まとめ

非同期式プロセッサの機能モジュールをプロセスとして記述し、これを分割して制御回路を実現する手法を示した。プロセスの記述が、与えられた制約を満たしていることを調べる手段、およびその制約を緩和し記述を容易にすることは今後の検討課題である。

なお、本研究の一部は文部省科学研究費補助金「02452156」によって行われたものである。

## 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] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666-677, August 1978.

[3] Alain J. Martin. Synthesis of asynchronous VLSI circuits. In J. Staunstrup, editor, *FORMAL METHODS FOR VLSI DESIGN*, chapter 6, pages 237-283, Elsevier Science Publishers B.V., 1990.