2007－SLDM－131 (4)
2007／10／26

# 束データ方式による非同期式回路の動作合成システムについて

濱田 尚宏† 志賀 雄城† 齋藤 寛††

† 会津大学コンピュータ理工学研究科 コンピュータシステム学専攻
†† 会津大学コンピュータ理工学部 コンピュータハードウェア学科

**あらまし** 本稿では，束データ方式による非同期式回路を対象とした動作合成システムについて検討を行う．動作合成システムは，C 言語によるアプリケーションの動作仕様より，束データ方式による非同期式回路の論理設計を自動生成する．これまでに提案してきた演算スケジューリング，リソースアロケーション，制御合成手法に加え，ビット長解析を行なう．また，システムを一部実装し，幾つかの例に適用した結果より，今後の改善点を述べる．

# A Behavioral Synthesis System for Asynchronous Circuits with Bundled-data Implementation

Naohiro HAMADA†, Yuuki SHIGA†, and Hiroshi SAITO††

† Dept. of Computer Systems, The Graduate School of The University of Aizu
†† Dept. of Computer Hardware, The University of Aizu

**Abstract** In this paper, we propose a behavioral synthesis system for asynchronous circuits with bundled-data implementation. Proposed behavioral synthesis system synthesizes an asynchronous logic circuit from a behavioral description written in a restricted C language. We evaluate the proposed system through a experiment where some benchmark circuits are synthesized.

## 1. はじめに

LSI の微細化に伴い，配線遅延のばらつき，消費電力の増加などの問題が顕著になってきている．グローバルクロック信号で回路全体を制御する同期式回路では，クロックスキューによる同期の失敗，クロックツリーにかかる消費電力の増加といった問題が生じる．また，高クロック周波数を用いた場合には，電磁放射なども問題となってくる．一方，ローカルなハンドシェーク信号で回路を制御する非同期式回路では前述のようなクロック信号にまつわる問題は発生しない．また，低消費電力，低電磁放射といった利点もある．これらの利点より，組み込み用途で実用化がなされている [1]．しかしながら，非同期式回路の設計では，ハザードのない回路，適切な遅延モデル，制御モデルの選択が必要となるなど非常に困難である．従って，非同期回路を対象とした設計自動化手法が必要となってくる．

こうした背景より，著者らは C 言語で記述されたアプリケーションの動作記述より束データ方式による非同期式回路の論理設計を自動生成する手法を提案してきた [2], [3]．本稿では，これまでに提案してきた手法を基にし，束データ方式による非同期式回路を対象とした動作合成システムについての検討を行う．

以下に，本稿の構成を述べる．2. 節では，提案システムの中間表現となる Data Flow Graph について述べる．3. 節では，本稿で対象とする束データ方式による非同期式回路について述べる．4. 節では，提案システムについて述べる．5. 節では，実験を通して提案システムの評価を行う．最後に，6. 節で今後の展望と本稿のまとめを述べる．

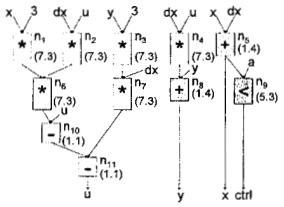


図 1 Data Flow Graph

## 2. Data Flow Graph

Data Flow Graph (DFG) とは、アプリケーションのデータフローを表した有向グラフであり，$G = (N, E)$ と表現される．$N$，$E$ はそれぞれ，ノードの集合，有向エッジの集合を表す．ノード $n_i \in N (i = 1, \ldots, m)$ は，演算の種類と遅延時間でラベル付けされる．また，$N$ には，参照ノードとして演算の開始を表す source ノードと終了を表す sink ノードが含まれる．演算は、予め用意されているリソースライブラリに定義されている演算器で実行される．演算の遅延時間はノードにラベル付けされた演算を実行するのに必要となるリソースの遅延時間や配線遅延の概算から得られた値である．有向エッジ $e_{n_i,n_j} \in E$ は，ノード $n_i$ からノード $n_j$ へのデータ依存を表している．なお，

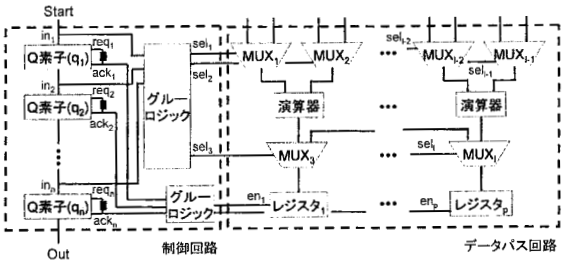


図 2　束データ方式による非同期式回路

本稿で用いる DFG では，演算を表すノード $n_i$ は 2 入力 1 出力 ($in1_{n_i}$, $in2_{n_i}$, $out_{n_i}$)，データ転送を表すノード $n_j$ は 1 入力 1 出力 ($in1_{n_j}$, $out_{n_j}$) であると仮定する．図 1 に，ラベル付けされた DFG を示す．図中の括弧内の数字は，遅延時間を表す．

## 3. 束データ方式による非同期式回路

束データ方式とは，非同期式回路におけるデータエンコーディング方式の一つである．束データ方式では，1 ビットのデータが一本の信号線に対応し，回路の制御にはローカルなハンドシェーク信号を用いる．

本稿で扱う束データ方式による非同期式回路は，図 2 に示されるようにデータパス回路と制御回路からなる．データパス回路は，各演算を実行する演算器，演算結果を保持するレジスタ，演算器とレジスタを共有するために必要となるマルチプレクサからなる．制御回路は，Q 素子 [4] と呼ばれる制御回路，遅延素子，共有リソースを制御するためのグルーロジックからなる．演算スケジューリングの結果から求められた各状態毎に一つの Q 素子が割り当てられる．各状態の実行時間は，遅延素子に依存する．

本稿で扱う束データ方式による非同期式回路の動作は以下の通りである．ある状態 $s_h(h = 1, \ldots, n)$ に対応した Q 素子 $q_h$ は，動作開始を表す入力信号 $in_h$ が 1 となると動作を開始する．また，Q 素子の入力信号 $in_h$ からデータパス回路上のマルチプレクサへの制御信号 $sel_k(k = 1, \ldots, l)$ が生成される．Q 素子 $q_h$ は，入力信号 $in_h$ が 1 になると次に，出力信号 $req_h$ を 1 にする．出力信号 $req_h$ は，対応した状態 $s_h$ に対応した遅延素子を通過し，入力信号 $ack_h$ として Q 素子 $q_h$ に入力される．また，Q 素子への入力信号 $ack_h$ からレジスタへの書き込み許可信号 $en_t(t = 1, \ldots, p)$ を生成する．Q 素子 $q_h$ は入力信号 $ack_h$ が 1 になると直ちに，出力信号 $req_h$ を 0 とし，入力信号 $ack_h$ が 0 となるのを待つ．入力信号 $ack_h$ が 0 となると，次状態 $s_{h+1}$ に対応した Q 素子 $q_{h+1}$ への入力信号 $in_{h+1}$ を 1 とする．

## 4. 束データ方式による非同期式回路の動作合成システム

図 3 に，提案する束データ方式による非同期式回路の動作合成システムの合成フローを示す．演算スケジューリング，制御回路合成，遅延時間の挿入は束データ方式による非同期式回路の性質を考慮したものであり，同期式回路を対象とした動作合成手法とは異なる．以下に，フローの各項目を解説する．

### 4.1 入　　力

提案システムの入力は，C 言語で記述されたアプリケーション

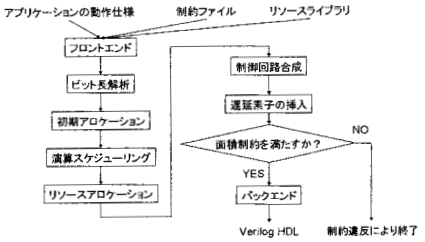


図 3　提案システム

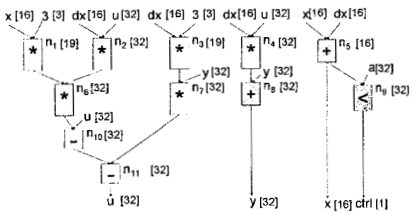


図 4　ビット長解析を行った DFG

の動作仕様記述，リソースライブラリ，設計制約の 3 つである．C 言語で記述されたアプリケーションの動作仕様は以下に制限される．

- 整数のデータ型宣言
- 代入文

リソースライブラリは，各リソースの面積，遅延時間，実行可能な演算と入出力のビット長を XML 形式で与える．設計制約は，設計目標となる時間制約と面積制約を XML 形式で与える．提案システムでは用いているアルゴリズムの性質上，時間制約の下で演算スケジューリングを行い，リソースアロケーション，制御回路合成，遅延素子の挿入後に面積制約を満たすかを判断する．面積制約を満たしていれば，バックエンドで目的となる非同期式回路の論理設計を出力する．

### 4.2 フロントエンド

フロントエンドでは，COINS [5] を用いて与えられた動作仕様記述の解析を行い，提案システムにおける中間表現である DFG を生成する．生成された DFG は，XML 記述として生成される．

### 4.3 ビット長解析

動作仕様記述上の各演算，各変数のビット長を解析し，解析結果に基づいて動作合成を行うことで，合成された回路の面積や性能が向上させることができる．本稿では，Stephenson らの提

案した手法 [6] を志賀らが拡張した手法 [3] を用いて，ビット長解析を行う．

DFG 上の各ノード $n_i$ のビット長 $b(n_i)$ を以下の情報から求める．

- 動作仕様記述における変数のデータ型
- 各ノードの入力のビット長の最大値
- ノード $n_i$ にラベル付けされた演算の種類

ビット長解析では，先ず，回路の出力に対応する変数のデータ型から各ノード $n_i$ のビット長の閾値 $b_{th}(n_i)$ を求める．動作仕様記述を C 言語で記述する関係上，変数の各データ型のビット長については，C 言語での各変数の型のビット長と一致すると仮定する．各ノード $n_i$ のビット長の閾値 $b_{th}(n_i)$ の計算後，各ノード $n_i$ の入力 $in1_{n_i}$ と $in2_{n_i}$ のビット長 $b(in1_{n_i})$ と $b(in2_{n_i})$ の最大値を動作仕様記述上の入力となる変数と定数から求める．ここまでに求められた各ノード $n_i$ の $b_{th}(n_i)$，$b(in1_{n_i})$，$b(in2_{n_i})$ とラベル付けされた演算の種類から各ノードのビット長 $b(n_i)$ を求める．

- ノード $n_i$ にラベル付けされた演算が加減算，論理演算の場合
  - $b(n_i) = min(b_{th}(n_i), max(b(in1_{n_i}), b(in2_{n_i})))$
- ノード $n_i$ にラベル付けされた演算が乗算の場合
  - $b(n_i) = min(b_{th}(n_i), b(in1_{n_i}) + b(in2_{n_i}))$
- ノード $n_i$ にラベル付けされた演算が除算の場合（被除数を $in1_{n_i}$，除数を $in2_{n_i}$ とする）
  - 除数 $in2_{n_i}$ が変数の場合: $b(n_i) = min(b_{th}(n_i), b(in1_{n_i}))$
  - 除数 $in2_{n_i}$ が定数の場合: $b(n_i) = min(b_{th}(n_i), b(\frac{in1}{in2}))$

交換則の成り立つ演算では，DFG の構造を変えることでビット長を最小化することが出来る．従って，提案システムでは，交換則の成り立つ演算についてノードの入力の入れ替えを行い，ビット長が最小となるように DFG の構造の変更を行う．

図 4 に，図 1 の DFG に対しビット長解析を行った例を示す．図中の角括弧内の数字は，解析されたビット長を示す．

### 4.4 初期アロケーション

DFG 上の各ノードのビット長解析後，解析されたビット長 $b(n_i)$ を基に，初期アロケーションを行う．初期アロケーションでは，各ノード $n_i$ にラベル付けされた演算を実行可能な演算器 $fu$ を割り当て各ノード $n_i$ の遅延時間 $d(n_i)$ を決定する．提案システムにおける初期アロケーションでは，リソース共有の最大化を図るため，同種類の演算については最大のビット長の演算と同じビット長の演算器を割り当てる．

初期アロケーション後に，As Late As Possible (ALAP) スケジュール [7] の計算を行う．ALAP スケジュールでは，ノード $n_i$ の実行を与えられた時間制約下で，可能な限り遅らせるようにスケジューリングを行う．求められた ALAP スケジュールより，リソースの使用量の概算を行い，必要となるマルチプレクサの見積もりを行う．見積もられたマルチプレクサの遅延，レジスタの遅延，演算器の遅延時間より各ノードの遅延時間を決定する．

### 4.5 演算スケジューリング

演算スケジューリングでは、各ノード $n_i$ の開始時間を決定する．提案システムでは，Paulin らが提案した Force-Directed Scheduling (FDS) アルゴリズム [8] に対し，ビット長を考慮し，束データ方式に適用した Asynchronous FDS (AFDS) アルゴリズム [3] を用いる．

FDS アルゴリズムは，与えられた時間制約の下，使用されるリソースの数が最適となるように演算スケジューリングを行う時間制約アルゴリズムである．FDS アルゴリズムでは，時間制約を一定間隔のコントロールステップ（以下ステップと呼ぶ）に分割し，ステップ全体で利用されるリソースの数が最適となるように演算スケジューリングを行う．ここで、ステップが一定間隔を持つのはクロック信号を想定しているためである．

一方，非同期式回路ではクロック信号が存在しないため，必ずしもステップを一定間隔とする必要はない．非同期式回路において演算は，前の演算の終了によって開始されるので，演算の開始時間候補の概算を行い，概算された開始時間候補よりステップを求めたほうが計算効率が良い [9]．以下では，AFDS アルゴリズムの解説を行う．

AFDS アルゴリズムでは，ステップの計算を行うに先立って，As Soon As Possible (ASAP) スケジュール [7] と ALAP スケジュールの計算を行う．ASAP スケジュールでは，ノード $n_i$ は実行可能になったら，直ちに演算が開始されるようにスケジューリングされる．ここで，ASAP スケジュールと ALAP スケジュールにおけるノード $n_i$ の開始時間をそれぞれ $ALAP_{start}(n_i)$，$ASAP_{start}(n_i)$ とする．ASAP スケジュールと ALAP スケジュールにおけるノード $n_i$ の終了時間は，それぞれの開始時間にノード $n_i$ の遅延時間 $d(n_i)$ を加算することで求まる．ASAP スケジュールと ALAP スケジュールより，ノード $n_i$ のスケジューリング可能な時間間隔 $TimeFrame(n_i)$ が次のように求まる．

$$TimeFrame(n_i) = ALAP_{start}(n_i) - ASAP_{start}(n_i)$$

全てのノードのスケジューリング可能な時間間隔の計算後，ノード $n_i$ の開始時間候補の集合 $StartTime(n_i)$ を求める．齋藤らの提案した手法 [10] では，依存関係のあるノード $n_j(\forall e_{n_j,n_i} \in E)$ とリソースの共有が可能なノードから開始時間候補の集合 $StartTime(n_i)$ を求めていた．しかし，本稿で検討している回路モデルでは，リソースの共有が行われていなくてもノード $n_i$ の直前に終わるノード $n_j$ の終了がトリガーとなってノード $n_i$ の演算が開始することが出来る．即ち，$ASAP_{start}(n_i) \leqq ASAP_{start}(n_j) + d(n_j) \leqq ALAP_{start}(n_i)$ となるノード $n_j$ はノード $n_i$ の直前に終了する可能性のあるノードの一つとなる．従って，ノード $n_i$ の開始時間候補の集合 $StartTime(n_i)$ はノード $n_i$ と依存関係のある直前のノード集合 $DirectPred(n_i)$，依存関係のない直前に終わる可能性のあるノード集合 $Pred(n_i)$ より求められる．

- $DirectPred(n_i) = \{\forall n_j, e_{n_j,n_i} \in E\}$
- $Pred(n_i) = \{n_j | n_j \notin DirectPred(n_i)$ かつ，ノード $n_i$ の直前に終了する可能性のあるノード $n_j\}$

$DirectPred(n_i)$ と $Pred(n_i)$ の計算後，ノード $n_i$ の開始時間候補の集合を次のように求める．

$$\begin{aligned} StartTime(n_i) = \{et_{n_j} | & \\ & \forall n_j \in \{DirectPred(n_i) \cup Pred(n_i)\}, \\ & et_{n_j} = ASAP_{start}(n_j) + d(n_j), \\ & ASAP_{start}(n_i) \leqq et_{n_j} \leqq ALAP_{start}(n_i)\} \\ & \cup \{ASAP_{start}(n_i), ALAP_{start}(n_i)\} \end{aligned}$$

ここで，$et_{n_j} \in StartTime(n_i)$ はノード $n_i$ の開始時間候補となる．また，ノード $n_i$ の前に終わる可能性のあるノード $\forall n_j \in DirectPred(n_i) \cup Pred(n_i)$ についても，同様に開始時間候補の集合 $StartTime(n_j)$ を求める．上記の計算を再帰的に行うことで，全てのノードの開始時間候補の集合を求める．ここで，求められた全てのノードの開始時間候補の集合の和集合を求めることで，スケジューリング可能なステップの集合 $Step = \{step_k | k = 1, \ldots, c\}$ を得る．

AFDS アルゴリズムでは，ステップの計算以外は，特に FDS アルゴリズムと同じである．全てのノードについて，開始時間候補の集合 $StartTime(n_i)$ の計算後，Probability と Distribution Graph の計算を行う．Probability とは，ノード $n_i$ がステップ $step_k$ にスケジューリングされる確率を示している．Probability の計算後，Distribution Graph という各ステップ毎に利用されるリソース数の見積もりを表したグラフを計算する．Probability と Distribution Graph の計算後，$SelfForce(n_i, step_k)$ の計算を行う．$SelfForce(n_i, step_k)$ は，ノード $n_i$ をステップ $step_k$ にスケジューリングした場合に，どの程度リソースの使用量が最適になるかを表すコスト関数である．$SelfForce(n_i, step_k)$ の値が，負の場合にはリソースの使用量が最適になることを示す．従って，AFDS アルゴリズムでは，最小となる $SelfForce(n_i, step_k)$ を持つノード $n_i$ を当該ステップ $step_k$ にスケジューリングする．最小となる $SelfForce(n_i, step_k)$ が複数あった場合，すでにスケジューリングされたノードと同じ開始時間にスケジューリングすることにより，状態数を抑える．提案システムでは，スケジューリング候補が複数存在した場合には，すでにスケジューリングされたノードと同じ開始時間にスケジューリングできるものを優先してスケジューリングを行う．

上記の手順を全てのノードがスケジューリングされるまで繰り返す．

### 4.6 リソースアロケーション

リソースアロケーションでは，各演算と各変数に演算器とレジスタの割り当てを行い，演算器とレジスタの共有が行われた場合にはマルチプレクサの割り当ても行い，データパス回路の生成を行う．

提案システムでは，Left-Edge (LE) アルゴリズム [11] に対し，ビット長の考慮と割り当てられるマルチプレクサ数が最小となるように拡張を施す．この拡張された LE アルゴリズムを本稿では Modified LE (MLE) アルゴリズムと呼ぶ．MLE アルゴリズムの概要は表 1 に示す．以下では，MLE アルゴリズムの解説を行う．

MLE アルゴリズムは，ノードリスト $L$ を用いてリソースアロケーションを行う．ここで，ノードリスト $L = \{(n_i, st_{n_i}) | \forall n_i \in N\}$ は，ノード集合に含まれるノード $n_i$ とそのノードの開始時間 $st_{n_i}$ の二項関係として定義する．また，ノード $n_i$ がリソース $r_l$ に割り当てられていることを示す割り当て変数 $map$ を定義する．割り当て変数 $map[n_i] = l$ のとき，ノード $n_i$ はリソース $r_l$ に割り当てられていることを表す．MLE アルゴリズムでは，リソースアロケーションに先立って，各ノードについての割り当て変数 $map[n_i](\forall n_i \in N)$ を 0 に初期化し，ノードリスト $L$ を開始時間について昇順にソートする．その後，演算スケジューリングの結果から必要となるリソースの最小数を求め，求められた最小値を上限値 $maxIndex$ とし，この値を超えないようにリソースアロケーションを行う．また，ノードリスト $L$ より各ノードの開始時間の集合 $S$ の計算を行う．次に，現在までに割り当てられたリソースの数を示す $index$ を 1 とし，割り当ての行われなかったノード集合を表す $P$ を $\emptyset$ とし，リソースの割り当てを行っていく．

表 1　Modified Left-Edge アルゴリズム

```
function Modified_Left_Edge(L)
for ∀n_i ∈ L do map[n_i] = 0
Sort(L) ; maxIndex = CalculateMinSize(L)
S = GetStartTime(L) ; index = 1 ; P = ∅
for ∀s_j ∈ S do
    SN = GetSameTimeIntervalSet(L, s_j)
    Map_Node_Set(SN, map, index, maxIndex)
end-for
while P ≠ ∅ do
    for ∀s_j ∈ S do
        SN = GetSameTimeIntervalSet(P, s_j)
        Map_Node_Set(SN, map, index, maxIndex)
    end-for
end-while
return map
function Map_Node_Set(SN, map, index, maxIndex)
    for ∀n_i ∈ SN do
        for l = 1 to index do
            Pr[n_i][r_l} = CalculatePriority(map, l, n_i)
        end-for
    end-for
    for l = 1 to index do
        pr = GetHighestPriority(Pr, map, l)
        n_i = GetHighestPriorityNode(Pr, map, l, pr)
        if pr > 0 then
            map[n_i] = l
            P = P \ {n_i}
        else if index ≦ maxIndex then
            map[n_i] = index
            P = P \ {n_i}
            index = index + 1
        else P = P ∪ {n_i}
        end-if
    end-for
```

MLE アルゴリズムでは，開始時間順に各ノードを各リソースに割り当てていく．まず，開始時間 $s_k \in S$ と同じ開始時間にスケジューリングされたノード集合 $SN = \{n_i | st_{n_i} = s_k\}$ を求める．次に，すでに割り当てられているノードに対する割り当て変数 $map$ について，現在割り当てを行っているノード $n_i \in SN$ との優先度 $Pr[n_i][r_l]$ を計算する．優先度 $Pr[n_i][r_l]$ は次のように計算を行う．

$$Pr[n_i][r_l] = \sum_{\substack{j=1 \\ (map[n_j]=l)}}^{m} \Big( SameIONumber(n_i, n_j) + (b(n_j) - b(n_i)) \Big),$$

計算式中の $SameIONumber(n_i, n_j)$ はノード $n_i$ と $n_j$ に共通する入出力の数を計算する関数である．$b(n_j) - b(n_i)$ では，すでに割り当てられているノード $n_j$ と割り当てを行っているノード $n_i$ とのビット長の差を求める．この値が負となる場合は，すでに割り当てられているノード $n_j$ のビット長を超えるため，よりビット長の大きいリソースを割り当てる必要があることを意

味する．0より大きい場合は，すでに割り当てられているノード $n_j$ のビット長に対応したリソースにノード $n_i$ の割り当てを行っても，よりビット長の大きなリソースが必要とならないことを意味する．なお，ノード $n_i$ をリソース $r_l$ に割り当てることが出来ない場合は，$Pr[n_i][r_l] = -1$ とし，割り当てが行われないようにする．全てのノード $n_i \in SN$ についての優先度 $Pr[n_i][r_l]$ の計算後，現在割り当てを行っているリソース $r_l$ に対する優先度 $Pr[n_i][r_l]$ が最大となるノード $n_i$ が存在した場合，リソース $r_l$ に対しノード $n_i$ を割り当てる．また，優先度 $Pr[n_i][r_l]$ が0以下の値であった場合，リソースに余りがあれば，余っているリソースに対して，ノード $n_i$ の割り当てを行う．ここで割り当てが行われなかったノード $n_i$ は $P$ に加える．以上の手順を各開始時間について行っていく．

次に，上記の手順で割り当ての行われなかったノード $n_i \in P$ についても同様の手順で割り当てを行っていく．このようにして，リソースアロケーションを行うことでビット幅の考慮を行うと同時に割り当てられるマルチプレクサ数を減らすことが出来る．

演算器とレジスタの割り当て後，共有された演算器とレジスタに対しマルチプレクサの割り当てを行い，データパス回路を生成する．

### 4.7 制御回路合成

制御回路合成では，演算スケジューリングとリソースアロケーションの後に生成されたデータパス回路の動作を制御する回路を生成する．制御回路合成に先立って，演算スケジューリングの結果より状態 $State$ を決定する．

$$State = \{st_{n_i} | \forall n_i \in N\}$$

ここで，$st_{n_i}$ はスケジューリングで決まったノード $n_i$ の開始時間を表す．また，$st_{n_i} \in State$ を昇順にソートし，第 $h$ 番目の要素を $s_h$ とする．

決定された各状態 $s_h \in State$ に対し，一つの Q 素子 $q_h$ を割り当てることで，制御回路を合成する．このような方針を採ることにより，状態数が多いアプリケーションであっても短時間で制御回路の合成が出来るという利点がある．

制御回路の合成後，マルチプレクサ制御信号 $sel_k$ とレジスタ書き込み許可信号 $en_t$ を各 Q 素子の入力信号 $in_h$ と $ack_h$ から生成するために，グルーロジックを挿入する．

### 4.8 遅延素子の挿入

生成されたデータパス回路上の各演算器，マルチプレクサ，レジスタの遅延時間より各状態 $s_h$ に対応した遅延時間の概算を行う．ここで概算された遅延時間を基に，各状態毎に遅延素子を生成する．遅延素子は AND ゲートやインバータなどで構成された論理ゲートのチェーンとして生成される．なお，バックエンドツールを用いて配置配線を行った際に判明する配線遅延による影響を考慮し，概算された遅延時間に対して，幾分かのマージンを加えて遅延素子を生成する．

### 4.9 出力と機能検証

提案システムでは，合成された回路と合成された回路に対応したシミュレーションモデルを出力する．これらは，Verilog HDL 記述で出力される．このように出力することで，バックエンドツールやシミュレーションツールとの親和性を高めることが出来る．また，合成された回路では Q 素子と遅延素子の間にフィードバックループが存在する．論理シミュレータを用いて検証を行う際，このようなフィードバックループが存在すると論理シミュレータは出力を生成するときにどの時点でのフィードバックループを用いてよいか認識できなくなり，正しい値を出力しなくなる．このため，シミュレーションモデルでは，フィードバックループに一定の遅延を付加している．

表 2　ベンチマーク

| ベンチマーク | ノード数 |
|---|---|
| dct | 84 |
| ewf | 34 |
| fft | 130 |

表 3　リソースライブラリ

| リソース名 | ビット長 | 面積 [スライス数] | 遅延時間 [ns] |
|---|---|---|---|
| mux2 | 32 | 9 | 0.15 |
| mux3 | 32 | 32 | 1.23 |
| mux4 | 32 | 30 | 0.43 |
| mux5 | 32 | 35 | 1.47 |
| mux6 | 32 | 40 | 2.18 |
| mux7 | 32 | 45 | 1.92 |
| mux8 | 32 | 50 | 0.65 |
| reg | 32 | 32 | 0.45 |
| sub | 16 | 8 | 1.16 |
| shiftL | 16 | 38 | 2.82 |
| shiftR | 16 | 38 | 2.89 |
| add | 32 | 16 | 1.37 |
| sub | 32 | 16 | 1.04 |
| mult | 32 | 0 | 7.25 |

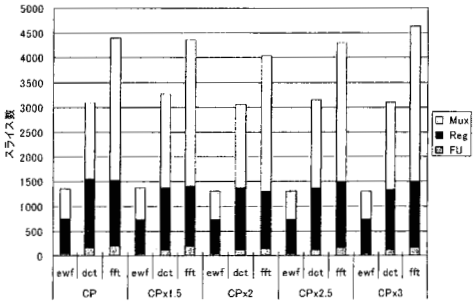


図 5　合成された回路の面積

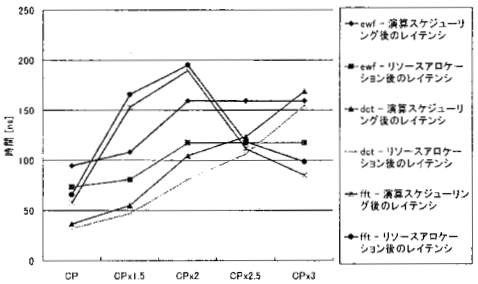


図 6　合成された回路の性能

## 5. 実験結果

実験では，提案システムを用いて幾つかのベンチマークアプリケーションの動作合成を行い，得られた回路について評価を行う．実験で用いたベンチマークの中には，提案システムで扱えない記述が存在するため，該当箇所については修正，または削除し回路を合成した．表 2 に，実験に用いたベンチマークアプリ

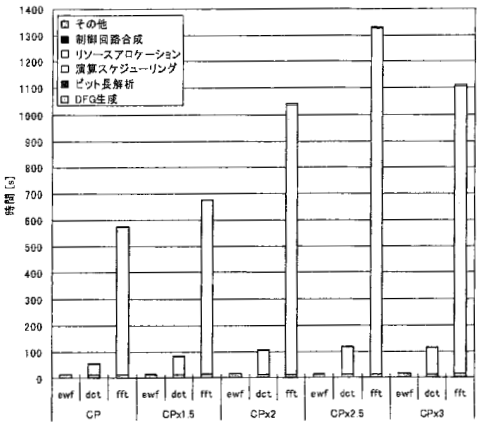


図 7　動作合成にかかった時間

ケーションと DFG のノード数を示す．提案システムは Java を用いて実装し，Pentium4 プロセッサ (3GHz) 一つと 2GB のメモリを搭載した Windows マシンを用いて合成を行った．

表 3 に実験で用いた各リソースのビット長，面積，遅延時間を示す．これらの値は，Verilog HDL で各リソースの動作を記述し，Xilinx 社の ISE WebPACK 9.2i [12] を用い，Vertex-4 を対象デバイスとして，論理合成を行った結果得られた値である．Vertex-4 では論理回路を構築するのにスライスと呼ばれるプログラマブルな論理ブロックを用いている．Vertex-4 では自らに搭載されている専用の乗算器を利用して乗算を行われるため，スライスを利用しない．本稿では，リソースの面積をスライス数として算出したため，上記の理由より乗算器の面積は 0 とし，実験を行った．また，時間制約はクリティカルパス (CP) の長さから 0.5 刻みに 3 倍まで変更しながら，実験を行った．

図 5，6 にそれぞれ，合成された回路の面積と性能を示す．図 7 に動作合成にかかった時間を示す．

次に，動作合成によって得られた回路に対し，Verilog HDL にてテストパターンを記述し，Mentor Graphics 社の ModelSim [13] を用いてシミュレーションを行った．動作の正しさを確認するために，Verilog HDL で記述したテストパターンを C 言語で記述し，C 言語レベルでシミュレーションを行ったものと結果を照合し，合成された回路が正しく動作することを確認した．

初期アロケーションでマルチプレクサとレジスタの遅延時間を考慮することで，演算スケジューリング後のレイテンシとリソースアロケーション後でのレイテンシに大きな差が生じないことが判った．従って，提案システムを用いることで，動作合成の段階である程度の精度で回路の評価を行うことが出来る．また，fft ではリソースアロケーション後にレイテンシが演算スケジューリング後のレイテンシよりも大きくなっている．これは，見積もられたマルチプレクサの遅延時間と割り当てられたマルチプレクサの遅延時間が異なっていることが原因である．fft では，多くのノードで初期アロケーション後に見積もられたマルチプレクサの遅延時間に対してリソースアロケーション後のマルチプレクサ遅延が上回っていた．この結果，リソースアロケーション後のレイテンシが大きくなってしまった．

動作合成にかかった時間については，ノード数が 80 程度の DFG であっても 2 分程度で合成が出来た．ノード数が 100 を超える DFG であっても，10 分から 20 分程度で非常に短い時間で合成を行うことが可能である．しかしながら，演算スケジューリングにかかる時間がその大部分を占めているため，現在の解を維持したまま，演算スケジューリングをより早く実行することが今後の課題として挙げられる．

## 6. まとめと今後について

本稿では，束データ方式による非同期式回路の動作合成システムについての検討を行った．提案システムを一部実装し，ベンチマーク回路を用いて評価を行った．提案システムでは、4.1 節に示されたように，扱える動作仕様が限られていることから，今後，扱える動作仕様を拡張していく．分岐やループといった制御構造を扱うために，制御回路に変更を加えるとともに演算スケジューリングとリソースアロケーションでも制御構造を考慮した変更を加える必要がある．配列を扱うために，メモリモデルの定義を行う必要がある．浮動小数点を扱うために，浮動小数点演算を実行できる演算器の準備を行う．また，スループットの向上を目的とし，パイプライン化された非同期式回路も扱えるように拡張を行っていく．パイプライン化された非同期式回路では，制御モデルを考慮するとともにパイプライン化を考慮した動作合成も考える必要がある．

## 謝　辞

本研究は，文部科学省科学技術研究費補助金 若手研究 (B)(課題番号 18700047) の研究助成による．

## 文　献

[1] ARM Ltd., “ARM996HS,” http://www.arm.com/
[2] 濱田，小西，齋藤，米田，南谷，“束データ方式による非同期式回路の動作合成手法の提案”，信学技報 VLD (デザインガイア 2006), vol.106, no.387, pp. 71-76, 北九州, 2006 年 11 月
[3] 志賀，濱田，小西，齋藤，“ビット長を考慮した束データ方式による非同期式回路の動作合成手法の提案”, DA シンポジウム，2007.
[4] F.U. Rosenberger, C.E. Molnar, T.J. Chaney, and T.-P. Fang, “Q-Modules: Internally Clocked Delay-Insensitive Modules,” IEEE Transactions on Computers, vol. 37, no. 9, pp. 1005–1018, Sept., 1988
[5] COINS-project, A compiler infra structure http://www.coins-project.org
[6] M. Stephenson, J. Babb, and S. Amarasinghe, “Bitwidth Analysis with Application to Silicon Compilation,” in Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, pp.108–120, 2000.
[7] G. De Micheli, “Synthesis and Optimization of Digital Circuits,” McGraw-Hill Higher Education, 1994
[8] P. G. Paulin and J. P. Knight, “Force-Directed Scheduling for the Behavioral Synthesis of ASIC’s,” IEEE Transactions on Computer-Aided Design, vol.8, no.6, pp.661–679, June 1989.
[9] 齋藤，米田，南谷，“Force-Directed Scheduling 手法の非同期式回路への適用と評価”, DA シンポジウム, pp.37–42, 2005.
[10] 齋藤，米田，南谷，“Integer Linear Programming を用いた束データ方式による非同期式回路のスケジューリング”, DA シンポジウム, pp.43–48, 2006.
[11] D. Gajski, N. Dutt, A. Wu, and S. Lin, High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers, 1992.
[12] XILINX Inc, ISE WebPACk 9.2i http://www.xilinx.com
[13] Mentor Graphics Corp, ModelSim http://www.mentor.com