設 計 自 動 化 79－5
（1996．2．9）

# エントロピーCODECの高位合成手法

鈴木 克青　戸川 望　佐藤 政生　大附 辰夫

早稲田大学理工学部電子通信学科
〒169 東京都新宿区大久保 3-4-1
E-mail: suzuki@sato.comm.waseda.ac.jp

**あらまし**

画像符号化処理において，エントロピー符号化・復号化は，高位合成CADとFPGAの組合せによって，高速かつ柔軟なハードウェアとして実現できる．本稿では，そのようなCADシステムの核となる，エントロピーCODECの動作記述を入力としたスケジューリング・アロケーション手法を提案する．スケジューリング処理では，制御の流れを表すグラフ（CFG）を入力とし，CFGの各節点を縮退させることで、実行時間が短く，ハードウェアコストの小さい結果を得る．アロケーション処理では，各演算をビット長の異なる機能ユニットに割り当てることを可能とする．このような処理によって，条件分岐が多く，かつ変数のビット長が異なるようなエントロピーCODECの動作記述から効率良くRTレベルの記述を得ることができる．提案手法を計算機上に実装し，評価実験を行った結果を報告する．

キーワード　高位合成，エントロピーCODEC，スケジューリング，アロケーション，コントロールフローグラフ

# High Level Synthesis for Entropy CODEC

Katsuharu SUZUKI　Nozomu TOGAWA　Masao SATO　Tatsuo OHTSUKI

Dept. of Electronics and Communication Engineering
Waseda University
3-4-1 Okubo, Shinjuku, Tokyo 169, Japan
E-mail: suzuki@sato.comm.waseda.ac.jp

**Abstract**

Entropy coding/decoding is implemented on FPGAs as a fast and flexible system in which high level synthesis technologies are key issues. In this paper, we propose scheduling and allocation algorithms for behavioral description of Entropy CODEC. The scheduling algorithm employs CFG as input and finds a solution with minimal cost and execution time by degenerating nodes in CFG. The allocation algorithm assigns each operation to functional units with various bit length. As a result RTL description is efficiently obtained from behavioral description of Entropy CODEC with many conditional branches and variable bit length. Experimental results demonstrate its efficiency and effectiveness.

**Key Words** *high level synthesis, entropy CODEC, scheduling, allocation, control flow graph*

## 1 まえがき

画像符号化は，画像データを，そのデータのもつ冗長性を削減してできるだけ短いビット列に変換する処理である．現在，多くの画像符号化アルゴリズムが提案されており [7],[8]，JPEG，MPEG などの標準化されたアルゴリズムが多数存在する [4],[8]．これらの符号化アルゴリズムの多くは，DCT 等の直交変換処理と，変換されたデータに最適な符号を割り当てるエントロピー符号化処理から構成される，

これらの処理を実現する CODEC（COder and DECoder）に DSP（Digital Signal Prcessor）を適用した場合，積和演算が主体となる直交変換処理は容易に実現できる．ところが，エントロピー符号化処理は (1) 条件分岐が多い，(2) 各データのビット長が異なるという理由から処理効率が直交変換処理ほど高くならない．エントロピー符号化部を布線論理で実現すれば，CODEC 全体のプログラマビリティが失われ効率的でない．

文献 [5] では，処理の高速性とプログラマビリティを同時に実現するためにエントロピー CODEC の動作を FPGA（Field-Programable Gate Arrays）によって実現する手法が提案されている．しかし，この手法では，ハードウェア記述言語を入力としているため，設計者はエントロピー符号化のアルゴリズムそのもの以外にタイミング，機能ユニットのビット長等を考慮して動作を記述する必要がある．完全なトップダウン設計環境を実現するためには，アルゴリズム記述にこれらの情報を付加するシステムが必要になる．

本稿では，時間情報のないエントロピー CODEC の動作記述から，タイミングおよび機能ユニットのビット長の情報を含んだハードウェア記述言語を生成する高位合成手法を提案する．本手法は，以下に示す特徴をもつ．

- エントロピー CODEC の動作記述には，条件分岐が多いため，制御の流れを表すコントロールフローグラフ（CFG）を入力として，CFG の各演算節点を縮退することによってスケジューリングを行う．
- スケジューリング結果に対して，機能ユニットに必要なビット長が最小になるように，演算のアロケーションを行う．

以下，本稿で扱う高位合成問題を定式化し，条件分岐，ビット長の異なる変数を考慮したスケジューリング・アロケーション手法を提案する．最後に，計算機上での実験結果を示し，提案手法の有効性を検証する．

## 2 エントロピー CODEC の高位合成問題

コントロールフローグラフ $CFG(V,E)$は，制御構造を含まない実行単位を節点とし，制御の流れを枝とする有向

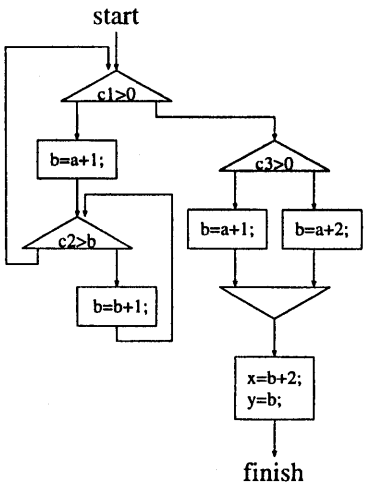


図 1: CFG

グラフである（図 1）．節点集合 $V$は，分岐節点集合 $V_f$，結合節点集合 $V_j$ および演算節点集合 $V_o$からなる．ここで，分岐節点とは，制御の流れがある条件によって分岐する節点のことをいい，図 1では，上向きの三角形で表している．結合節点とは，2 つ以上の制御の流れが 1 つに合流する節点のことをいい，図 1では，下向きの三角形で表している．演算節点集合とは，分岐節点，結合節点以外の節点のことをいい，図 1では，矩形で表している．$CFG(V,E)$ の各節点 $v \in V$は演算集合 $O_v$を持つ．$CFG$ の節点 $v \in V$の持つ演算 $o_1 \in O_v$と $o_2 \in O_v$は互いにデータ依存（例えば文献 [6] を参照）を持たないとする．演算集合全体の集合を $O = \bigcup_{v \in V} O_v$ とする．

制御ユニットは，有限状態機械（FSM）で表され，そのステート集合を $S$とする．高位合成によって，$CFG$ の節点 $v \in V$はステート $s \in S$に割り当てられる．節点 $v$がステート $s \in S$に割り当てられたとき，$v$の持つ全ての演算は $s$ に割り当てられる．

機能ユニットは，演算を実行するユニットである．機能ユニットの集合を $FU$とする．機能ユニット $fu \in FU$は タイプ $type(fu)$ をもつ．タイプの集合を $T$とする．タイプ $t \in T$の機能ユニットの個数の上限 $N_t$はあらかじめ与えられているものとする．高位合成によって，$CFG$ の演算 $o \in O$は機能ユニット $fu \in FU$に割り当てられる．

エントロピー CODEC の高位合成問題は，コントロールフローグラフ $CFG(V,E)$ と使用できる機能ユニットの個数 $N_t$ を入力とし，節点 $v \in V$のステート $s \in S$への割当ておよび演算 $o \in O$の機能ユニット $fu \in FU$への割当てを行うことをいう．その際，タイプ $t$の機能ユニットの個数が $N_t$を越えないという制約（資源制約と呼ぶ）のも

とで，制御ユニットのステート数と機能ユニットのビット数を最小化することを目的とする．

## 3 エントロピー CODEC の高位合成手法

本章では，条件分岐と，変数のビット長が異なることを考慮した高位合成手法を提案する．本手法は，既存の高位合成手法 [1],[2] と同様に，演算をコントロールステップに割り当てるスケジューリングと，演算を機能ユニットに割り当てる演算アロケーションの 2 つの段階から構成される．

スケジューリング処理では，CFG を入力とし，CFG の節点を分岐節点と結合節点を境界として分割した後，資源制約を満たしている限り境界を取り除いていくことによって，制御ユニットのステート数ができるだけ最小となるように CFG の節点 $v \in V$ をステート $s \in S$ に割り当てる．

アロケーション処理では，節点のステートへの割り当てが決定した CFG を入力とし，必要となるビット長の少ない演算同士をできるだけ同一の機能ユニットに割り当てる．機能ユニットのビット長の最小化が期待される．

以下，スケジューリング手法（3.1 節），アロケーション手法（3.2 節）を提案する．

### 3.1 スケジューリング手法

第 1 段階のスケジューリングでは，コントロールフローグラフ $CFG(V,E)$ とタイプ $t$ の機能ユニットの個数の上限 $N_t$ を入力とし，演算 $o \in O$ のステート $s \in S$ への割り当てを行う．その際，タイプ $t$ の機能ユニットの個数が $N_t$ を越えないという制約のもとで，制御ユニットのステート数の最小化を目的とする．

提案するスケジューリング手法を示す．

1. 条件分岐の前後でステートに分割（初期分割）
2. 資源制約の範囲内で，資源コストがなるべく増加しないようにステートを縮退（ステート縮退）

#### 初期分割

文献 [3] で提案されている CFG を対象としたスケジューリング手法にならい，はじめに，CFG の分岐節点の前および結合節点の後を境界として CFG を分割する．分割された 1 つの部分グラフ内にある全ての節点を同じステートに割り当てる．

#### ステート縮退

次に，初期分割で得られたスケジューリング結果を初期解としてステートを縮退する．本手法では，ステートの縮退を行うために，State Relation Graph（SRG）と呼ばれるグラフを内部表現として用いる．

**SRG** SRG は節点集合 $VS$，遷移枝集合 $EST$ および依存枝集合 $ESD$ からなるグラフである．節点 $vs \in VS$ は，CFG の部分グラフが割り当てられたステートを表す．遷移枝 $est \in EST$ は，節点間にある制御の流れを表し，節点 $vs_i \in VS$ の表すステートが節点 $vs_j \in VS$ の表すステートに遷移するとき $vs_i$ から $vs_j$ に向かって付加される．依存枝 $esd \in ESD$ は，節点間にあるデータ依存を表し，節点 $vs_i \in VS$ の表すステートに割り当てられた演算が節点 $vs_j \in VS$ に割り当てられた演算に対してデータ依存があるとき，$vs_i$ から $vs_j$ に向かって付加される．

**$enable(est)$** 遷移枝 $est \in EST$ は，その枝の結ぶ 2 つの節点が表すステートが 1 つのステートに縮退できるならば $TRUE$，縮退できなければ $FALSE$ となるような 属性 $enable(est)$ をもつ．

**$weight(est)$** $enable(est) = TRUE$ である依存枝 $est$ はその枝が結ぶ 2 つの節点を縮退したときの機能ユニットのハードウェアコスト

$$\sum_{t \in T} C_t \times n_t$$

を重み $weight(est)$ としてもつ．ここで，$C_t$ はタイプ $t$ の機能ユニットのハードウェアコストを，$n_t$ はタイプ $t$ の機能ユニットの個数を表す．

本手法では，初期分割された CFG から SRG を生成する際，スケジューリングを容易にするため以下の規則を加える．

1. CFG にあるフィードバック枝に対応する枝を SRG では付加しない
2. CFG において，SRG の節点 $vs$ が表す部分グラフに向かう枝が複数ある場合，$vs$ に向かう遷移枝 $est \in EST$ は $FALSE$ の属性をもつ

図 2に図 1の CFG に対して初期分割を行い，得られた SRG を示す．同図 (a) は CFG に対する初期分割の結果である．図中 CFG で，直線が引かれている枝は，分割の境界となっている枝であり，その枝によって分割された部分グラフは点線で囲まれている．同図 (b) は，初期分割の結果得られた SRG である．図中丸で表された節点は，(a) の点線で囲まれた部分グラフが割り当てられたステートに対応し，実線，点線で描かれた枝はそれぞれ遷移枝，依存枝を表す．なお，2 本の平行な直線が横切っている遷移枝は，属性 $enable$ が $FALSE$ である枝，すなわち，その枝の結ぶ 2 つの節点が縮退できないことを示している．図 2 では，state1 → 2，1 → 3，3 → 4 に向かって遷移枝が引かれており，内，1 → 2 の枝は，CFG において state2 へと

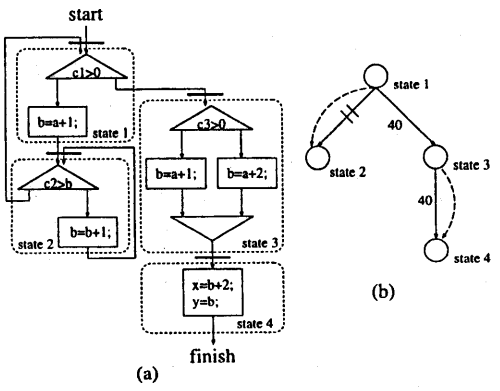


図 2: SRG

向かう枝が2本あるために，縮退不可となっている．依存枝はstate1 → 2，3 → 4へと引かれている．

このように得られたSRGの節点を縮退することで，ステート数を最小化する．節点の縮退は，重みが最小となる遷移枝が結ぶ2つの節点を縮退する操作を，資源制約の範囲内で，もはや枝が存在しなくなるまで繰り返すことによって行う．

本スケジューリング手法を以下に示す．

**Step 1** 条件分岐の前，および結合の後でCFGを分割する．

**Step 2** Step 1の結果からSRGを構築する．

**Step 3** 縮退可能な遷移枝に重みを付ける．

**Step 4** 未処理の遷移枝がなくなるか，機能ユニットの個数が制約を越えるまで以下を繰り返す．

**Step 4.1** 重みが最小の枝を探索する．

**Step 4.2** 重み最小の遷移枝が結ぶ節点を縮退し，遷移枝の重みを更新する．

Step 4.1で，重みが最小となる遷移枝が2つ以上ある場合には，それらの内で，縮退すると依存枝の始点と終点が同じ節点に含まれるようになるような縮退を優先的に行う．これは，依存枝の始点と終点が同じ節点になることによって，変数の定義と参照が1ステート内で完了するために，回路合成の際，変数を記憶ユニットではなく，接続ユニットにマッピングできるからである．

図3に，図2(b)のSRGの節点が縮退される過程を示す．同図(a)のSRGでは，最小の重み40を持つ遷移枝が2つある．この場合，依存枝の始点と終点が同じ節点になるよう3 → 4の遷移枝が結ぶ節点を縮退する．その結果，SRGは同図(b)のようになる．同様に最小の重み40

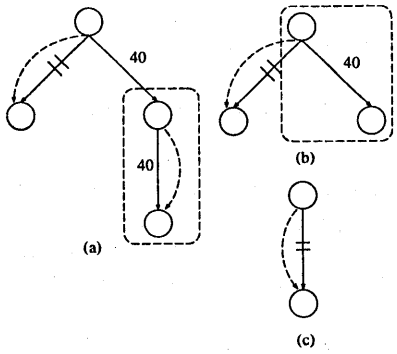


図 3: ステートの縮退

をもつ遷移枝が結ぶ節点が縮退され，処理を終了する（同図(c)）．ステート縮退の結果，初期分割（図2(a)）におけるstate1，3，4が1つのステートに縮退された．

### 3.2 アロケーション手法

第2段階のアロケーションでは，全ての演算$o \in O$のステート$s \in S$への割当てが決定したコントロールフローグラフ$CFG(V,E)$を入力とし，演算$o \in O$の機能ユニット$fu \in FU$への割り当てを行う．その際，機能ユニットのビット数を最小化することを目的とする．

提案するアロケーション手法の概要は以下のようになる．

1. 各機能ユニットのビット数の決定（ユニット選択）
2. 演算の機能ユニットへの割付け（ユニット割付け）

これらの処理は，アロケーションテーブルを用いて行われる．アロケーションテーブルはstate $i$で機能ユニット$fu_j$に割り当てられた演算$o_{ij}$を$i$行$j$列に保持する．図4にアロケーションテーブルの例を示す．ただし，$i$行$j$列に保持されている数値でstate $i$において機能ユニット$fu_j$が必要とするビット数を表す．この例では，state 1には演算が3つ存在し，それぞれの演算に必要となるビット数は12bit，8bit，2bitであることを示している．他のstateも同様である．以下，アロケーション手法を1ステート内に条件分岐を含む場合と含まない場合とに分けて説明する．

#### 1ステート内に条件分岐を含まない場合

#### ユニット選択

はじめに，機能ユニットに必要となるビット数を決定する．state $i$のアロケーションテーブル内の演算をビット数の降順にソートする．このとき，必要となる機能ユニットの個数は，明らかに全てのステートにおける演算の個数

| | $fu_1$ | $fu_2$ | $fu_3$ |
|---|---|---|---|
| state 1 | 12 | 8 | 2 |
| state 2 | 8 | 6 | |
| state 3 | 16 | 10 | 6 |
| state 4 | 12 | 2 | |
| max | 16 | 10 | 6 |

図 4: ソートされたアロケーションテーブル (数字は演算に必要なビット数)

| | $fu_1$ | $fu_2$ | $fu_3$ |
|---|---|---|---|
| state 1 | 12 | 8 | 2 |
| state 2 | 8 | 6 | |
| state 3 | 16 | 10 | 6 |
| state 4 | 12 | | 2 |
| max | 16 | 10 | 6 |

図 5: アロケーション結果

の最大値であり，機能ユニット $fu_j$に必要かつ十分なビット数は，$j$列目の数値の最大値で与えられる．

図 4の例では，$fu_1$の最大値は 16，$fu_2$の最大値は 10，$fu_3$の最大値は 6 であるから，16bit，10bit，6bit の機能ユニットをそれぞれ 1 つずつ用意すれば良いことが分かる．

### ユニット割付け

続いて，演算の機能ユニットへの割当てを決定する．ユニット選択によって，$fu_j$のビット数が決定したアロケーションテーブルでは，$fu_j$に割り当てられた演算のビット数は必ず $fu_j$のビット数よりも少ない．しかし，割り当てられた演算の中で，よりビット数の少ない機能ユニットで実行可能な演算が存在する可能性がある．このような割当ては合成の結果，冗長な回路となる．これを回避するため state $i$ のアロケーションテーブル内の演算を機能ユニット $fu_j$のビット数を越えない限り右にシフトする．

図 4に以上の処理を行うとテーブルは図 5のようになる．

### 1 ステート内に条件分岐を含む場合

スケジューリングの際に 1 つのステートの中に条件分岐が存在するスケジューリング結果を出力する可能性があるため，1 ステート内に，条件によっては実行されない演算が存在する可能性がある．

今，図 6のように 1 つの条件分岐を含むステートがスケジューリングによって得られたとする．以下図中の左側のパスをパス 1，右側のパスをパス 2 とよぶ．パス 1 では，10 ビットの加算と 6 ビットの加算が，パス 2 では，8 ビットの加算 2 つがあったとする．ここで，state $i$ の $p$ 番目

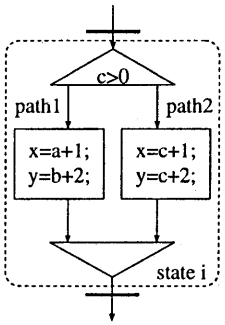


図 6: ステート内に条件分岐のある CFG

| | $fu_1$ | $fu_2$ |
|---|---|---|
| path 1 | 10 | 6 |
| path 2 | 8 | 8 |
| max | 10 | 8 |

図 7: ステート内に条件分岐がある場合のアロケーション結果

のパスにおいて $o_{ij}$ に割り当てられる演算を $p$ 行 $j$列に格納するテーブルを用意する（図 7）．

このテーブルにおいて $j$列目の数値の最大値を演算 $o_{ij}$ に必要なビット数とする．図 7においては，state $i$ で 10bit と 8bit の加算がそれぞれ 1 つずつ必要になることがわかる．このアロケーションテーブルに対し，ビット数の少ないパス上の演算は $o_{ij}$のビット数をこえない限り右にシフトする．

このように本手法では，ステート内，ステート間の 2 段階に分けてアロケーション手法を適用する．以下に，アロケーション手法を示す．

**Step 1** アロケーションテーブル Tbl を用意する．

**Step 2** 各ステートに割り当てられた全ての演算を実行するのに必要な機能ユニットのビット数を決定する（ステート内アロケーション）．

**Step 3** 各ステートについて Step 2 で決定されたビット数の降順に演算をソートし，アロケーションテーブルに保持し，各ステートで必要となるビット数の最大値を機能ユニットのビット数とする．続いて，各ステートのアロケーションテーブル内の演算を割り当てられる機能ユニットのビット数を越えない限りビット数の少ない演算から右にシフトすることで，よりビット数の少ない機能ユニットへ各演算を割り付ける（ステート間アロケーション）．

表 1: スケジューリング結果

| CFG | #states | #ALUs | CPU Time[s] |
|---|---|---|---|
| 1 | 3 | 2 | 0.02 |
| 2 | 1 | 1 | 0.08 |

表 2: アロケーション結果

| CFG | ALU | #bits | CPU Time[s] |
|---|---|---|---|
| 1 | ALU1 | 6 | 0.02 |
| | ALU2 | 6 | |
| 2 | ALU1 | 12 | 0.06 |

## 4 計算機実験結果

提案手法を SUN Sparc Station 2 (28.5 MIPS) 上に C 言語を用いて実装し，エントロピー CODEC の動作記述に対して合成を行った結果について報告する．

実験では、2 つの CFG を入力として用いた．CFG1 は，文献 [5] 内の JPEG 用ハフマン符号化器の AC 符号化モジュールの動作記述（ステートメント数 15）である．CFG2 は，文献 [5] 内の JPEG 用ハフマン符号化器の DC 符号化モジュールの動作記述（ステートメント数 29）である．全ての演算は ALU で行うこととし，使用する ALU の個数の上限 5 を制約として与えた．

表 1に, スケジューリング結果，スケジューリング実行時間を示す．表において最終的に得られた#states はステートの総数を，#ALUs は使用する ALU の総数を表す．また，CFG1 とそのスケジューリング結果を図 8に示す．

表 2に, アロケーション結果，アロケーション実行時間を示す．表において#bits は ALU のビット数を表す．

本手法によって得られたスケジューリング・アロケーション結果は，文献 [5] に示されているものと同一である．文献 [5] では，手設計によってスケジューリング・アロケーションが行われている．本手法では，手設計による設計と同一の合成結果が高速に得られており，提案手法の有効性が確かめられたといえる．

## 5 むすび

本稿では，エントロピー CODEC の高位合成手法を提案した．提案手法は，条件分岐が多く，変数のビット長が異なるエントロピー CODEC の動作記述に対して，CFG の節点を縮退させるスケジューリングと，演算のビット長を格納したテーブルを用いるアロケーションを行うことで効果的に合成を行うことができる．計算機実験の結果，提案手法の有効性を確認した．今後の課題として，提案手法に対し，CFG の構造変形，接続を考慮したアロケーションなどが挙げられる．

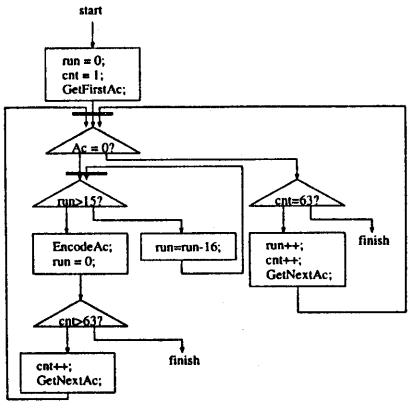


図 8: AC 符号化アルゴリズムのスケジューリング結果


## 謝辞

本研究の一部は，文部省科学研究費補助金（特別研究員奨励費）の援助を受けた．また，第 3 著者は財団法人神奈川科学技術アカデミー研究助成金の援助のもとに研究を行った．


## 参考文献


[1] D. Gajsky, D. Nutt, A. Wun and S. Lin, *High-Level Synthesis: Introduction to Chips and System Design*, Kluwer Academic Publishers, 1992.

[2] P. Michel, U. Lauther and P. Duzy, *The Synthesis Approach to Digital System Design*, Kluwer Academic Publishers, 1992.

[3] T. Miyazaki, "High-Level Synthesis Using Given Datapath Information," IEICE Trans. on Fundamentals, Vol. E76-A, No. 10, pp. 1617–1625, 1993.

[4] W. B. Pennebaker and J. L. Mitchel, *JPEG: Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.

[5] 鈴木純次, 小野定康, "PARTHENON を用いたエントロピー CODEC の設計," 第 5 回パルテノン研究会資料, pp. 109–130, 1994.

[6] 高橋隆一, 吉村猛, "ハイレベルシンセシスの動向," 信学論 (A), J74-A, No. 2, pp. 143–151, 1991.

[7] テレビジョン学会編, 画像情報圧縮, オーム社, 1991.

[8] 安田浩編, マルチメディア符号化の国際標準, 丸善, 1991.