複合演算を用いた高位合成の実装

平城景士[†1]　松永多苗子[†2]

高位合成とは、動作記述から LSI を設計するための RTL 記述を合成する技術である。本論文では、FPGA を対象とした高位合成において、多入力加算や積和演算等の複合演算を想定した手法について述べる。FPGA では高速なキャリーチェイン機構や演算マクロがあり、複数の演算を統合して実行しても遅延時間の増加が小さい傾向がある。実験結果では、複合演算を用いることにより 50%以上のサイクル数削減を確認するとともに、制約の与え方によって面積・サイクル数の異なる特徴をもつ回路構成を生成することを確認した。

キーワード：高位合成、FPGA、算術演算

High-level synthesis using composite functional units

KEIJI HIRAKI[†1]　TAEKO MATSUNAGA[†2]

High-level synthesis is a design automation process which reads behavioral descriptions and generates RTL descriptions to implement LSI circuits. This paper discusses an approach for high-level synthesis considering composite functional units which perform such as parallel additions and fused multiply-add operations in one step. Arithmetic operations can efficiently implemented by utilizing carry-chain mechanism and arithmetic operation modules on FPGAs. Experimental results show that more than 50% reduction of the number of cycles has been achieved with composite functional units, and various types of RTL structures can be generated with different area and delay features.

Keywords: High-level synthesis, FPGA, Arithmetic operation

1. はじめに

集積回路の技術発展に伴い ASIC(Application Specific Integrated Cicuit)や FPGA(Field-Programmable Gate Array)のゲート数は年々増加するようになった。かつてゲート数が数千個の時代はすべて人の手により集積回路の設計が行われた。しかし、数万ゲート数十万ゲートとゲート数が増えるにつれて、集積回路の設計が人の手では困難になり始め、集積回路を設計するために EDA（Electronic design automation）ツールが使われるようになった。現在ゲート数は数千万ゲートの集積回路が可能になったが、大規模な集積回路の設計には膨大な設計期間が必要となり、動作記述から設計する高位合成の研究が注目されるようになった[1]。高位合成は、動作記述からの設計のためソフトウェア開発者でもハードウェア開発が可能で、集積回路の設計が容易になり設計期間を大幅に削ることができる。また、現在ハードウェアの知識がなくても使えるような様々なフリーの高位合成ツールが開発されている。ただし、これらの高位合成は、まず動作記述をハードウェアで実現することを目的としたものが多く、合成時のアルゴリズムの改善の余地があるものが多い。

本研究では、簡単な記述から回路を合成する高位合成ツールを独自に開発し動作記述から FPGA までの開発環境を構築する。特に、処理の高速化を図るために、多入力加算器や積和演算器を導入した高位合成を行う。本稿ではこのような多入力加算器や積和演算器を複合演算とよぶこととする。複合演算を用いた高位合成には、サイクル数の減少の効果がある。しかし、複合演算の回路の大きさは 2 入力演算回路の大きさより大きくなるため、高位合成時には回路の大きさを考慮しなければならない。

2. 高位合成の原理

高位合成とは、C 言語や Python などの動作記述から RTL(Register Transfer Level)記述を生成する技術である。動作記述から設計するため、RTL 設計からでは困難であった、複雑なアルゴリズムをハードウェアに実装することが容易になり、設計期間の短縮や、並列処理による高速化などが実現できるようになった[3][4][6]。

†1 九州産業大学大学院工学研究科　産業技術デザイン専攻
Graduate School of Engineering, Kyushu Sangyo University
†2 九州産業大学　工学部　電気情報工学科
Electrical Engineering and Information Technology, Faculty of Engineering, Kyushu Sangyo University

図 1　高位合成の工程

高位合成の工程を図 1 に示す。最初に、通常のコンパイラと同様に動作記述をプログラムの最小の意味のあるコードであるトークンに分割する字句解析を行い、このトークンを構文解析により、解析木を生成し、演算間のデータ依存関係や制御依存関係を表す CDFG(Control Data Flow Graph)を作成する。CDFG 最適化では、言語の構文に依存した冗長性を取り除くため、CDFG 上で幾つかの変換処理を行う。主に、定数伝播、共通演算除去、デッドコード除去、並列性の抽出などがあげられる。この工程はソフトウェアのコンパイラでもよく用いられる処理である。アロケーションでは、必要となるハードウェア量を考慮しながら、与えられた動作を実現できる RTL の回路構成を決定する。スケジューリングでは、全体のハードウェア量や、総制御ステップ数などの時間制約を考慮しながら、動作記述に現れる各演算を具体的にどの制御ステップで実行するかを決定する。バインディングでは、動作記述上の演算子を演算器に、変数や配列をレジスタやメモリにマッピングする。アロケーションからバインディングはアルゴリズムによって何度も繰り返すことや、順番を変更することがある。FSMD 生成では、制御用の有限状態機械(Finite State Machine) と演算を実行するデータパスからなるシーケンシャル動作実行可能なマイクロアーキテクチャを生成する。

現在、様々な入力言語を対象とした手軽な高位合成が開発されている。Python を入力言語とした Polyphony、java を入力言語とした Synthesijer 等がある。これらのツールはソフトウェア開発者が、ハードウェアの知識が少なくても手軽に導入できる高位合成である。

3. 提案手法

本研究で作成した高位合成は、簡易言語を入力言語とし、入力言語を字句解析、構文解析、意味解析を行うことで CDFG を作成する。ここで既存の手法でスケジューリング、アロケーション、バインディングを行い、RTL 記述を出力する。現状 IF 文 WHILE 文などの制御がない基本ブロックのみの高位合成である。

本研究では、この基本的な高位合成に対して、複合演算を導入することで改良を図るものとする。通常高位合成において使用される演算回路は 2 入力の演算を想定したものが多い。例として 3 入力加算を考える。図 2 の(a) は 2 入力の加算器だけで考えた場合の CDFG を示す。ここでは 2 入力加算器を 1 つ、サイクル数が 2 サイクルとなっている。次に(b) は多入力加算器を使用した場合の CDFG を示す。多入力加算器の場合、多入力加算を 1 つ、サイクル数が 1 サイクルとなっている。加算器は 2 入力の場合と多入力の場合、回路の大きさは大きくなるが、遅延時間は FPGA での実装の場合、余り差がない。そのため、本実験で使用した高位合成の多入力加算回路の遅延時間はすべて 1 サイクルで動作するスケジューリングを行う。

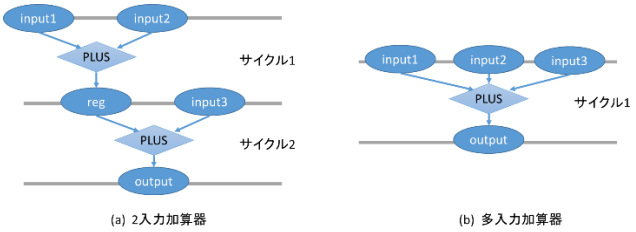

図 2 2 入力加算器と多入力加算器の CDFG

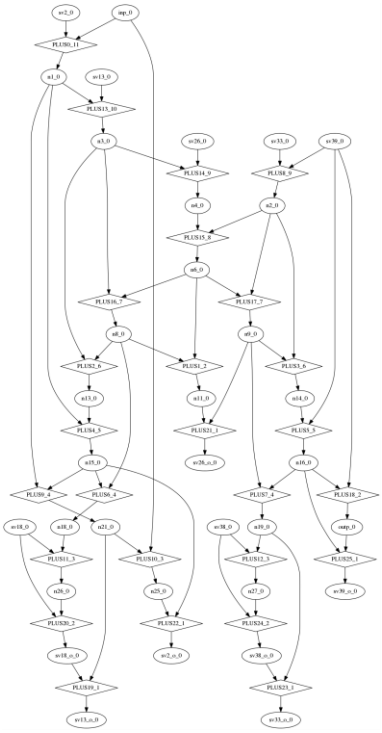

図 3　2 入力加算器を使用した CDFG

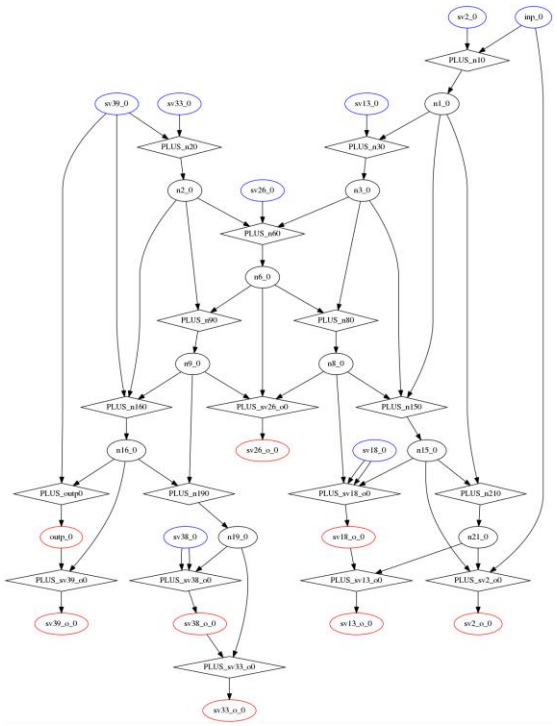

図 4 複合演算を用いた高位合成

例として、2 入力加算器だけで構成された CDFG を図 3 に示す。この CDFG から複合演算器を選択する順番は、最初に高位合成の CDFG 最適化の工程で、変数と使用された演算を確認し、同じ演算を使用している場合を複合演算器に選択する。複合演算を用いた結果を図 4 に示す。次に演算の使用数を計算する。演算の使用数と資源制約からクリティカルパスを計算し、クリティカルパスの演算を実行する順番の早い演算から複合演算器を使用できるかを確認し、2 つの演算を合成する。図 4 では n1_0=sv2_0+inp_0 と n2_0=sv13_0+n1_0 を複合演算で合成する。この工程を繰り返し、ユーザーが決めた資源制約の限界まで行う。

4. 実験

複合演算を用いた高位合成を Python2.7.7 で実装し、ベンチマークに対して評価実験を行った。ベンチマークには、HLSynth'92 の楕円フィルタの計算の ellipf を使用した[2]。FPGA は Cyclone III 3C16、LE(Logic Element)数 15408 個[5]、開発ソフトを Quartus II 64Bit Version 13.1.0 を使用する。また、演算器の入力ビット数は 8 ビットとする。

ellipf は 2 入力加算器 1 個と制限した場合、サイクル数は 26 サイクル必要になる。次に、加算器の個数を制限なくしできるだけ並列化できる演算を並列に行うと、2 入力加算器 4 個に対しサイクル数は 11 サイクルになる。この 2 入力加算器を 4 個以上増やしてもサイクル数はこれ以上削れない(図 3 参照)。

複合演算器を用いた高位合成を使用した場合 2 入力加算器 3 個、3 入力加算器 1 個、4 入力加算器 1 個の場合 8 サイクルになる(図 4 参照)。複合演算を増やした時の計算を表 1 に示す。この回数とは共有されていない変数を複合演算で削減した CDFG を生成し(図 4)、クリティカルパスから 2 個の演算を複合演算器変更した回数のことである。本実験ではサイクル数とその時の複合演算器の数を比較したいので複合演算器の制限は設けなかった。

表 1 複合演算器を用いた高位合成

回数	サイクル数	2	3	4	6	9	11	14	16	19	25
0	8	3	1	1							
1	7	3	1	1							
2	6	3	1		1						
3	6	1	2			1					
4	5	2	1			1		1			
5	4	2	1			1		1			
6	4	1	1	1		1			1		
7	3	1	1			1	1	1		1	
8	3	1	1		1	1		1	1		1

複合演算器を用いた高位合成の演算器を FPGA で実装した場合の総 LE 数とサイクル数をグラフ化したものを図 5 に示す。演算器を複合することで順調にサイクル数を減少していることがわかる。しかし、3 サイクル以下になると、LE の数が倍近くになり効率が悪いことが考えられる。

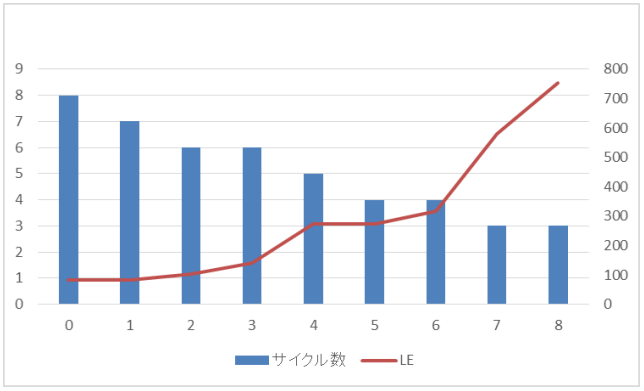

図 5 サイクル数と LE

以上の結果から、2 入力加算器を 1 個の制限とした場合と比較して、多入力加算器を使用した場合は半分以上のサイクル数を短縮することができ、LE の数は 10 倍から 30 倍になることがわかる。しかし本研究で使用した FPGA の LE 数は 3%も満たないため、LE 数は問題ないと考える。したがって、複合演算を用いた高位合成は、サイクル数を大幅に短縮し、回路の大きさには問題ないと考えられる。

5. おわりに

本論文では、複合演算を用いることによりサイクル数を少なくする手法を用いた高位合成を作成した。ベンチマークからの高位合成では、2 入力演算を使用した場合と複合演算を用いた場合、サイクル数は半分以上の短縮を行うことができた。また、複合演算を FPGA に実装する場合の LE

の数は問題ない数で合成されていることを確認した。

今後の課題として積和演算の実装による、サイクル数の削減を目指す。また、LE 数が少ない FPGA で実装する場合、不必要な複合演算を実装する FPGA の LE 数に合わせて分解した演算器にするスケジューリング手法を検討する。

参考文献

[1] 若林一敏. ハードウェア記述を合成する高位合成技術. IEICE Fundamentals Review Vol.6 No.1. (2012)
[2] " HLSW92 Benchmark Coordinator". http://ftp.ics.uci.edu/pub/hlsynth/HLSynth92/GUIDELINES
[3] 藤田昌宏. システム LSI 設計工学 (IT Text). オーム社.(2006)
[4] 藤田昌宏. VLSI 設計工学 - SoC における設計からハードウェアまで (新・電子システム工学). 数理工学社.(2009)
[5] “Cyclone III Device Handbook”. https://www.altera.co.jp/ja_JP/pdfs/literature/hb/cyc3/cyc3_ciii5v1.pdf
[6] Daniel Gajski. HIGH-LEVEL SYNTHESIS Introduction to Chip and System Design.Kluwer Academic Publishers.(1992)