HDL記述によるディジタル回路の複数FPGAに対する分割手法

2B－3

空岡 誠実　田中 康一郎　久我 守弘
九州工業大学

1　はじめに

近年，集積化技術の向上によりLSIに集積可能な回路規模が増加し，それに伴い設計検証時間が増大している．そこで，LSI仕様の決定とそれに伴う機能検証の高速化のために，ソフトウェアによるシミュレーションに代わりFPGA（Field Programmable Gate Array）を核としたエミュレータを用いる方法が注目を浴びている．このエミュレータでは，ソフトウェアによるシミュレータと比較して数百～千倍の速度で検証を行える．しかし，FPGAに実装するための論理合成および複数のFPGAに実装するための回路分割といった2つの処理が必要であり，設計修正毎に行うこれらの処理をいかに短縮するかがエミュレータを用いる際の重要な鍵となる．そこで，我々はHDL記述をソースレベルで単一FPGA用に分割し，並列に論理合成/実装を行うことで実装時間の短縮を図る方法を提案する．

2　実装までの流れ

現在の回路分割方法は，図1に示すように論理合成後のネットリストを分割する方法である．この方法ではネットリストレベルで分割が行われており，設計変更のたびに回路全体の論理合成および分割処理を要してしまう．一方，我々の提案する方法はHDL記述レベルで分割する．分割手順としては，まずHDL記述を回路構成に応じて**細分化**する．この際，細分化した回路間の結線情報も生成しておく．次に，細分化した回路をそれぞれ論理合成を行い，合成結果のレポートファイルから細分化した回路の規模を調べておく．最後に，ここまでの段階で得られた情報（回路間の結線情報，回路規模など），および実装するエミュレータ上のFPGAの規模やFPGA同士の結線情報から，各々のFPGAへ実装する回路を決定するとともに，実装のために必要となるHDL記述を生成する（**グループ化**）．

上記の方法でHDL記述を分割することにより，以下の理由から論理合成と配置配線の時間を短縮することが期待できる．i) HDL記述のソースに修正を行って再び細分化およびグループ化をしたとき，前回の分割と比較して変更されている部分のHDL記述のみを論理合成行えば良い．ii) HDL記述を細分化することで並列に論理合成することが可能となる．

3　細分化手順

本研究で対象としているHDLは，業界で広く用いられているVerilog HDLを対象とする．また，変換を行う記述は論理合成可能な構文を使用したものに限定する．

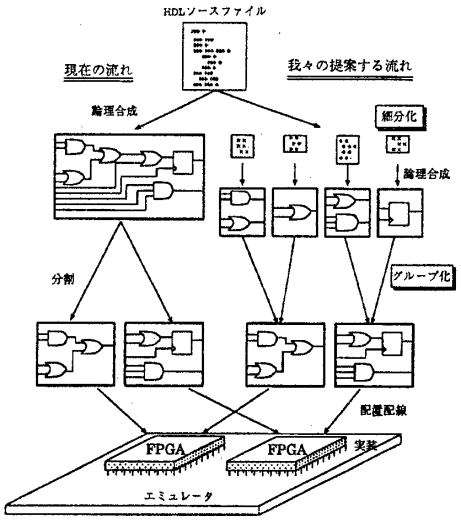

図1: 実装までの流れ

分割する際に用いるアルゴリズムを決定する際には，i) 高速に分割できること，ii) 分割前と分割後の動作が同一であること，iii) 細分化後の時点では細分化前と比較して回路規模が増大しないこと，iv) 複数のFPGAに十分実装可能であること，を満足する必要がある．

以下，本稿では分割アルゴリズムのうち細分化の手順のみについて述べる．

[1] 並行動作記述部分の分割

順次実行文以外の並行動作を記述している部分は式単位でモジュール化する．

[2] task，functionの部分の分割

taskおよびfunctionにより記述されている部分を組合せ回路として分割する．その際，taskおよびfunctionの引数を代入文へ変更する．taskおよびfunctionの回路が大きい場合は，さらにそれらを細分化する．

[3] 順次実行文の分割

問題になるのは1つのモジュール内のある1つの順次実行文（always文）が単一のFPGAに実装できないほど規模が大きい場合である．基本的な細分化方針は，図2に示すように代入文のディスティネーションとなる変数（図2ではacc）を基準として細分化する．

変数は，他の複数の変数から代入および参照される可能性がある．複数の代入がある場合は，その入力を選択するためのマルチプレクサとその値を保持するフリップフロップが必要になる．また，参照がある場合，その同一クロック中で変数に代入されている値があればその値を，そうでなければフリップフロップの保持している値を参照する必要があるため，出力を選択するためのマルチプレクサが必要になる．ただし，変数への代入がノンブロッキング代入文で行われている場合や，ブロッキング代入文であっても同一クロックサ

A Partitioning Technique of a Digital System Described in HDL for Multiple FPGAs.
Makoto Soraoka　Koichiro Tanaka　Morihiro Kuga
Kyusyu Institute of Technology

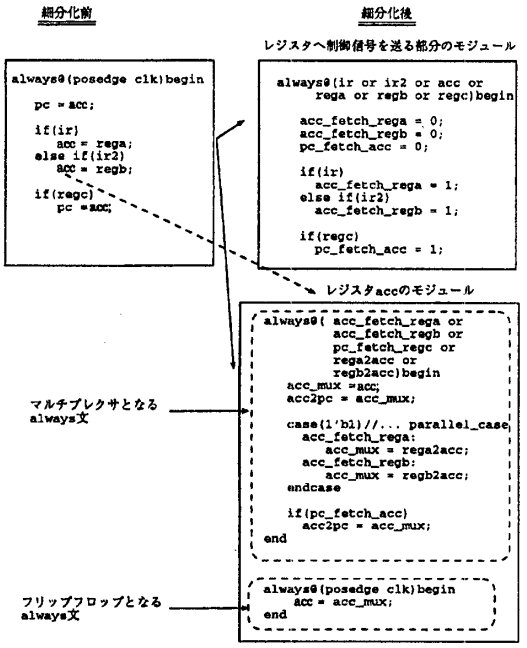

図 2: 順次実行文の細分化

イクル中に変数に対しての代入と参照が同時に起きない場合には，出力は必ずフリップフロップの値となるためマルチプレクサは必要ない．また，変数が値を保持しないワイヤとして使用される場合，値を保持するためのフリップフロップは取り除くことができる．以上の規則に従って，変数を図2右下のようにモジュール化する．

変数のモジュール化に伴い，そのモジュールを制御するためのHDL記述が必要となる．変数を制御するための記述は図2右上に示すように細分化前のHDL記述から生成できる．基本的には，各々の代入文をユニークな変数名を持つ変数に対して"1" を代入する式に変更し，それらを参照および代入されていた変数のモジュールのマルチプレクサの制御信号とする．また，そのalways文を順序回路から組み合わせ回路に変更し，各モジュールと接続することでモジュールを制御するためのHDL記述を生成できる．

4 細分化手順の評価

3章で提案した手順を用いた場合の細分化の効果について評価するために，i) 細分化前と細分化後の動作の同一性，ii) 細分化前と細分化後の論理合成時間，iii) 細分化前と細分化後の回路規模，について調査を行った．

対象とした回路は 本学で教育用に開発されたKITE-1マイクロプロセッサ [1] である．細分化前のHDL記述は1つのモジュールにより全体の動作を記述している．また，動作のほぼ全てを1つの順次実行文により記述しており，ALUはtaskを用いて記述している．

評価の際に用いた環境は以下のとおりである．

使用計算機 ：米国 Sun Microsystems 社製 SPARC station 20 model 151 (主記憶 64MB)
使用 EDA ツール ：米国 Cadence 社製 Verilog-XL Ver. 1.6a，米国 Synopsys 社製 Design Compiler Ver. 3.4b,

表 1: 論理合成時間

細分化前		11 分 14 秒
細分化後	制御信号生成	3 分 48 秒
	ALU	1 分 20 秒
	その他	30 秒～1 分

米国 Xilinx 社製 XACTstep(Ver 5.2)
実装デバイス ：米国 Xilinx 社製 XC4013pg223-5

なお，今回は細分化手順のみについて評価を行うため，細分化後の回路は1個のFPGAへ実装している．

以下に評価結果および考察について述べる．

細分化前と細分化後の動作の同一性については，細分化前と細分化後においてシミュレーションによる動作確認と，デバイスへ実装しての動作確認により動作の同一性を確認している．

次に，**細分化前と細分化後の論理合成時間**の結果を表1に示す．1つの順次実行文であった記述を細分化すると，23個のモジュールに分割された．このうち，最も合成時間の長いモジュールはマイクロプロセッサの制御信号生成部分であり3分48秒であった．その他のモジュールであるALUやレジスタはすべて制御信号生成部分の論理合成に要した時間以内に終了している．この結果より，HDL記述のソースに修正を行って再び論理合成を行ったとき，並列に論理合成を行える環境が用意できれば，細分化後のHDL記述は細分化前よりも必ず短時間で再合成することができることが確認できた．

最後に，**細分化前と細分化後の回路規模**については，細分化前のHDL記述では約75%のルックアップテーブルを利用したのに対して，細分化後のHDL記述では約66%のルックアップテーブルを利用した．フリップフロップ数については，細分化前と細分化後では同一数（113個）利用している．回路規模が減少した理由についてはいくつかの要素が考えられるが，詳細については現在調査中である．

5 おわりに

エミュレータ上の複数FPGAにHDLで記述された回路をマッピングする際に必要となる回路分割を，回路の細分化およびグループ化により行う手法について提案した．このうち，提案する細分化手順について実際の回路に適用し，その効果を確認した．

本稿で述べていないグループ化の具体的な手順については現在検討中である．また，エミュレータによる検証を行う際，分割実装した回路はできるだけ高速に動作できることが望ましい．従って，記述を分割するときのアプローチとして，回路規模優先ではなく動作速度を優先させる方法や，回路規模と動作速度のトレードオフを考慮した分割方法のについても検討を行う予定である．

参考文献

[1] 末吉 田中 久我：“教育用マイクロプロセッサ KITE による設計教育事例 (第2報)” 情報処理学会研究報告 (93-ARC-110-13, 93-DA-73-13) 1994.