システムＬＳＩ設計技術　94－11
（2000．1．12）

高位合成システムによるCPLD設計

中條　新，川上　洋史，檀　良

法政大学

〒184　東京都小金井市梶野町３－７－２
TEL：(0423) 87－6208　FAX：(0423) 87－6381
E-mail：jyo@dang.k.hosei.ac.jp

あらまし

本稿では高位合成システムと,このシステムからの CPLD 設計例について述べる.本システムは Verilog-HDL による動作記述を入力とし,Verilog-HDL によるデータパス部とステートマシン部の RTL 記述情報を得る.この情報より得られる RTL 記述は論理合成可能であり,配置・配線後のタイミンク検証でも入力動作記述を満たす正確な結果を得ることができる.また演算器ライブラリーには,３２ビット単精度の演算器,ALU,比較器をもつ.

キーワード　動作合成，CPLD，ステートマシン

CPLD design by A High-Level Synthesis System

Shin Nakajyo and Hiroshi kawakami and Ryo Dang

Hosei University

3－7－2　Kajino-cho, Koganei-shi, Tokyo 184, JAPAN
TEL：(0423) 87－6208　FAX：(0423) 87－6381
E-mail：jyo@dang.k.hosei.ac.jp

Abstract

This paper presents a High-Level Synthesis and examples of CPLD design by this system. This system takes as input behavioral description written by Verilog-HDL and outputs RTL description information of data path circuit and state machine circuit written by Verilog-HDL. RTL description is obtained from RTL information is ready to be used for logic synthesis, therefore we gain results which satisfy input behavioral description at timing analysis after layout. This system has an arithmetic operation unit, an ALU and a comparator, all with 32bit single precision.

High-Level Synthesis, CPLD, statemachine

key words

1. はじめに

近年の LSI 設計は HDL や CAD を用いる自動設計化により短期間で行なえるようになってきたが,システム仕様の多様化により更なる設計の設計期間の短縮が望まれる.

本稿では自動設計の中での論理合成の上位に位置し,回路の動作記述から RTL 記述情報を得る高位合成システムを実現し,このシステムから得られる RTL 情報をもとに CPLD 設計例について述べる.

2. システムについて

本システムでは,スケジューリングとアロケーションを独立させ,スケジューリング,アロケーションの順で合成をおこなう.CDPL 設計までの流れを図 1 に示す.

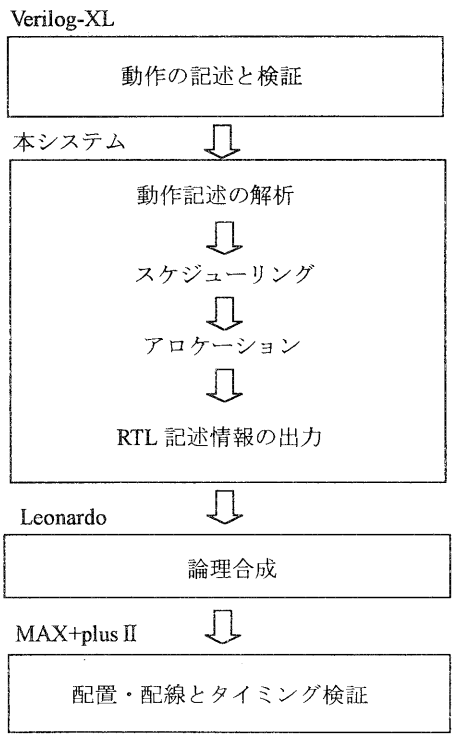


図 1　本システムと設計までの流れ

2.1　動作の記述と検証

動作は Verilog-HDL を用いて記述し Verilog-XL で動作検証を行なう.Verilog-HDL は c 言語ように記述でき,ビット幅を明確に宣言できる.また,テストパターン記述をもちいて動作記述のエラーチェック,機能検証ができるので,回路動作の記述に適すると考える.バブルソートの動作の記述例を図 2 に示す.

```
// clock の定義
`define clock   begin (posedge clk) end;
module   buble_sort(undata,clk,reset,sodata);
parameter N=3;
input clk,reset;
input[31:0] undata;
output[31:0] sodata;
reg[310] sodata;
reg[31:0] i,j, p,q, s,t;
reg[31:0]   unreg   [1:N];
initial begin
    // データの入力
    for(i=1; i<=N; i=i+1)begin
      `clock
       unreg [i]=undata;
    end
    //   演算処理
    for(p=1; p<=N-1; p=p+1)begin
        for( q=N; q>p; q=q-1;) begin
              s=unreg[q-1];
              t= unreg [q   ];
              if(s>t)begin
                  unreg [q-1]=t;
                  unreg [q   ]=s;
              end
          end
       end
      // データの出力
      for(j=1;j<=N; j=j+1)begin
      `clock
      sodata= unreg [j];
      end
  end
endmodule
```

図 2　バブルソートの動作記述の例

図 2 の clock の定義の部分では,clock の立ち上がりを待つ関数　`clock　を定義している.この関数を使うことにより,データ入力,出力の記述で 1 つの

入力,出力に複数のデータの受け渡しを記述できる.動作記述の変数はすべてレジスタで宣言する.また動作記述では if 文,while 文,for 文が使える.for 文を使う際,for の終了条件を表す変数は値が確定していなければならない.

2.2 動作記述の解析

ここではパラメータ宣言の変数を値に置き換え,for 文の展開を行なう.また条件分岐演算に支配されるブロック(条件演算ブロック)とそうでないブロック（演算ブロック）に分ける.その後同じ名前の変数の接続関係から変数の依存関係を求める.これにより変数や演算の接続データを構築していく.

```
for(i=1; i<=3; i=i+1)begin
    `clock
     unreg [i]=undata;
end
```

⇩

```
    `clock
   unreg [1]=undata;
    `clock
   unreg [2]=undata;
    `clock
   unreg [3]=undata;
```

図 3 for 文展開の例

演算ブロック

```
s= unreg [2];
t=unreg[3];
```

条件演算ブロック

```
if(s>t)begin
  unreg [2]=t;
  unreg [3]=s;
end
```

図 4 演算ブロックと条件演算ブロック

2.3 スケジューリング

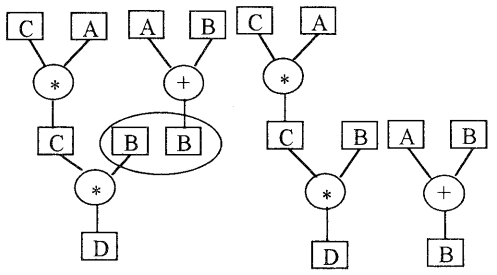
スケジューリングでは始めに 2.2 で構築した接続データに ASAP 法でスケジューリングを行ないコントロールステップを割り当てる.演算ノードにつながるレジスタノードのステップ数は演算ノードと同じとする.条件分岐演算がある場合,条件分岐演算と条件分岐中の演算をどちらを先に行なうかにより,必要なコントロールステップが変わってくる[4]が,本研究では簡単化のためは条件分岐の入れ子は考えないものとする.ASAP 法は以下のルールにしたがって割り当てることができる.

1) input ノードのステップ数は 0 とする.
2) 先行ノードが条件演算ブロックある場合,参照するレジスタのステップ数は条件演算ブロックの最大ステップとする.
3) 演算ノードのステップ数は参照しなければならない 2 つのレジスタうち,ステップの数の大ステップに＋1 する.
4) 条件演算ブロック中の演算ノードで,先行ノードがその条件演算ブロック中にない場合,そのノードのステップ数は条件分岐演算ノードのステップ+1 とする.
5) 条件演算ブロック中のレジスタノードで,先行ノードがその条件演算ブロック中にない場合,そのノードのステップ数は条件分岐演算ノードのステップとする.
4) レジスタノード,output ノードのステップ数はそのノードに代入するノードのステップ数は同じとする.
5) output ノードのステップ数をもっとも遅い output ノードのステップ数にそろえる.

さらに 2) のままでは式 1~式 2 のような記述がある場合,式 4 と式 5 の順序が変わりノードの依存関係を崩してしまうので,ステップ数を修正する必要がある.

```
A=X;        式 1
B=Y;        式 2
C=C*A;      式 3
D=C*B;      式 4
B=B+A;      式 5
```

式 1,式 2 は同じステップに割当てられていると仮定するとデータの流れを示す DFC(データフローグラフは図 5 のようになりコントロールステップの修正が必要になる.



依存関係が崩れてしまう　　ステップ修正

図 5　依存関係の修正が必要な記述

この ASAP スケジューリング手法は本研究室の回路シミュレータ SPICE ライクな独自開発の行列計算に適応させ,正確な計算結果を確認でき,共有メモリ型コンピュータを使って,従来の行列計算よりも速い時間で計算できることが分かっている.

ASAP 法のスケジューリング結果から,条件演算ブロックとそうでない演算ブロック分けて,それぞれのブロックの演算ノードに ALAP 法でスケジューリングする.ALAP 法は ASAP 法よりも求めた最大ステップの範囲内で演算ノードの可動範囲を求める.その可動範囲から GA を用いてブロックごとに最適なコントロールステップ数を割り当てる.

スケジューリングの後に動作記述内に`clock 関数がある場合,input ノードから入力されるレジスタと,output ノードのコントロールステップ数を`clock 関数の出現順に足していく.図 2 のバブルソートの動作記述のスケジューリング結果を表す DFG を図 6 に示す.図 6 で太い点線で囲まれたステップは条件演算ブロックを表す.それ以外のステップは演算ブロックを表す.この例の場合は両方のブロック内で可動範囲がないので ALAP 法,GA による,スケジューリングはないが,この結果から,行なう演算,実行順序,演算に使うレジスタなどが分かる.最終的なスケジューリング結果がステートマシン部 RTL 情報となる.

2.4　アロケーション

アロケーションはスケジューリング後の DFG の中で,同じコントロールステップ内で使わない演算ノードやレジスタノードを 1 つのハードウエアで共有化しハードウエア資源を最小化する問題である.本システムのアロケーションは,演算器割り当て,レジスタ割り当て,バス割り当ての順で行なう.それぞれの割り当てはクリーク分割問題であり,その補グラフを論理関数で表し SBDD をもちいて簡単化を行なう手法[2]にもとずく.またバス割当てでは出力が 2 本以上あるハードウエア資源を対象として割り当てを行なう.

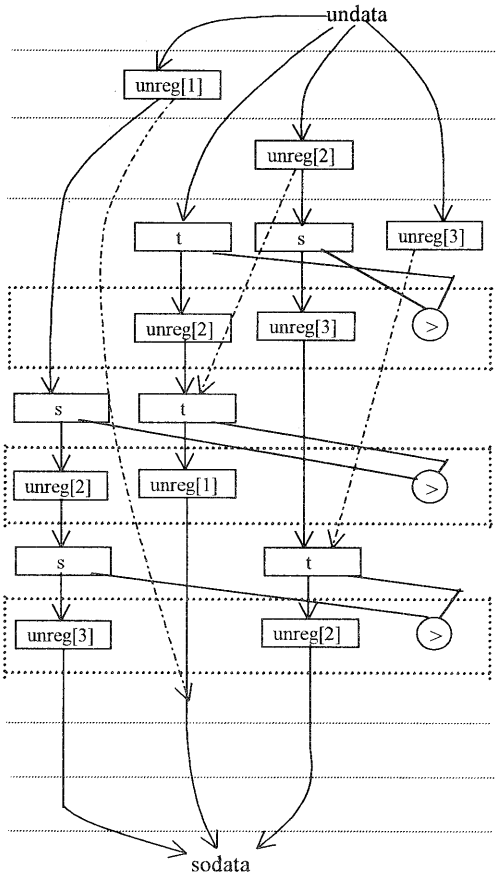


図 6　　スケジューリング結果

アロケーション後にレジスタや演算器の個数と接続関係が得られる.この結果から複数入力のある演算器,レジスタにマルチプレクサを挿入し,データパス回路が構成される.この結果がデータパス部の RTL 情報となる.図 2 のバブルソートのデータパス回路を図 7 に示す.

3. RTL 記述への変換について

動作記述を CPLD 回路で実現するために,スケジ

ューリング,アロケーションの結果から得られるRTL情報をRTL記述に変換し論理合成,配置・配線を行なわなければならない.ここではRTL記述はVerilog-HDLで記述する.そして回路の演算部分は表1に示す32ビット単精度の演算器を用いる.

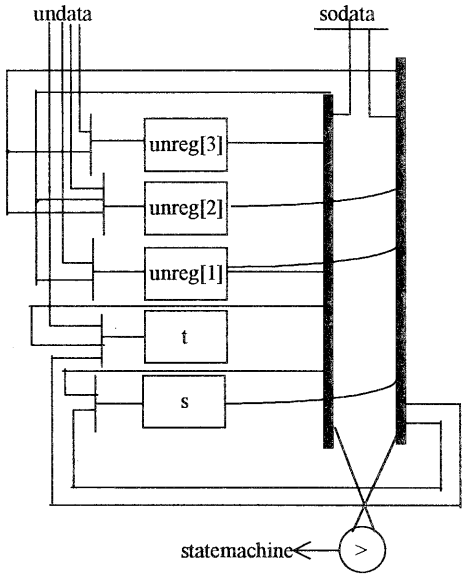


図7　バブルソートのデータパス回路

表1　演算器ライブラリー

	LCs	Delay	DFFs	TRIs	Pis	Pos	CLK
divider	244	14	115	0	67	33	27
multiplier	245	11	115	0	67	33	27
addsub	750	28	181	0	68	34	17
subst	718	28	177	0	67	33	5
adder	647	28	112	0	67	33	5
comparatorg	31	8	0	0	64	1	0
comparator	31	8	0	0	64	1	0

演算器には実行クロック時間にばらつきがり,スケジューリングにより割り当てたコントロールステップ内で演算を実行するためには,クロックの同期をとらなければならない.クロックの同期をとるために比較器以外の演算器は,演算スタートさせる信号(key 信号)が入ると計算を始め,演算が終わると演算の終わりを告げる信号（finish 信号）を出力する,仕様に設計されている.よってステートマシンはあるステートに複数の演算器が割り当てある場合,それらの演算器から出力されるすべてのfinish信号が出そろうまでそのステートで待機している.また演算器とレジスタのデータ転送に余裕を持たすために演算開始のまえに1クロック余裕をもたせる.Verilog-HDLの記述は図8のようになる.

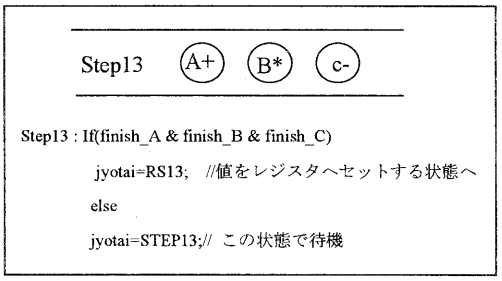


図8　Verilog-HDLのcase文をもちいたステートマシン記述(一部抜粋)

4. 合成結果

RTL情報から先にのべたRTL記述形式で論理合成した.合成に使用したツールは,Exemplar logic社のleonardo spectrum 配置・配線に使用したツールはALTERA社　のMAX+plus II.合成は図2に示したバブルソートの記述と,図9に示す微分方程式の解を求める動作記述で行なった.

```
module DIFFEQ(Xinport,Xoutport,DXport,Aport,
              Yinport,Youtport,Uinport,Uoutport);
input [31:0] Xinport,Yinport,Uinport,Dxport,Aport;
output [31:0] Xoutport,Youtport,Uoutport;
reg[31:0] Xoutport,Youtport, Uoutport;
reg[31:0] x,y,u,a,dx, xl,yl,ul;
initial begin
x =Xinport; a =Aport;dx =DXport;
y =Yinport;u =Uinport;
while(x<a)begin
ul=u-(u * dx) *(3 * x)- dx *( 3 * y ) ;
yl =y+u*dx;
xl= dx + x;
x=xl;   y=yl;   u=ul;
end
Xoutport = x;
Youtport = y;
Uoutport = u;
end
endmodule
```

図9　微分方程式の解を求める動作記述

デバイスは FLEXEPF10K100EQC208 を使用した.合成結果を表 2 の上段にバブルソートの記述の結果,下段に微分方程式の解を求める記述の合成結果を示す.

表2　合成結果

LCs	Delay	DFFs	TRIs	Pis	Pos	最大動作周波数	配線後 LCs
430	27	165	32	34	32	34.98MHz	435
2509	28	817	32	162	101	21.92MHz	2530

MAX+plus II のタイミング検証を行ない 2 つの例で配置・配線後の正しい結果を得ることができた.結果を図 10,図 11 に示す.

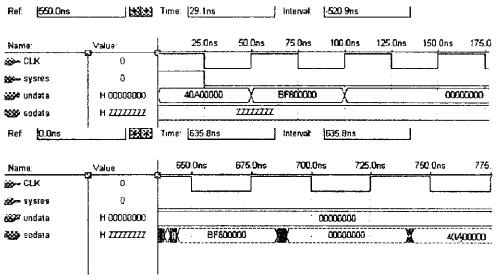


図 10　バブルソートのタイミング結果

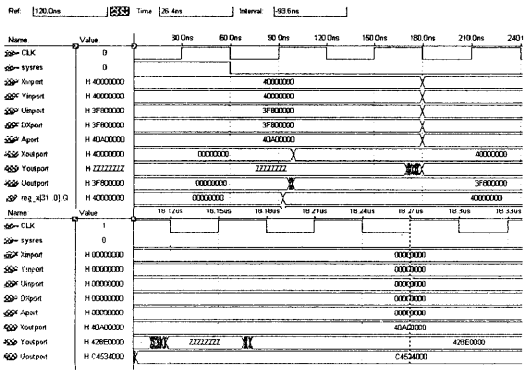


図 11　微分方程式の解を求める動作のタイミング結果

5. おわりに

本稿では Verilog-HDL による動作記述から RTL 情報を得る高位合成システムと,RTL 情報をもとにして記述した RTL 記述が論理合成可能で,配置・配線後のタイミング検証でも満足できる結果が得られることを示した.今後の課題としては RTL 記述の自動生成,本システムに対応する動作記述レベルの向上,スケジューリングの段階での最適化などがあげられる.

6. 参考文献

[1] 田中　敏明, 小林　勉 ."FORTRAN 記述からのデータパス合成法." 電子情報通信学会論文誌 pp.973-971 '88/4 Vol.J-71A No.4

[2] 高橋　隆一, 吉村　猛 ."ハイレベルシンセシスの動向." 電子情報通信学会論文誌 pp.143-151 '91/2 Vol.J-74A No.2

[3] 宮崎　敏明,."データパス合成のためのリソース割り付け法." 情報処理学会論文誌 pp.1663-1669 Vol.34 No.7 july 1993

[4] 山田　晃久, 石崎　年樹,石浦菜岐佐,神戸　尚志,白川　功,."条件制御分岐のあるデータパスのスケジューリング方法." 信学技法,VLD93-52, pp.7-14　1993