情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SLDM-144 No.17
Vol.2010-EMB-16 No.17
Vol.2010-MBL-53 No.17
Vol.2010-UBI-25 No.17
2010/3/26

仕様記述言語 SpecC によるサイクル精度記述の一試行

泉知論[†‡]　吉川寿広[*]　荒木大[*]

概要：システム設計の効率化を求めて，動作記述から回路を自動的に合成する動作合成の技術が実用化されてきており，また，より抽象度の高いシステムレベルの仕様記述言語も提案されている．一方で，高度な最適化のため，あるいは，外部回路とのインターフェースをとるため，サイクル精度のタイミングまで考慮した詳細な設計が求められる場面も依然存在する．仕様記述によるモデリングと検証，ハードウェア化する部分については動作記述からの回路合成，そして必要に応じてサイクル精度記述による最適化，と設計を進めていくにあたって，記述言語や処理系の断絶なく設計レベルを徐々に深めていくことが望ましい．そこで，仕様記述言語 SpecC では 2.0 版でレジスタ転送レベル記述のための言語仕様が追加されている．しかし一般には，シミュレータやシンセサイザなどの処理系が必ずしも言語仕様上可能なすべての記述に対応するわけではない．そこで，Spec−C 2.0 と現在利用可能なシミュレータを対象に，基本的な回路例を取り上げながら，サイクル精度の記述を試行し，記述法を確認していく．

Trial Designs by Cycle-Accurate Hardware Description in Specification Description Language

Tomonori Izumi[†‡]　T oshihiro Kikkawa[*]　Da i Araki[*]

Abstract: I n or der to cope with the des ign of recent huge and co mplicated s ystems, higher ab stracted design methodologies ha ve b een pr oposed in cluding, b ehavioral synthesis, h ardware-software codesign , specification d escription languag es, etc . However, cycle-accurate design is still needed, to optimize the detailed architecture more deeply, to inter face with per ipheral har dwares to be coupled tigh tly, o r to i mport intellectual properties predesigned as cycle-accurate modules. Starting with specification description langu age, designers will break the target into more detailed ones repeatedly and may us e so me languag e and design too ls at each lev el. SpecC was originally proposed as a specification description language and have been enhanced to support RTL (register transfer level) descriptions in order to provide an unified and seamless design environment fro m spe cifications to det ailed designs. Ho wever, th ere is generally so me gap between the ideal language specification and the subset and/or some kind of implicit description rule s for ea ch specific tool. This manuscript presents some trial designs in RTL SpecC with a simulator and a synthesizer and proposes a set of description rules for practical and efficient designs.

1. はじめに

近年のＬＳＩの集積度向上に伴うシステムの大規模化・複雑化により，設計期間の長期化，開発コストの増大が大きな問題となっている．そのため，集積回路黎明期のトランジスタレベル設計，ゲートレベル設計から，現在のハードウェア記述言語によるレジスタ転送レベル設計，近年実用化されてきた動作レベル設計，さらにシステムレベル設計の研究開発と，設計の抽象度をあげての効率化が求められ進められてきている．そのような高い抽象での設計の為，システムを記述する言語と処理系が必要となる．SpecC[1]はシステムの仕様を記述するための言語として提案され，これによりシステムのモデリングと機能検証，システムレベルのタイミング仕様の記述と検証，コンポーネント分割や性能見積りを含むシステムアーキテクチャの検討，などの上流設計の効率化を狙っている[2].

一方で，高度な最適化のため，あるいは，外部回路とのインターフェースをとるため，サイクル精度のタイミングまで考慮した詳細な設計が求められる場面も依然存在する．一般には，システムレベルでの記述と検討，コンポーネント分割，ハードウェア化する部分については動作記述からの回路合成，そして必要に応じてサイクル精度記述による最適化，と設計を進めていくことになるが，段階が進むたびに記述言語や処理系が変わると，効率の低下をまねき，またミスが混入しやすくなる．そこで，SpecC では 2.0 版でレジスタ転送レベル記述のための言語仕様が追加され，回路における信号，レジスタ，状態遷移などが表現可能となった[3].

言語は仕様書ならびにリファレンスコンパイラにより規定されるが，実際に使用するにはシミュレータや合成ツールなどの処理系が必要である．SpecC の実使用環境の一例として，主に宇宙機器のための電子機器開発支援システム ELEGANT が挙げられる．ELEGANT システムでは SpecC が記述言語のひとつとして用いられ，設計詳細化支援ツール[4]やシミュレータ[5]，動作合成ツール[6]などの処理系が提供されている．さらに，このプラットホームを使用してのコンポーネント分割の研究開発[7]などがすすんでいる．詳細設計については，ハードウェア記述言語との相互変換や合成などの

† 立命館大学 理工学部／大学院 理工学研究科
Faculty of Science and Engineering, Ritsumeikan University
http://w ww.ritsumei.ac.jp/se/re/izumilab/
‡ 株式会社シンセシス
Sy nthesis Corporation
http://w ww.synthesis.co.jp/
＊株式会社インターデザイン・テクノロジー
InterD esign Technologies, Inc.
http://w ww.interdesigntech.co.jp/

Vol.2010-SLDM-144 No.17
Vol.2010-EMB-16 No.17
Vol.2010-MBL-53 No.17
Vol.2010-UBI-25 No.17
2010/3/26

研究開発が進められており，一部試用段階に入りつつある．

一般に，支援ツールがサポートするのは必ずしも言語仕様のすべてではなく，サブセットであったり，特定の記述スタイルを要求するものであったりすることも多い．また,実用上の必要から,本来の言語仕様に対して独自の拡張がなされる場合もある．設計現場に受け入れられるためには，言語と処理系は，試用によって磨かれねばならない[8].

本稿では，Spec-C シミュレータ VisualSpec 4 [9]と開発中の合成支援ツール（SpecC から Verilog-HDL への変換ツール）を対象に，デジタル回路設計者の立場から，サイクル精度の記述法の検討を行い，この環境における記述スタイルを確立する．試行例として，回路表現の基礎である組合せ回路，順序回路，またＲＴＬ回路設計上の基本構成要素であるレジスタ（フリップフロップ）やラッチ，バス，それらの相互接続，フィードバックループ，多段階層化などの基本的な構造化記述，それらを組み合わせての設計例などを取り上げる．

2. 記述言語と設計環境

本設計試行においては，SpecC 2.0 をベースに株式会社インターデザイン・テクノロジーで拡張されたものを記述言語として用いる．以下, 便宜上 SpecC.IDT2009 と呼ぶ．

本来抽象度の高い仕様記述言語である SpecC に対して, SpecC 2.0 ではレジスタ転送レベル(RTL)の記述のために次の仕様が追加された．

- signal 変数：RTL の信号線に相当する．代入によりイベントが発生し，それに基づく動作を記述することができる．
- bufferd 変数：RTL のレジスタやフリップフロップに相当する．ある特定の信号（すなわち，クロック）に同期して値が更新される．
- fsmd 構文：データパス付有限状態機械(finite state machine with datapath)を直接的に記述できる．

SpecC.IDT2009 では，さらに次の変更，拡張が加えられている．

- signal 変数に関するセマンティクスの変更：値代入でのイベント発生から値変化でのイベント発生に変更．これによりフィードバックループを記述した場合の無限ループが回避される．
- bit4 型の追加：0,1 に加えて不定値 x やハイインピーダンス z を扱う 4 値ビットベクタ．これにより３ステートバスを記述することができる．
- par 構文の拡張：並列実行の par 文に書けるのは behavior の起動のみという制約を緩和し，関数や動作ブロックを書けるものとした．これにより記述性が大幅に向上する．
- タイムアウト付イベント wait 機能
- 演算式における符号処理の明確化

本設計試行の設計環境の概要を図 1 にまとめる．設計者は SpecC.IDT2009 で設計対象を記述する．インターデザイン・テクノロジー社のシミュレーション環境 VisualSpec 4.3 を用いて SpecC 記述の動作を確認する．VisualSpec の内部ではまず C++記述に変換し，C++コンパイラを起動して実行形式を生成し，それを実行することでシミュレーションを行っている．シミュレーション結果は VCD(value change dump)形式で保存され，波形表示ツールを起動して設計者に結果を示す．設計者は結果を受けて記述の変更を行う．ここではマイクロソフト社の VisualStudio ならびに BSI の GTKWave を使用している．シミュレーションによる動作検証が完了すると，回路合成にすすむ．インターデザイン・テクノロジー社の SpecC からハードウェア記述言語への変換ツール(scrtl2verilog)を用いて Verilog-HDL に変換し，従来の合成ツールを用いて回路合成を行う．ここでは ALTERA 社の Quartus-II を使用している．なお，必要に応じて，SpecC 2.0 準拠の記述の動作確認のために Center for Embedded Computer Systems, University of Clifornia, Irvine から提供されている SpecC Reference Compiler (scrc) 2.0 を用いる．

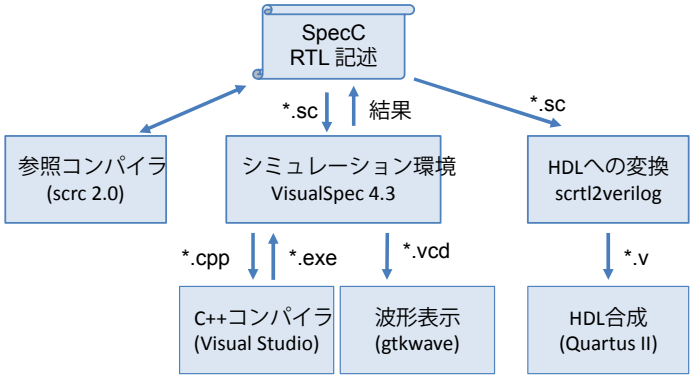

図 1　設計環境.

Vol.2010-SLDM-144 No.17
Vol.2010-EMB-16 No.17
Vol.2010-MBL-53 No.17
Vol.2010-UBI-25 No.17
2010/3/26

3. 基本的な記述スタイル

SpecC では behavior と呼ぶ動作ブロックを単位として記述する．ここでは，ひとつの behavior をひとつの回路モジュールとして記述するものとする．behavior の引数がモジュールの入出力信号に相当し，behavior 直下の変数がモジュール内の信号線やレジスタに相当する．原則，ひとつの behavior 内にはひとつの関数(main())を持ち，そこに動作を記述する．また，子 behavior のインスタンス化と引数指定が，子モジュールのインスタンス化と接続指定に相当する．

behavior の引数は必ず signal 変数とし，main 関数の中で入力信号やクロック同期のイベントを wait する無限ループを置いて，その中に動作を記述する．モジュール内のレジスタやフリップフロップを buffered 変数として宣言する．このスタイルにしたがった記述例を図 2 に示す．左は組合せ回路の記述例，右はクロック同期のフリップフロップによる順序回路の動作記述例である．この記述では敢えて順序回路の一般形である Mealy 型で記述しているが，もちろん出力が状態のみに依存する（入力に依存しない）Moore 型の順序回路も同様に記述可能である．

```
behavior combi_bh(
   in  signal unsigned bit[8] x,
   in  signal unsigned bit[8] y,
   out signal unsigned bit[8] z
 ){
 unsigned bit[8] a;
 void main(void) {
  while (1) {
    a = x & y;
    z = a;
    wait x, y;
  }
 }
};
```

```
behavior seq_bh(
 in  signal unsigned bit[1] clk,
 in  signal unsigned bit[8] x,
 out signal unsigned bit[8] y){
 unsigned bit[9] sum;
 buffered [clk rising]
  unsigned bit[8] r = 0;
 void main(void) {
  while (1) {
    sum = r + x;
    r = sum[7:0];
    y = sum[7:0];
    wait x, clk rising;
  }
 }
};
```

図 2　回路の基本的な動作記述例．組合せ回路（左）とフリップフロップによる Mealy 型順序回路（右）．

fsmd 構文を用いることで明示的に状態遷移機械を記述することも可能であり，そこでは状態の符号化や状態符号割り当てを陽に考慮する必要がないため効率のよい記述が可能となる．反面，リセット動作やデフォルト動作など詳細なセマンティクスの誤解が心配される．fsmd 構文についての詳細は SpecC 2.0 仕様書を参照されたい．

モジュールとそれらの接続による構造記述は子 behavior をインスタンス化することで行う．相互接続には signal 変数を用いる．構造記述を行う behavior の main()関数内で子 behavior のインスタンスのメイン関数を起動する．par 構文を用いて並列に起動する．図 3 に構造記述の例を示す．

```
behavior connect_bh(
 in  signal unsigned bit[1] rst,
 in  signal unsigned bit[1] clk,
 in  signal unsigned bit[1] set,
 in  signal  signed bit[8] x,
 out signal  signed bit[8] y){
 signal signed bit[8] xm,xp;
 signal signed bit[8] ym,yp;
 fork_bh fork(set,x,xm,xp);
 node_bh m1(rst,clk,set,xm,ym);
 node_bh m2(rst,clk,set,xp,yp);
 join_bh join(ym,yp,y);

          :
```

```
          :

 void main(void) {
  par {
   fork.main();
   m1.main();
   m2.main();
   join.main();
  }
 }
};
```

図 3　モジュールと相互接続による構造記述例．

構造記述を繰り返し適用することにより，木構造の階層化設計を行うことができる．ひとつのモジュールを任意の場所で複数インスタンス化することも可能であるが，ハードウェアであるがゆえに再帰となるインスタンス化は許されない．拡張 par 構文を用いれば，構造記述と動作記述をひとつの behavior に混在させることができる．

モジュールを構造化する際，信号の流れがループを形成する場合がある．信号処理におけるフィードバックループ，暗号化の繰り返し処理，ローカルバッファモジュールとのやり取りなど，がその例である．さらに，複雑なデータパスの切り替え部分など，組合せ回路中に実際には発現しないフォルスループが形成される場合がある．少なくとも，順序回路として記憶素子を挟んでのループは一般的で正常な回路であり，回路シミュレータでの動作検証や合成などは問題なく扱うことができる．しかし，信号の変化と安定の詳細を省いて抽象化した環境下では，これがシミュレーションの無限ループとなってしまう場合がある．図 4 にループ構造を持つ単純な回路の例と図 5 にその記述例を挙げる．SpecC の本来のセマンティクスを示す参照コンパイラでは，

Vol.2010-SLDM-144 No.17
Vol.2010-EMB-16 No.17
Vol.2010-MBL-53 No.17
Vol.2010-UBI-25 No.17
2010/3/26

この記述は無限ループを引き起こしてしまう．それを解消するためには出力 signal 変数に代入する前に中間変数を置いて代入し，値変化を確認してから代入する，などの回避策が必要となる．SpecC.IDT2009 では，signal 変数のイベントは代入ではなく値変化により発生するため，この記述で所望の動作が実現できる．

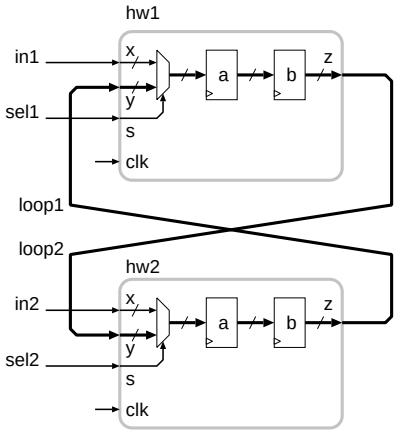

図 4　ループ構造を持つ単純な回路の例．

```
behavior Main(void){
behavior Main(void){
    :
 hw_bh hw1(clk, sel1,in1,loop1,loop2);
 hw_bh hw2(clk, sel2,in2,loop2,loop1);
    :
 int main(void) {
  par {
    :
   hw1.main();
   hw2.main();
    :
  }
 }
};
```

```
behavior hw_bh(
  in  signal unsigned bit[1] clk,
  in  signal unsigned bit[1] int s,
  in  signal int x,
  in  signal int y,
  out signal int z){
  buffered [clk rising] int a;
  buffered [clk rising] int b;
  void main(void) {
   while (1) {
    if (s==0) a = x; else a = y;
    b = a;
    z = b;
    wait s, x, y, clk rising;
   }
  }
};
```

図 5　ループ構造の記述例．

これまで述べてきた組合せ回路，クロック同期フリップフロップに基づく順序回路（Mealy 型，Moore 型），そして，それらの階層化とループの記述により，安全確実で広く使用されている設計スタイルである単相同期型のハードウェアが記述可能となる．

しかしここで，シミュレーション上動作しても回路として実現不可能な記述となることもあり得ることに注意されたい．ソフトウェア言語における変数の挙動が，必ずしも回路中のワイヤやフリップフロップの挙動と等しくはないため，記述によっては，ワイヤとも記憶素子ともつかない変数，ハードウェアとして不可能なタイミングでの値の記憶などといった問題が発生する為である．これは，ハードウェア記述言語でのブロッキング代入による動作記述での問題と同様のものであり，ハードウェア記述言語に変換できてハードウェア記述言語によるシミュレーションで動作したとしても，回路としては実現不可能であることには変わりない．ハードウェア記述言語による記述と同様，結果の回路を思い描きながら記述を進めていく必要がある．

回路の設計最適化のために，必要最小限の単相同期回路からさらに踏み込んだ回路設計が必要になることもある．小規模回路のタイミングの最適化において，ラッチを用いる場合がある．クロックに同期したフリップフロップやレジスタは buffered 変数を用いて陽に記述するが，ラッチは通常のソフトウェアの変数としての記憶動作により表現する．図 6 にラッチの記述例を示す．

また、データパスの切り替えなど，通常の０，１の信号によるセレクタ論理ではなく３ステートバスを用いる場合がある．本来の SpecC の仕様ではハイインピーダンスを含む３ステートを扱うことはできないが，SpecC.IDT2009 で拡張された bit4 型を用いて扱うことができる．３ステートバスの端子と接続信号は bit4 型で宣言し，３ステートゲート部分の記述では切り離し時にハイインピーダンスを代入する．複数のモジュールから駆動されるバスは，値の衝突を解決せねばならない．そのため，変数宣言で resolved を指定する．図 7 に出力ゲートと入力ラッチの記述例を示す．

Vol.2010-SLDM-144 No.17
Vol.2010-EMB-16 No.17
Vol.2010-MBL-53 No.17
Vol.2010-UBI-25 No.17
2010/3/26

```
behavior latch1_bh(
  in  signal unsigned bit[1] g,
  in  signal unsigned bit[8] i,
  out signal unsigned bit[8] o
  ){
  unsigned bit[8] v;
  void main(void){
    while (1) {
      if (g) v=i;
      o=v;
      wait i, g;
    }
  }
};
```

図 6　ラッチの記述例.

```
behavior trigate8_bh(
  in signal unsigned bit[1] gate,
  in signal unsigned bit[8] i,
  out signal resolved
        unsigned bit4[8] o){
  void main(void){
    while (1) {
      if (gate)
          o=i;
      else
          o=0qzzzzzzzzu;
      wait i, gate;
    }
  }
};
```

```
behavior trilatch8_bh(
    in  signal unsigned bit[1] gate,
    in  signal resolved
          unsigned bit4[8] i,
    out signal unsigned bit[8] o
  ){
  void main(void){
    while (1) {
      if (gate) o=i;
      wait i, gate;
    }
  }
};
```

図 7　３ステートバスの出力ゲート（左）と入力ラッチ（右）の記述例.

4.　応用設計例…多相ラッチと３ステートバスによる MPU

ここでは，これまで述べてきた基本記述スタイルを踏まえて，応用的な設計を試みる．十分な試行たるべく様々な要素が盛り込まれた複雑さと，しかし問題の本質を見通しよく理解するための単純さとを併せ持ったモチーフのひとつとして，多相ラッチと３ステートバスによる MPU の設計例をとりあげる．これは実在する組込み制御用 MPU の回路的特徴を抽出して単純化したものである．MPU の概要とブロック図を図 8 に示す．このＭＰＵはひとつのクロックとこれを 4 分周した 4 相のラッチで動作する．命令レジスタ(IR)ならびに即値レジスタ(IV)はクロックエッジで動作し，一般のレジスタ類(R0,R1,R2,R3,carry)は第３相のラッチとして，ALU の入力を保持する中間レジスタは第 1 相のラッチとして動作する．演算ユニットの入出力は３ステートの内部バス(abus, bbus, sbus)により接続される．外部から命令と即値を入力し，それを解釈して計算を実行する．命令は８ビットからなり，演算種別として算術加算，論理積，論理和，論理否定を，入力として即値あるいはレジスタ R1, R2, R3　を，出力として R0,R1,R2,R3 を選ぶことができる．このＭＰＵから外部へは R0 と carry フラグが出力される．

命令レジスタ IR
即値レジスタ IV
ALU
　入力バス abus, bbus
　（ゲート AR1,AR2,…,BIV）
　中間ラッチ AL, BL
　出力バス sbus
汎用レジスタ R1,R2,R3
出力レジスタ R0,carry
命令デコード＆制御部

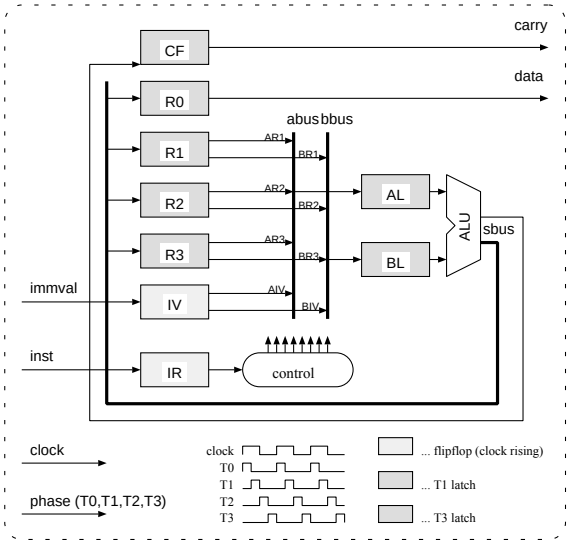

図 8　設計 MPU の概要（左）とブロック図（右）.

Vol.2010-SLDM-144 No.17
Vol.2010-EMB-16 No.17
Vol.2010-MBL-53 No.17
Vol.2010-UBI-25 No.17
2010/3/26

この MPU をこれまで述べてきた記述スタイルに倣って SpecC.IDT2009 により記述した．図 9 のサンプルプログラムを VisualSpec 上で実行させた結果の波形を図 10 に示す．バスやラッチが所望のタイミングで所望の動作をしている．また当該記述を scrtl2verilog を用いてハードウェア記述言語に変換し，回路合成ツールで回路に合成することができた．図 11 はトップモジュールの合成結果の図である．

```
機械語          意 味
0x5001 … R1 = 0x01 & 0x01 (R1 = 0x01)
0x6002 … R2 = 0x02 & 0x02 (R2 = 0x02)
0x7003 … R3 = 0x03 & 0x03 (R3 = 0x03)
0x0b** … R0 = R2+R3
0x55** … R1 = R1 & R1 ( NOP)
```

図 9 被検 MPU のサンプルプログラム．

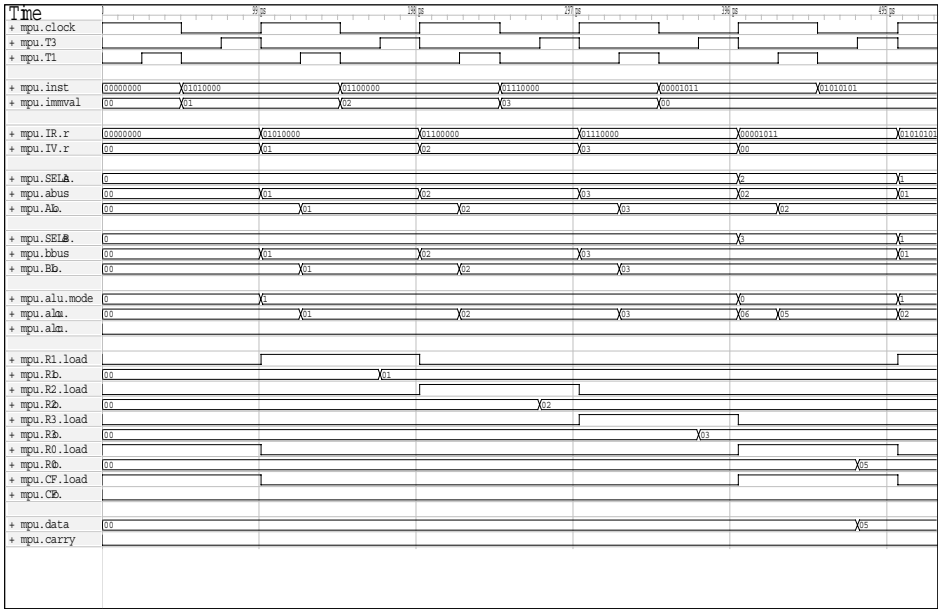

図 10 シミュレーション結果の波形．

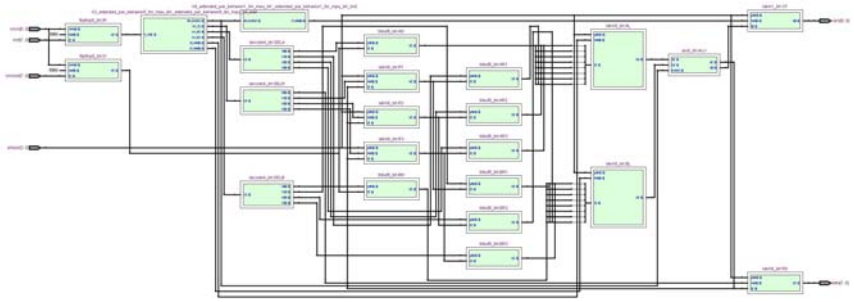

図 11 MPU の合成結果．トップモジュールの構成図．

5. まとめ

仕様記述言語である SpecC について，レジスタ転送レベル記述のために拡張した言語仕様 SpecC 2.0 ならびに SpecC.IDT2009 を用いた回路設計を試行した．基本的な回路要素，記述法項目を整理し，実際にシミュレータと合成系の動作を確認しながら記述法を精査検討し，記述スタイルをまとめた．さらに，応用的な設計を試行しその動作，合成を確認した．今後は，さらに応用事例の幅を広げて適応性・表現力を検証し，またより効果的な記述スタイルを模索し，明確な記述ルールあるいは記述チェッカなど，厳密な運用が可能なものとしていく必要がある．また，実際に上流の仕様記述から詳細設計までの設計フローを試行し，異なるレベルの設計との整合性を確認し，全体としての効果的な設計スタイルを検討していく必要がある．

参考文献

1) Domer, R., Gerstlauer, A. and D. Gajski, D.: SpecC Language Reference Manual, Ver.2.0, SpecC Technology Open Consortium (2002). http://www.specc.gr.jp/tech/SpecC_LRM_20.pdf
2) Gajski, D. et.al 著, 木下常雄, 冨山宏之訳: SpecC 仕様記述言語と方法論, ＣＱ出版 (2000).
3) Fujita, M.: SpecC Language Version 2.0: C-based SoC Design from System level down to RTL, tutorial, Asia and South Pacific Design Automation Conference (ASPDAC) (2003).
4) Gerstlauer, A., Peng, J., Shin, D., Gajski, D., Nakamura, A., Araki, D., Nishihara, Y.: Specify-explore-refine (SER): from specification to implementation, in Proc. Design Automation Conference (DAC) pp.586-591 (2008).

Vol.2010-SLDM-144 No.17
Vol.2010-EMB-16 No.17
Vol.2010-MBL-53 No.17
Vol.2010-UBI-25 No.17
2010/3/26

5) 荒木大：ELEGANT の SpecC シミュレーションと設計詳細化について、第三回先端宇宙情報技術ワークショップ講演集 (2007).
6) 若林一敏：ELEGANT で使う動作合成について、第三回先端宇宙情報技術ワークショップ講演集 (2007).
7) 松永惇弥、村岡道明、荒木大：ソフトウェア並列化を考慮したハードウェア/ソフトウェア分割手法、電子情報通信学会技術研究報告 VLD2009-71, CPSY2009-53, RECONF2009-56 (2010).
8) 井上諭、橋詰大毅、泉知論、福井正博：高位記述言語を用いた画像処理向けディジタルシステム設計の効率化手法、電子情報通信学会技術研究報告 VLD2006-123, ICD2006-214 (2007).
9) V isualSpec Version 4 利用ガイド, InterDesign Technologies, Inc. (2008).