ハードウェア設計環境COMET

1U-7

澁谷利行　進藤達也　河村薫

(株) 富士通研究所

1. はじめに

数万～十万ゲート級のカスタムLSIが実用化されている．このような大規模なLSIを正しく短期間に設計するには，組織的な設計手法が重要となる．

本稿では，同期型デジタル回路を対象としたトップダウンな設計手法を提案し，これを支援するための設計環境COMET (2) について述べる．COMETは，ハードウェア記述言語と，そのシミュレータで構成されている．

2. トップダウン設計手法

COMETでは，以下の項目を前提としたトップダウンな設計手法を想定している．

①データパスと制御回路の分離

ドキュメント性の高い仕様を作成し，詳細化を容易にするために，データパスと制御回路を分離して設計を行う．

設計初期段階において，データパスはブロック図として表現し，制御回路は，論理式あるいは状態遷移表現として表す．

②全回路の機能シミュレーションの実施

設計ミスの早期発見を行うために機能シミュレーションを行う．さらに，シミュレータ上のモデルを用いてファームウェア開発や，システム全体の性能評価を行う．

③回路の階層化

回路の大規模化に対応するために，階層設計は必要不可欠である．分割された部分回路を，それぞれ別の設計者が担当することで，複数設計者による共同作業を行う．

3. COMET言語

上記のトップダウン設計を実現するために，以下の方法でその支援を行っている。

① 記述レベルの混在

COMETでは，次のようなレベルの記述を同一回路内に混在できる．

(a)動作レベル・・・C言語の関数で実現

(b)機能レベル・・・状態遷移，レジスタ転送表現

(c)構造レベル・・・ネットの接続表現

②階層的記述

1つの回路をクラスとして記述を行う．他で記述されたクラスを用いて内部ファシリティを宣言することにより階層的記述を実現する．

③インターフェイス部と実現部の分離

回路の入出力端子を定義するインターフェイス部と，回路機能を定義する実現部を，それぞれ別のクラスとして記述する．インターフェイス部を記述したクラスは，実現部を記述したクラスを継承することにより，回路記述を完成させる．

以上の機能を持ったCOMET言語を，C++ (3) を拡張することにより実現した．

COMET言語によるクラス定義の構成を図1に示す．

4. 回路機能の抽象化

回路機能の抽象化の定義とは，回路の利用者に対して，その回路へのアクセス方法のみを公開し，その回路の実現法をブラックボックス化することである．

回路機能の抽象化の目的は，複数の設計者により作業を進める際に，互いにインターフェイス部の記述が更新されない限り，独立の詳細化が行えることを保証することである．

COMETでは，次の2つの方法により抽象化を支援する．

① 端子による抽象化

端子による抽象化とは，回路を使用する側には，端子，すなわちインターフェイス部のみを見せて，実現部はブラックボックスとして扱うことである．その回路を詳細化する側は，インターフェイス部は触れずに実現部のみに変更を加えて行く．

端子による抽象化は，構造レベル記述における詳細化で用いる．

② 関数による抽象化

COMET - A Hardware Design Environment

Toshiyuki SHIBUYA　Tatsuya SHINDO　Kaoru KAWAMURA

FUJITSU LABORATORIES LTD.

関数による抽象化とは，回路の使用目的ごとに回路のメンバ関数を用意し，使用する側にその関数の呼び出し方法のみ見せることである（1）．回路を実現する側は，メンバ関数の定義として，その使用目的に必要な信号線の設定を記述する．

例として，メモリ回路に対する抽象化のためのメンバ関数を図2に示す．メモリへのアクセスは，関数read, write を用いることで容易に行なえる．

このように，使用目的に対応する関数を作成しておけば，その部品の端子の定義を知らなくても，関数を呼び出すだけで部品が使用できる．

関数による抽象化は，動作レベル・機能レベル記述において用いる．

5. COEMT シミュレータ

シミュレータは，対象が同期型デジタル回路であることと，高速性の必要からレベルソート方式を採用した．

シミュレータの拡張性を高めるため，また制御のための特別な言語を覚える負担を無くすために，シミュレーションの制御用の言語としてはC++ をそのまま用いる．これにより，COMET では回路記述もシミュレーション制御記述も同一の言語で統一的に記述できる．

例えば，シミュレーション制御の記述において，C++ で作成した関数を自由に呼び出すことができるので，グラフィック機能を利用したユーザインターフェイスを構築できる．また，詳細化前と詳細化後の記述を同時にシミュレーションし，結果を毎クロックごとに照合するような操作も容易に行える．

このシミュレーション制御記述とハードウェア記述を，COMET コンパイラによって，一度C言語へ変換してから，実行モジュールを作成する．

5. まとめ

本稿では，トップダウンな設計を支援する環境COMET の言語と，それを用いた回路機能の抽象化法を紹介した．また，シミュレータについて述べた。

現在，回路設計に適用して評価中である。

参考文献

1)進藤，河村：「ハードウェアのオブジェクト指向記述と実行」，信学会 CAS87-111 ，PP. 23～31
2)澁谷，進藤，河村：「トップダウン設計のためのハードウェア記述言語COMET 」，信学会 VLD88-24
3)B. Stroustrup：「The C++ Progamming Language 」, Addison-Wesley, 1986

```
/* インターフェイス部の定義 */
/* 端子によるインターフェイスの宣言 */
class 定義する部品名 : 実現クラス名 {
public:
     InputTerminal 入力端子名;
     OutputTerminal 出力端子名;
     Clock クロック端子名;
     メンバ関数名;
};

/* 関数によるインターフェイスの定義 */
部品クラス名::メンバ関数名(引数宣言,..)
{
/* 信号線の設定による動作レベルの記述 */
}
```

図1（a）　インターフェイス部の定義

```
/* 実現部の定義 */
/* ファシリティの宣言 */
class 実現クラス名 {
        使用部品クラス名   使用部品名 ;
                .
                .
};

/* ネットによる接続関係の定義 */
実現クラス名::Net()
{
/* 接続関数による構造レベル記述 */
     connect(接続端子名,.....);
                .
                .
}

/* 機能動作の定義 */
実現クラス名::operation()
{
/* 状態遷移、レジスタ転送表現による
                          機能レベル記述 */
}

/* クロック動作の定義 */
実現クラス名::クロック端子名()
{
/* クロック入力時における動作の記述 */
}
```

図1（b）　実現部の定義

```
/* 端子によるインターフェイス部の宣言 */
class Memory:primitive {
public:
     InputTerminal  a; /*アドレス端子*/
     InputTerminal  i; /*データ入力端子*/
     InputTerminal  cs;/*チップセレクト端子*/
     OutputTerminal o; /*データ出力端子*/
     read(Signal);
     write(Signal, Signal);
};

/* 関数によるインターフェイス部の定義 */
Memory::read(Signal address)
{
     a = address;
     cs = 0;
     we = 1;
     return o;
}

Memory::write(Signal address, Signal data)
{
     a = address;
     i = data;
     cs = 0;
     we = 0;
     return o;
}
```

図2　メモリのインターフェイス部の定義