システムＬＳＩ設計技術　103－10
（2001．11．29）

信号の型に忠実なSystemCからVerilog-HDLへの言語変換

長尾　文昭†

† 三洋電機(株) システムLSI事業部
〒503–0195 岐阜県安八郡安八町大森180

E-mail: †naga046053@swan.sanyo.co.jp

あらまし　システムレベルの設計言語SystemCから論理合成可能なハードウエア記述言語VerilogによるRTL記述を作成する変換ツールsc2vを作成した。sc2vは、SystemCのオブジェクト指向の記述に対応することを目指している。sc2vの開発にあたり、ハードウエアのオブジェクトをモジュール、接続、信号の３つであると想定した。本稿では、この３つのオブジェクトのうち特に信号のオブジェクト指向への対応と型の特性を忠実に変換する方法について説明する。

キーワード　システムレベル設計、システムレベル記述言語、SystemC、オブジェクト指向、言語変換、verilog

The translation from SystemC to Verilog-HDL keeping the character of signal

Fumiaki NAGAO†

† System LSI Division, SANYO Electric Co.,Ltd.
180, Ohmori, Anpachi-cho, Anpachi-gun, Gifu, 503–0195 Japan

E-mail: †naga046053@swan.sanyo.co.jp

Abstract　I developed the translator "sc2v" from the system level design language SystemC to the RTL level of the hardware description language Verilog. Main concept of this translator is to support several object oriented syntaxes of SystemC. A hardware design has three kinds of object, which are module, connection and signal. In this paper, I explain how to translate signal objects and how to keep the characteristic of them in full .

Key words　System level design, Syetem level description language, SystemC, Object oriented, Language translation, verilog

1. はじめに

近年、システムのほとんどの部分を1個または数個のLSIで実現するようになってきた。そのためシステムのアルゴリズムがLSIの仕様と見なせるようになりLSIの設計もアルゴリズムを検証するためのC/C++言語を用いることが望まれるようになってきている。近年、この様な背景でC/C++言語をベースとしたシステムレベル記述言語がいくつか提案されシミュレーション環境が提供されている。この中で、SystemC [1] はオブジェクト指向のC++言語にライブラリを付け加える形で実現されており、オブジェクト指向の記述スタイルを実現しやすい。このSystemCの特徴を生かしてハードウエア設計で利用するために、オブジェクト指向で使われるクラスやテンプレートの構文を解釈できるSystemCからVerilog-HDLへの変換ツールsc2vを開発中である。

SystemCからHDLを生成するための合成ツールや変換ツールは幾つか[2]～[4]公表されているが、オブジェクト指向や信号の型の扱いに焦点を当てた物は見当たらない。そこで本稿ではsc2vで行われているそれらの手法について報告する。

2. オブジェクト指向のハードウエア設計

ハードウエアの表現においてオブジェクトを識別すると機能からオブジェクトを識別した場合と異なるオブジェクトの抽出になる可能性がある。現段階のsc2vは論理合成ツールへの言語変換を行うツールとして開発されているので、設計者はハードウエアを想定してオブジェクトを識別しなければならない。また現在、LSIの仕様からRTレベルのハードウエアへインプリメントするまでの設計工程で自動化できているのは、スケジューリングとリソースシェアリングである。これを機能合成と呼んでいる。現状では機能合成ツールを用いる場合においても設計者はハードウエアを想定してオブジェクトを識別すべきと考えている。sc2vの開発においてハードウエア設計のために重要と考えているオブジェクトは

- モジュール
- 接続
- 信号

の3つである。

ハードウエアのモデリングにおいては、最初にブロック分割を行う。関係が疎であるものを別々のブロックに分ける。これをSystemCやVerilogではモジュールと呼んでいる。

モジュール間でのデータ交換に必要となるなる物をここでは接続と呼ぶことにする。接続はモジュールのポートを介して行われる。ハードウエアの場合、モジュール内部へのアクセスはポートを介さない限り行うことができない。SystemCで記述する接続は、単純に値を引き渡すものから一定の手続きを踏んでデータや制御を交換する複雑なものまで表現することができる。

モジュール内は、さらにブロック分割されるか、そのモジュールの機能をデータと処理に分けて考察する。データや処理の中には相互の関係の深い物がある。この場合、データとそのデータ固有の処理をセットにして信号と呼ぶことにする。SystemCではモジュールの中にプロセスと呼ばれる信号を使った処理手順を1個から複数個記述する。プロセスは接続や信号の中のデータの値の変化によって起動されるモジュールの手続きである。

この様に考えれば、モジュール、接続、信号の3つのオブジェクトを設計すればハードウエアを設計できることがわかる。SystemCでは、モジュール、インタフェース、信号の仕様をクラスで定義している。すなわち、この3つをSystemCではオブジェクトとして扱っておりオブジェクト指向の設計手法を採り入れることが可能になっている。

3. SystemCとモジュール

SystemCのモジュールはハードウエアのブロックを表現している。シミュレーションの最初に生成されたら最後まで存在する。コンストラクタによって同期処理のための仕掛けをつくり出している。デストラクタはない。

モジュールのカプセル化はC++の通常の手法と同じではない。モジュールの定義でメンバー関数を定義するがハードウエアの記述として見ればモジュールの外部からこれを直接アクセス(注1)することはできない。C++を使ったオブジェクト指向プログラミングでは公開されたメンバ関数を直接アクセスすることでメッセージとするけれども、SystemCのモジュールではポートを介してのみメッセージをやり取りすることができる。メッセージの交換手順はインタフェースによって定義される。

(注1)：SystemCにおいてモジュールのメンバ関数を直接アクセスする記述でシミュレーションを行うことは可能である。しかし、ハードウエアに変換する記述としてみれば規則違反の記述であり、変換ツールは構文エラーとすべきである。

モジュール内のメンバ関数はコンストラクタにおいてセンシティビティを定義することでメッセージと結びつけられる。また、メンバー関数を起動するセンシティブはモジュール内部の信号でも良い。コンストラクタ内でセンシティビティと結びつけられたメンバ関数をプロセスと呼ぶ。

モジュール内のメンバ変数はポートか信号である。メンバ関数内部で呼び出されて使われる。そのデータは実行終了まで記憶される。

シミュレーション時にSystemCでは、デバッグのしやすさからメンバ変数はC++の観点から見れば外部からアクセス可能とみなされる記述を行うことが多い。しかし、ハードウエアに変換する観点から見れば、ポート以外のメンバ変数を外部からアクセスする記述は変換できない。したがって、内部信号を直接アクセスする記述はハードウエアに変換する部分では使用できない。デバッグ用としてであれば記述可能である。

SystemCにおけるモジュールはC++の継承とテンプレートを利用することができる。以上より、ハードウエアへの変換ツールが対応すれば、カプセル化、多態性、継承、テンプレートを使うオブジェクト指向でモジュールを設計可能である。

4. SystemCと接続

SystemCもsc2vも開発途上にある。SystemCは、バージョンが上がるにつれて接続部分の仕様が拡充されている。sc2vの開発に取り掛かった時点でターゲットとしたSystemCのバージョンは1.0であり、接続の記述は単純でデータをやり取りするだけであった。そのため、sc2vはまだ単純なポートしか扱えない。

最新のSystemCバージョン2.0では、ブロック転送などの複雑な接続を表現する手法が追加されている。接続部分をインタフェース、ポート、チャネルの3つの概念に分けて記述する。ポートはモジュールにおける外部アクセスの窓口である。インタフェースによってポートのアクセス手順が規定される。インタフェースは関数をまとめて定義した物でありデータの宣言はできない。チャネルはデータをモジュール間で受け渡す時の媒介である。チャネルの定義ではデータと手順を記述する。複雑な接続はチャネルとインタフェースを定義することでモジュールと独立して設計することができる。sc2vでのチャネルへの対応は今後の課題である。

5. SystemCと信号

C++においては抽象データ型をクラスで定義して作り出すことができる。SystemCで新たに用意された信号の型はクラステンプレートで実現されている。テンプレートの引数でビット幅などのパラメータを与える仕掛けになっている。

SystemCでは、同期処理のために`sc_signal`型が用意されている。`sc_signal`型はテンプレートであり信号の型を引数で与える。`sc_signal`型のオブジェクトから読み出される値はそのプロセスで書き込まれた値ではなく1つ前に処理された値になる。すなわち、`sc_signal`型への代入はVerilogのノンブロッキング代入文と同じ動作をする。

従来のHDLにない特徴的な型として固定小数点型がある。この型は演算においてシフト制御をするだけでなく、オーバーフロー時に飽和処理を行ったり、下位のデータを取り除く時にまるめを行ったりする処理を含ませることができる。

さらに任意の型をクラスで定義することができる。C++は演算子の多重定義ができるので、新たに定義した型の演算は任意の処理をさせることができる。すなわち、信号の型はデータと処理をセットにして定義される。モジュールの場合と異なり信号のカプセル化は、C++の手法をそのまま利用する。継承、テンプレート、多態性もC++そのままに利用することができる。

6. SystemCのVerilogへの変換

SystemCバージョン1.0で用意された記述方法はRTレベルのものでVerilogに変換しやすい。論理合成可能なVerilog記述に変換するため、sc2vは論理合成ツールがVerilog記述に対して与える制限と同じようにSystemCでの記述方法を制限している。

SystemCとVerilogの構文には1対1で対応させることができる部分が多数ある。表1にsc2vで行っている変換のうち1対1で対応しているものを示す。

sc2vは、変換後の記述が論理合成可能になることを前提としているのでSystemCの3種類のプロセスのうちMETHODプロセスのみ対応している。CTHREADプロセスをそのまま変換すると機能合成ツール向きの記述になる。センシティビティを限定しないTHREADへの対応は難しい。

SystemCによる4ビットカウンタの記述例を以下に示す。非同期リセットrstで0に初期化され、クロックckで1カウントアップする記述になっている。

表 1　SystemC と Verilog の対応

SystemC	Verilog
プリプロセッサ #define 他	プリプロセッサ `define 他
モジュール SC_MODULE	モジュール module
モジュールのテンプレート	パラメータ
ポート sc_in 他	ポート input 他
プロセス	always 文
センシティビティ	センシティビティリスト
戻り値のないメンバ関数	task
戻り値のあるメンバ関数	function
if 文	if 文
switch 文	case 文

```
SC_MODULE(count){
  sc_in_clk  ck;
  sc_in<sc_bit > rst;
  sc_inout<sc_uint<4> > do;

  void entity(){
    if(rst)
      do = 0;
    else
      do = do.read() + sc_uint<4>(1);
  }
  SC_CTOR(count){
    SC_METHOD(entity);
    sensitive_pos << ck << rst;
  }
};
```

sc2v の変換結果は次のようになる。実際は、コメント文や他の場合を想定した文が付与されているが説明のため基本部分のみを示す。ここでは SystemC が Verilog-RTL と同様なレベルで記述されており、1 対 1 対応で変換されている。

```
module  count(ck, do, rst);
  input  ck ;
  output [3:0] do ;
  input  rst ;
  reg [3:0] do ;

  always @(posedge ck or
           posedge rst) begin
    if(rst) begin
      do <= 0;
    end else begin
      do <= ({1'b0,do})+(1);
    end //if
  end //always
endmodule //count
```

オブジェクト指向で用いられる構文は Verilog にないため素直に変換できない。sc2v では、表 1 に示される 1 対 1 対応の変換の他に幾つかの C++のオブジェクト指向の構文を解釈する。現状の sc2v のオブジェクト指向への対応状況を表 2 に示す。モジュールは、Verilog においても継承以外を対応していると見なせる。今回特に注力したのは信号に関してである。継承の実装はそれほど困難とは思えないが、我々の設計において優先度が高くないので後回しになっている。

表 2　sc2v のオブジェクト指向への対応

構文	モジュール	インタフェース	信号
カプセル化	○	×	○
多態性	○	×	○
継承	×	×	×
テンプレート	○	×	○

sc2v は、データパスとコントローラの分離機能を持っている。信号に MEM, DATA, TEMPORARY, STEP, STATE の属性を持たせると、図 1 に示されているようなデータパスとコントローラを分離した記述が生成される。メモリインタフェースとセレクタは自動的に生成される。状態遷移を元にした記述から強制的にデータパスを抽出したい場合に使用される。

SystemC は、固定小数点型が宣言可能であることや任意の抽象データ型を定義できることなど信号においても Verilog にない特徴を持つ。これらについては特に取り上げて次の章以降で考察する。

7. 信号の型の整合性

Verilog は 2001 年に改定された規格 (Verilog-2001) からビットベクタで符号が扱えるようになったが、1995 年に制定されたもの (Verilog-1995) は符号が扱えない。sc2v は論理合成ツールの対応状況から Verilog-1995 用に変換する。また、Verilog のビットベクタは小数点が扱えない、演算は式中の変数のビット数を越えられない、飽和やまるめについて考慮できないなど SystemC で扱えるビットベクタ、特に固定小数点型との間には大きな違いが存在する。さらに、SystemC は任意の抽象データ型を定義できる。このように SystemC と Verilog には信号の表現力に大きな違いがある。sc2v は、SystemC と Verilog の

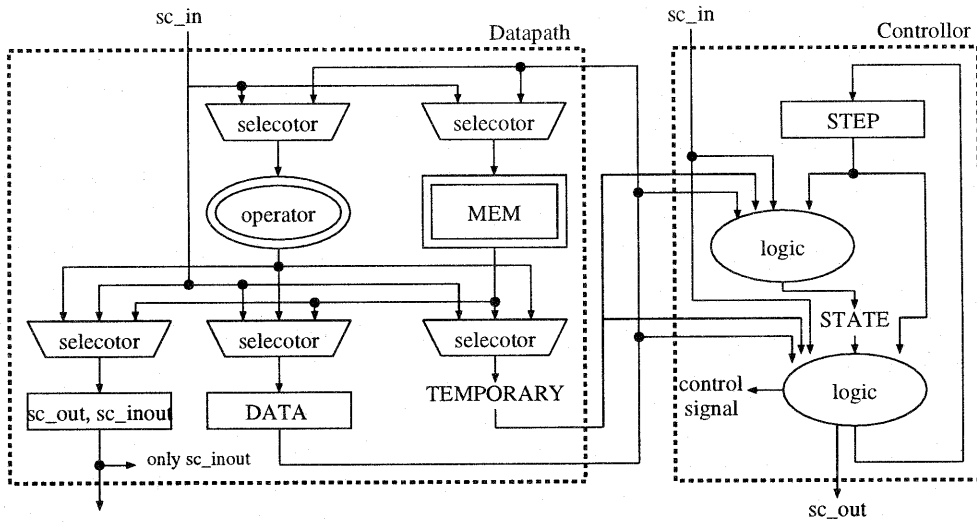

図 1　データパスとコントローラの分離

間の信号に関する表現のギャップを自動的に補って変換している。

8. 信号の型に忠実な変換

sc2v は、型に付随する合成用の情報として

- 種類
- ビット数
- 小数点位置
- 量子化の選択
- オーバーフロー処理の選択

を保持している。型の種類から変換可能かどうかと符号付かどうかを認識する。SystemC の固定小数点型以外の型ではテンプレートのパラメータとしてビット数のみ与える。そこで、固定小数点型以外の型では小数点位置は右端に設定し、量子化とオーバーフロー処理は切捨てとラップに固定される。

固定小数点型の量子化やオーバフロー処理の選択には多数のオプションがあるが、sc2v は量子化においてはまるめまたは切捨て、オーバーフロー処理においては飽和またはラップの 2 つずつの選択を可能にしている。その他のオプションは使用できない。

例として次に示す乗算を検討する。

```
sc_fixed<3,1,SC_RND, SC_SAT> a,b,y;
y = a * b;
```

変数 a,b,y は図 2 に示されるように 3 ビットのデータで右から 1 ビット目と 2 ビット目の間に小数点がある。この型の乗算が取り得る値の範囲は 6 ビットで小数点は右から 2 ビット目と 3 ビット目の間にある。符号付きの型なので Verilog に乗算させる場合には相手の変数のビット幅ほどすなわち変数 a,b は 3 ビットずつ符号拡張を行う。また、変数 y に乗算結果を代入するためには、値の両側のビットを削る必要がある。このビットを削る際に変数 y の型宣言で指定されている SC_RND,SC_SAT に従ってまるめと飽和処理を行う。変換して得られる Verilog 記述は次のようになる。

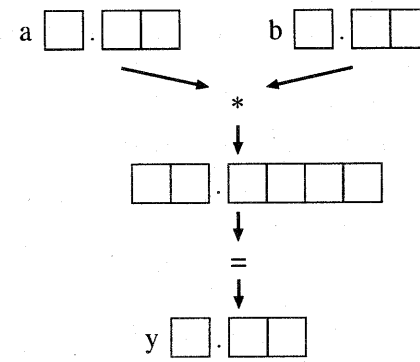

図 2　固定小数点の乗算

```
reg[2:0] a,b,y;
y <= sssat_3_5(rnd_5_7(sup_7_6(
       {{3{a[2]}},a}*{{3{b[2]}},b})));
```

sssat_3_5 は 5 ビットの変数から左 2 ビットを飽和処理して取り除く関数、rnd_5_7 は 7 ビットの変数から右 2 ビットをまるめて取り除く関数、sup_7_6 は 6 ビットの変数を 7 ビットに符号拡張する関数で

ある。この場合の乗算結果の符号拡張は不要であるが、丸めを元に判断して付与されている。

飽和処理には処理前と処理後の変数の符号の有無を考慮して4通りの関数が用意されている。いずれの関数も変換前の値で変換後の変数の型で扱える範囲を外れる値は最も近い表現可能な値、限界値に変換される。sssat関数は符号付きから符号付きへの飽和処理であり範囲外の値は正の最大値か負の最小値に変換される。

例では行われていないが、代入する変数の右側ビットが増える場合はかさあげが行われる。以上のビット幅や小数点位置の調整、飽和、まるめ、かさあげの処理は次の様な場合に行われる。

- 演算子の2項の型が異なる場合
- 関数の引数の型と実際に代入された型が異なる場合
- 左辺の値が代入される変数の型と右辺の演算結果の型が異なる場合

sc2vは、これらの処理の必要な時に自動的にその記述を生成する。したがって、sc2vを使用する場合は変数の型を決める時にのみ型の変換に気を配れば良い。

9. 任意のデータ型の導入

浮動小数点やガロア体などのあらかじめ用意されていない演算規則を持つ信号は、クラス定義を行うことができる。sc2vでは、データ型のクラス宣言の中でメンバ変数を1つだけ宣言する。このメンバ変数に対して行われる処理をメンバ関数やオペレータのオーバーロードで宣言する。

浮動小数点の定義例を次に示す。簡単のため演算の中身の定義は省略している。

```
class float_25{
  sc_bv<25> data;
public:
  float_25();
  float_25(sc_fixed<16,1> di);
  sc_fixed<16,1> operator +(float_25);
  sc_fixed<16,1> get_value();
  bool operator ==(float_32 di){
    return di.data == data;
  }
}
```

新しい信号型を定義しておけば、次に示すように新しい信号型の処理内容を意識することなく処理手順を記述することができる。

```
sc_fixed<16,1> da,db,dy;
float_25 fy;
fy = float_25(da)+float_25(db);
dy = fy.get_value();
```

型が違っても同じ意味合いを持つ演算子や関数は同じ名前を使える。このため型を意識するのは変数の型宣言の時点においてのみであり、それ以外は手順の設計に集中できる。

10. 今後の課題

SystemCのバージョン1.0は、RTレベルと動作レベルの記述に対応している。現状のsc2vは、このうち論理合成を前提としたRTレベルへの対応になっている。次の段階として機能合成を前提とした動作レベルの記述に対応させたい。また、SystemCのバージョン2.0から接続に関する記述方法が充実してきたので接続を独立して設計できる変換機能を持たせたいと考えている。

11. ま と め

SystemCでC++言語のオブジェクト指向の構文を利用する方法を示した。SystemCからVerilogへ変換するsc2vは、現状でモジュールと信号をカプセル化して設計することを可能にしている。従来の再利用はモジュールに対するものであった。sc2vの開発で信号のカプセル化を可能にすることにより信号に対する処理をハードウエア設計においてもソフトウエアのクラスライブラリのように再利用性を高めることができた。

文 献

[1] http://www.systemc.org
[2] 高井幸輔他, "SystemCを用いたハードウェア設計 — SystemCのRTL記述からHDLへの変換", 電子情報通信学会総合大会 A-3-13 ,2001
[3] George Economakos, Petros Oikonomakos, Ioannis Panagopoulos, Ioannis Poulakis and George Papakonstantinou, "Behavioral Synthesis with SystemC", DATE 2001 1B
[4] http://www.synopsys.co.jp/products/cocentric/cocentric_systemC.html
[5] Stuart Sutherland, "The IEEE Verilog 1364-2001 Standard, What's New, and Why You Need It", http://www.verilog-2001.com/verilog-2001_presentation.pdf
[6] 土居範久, 「オブジェクト指向のおはなし」, 日本規格協会 ,1995