Bluespec SystemVerilog によるIP流通と効果的なRTLのデバッグ

他人の書いたRTLを正確に理解し、把握することは簡単なことではありません。他人の書いたRTLに変更を加えてエラーなくインプリメントすることはさらに困難です。これを成功させるためには、設計者はRTLコード全体の設計スタイルに慣れ、コードの詳細を把握し、アーキテクチャとマイクロアーキテクチャを完全に完全に理解する必要があります。これは、RTLに関する笑い話ではなく、切実な問題として考える必要があります。

Bluespec は本質的に RTL よりも良い IP 流通システムである

- Bluespec のソースコードは設計仕様をそのまま記述し、より読みやすい
- Bluespec のソースコードは変更が容易で、安全に機能拡張できる
- RTL のテクノロジでは不可能な、強力にパラメータ化する能力を有し、パラメータ化したまま設計を進めることができる
  ・・・この能力によって IP ベンダや設計者は、設計対象のコア部分のカスタマイズ能力を組み込むことができるようになります。
- 設計者は出力される RTL に対して完全にコントロール可能です
  - 各部の名前
  - インライン化する構造
  - モジュールの階層
  - デバッグのための信号線の挿入
  - シミュレーション出力のような初期値など
- RTL に比べて Bluespec の設計は非常に高い品質をもっている
  - IP ユーザは、ベンダの供給する IP にはバグがあり顧客自身がバグを発見する必要があると予想している
  - 高い品質に加え、Bluespec の設計は修正が容易
- Bluespec の環境では、Bluespec のインタフェースは IP の適切な接続性を保証することが可能で、それ自身が暗黙に行われるアサーションのドキュメントとしてコンパイル時にフォーマルチェックされる

Bluespec は適切に構造化され、読みやすく、組み込みやすく、一般的な Verilog RTL を出力

- Bluespec の出力する RTL は Bluespec のソースコードと直接的な関係をもち、同一のアーキテクチャとマイクロアーキテクチャに従い、ステートエレメントを追加したり削除したりしない
- Bluespec の出力する RTL は一貫性のある、明確な構造により構成されている
  ・・・デザインエレメントやコメントは、モジュール、インタフェース、ステートエレメント、順序回路と組み合わせ回路のように明確にグループ化して構成されています。これによって生成されたコードを理解したり、変更を加えることを容易にします。
- 命名規則はソースからコントロールでき、RTL に適用される
- RTL に挿入するコメントは、コンパイラが自動的に付与するものも含め、ソースコード内で RTL の各モジュールのヘッダになる部分、ステートエレメント、ルール部分に記述することが可能
- Bluespec は、実績のある、幅広い相互運用性をもつ Verilog RTL の出力を自動化できる
  ・・・これらの IP を販売する際には特に、使用する EDA ツールによって受け入れられる Verilog の構造やテクノロジが異なるため確認が必要です。

生成する RTL の見本

添付のコードはBluespecのソースコードと生成されるRTLの見本のための簡単なモデルで、シフトと加算によって2つの数字を乗算する回路です。

Bluespec SystemVerilog design example: mkWidget

```
package Widget;

//
// Interface to the multiplier module
//
typedef Bit#(16) Tin;
typedef Bit#(32) Tout;

interface Widget_IFC;
   (*ready = "StartIsReady", enable = "StartShouldGo", prefix = ""*)
    method Action  start (Tin m1, Tin m2);
// Leave the rest of the methods for standard naming by BSV
    method Tout    result();
    method Action acknowledge();
endinterface


//
// Simple (naive) binary multiplier
//
 (* synthesize *)

(* doc = "This module performs a simple (naive) multiplication of two input values" *)
module mkWidget( Widget_IFC );


//
// State elements
//
(* doc = "The 'product' register holds the result of the multiply" *)
   Reg#(Tout)    product   <- mkReg(0);

(* doc = "The 'mcand' reg holds the value of the multiplicand, which is the intermediate result" *)
   Reg#(Tout)    mcand     <- mkReg(0);

(* doc = "The 'mplr' reg holds the value of the multiplier" *)
   Reg#(Tin)     mplr      <- mkReg(0);

(* doc = "The 'available' reg indicates whether the unit is currently available for further calculations" *)
   Reg#(Bool)    available <- mkReg(True);


(* doc = "The 'cycle' rule defines the core functionality of the multiply" *)
(* doc = "The rule does shift and adds to perform each stage of the multiply. \n The rule ‘fires’ (executes) on
any cycle that mplr!=0.\n If the LSB of the mplr is 1, then mcand is added to the interim calculation.  \nOn every
cycle, mcand is shifted left and mplr is shifted right." *)

// This rule will ‘fire’ or run every cycle that (mplr != 0)
   rule cycle ( mplr != 0 );
      let localsum = product+mcand;
      if (mplr[0] == 1) product <= localsum;
      mcand   <= mcand << 1;
      mplr    <= mplr  >> 1;
      $display("rule cycle just fired!");
   endrule


//
// Interface Methods
//
   method Action start(Tin m1, Tin m2) if (mplr == 0 && available);
       product <= 0;
       mcand   <= {0, m1};
       mplr    <= m2;
       available <= False;
   endmethod

   method Tout result() if (mplr == 0);
      return product;
   endmethod

   method Action acknowledge() if (mplr == 0 && !available);
      available <= True;
   endmethod

endmodule : mkWidget

endpackage : Widget
```

Verilog for mkWidget produced by Bluespec's compiler

```
//
// Generated by Bluespec Compiler, version 3.8.67 (build 8255, 2006-04-11)
//
// On Wed May 10 13:51:13 EDT 2006
//
// Method conflict free info:
// [result CF [acknowledge, result], start CF acknowledge, result SB start]
//
// Ports:
// Name                         I/O  size props
// StartIsReady                   O     1
// result                         O    32 reg
// RDY_result                     O     1
// RDY_acknowledge                O     1
// CLK                            I     1
// RST_N                          I     1
// m1                             I    16
// m2                             I    16
// StartShouldGo                  I     1
// EN_acknowledge                 I     1
//
// No combinational paths from inputs to outputs
//
//
// This module performs a simple (naive) multiplication of two input values
//
//

`ifdef BSV_ASSIGNMENT_DELAY
`else
`define BSV_ASSIGNMENT_DELAY
`endif

module mkWidget(CLK,
                RST_N,

                m1,
                m2,
                StartShouldGo,
                StartIsReady,

                result,
                RDY_result,

                EN_acknowledge,
                RDY_acknowledge);
  input  CLK;
  input  RST_N;

  // action method start
  input  [15 : 0] m1;
  input  [15 : 0] m2;
  input  StartShouldGo;
  output StartIsReady;

  // value method result
  output [31 : 0] result;
  output RDY_result;

  // action method acknowledge
  input  EN_acknowledge;
  output RDY_acknowledge;

  // signals for module outputs
  wire [31 : 0] result;
  wire RDY_acknowledge, RDY_result, StartIsReady;

  // register available
  // The 'available' reg indicates whether the unit is currently available for further calculations
  reg available;
  wire available$D_IN, available$EN;

  // register mcand
  // The 'mcand' reg holds the value of the multiplicand, which is the intermediate result
  reg [31 : 0] mcand;
  wire [31 : 0] mcand$D_IN;
  wire mcand$EN;

  // register mplr
  // The 'mplr' reg holds the value of the multiplier
  reg [15 : 0] mplr;
  wire [15 : 0] mplr$D_IN;
```

```
wire mplr$EN;

// register product
// The 'product' register holds the result of the multiply
reg [31 : 0] product;
wire [31 : 0] product$D_IN;
wire product$EN;

// rule scheduling signals
wire CAN_FIRE_RL_cycle,
     CAN_FIRE_acknowledge,
     CAN_FIRE_start,
     WILL_FIRE_RL_cycle,
     WILL_FIRE_acknowledge,
     WILL_FIRE_start;

// inputs to muxes for submodule ports
wire [31 : 0] MUX_mcand$write_1__VAL_1,
              MUX_mcand$write_1__VAL_2,
              MUX_product$write_1__VAL_1;
wire [15 : 0] MUX_mplr$write_1__VAL_2;
wire MUX_product$write_1__SEL_1;

// action method start
assign StartIsReady = mplr == 16'd0 && available ;
assign CAN_FIRE_start = StartShouldGo ;
assign WILL_FIRE_start = StartShouldGo ;

// value method result
assign result = product ;
assign RDY_result = mplr == 16'd0 ;

// action method acknowledge
assign RDY_acknowledge = mplr == 16'd0 && !available ;
assign CAN_FIRE_acknowledge = EN_acknowledge ;
assign WILL_FIRE_acknowledge = EN_acknowledge ;

// rule RL_cycle
//   The 'cycle' rule defines the core functionality of the multiply
//   The rule does shift and adds to perform each stage of the multiply.
//    The rule fires (executes) on any cycle that mplr!=0.
//    If the LSB of the mplr is 1, then mcand is added to the interim calculation.
//   On every cycle, mcand is shifted left and mplr is shifted right.
assign CAN_FIRE_RL_cycle = mplr != 16'd0 ;
assign WILL_FIRE_RL_cycle = CAN_FIRE_RL_cycle ;

// inputs to muxes for submodule ports
assign MUX_product$write_1__SEL_1 = WILL_FIRE_RL_cycle && mplr[0] ;
assign MUX_mcand$write_1__VAL_1 = { 16'd0, m1 } ;
assign MUX_mcand$write_1__VAL_2 = { mcand[30:0], 1'd0 } ;
assign MUX_mplr$write_1__VAL_2 = { 1'd0, mplr[15:1] } ;
assign MUX_product$write_1__VAL_1 = product + mcand ;

// register available
assign available$D_IN = !StartShouldGo ;
assign available$EN = StartShouldGo || EN_acknowledge ;

// register mcand
assign mcand$D_IN =
           StartShouldGo ?
             MUX_mcand$write_1__VAL_1 :
             MUX_mcand$write_1__VAL_2 ;
assign mcand$EN = StartShouldGo || WILL_FIRE_RL_cycle ;

// register mplr
assign mplr$D_IN = StartShouldGo ? m2 : MUX_mplr$write_1__VAL_2 ;
assign mplr$EN = StartShouldGo || WILL_FIRE_RL_cycle ;

// register product
assign product$D_IN =
           MUX_product$write_1__SEL_1 ? MUX_product$write_1__VAL_1 : 32'd0 ;
assign product$EN = WILL_FIRE_RL_cycle && mplr[0] || StartShouldGo ;

// handling of inlined registers

always@(posedge CLK)
begin
  if (!RST_N)
    begin
      available <= `BSV_ASSIGNMENT_DELAY 1'd1;
      mcand <= `BSV_ASSIGNMENT_DELAY 32'd0;
      mplr <= `BSV_ASSIGNMENT_DELAY 16'd0;
      product <= `BSV_ASSIGNMENT_DELAY 32'd0;
    end
  else
    begin
```

```
        if (available$EN) available <= `BSV_ASSIGNMENT_DELAY available$D_IN;
        if (mcand$EN) mcand <= `BSV_ASSIGNMENT_DELAY mcand$D_IN;
        if (mplr$EN) mplr <= `BSV_ASSIGNMENT_DELAY mplr$D_IN;
        if (product$EN) product <= `BSV_ASSIGNMENT_DELAY product$D_IN;
      end
  end

  // synopsys translate_off
  `ifdef BSV_NO_INITIAL_BLOCKS
  `else // not BSV_NO_INITIAL_BLOCKS
  initial
  begin
    available = 1'b0 /* unspecified value */ ;
    mcand = 32'hAAAAAAAA /* unspecified value */ ;
    mplr = 16'b1010101010101010 /* unspecified value */ ;
    product = 32'hAAAAAAAA /* unspecified value */ ;
  end
  `endif // BSV_NO_INITIAL_BLOCKS
  // synopsys translate_on

  // handling of system tasks

  // synopsys translate_off
  always@(negedge CLK)
  begin
    #0;
    if (RST_N) if (WILL_FIRE_RL_cycle) $display("rule cycle just fired!");
  end
  // synopsys translate_on
endmodule  // mkWidget
```

お問い合わせ先：

CYBERNET

サイバネットシステム株式会社
新事業統括部
〒101-0022 東京都千代田区神田練塀町３富士ソフトビル
Tel: 03-5297-3295　Fax: 03-5297-3637

e-mail: bluespec@cybernet.co.jp
http://www.cybernet.co.jp/bluespec/