システムLSI用CPUコアの設計

前田一樹　柏井啓嗣　浅上雅史　恩田周平　國信茂郎

高知大学理学部 〒780-8520 高知県高知市曙町 2-5-1

E-mail:　kuninobu@is.kochi-u.ac.jp

あらまし　Verilog-HDLを用いて３２ビットRISCマイクロプロセッサのコアをトップダウン設計により行った。具体的には、データパス、および制御部のHDLによる各構成要素の設計と各構成要素の接続による上位階層での全体の実現である。その後、テストベンチの記述とシミュレーション、論理合成とFPGAによる配置配線を行った。本設計の目的は（1）種々の応用に適応するシステムLSI用コアを用意すること。（2）学生の設計教育のためのひな形を用意することである。

キーワード　HDL、トップダウン設計、マイクロプロセッサコア、システムLSI

CPU Core Design For System LSI Using Verilog HDL

Maeda Kazuki　Hiroaki Kashiwai　Masafumi Asami　Shuhei Onda　and　Shigeo Kuninobu

Faculty of Science, Kochi University　2-5-1 Akebono-cho, Kochi, 780-8520, Japan

E-mail:　kuninobu@is.kochi-u.ac.jp

Abstract　We have proposed a topdown design from RTL to layout using Verilog HDL. Target is a 32-bit RISC processor. The processor is a core of system LSI and a model for education. The design covers (1) HDL description of data path and control unit, (2) Test bench description, (3) Simulation (4) Logic synthesis, and (5) Place and route for FPGA.

Keyword　HDL, Topdown Design, Microprocessor Core, System LSI, FPGA

１．はじめに

近年の半導体プロセス技術の進展による集積度の向上、および設計技術の向上によりチップ上にシステムを構成できるシステムLSIの時代になっている。システムLSIを構成するモジュールとしてメモリ、種々の機能要素をはじめとして核となるプロセッサを設計記述言語を基に準備をすることがシステムLSIにおいて非常に重要である。特にプロセッサ部分の設計として、

（1）　命令セットアーキテクチャの設計、

（2）　プロセッサの各種のモジュールの設計、

がシステムLSIの開発でハードウェアの面からも、ソフトウェアの面からも核となるからである。本論分ではシステムLSI用CPUコアとしてMIPSマイクロプロセッサ[1]を基にVerilog HDLによる設計を述べる。本設計の他の目的は学生の設計教育のためのひな形を用意することである。

２．CPUのHDL設計

MIPS命令のサブセットを用いたCPUのHDL設計を行う。データパス設計、および制御回路設計のverilogHDLによる記述を述べる。

データパス設計と制御線

データパスを設計するにあたって、まずは使用する各命令を実行するのに必要な構成要素を洗い出し、それらの要素に基づいて各命令タイプ別にデータパスを設計していく。最後にそれらを結合して全体のデータパスを作る。

次に、それに対応する制御線を設計する。

なお、パイプラインやキャッシュなどの技法を用いなかったため省略する。

全データパスの構成要素と必要な制御線を図 2．1に示す

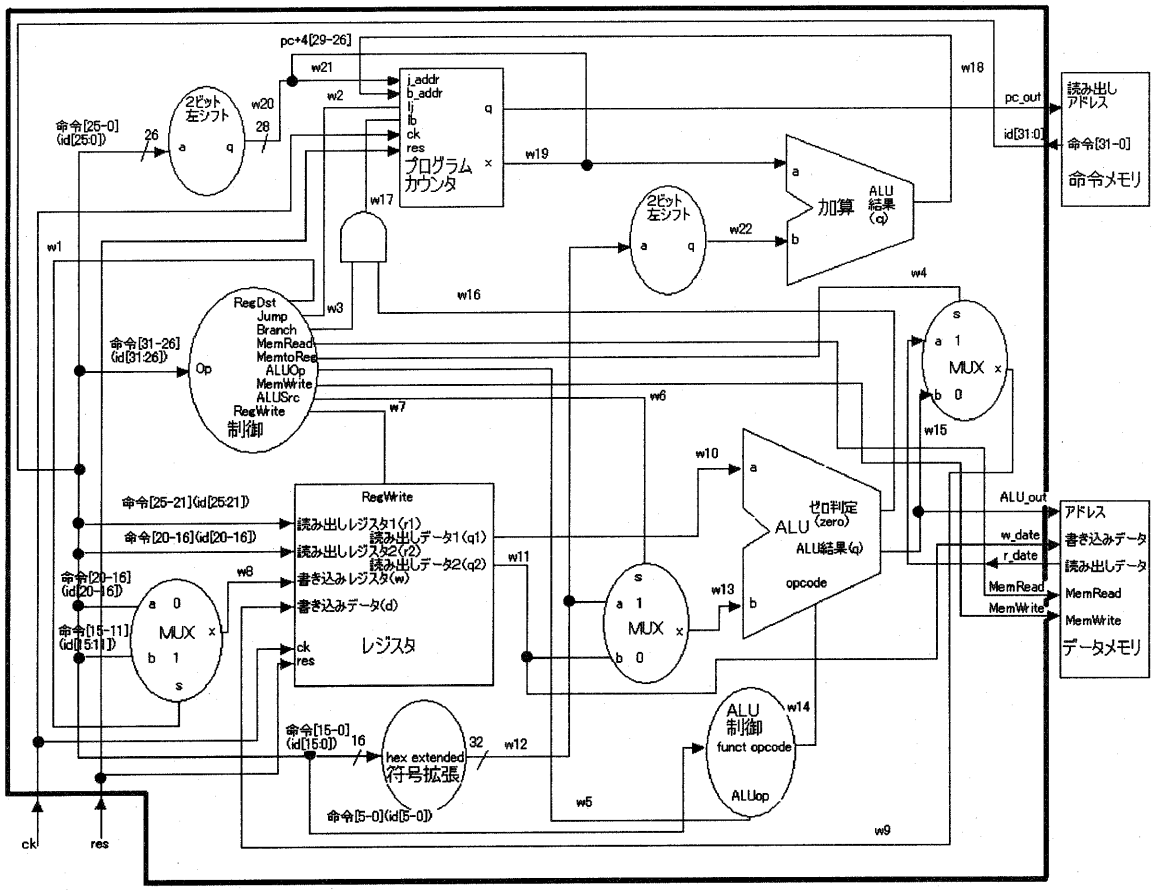


図２．１　全データパス構成要素と必要な制御線

図２．１より、構成要素は次のものになる。

データパス部：

（１）レジスタ
（２）ALU
（３）加算器
（４）符号拡張
（５）プログラムカウンタ
（６）シフタ
（７）マルチプレクサ（５ビット、３２ビット）

制御部：

（８）主制御ユニット
（９）ALU 制御ユニット

上記構成要素のうち、主たる構成要素の HDL 記述を以下に示す。

２．１　レジスタ

```
/*レジスタの HDL 記述*/

module register32 (d, r1, r2, w, RegWrite, ck,
                   res, q1, q2);

input  [31:0]  d;
input  [4:0]   r1, r2, w;
input          RegWrite;
input          ck, res;
output [31:0]  q1, q2;
reg    [31:0]  regf [0:31];
integer        i;

always @(posedge ck) begin
 if (res == 1'b0)
```

```
  for (i = 0; i < 32; i = i + 1)
    regf[i] <= 32'h00000000; /*リセット*/
    else if (RegWrite == 1'b1)
    if (w != 5'b0)  /*0番地には書き込み禁止*/
    regf[w] <= d;              /*書き込み*/
end

 assign q1 = regf[r1],         /*読み出し*/
            q2 = regf[r2];
endmodule
```

2．2　ALU

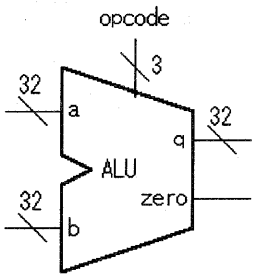


図 2.2 ALU のシンボル図

```
/*ALU の HDL 記述*/
/*マクロ名の定義*/
`define AND 3'b000
`define OR  3'b001
`define add 3'b010
`define sub 3'b110
`define slt 3'b111

module ALU32 (a, b, q, zero, opcode);

 input  [31:0] a, b;
 input  [2:0]  opcode;
 output        zero;
 output [31:0] q;
 reg    [31:0] q;

     /*slt 命令の関数*/
 function [31:0] slt_q;
 input [31:0] a, b;
 if (a < b) slt_q = 32'h00000001;
 else       slt_q = 32'h00000000;
 endfunction

 always @(a or b or opcode) begin
 case(opcode)
   `AND : q <= a & b;
   `OR  : q <= a | b;
   `add : q <= a + b;
   `sub : q <= a - b;
   `slt : q <= slt_q(a, b);
   default:q       <=32'hxxxxxxxx;
 endcase
end

 /*ゼロ判定*/
 assign zero = ~|q;
endmodule
```

2．3　主制御ユニット

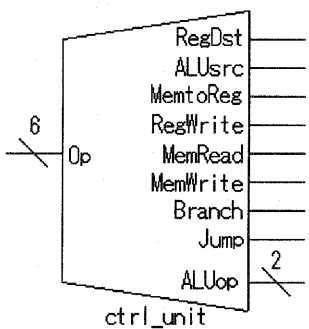


図 2.3　主制御ユニットのシンボル図

図２．３のシンボル図は６ビットの入力 Op（命令のオペコード）を１０ビットの出力（各制御信号）とするデコード回路である。

```
 /*制御ユニットの HDL 記述*/

module ctrl_unit(Op,  RegDst,  ALUsrc,  MemtoReg,
RegWrite, MemRead, MemWrite, Branch, ALUop, Jump);

 input [5:0]   Op;
 output[1:0]   ALUop;
 output        RegDst, ALUsrc, MemtoReg,
               RegWrite,  MemRead,  MemWrite,
               Branch, Jump;
 reg   [1:0]   ALUop;
 reg           RegDst, ALUsrc, MemtoReg,
               RegWrite, MemRead, MemWrite,
               Branch, Jump;
```

```
  /*制御信号の生成*/
  always@(Op) begin
  /*R 形式*/
  if ( {Op[2], Op[1], Op[0]} == 3'b000 ) begin
{RegDst, ALUsrc, MemtoReg, RegWrite, MemRead,
MemWrite, Branch, ALUop, Jump} <= 10'b1001000100;

  /*beq 命令*/
  end else if ( {Op[2], Op[1], Op[0]} == 3'b100 )
  begin
  {RegDst, ALUsrc, MemtoReg, RegWrite, MemRead,
MemWrite, Branch, ALUop, Jump} <= 10'bx0x0001010;

  /*j 命令*/
 end else if ( {Op[2], Op[1], Op[0]} == 3'b010 )
 begin
  {RegDst, ALUsrc, MemtoReg, RegWrite, MemRead,
MemWrite, Branch, ALUop, Jump} <= 10'bxxx000xxx1;

  /*lw 命令*/
  end else if ( {Op[5], Op[3]} == 2'b10 )
  begin
  {RegDst, ALUsrc, MemtoReg, RegWrite, MemRead,
MemWrite, Branch, ALUop, Jump} <= 10'b0111100000;

  /*sw 命令*/
  end else if ( {Op[5], Op[3]} == 2'b11 )
  begin
  {RegDst, ALUsrc, MemtoReg, RegWrite, MemRead,
 MemWrite, Branch, ALUop, Jump} <= 10'bx1x0010000;

  /*その他*/
  end else begin
  {RegDst, ALUsrc, MemtoReg, RegWrite, MemRead,
MemWrite, Branch, ALUop, Jump} <= 10'bxxxxxxxxxx;
  end
  end

  endmodule
```

（2）ALU 制御ユニット

（i）機能

2.4.3 HDL 記述全体

以上で各部品の VerilogHDL で記述することができた。最後に先ほど記述した各部品を上位階層の CPU と接続する。上位階層回路を VerilogHDL で記述し、HDL 記述の完成とする。なおこの記述の信号などは図2．1全データパス構成要素と必要な制御線に対応する。

```
  /*CPU の HDL 記述*/

  module KOA (ck, res, pc_out, id, ALU_out, w_data,
r_data, MemRead, MemWrite);

 input          ck, res;
 input  [31:0]  id, r_data;
 output [31:0]  pc_out, w_data, ALU_out;
 output         MemRead, MemWrite;

 wire           w1, w2, w3, w4, w6, w7, w16, w17;
 wire   [1:0]   w5;
 wire   [2:0]   w14;
 wire   [4:0]   w8;
 wire   [27:0]  w20;
 wire   [31:0]  w9, w10, w11, w12, w13, w15, w18,
                w19, w21,w22;

 assign w17 = w3 & w16;
 assign w21 = {w19[29:26], w20};
 assign w11 = w_data;
 assign  w15 = ALU_out;

 ctrl_unit                                      i0
(.Op(id[31:26]), .RegDst(w1), .Jump(w2), .Branch
(w3), .MemRead(MemRead), .MemtoReg(w4), .ALUop(w
5), .MemWrite(MemWrite), .ALUsrc(w6), .RegWrite(
w7));

 register32                                     i1
(.ck(ck), .res(res), .r1(id[25:21]), .r2(id[20:1
6]),  .w(w8), .q1(w10), .q2(w_data), .RegWrite(w
7), .d(w9));

 ALU32                                          i2
(.a(w10), .b(w13), .zero(w16), .q(ALU_out), .opc
ode(w14));

 ALU_ctrl                                       i3
(.funct(id[5:0]), .opcode(w14), .ALUop(w5));

 add i4 (.a(w19), .b(w22), .q(w18));
```

```
 sign_extend i5 (.hex(id[15:0]), .extended(w12));

 pc2                                              i6
(.ck(ck), .res(res), .q(pc_out), .lb(w17), .lj(w
2), .j_addr(w21), .b_addr(w18), .x(w19));

 l2_shift2 i7 (.a(id[25:0]), .q(w20));

 l2_shift i8 (.a(w12), .q(w22));

 mux5                                             i9
(.a(id[20:16]), .b(id[15:11]), .s(w1), .x(w8));

 mux32 i10 (.a(r_data), .b(w15), .s(w4), .x(w9));

 mux32 i11 (.a(w12), .b(w11), .s(w6), .x(w13));

endmodule
```

3．テストベンチとシミュレーション

CPU のシミュレーションは､CPU を構成するカウンタやレジスタ、ALU 等を各々シミュレーションし、次にジャンプ命令に対応したプログラムカウンタや R 形式の命令を実行可能なデータパスなど機能別のデータパスにおいてのシミュレーションを行い、最終的にすべてのデータパスを統合して CPU としてシミュレーションを行い動作を確認した。また、簡単な演算プログラムによるシミュレーションにより動作を確認した。

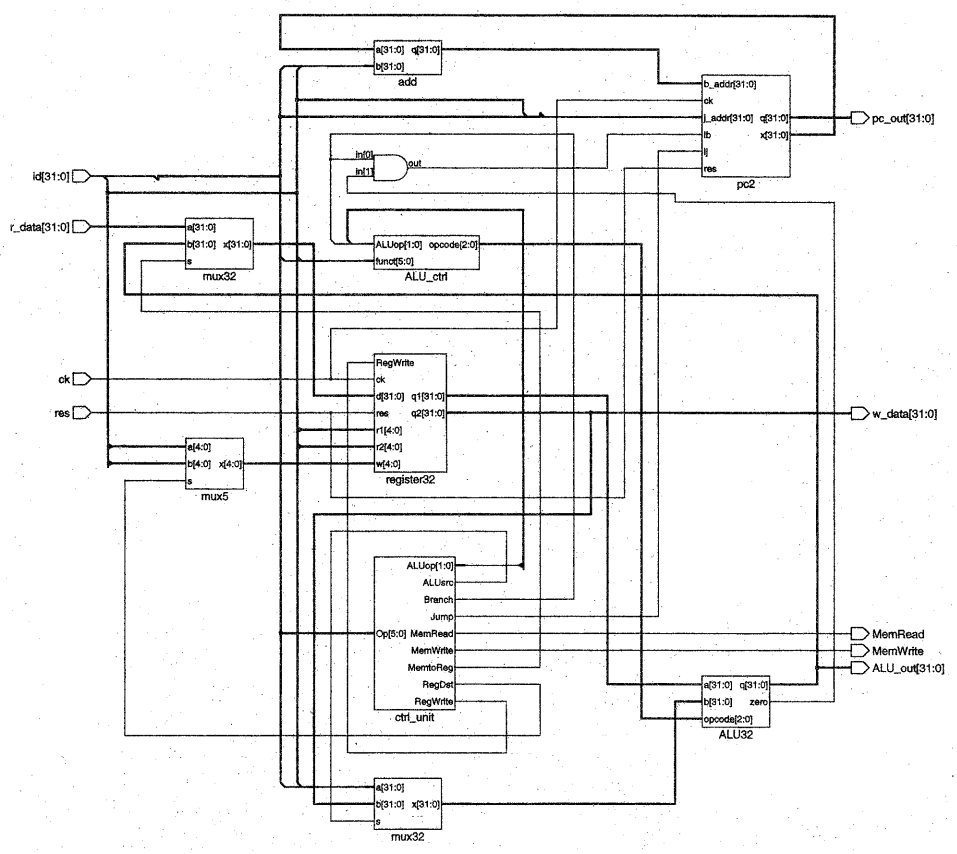
プログラム例は（１）メモリにある任意の２数の値のロードと実行する演算の選択、（２）各種演算（加算、減算、乗算、除算、累乗計算、階乗計算）である。

4．論理合成と FPGA の配置配線

4．1　論理合成の実行とゲートレベルシミュレーション

論理合成は HDL 記述をゲートレベル（テクノロジー独立のネットリスト）に変換する論理変換とテクノロジーマッピングに分けられる。

第４．１図に論理変換の例を示す。



第４．１図　　論理変換の例

４．２　FPGA の配置配線の実行

次に、配置配線（Place & Routing, P&R）を行う。FPGA では、ネットリスト上のインスタンスを FPGA のロジックセルに割り当て、ロジックセル間の接続を行う。

５．まとめ

本研究では、HDL からのトップダウン設計で、HDL 記述、シミュレーション、ターゲットとした FPGA の論理合成、そして配置配線を行なった。今後はタイミング検証を実施し、正確な動作速度を見積もりコアの作成を行いチップとして実証を行う。また、このコアおよび浮動小数点プロセッサコア[2]を用いて種々の応用に適したプロセッサの設計を行う。

参考文献

[1]D. A. Pattersopn, and J. L. Hennessy, Computer Organization & Design, Second Edition, Morgan Kaufmann Publishers, Inc. Sanfrancisco, California.

[2]H. Edamatsu, T. Taniguchi, T. Nishiyama, and S. Kuninobu, A 33MFLOPS Floating Point Processor using Redundant Binary Representation, IEEE、ISSCC1988, pp. 152-, February 1988.