# 基于 FPGA 的单周期 CPU 设计与实现

杨 杨
(南京师范大学强化培养学院，南京 210046)

**摘 要：** 选择 XINLINX 公司的 FPGA 芯片，采用 ISE9.1i 开发工具，介绍了单周期 CPU 编译程序与仿真功能。

**关键词：** Verilog HDL；FPGA；单周期 CPU

# Designing and Implementing of Single-cycle CPU by Using FPGA

YANG Yang
(Honor School, Nanjing Normal University , Nanjing 210046)

**Abstract:** In this paper, we introduce compile program and simulate function of a single-cycle CPU by using ISE9.1i development tool for FPGA Chip by XINLINX Company.

**Key words:** Verilog HDL ; FPGA ; single-cycle CPU

## 1 引言

在已有研究的基础上，围绕单周期 CPU 设计，进行深入的探索，给出一种用 VHDL 语言和 FPGA 芯片实现的单周期 CPU 精简指令系统，该系统包括指令控制、时间控制、操作控制和数据加工等功能。通过该系统的设计过程，可充分展示出 FPGA 的强大功能和优越特性。此外，自己开发精简指令 CPU 还具有以下优点：（1）可以根据不同课题设计需要，临时增加一些硬件资源和指令；（2）开发者可熟悉 CPU 的资源和指令；（3）可控制 CPU 的规模，节约 FPGA 的面积.

## 2 单周期 CPU

### 2.1 核心内容

CPU 设计侧重涉及到数据通路的设计和控制逻辑的设计，其核心内容如下：

（1）分析每条指令的功能。

（2）依据指令的功能给出所需的元件，并考虑如何将他们互联。

（3）确定每个元件所需控制信号的取值。

（4）汇总各指令设计的控制信号，生成反映指令与控制信号之间的关系表。

（5）根据关系表，得到每个控制信号的逻辑表达式，据此设计控制电路。

一个指令系统往往有几十到几百条指令，实现一个完整指令系统的处理器是一项具有挑战性的任务。

### 2.2 实现

设计 CPU 的首要步骤就是确认每条指令的功能，表 1 给出了 11 条 MIPS 指令的 RTL 描述。

表 1 11 条目标指令功能的 RTL 描述

| 指令 | 功能 |
|---|---|
| Add rd,rs,rt<br>Sub rd,rs,rt | M[PC],PC←PC+4<br>R[rd]←R[rs]±R[rt] |
| Subu rd,rs,rt | R[rd]←R[rs] - R[rt] |
| Slt rd,rs,rt | If(R[rs]<R[rt]) R[rd]←1 else R[rd]←0 |
| Sltu rd,rs,rt | If(R[rs]<R[rt]) R[rd]←1 else R[rd]←0 |
| Ori rt,rs,imm16 | R[rt]← R[rs]\|ZeroExt(imm16) |
| Addiu rt,rs,imm16 | R[rt]← R[rs]+SignExt(imm16) |
| Lw rt,rs,imm16 | Addr← R[rs]+SignExt(imm16)<br>R[rt]←M[Addr] |
| sw rt,rs,imm16 | **Addr← R[rs]+SignExt(imm16)**<br>**M[Addr] ← R[rt]** |
| Beq rt,rs,imm16 | Cond← R[rs]-R[rt]<br>If(Cond eq 0) PC←PC+(SignExt(imm16)×4) |
| J target | PC<31:2>← PC<31:28>\|target<25:0> |

依据表 1，单周期 CPU 程序的描述如下:

```
module  /* 数据通路 */
SingleDataLoad (clk,RegWr,RegDst,ExtOp,ALUSrc,Branch,
Jump,MemtoReg,MemWr,ALUctr) ;
  input clk;
//input [31:0]  Instruction;
output RegWr,RegDst,ExtOp,ALUSrc,Branch,Jump,
MemtoReg,MemWr;
output [2:0]  ALUctr;
wire RegWr,RegDst,ExtOp,ALUSrc,Branch,Jump,MemtoReg,
MemWr;
wire [2:0]  ALUctr;
wire [31:0]  Instruction;
wire [4:0]  Rd,Rt,Rs;
wire [5:0]  op,func;
wire [4:0]  Rx;
wire P1,P2,Zero,Overflow;
wire [31:0]  busW,busA,busB,out1,dataout,busB1,busBo;
wire [15:0]  imm16;
 Control  con ( Instruction,RegWr,ExtOp,ALUSrc,ALUctr,
Branch,Jump,RegDst,MemtoReg,MemWr) ;
assign op=Instruction [31:26] ;
assign func=Instruction [5:0] ;
assign Rs=Instruction [25:21] ;
assign Rt=Instruction [20:16] ;
assign Rd=Instruction [15:11] ;
assign imm16=Instruction [15:0] ;
assign P1=P2&RegWr;
MUX2 mux2 (RegDst,Rt,Rd,Rx) ;
assign busB1= {{16 {imm16 [15]  & ExtOp}} ,imm16
[15:0]} ;
MUX2TO1 mux1 (ALUSrc,busB,busB1,busBo) ;
```

**收稿日期：** 2011-09-28

```
    Registers Reg (clk,busW,P1,Rx,Rs,Rt,busA,busB) ;
    ALU alu (busA,busBo,ALUctr,out1,Overflow,Zero) ;
    assign P2=! Overflow;
    DataStore datas (clk,busB,out1 [4:0] ,MemWr,dataout) ;
    MUX2TO1 mux3 (MemtoReg,out1,dataout,busW) ;
    GetCode get (Branch,Zero,Jump,clk,Instruction) ;
    endmodule
    module MUX2TO1 (op,X,Y,Z) ;
    input op;
    input [31:0] X,Y;
    output [31:0] Z;
    reg [31:0] Z;
    always@ (op)
      begin
           if (op==1)
                Z=Y;
           else
                Z=X;
      end
    endmodule
    module MUX2 (op,x,y,z) ;
    input op;
    input [4:0] x,y;
    output [4:0] z;
    reg [4:0] z;
    always@ (op)
      begin
           if (op==1)
                z=y;
           else
                z=x;
      end
    endmodule

module alu (A,B,ALUctr,Zero,Overflow,Result) ; /* 算数逻辑运
算单元 */
 parameter n = 32;
 input [2:0] ALUctr;
 input [n-1:0] A, B;
 output [n-1:0] Result;
 output Zero,Overflow;
   wire   [ n -1:0]   I,H,M,G,K,L,Add_carry,Add_Overflow,
Add_Sign,Add_Result;
 wire SUBctr,OVctr,SIGctr;
 wire [1:0] OPctr;

 changek change (ALUctr,SUBctr,OVctr,SIGctr,OPctr) ;
 assign H=B^ {n {SUBctr}} ;
   adderk   adder ( SUBctr,A,H,Add_carry,Zero,Add_Overflow,
Add_Sign,Add_Result) ;
 mux2to1k mux1 (SIGctr,I,K,G) ;
 mux2to1k mux2 (G,1,0,L) ;
 mux3to1k mux3to1 (OPctr,Add_Result,M,L,Result) ;
 assign M=A^B;
 assign K=SUBctr^Add_carry;
 assign I=Add_Sign^Add_Overflow;
 assign Overflow =OVctr ^Add_Overflow;
endmodule

module adderk (Cin,X,Y,scarry,szero,soverflow,ssign,sresult) ;
 parameter k=32;
 input [k-1:0] X,Y;
 input Cin;
 output [k-1:0] sresult;
 output scarry,szero,soverflow,ssign;
 reg [k-1:0] sresult;
 reg scarry,szero,soverflow,ssign;

 always @ (X or Y or Cin)
  begin
    {scarry,sresult} =X+Y+Cin;
   if (sresult==0)
    szero=1;
   else
    szero=0;
   soverflow=X [k-1] & Y [k-1] & ! sresult [k-1] | ! X
[k-1] &! Y [k-1] & sresult [k-1] ;
   if (sresult [k-1] ==1)
    ssign=1;
   else
    ssign=0;
 end
endmodule

module mux2to1k (sel,X,Y,S1) ;
 parameter k=32;
 input [k-1:0] sel,X,Y;
 output S1;
 reg S1;

 always @ (X or Y or sel)
  if (sel==1)
    begin
      S1=X;
    end
  else
    begin
      S1=Y;
    end
endmodule

module mux3to1k (N,X,Y,Z,S) ;
 parameter k=32;
 input [1:0] N;
 input [k-1:0] X,Y,Z;
 output [k-1:0] S;
 reg [k-1:0] S;

 always@ (X or Y or Z or N)
     if (N==2'b00)
       begin
       S=X;
```

```
      end
     else
      if  (N==2'b01)
        begin
        S=Y;
        end
      else
       if  (N==2'b10)
         S=Z;
endmodule
module changek (al,su,ov,si,op) ;
 input [2:0]  al;
 output su,ov,si;
 output [1:0]  op;
  assign su=al [2] ;
  assign ov=! al [1]  & al [0] ;
  assign si=al [0] ;
  assign op [1] =al [2]  & al [1] ;
  assign op [0] =! al [2]  & al [1]  & ! al [0] ;
endmodule
module jcqz (clk,RegWr,Rw,busW,Ra,Rb,busA,busB) ; /* 数据寄存器 */
input clk,RegWr;//run;
input  [4:0]  Rw;
input  [31:0]  busW;
input  [4:0]  Ra,Rb;
output  [31:0]  busA,busB;
reg  [31:0]  busA,busB;
reg  [31:0]  mem [31:0] ; /*32 个 32 位寄存器 */
reg y=1;
always@ (negedge clk)
if  (y==1)
 begin
      mem [1]  <= 10;
      mem [2]  <= 10;
      y <= 0;
 end
 else
   if (RegWr==1)
    begin
     mem [Rw] <=busW;
    end
always@ (Ra or Rb )
 begin
  busA<=mem [Ra] ;
  busB<=mem [Rb] ;
 end
endmodule
module datamem (datain,addr,WE,clk,dataout) ;
parameter n=32;
input  [n-1:0]  datain;
input  [4:0]  addr;
input WE,clk;
output  [n-1:0]  dataout;
wire  [n-1:0]  dataout;
reg  [n-1:0]  Mem [32:0] ;
always@ (negedge clk)
begin
  if (WE)  Mem [addr] <=datain;
end
assign dataout=Mem [addr] ;
endmodule

module GetCode (Branch,Zero,Jump,clk,Instruction) ; /* 取指令部件 */
input Zero,Branch,Jump,clk;
output  [31:0]  Instruction;
wire  [31:0]  Instruction;
reg  [29:0]  PC=28;
reg reset=1;
wire  [4:0]  addmem;
assign addmem= {PC [2:0] ,2'b00} ;
wire  [29:0]  a,e,c;
wire h;
wire  [29:0]  b,d,f,g;
assign h=Zero & Branch;
geti geti1 (addmem,Instruction) ;
always@ (negedge clk)
      begin
    if  (reset==1)
          begin
          PC <=0;
          reset<=0;
          end
    else
          begin
          PC <= g;
          end
      end
assign c= {{14 {Instruction [15]}} , {Instruction [15:0]}} ;
assign a=PC;
Adder adder1 (a,1,b) ;
Adder adder2 (b,c,d) ;
MUX2T1 mux1 (h,b,d,f) ;
assign e= {a [29:26] ,Instruction [25:0]} ;
MUX2T1 mux2 (Jump,f,e,g) ;
endmodule
module Adder (X,Y,Z) ;
input  [29:0]  X,Y;
output  [29:0]  Z;
wire  [29:0]  Z;
assign Z=X+Y;
endmodule
module MUX2T1 (op,X,Y,Z) ;
input op;
input  [29:0]  X,Y;
output  [29:0]  Z;
reg  [29:0]  Z;
always@ (op)
      begin
          if (op==1)
```

```
            Z=Y;
        else
            Z=X;
    end
endmodule
module geti (addmen,Instruction) ;
input [4:0] addmen;
output [31:0] Instruction;
reg [31:0] Instruction;
reg [31:0] Mem [31:0] ;
always @ (*)
    begin
                                    Mem [0] <=
{6'b000100,5'b00001,5'b00010,5'b00000,5'b00000,6'b000001};
                                    Mem [8] <=
{6'b000010,5'b00000,5'b00000,5'b00000,5'b00000,6'b000010} ;
  end
always @ (addmen)
    begin
        Instruction <= Mem [addmen] ;
    end
endmodule
moduleControl (Instruction,RegWr,ExtOp,ALUSrc,ALUctr,
Branch,Jump,,RegDst,MemtoReg,MemWr) ; /* 总控制器 */
output  RegWr,RegDst,ExtOp,ALUSrc,Branch,Jump,MemtoReg,
MemWr;
reg  RegWr,RegDst,ExtOp,ALUSrc,Branch,Jump,clk,MemtoReg,
MemWr;
output [2:0] ALUctr;
reg [2:0] ALUctr;
input [31:0] Instruction;
wire [5:0] op,func;
assign op=Instruction [31:26] ;
assign func=Instruction [5:0] ;
parameter S0 =6'b100000,S1 =6'b100010,S2 =6'b100011,S3 =
6'b101010,S4 =6'b101011,S5 =6'b001101,S6 =6'b001001,S7 =
6'b100011,S8=6'b101011,S9=6'b000100,S10=6'b000010;
always@ (op or func)
begin
    if (op==6'b000000)
        begin
            case (func)
                S0:
                begin
                    Branch=0;
                    Jump=0;
                    RegDst=1;
                    ALUSrc=0;
                    ALUctr=3'b001;
                    MemtoReg=0;
                    RegWr=1;
                    MemWr=0;
                end
                S1:
                …
            endcase
        end
    else
        begin
            case (op)
                S5:
                begin
                    Branch=0;
                    Jump=0;
                    RegDst=0;
                    ALUSrc=1;
                    ALUctr=3'b010;
                    MemtoReg=0;
                    RegWr=1;
                    MemWr=0;
                    ExtOp=0;
                end
                …
            endcase
        end
end
endmodule
```

## 3 仿真结果

为了说明设计的单周期 CPU 的有效性，采用 MAX + PLUS Ⅱ软件编译并仿真，部分仿真波形图如图 1 所示。

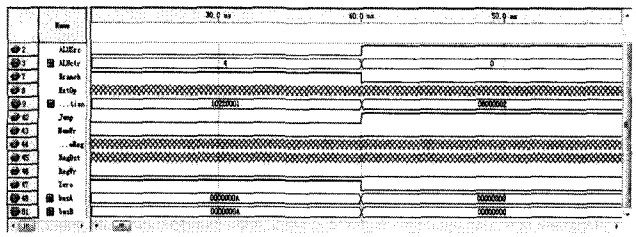

图 1 单周期 CPU 的时序图

## 4 结语

为深入掌握 CPU 的功能，提出了基于 FPGA 的单周期 CPU 设计方案，并采用 ISE9.1i 开发工具进行了程序的编译与功能仿真，实现了单周期 CPU 的硬件电路描述。

## 参考文献


[1] 翟文正，管功湖. EDA 在《计算机组成与结构》课程设计中的实践与探索. 微型电脑应用，2009,25 (12) :10-12.

[2] 吴秀敏，王晓兰，方运潭. FPGA 在硬件设计 CPU 中的应用. 高等工程教育研究，2008 (增刊)：137-138.

[3] 方恺晴，徐成，刘峰. 基于 EDA 技术的教学型 CPU 的设计与实现. 实验技术与管理，2005,22 (9) :41-43.

[4] 赖先志. 基于 FPGA 的简单 CPU 设计. 重庆职业技术学院学报 (工程技术版)，2005,14 (1) : 117-119.

[5] 周宁宁，刘胜. 基于 FPGA 技术的 CPU 模型机的设计与实现.南京邮电学院学报，2003, 23 (1) :77-80.

[6] 张有志，孙科. 一种基于 FPGA 的微处理器系统. 山东大学学报 (工学版)，2003, 33 (4) : 407-409.