电 子 科 技 大 学

数字设计与微处理器系统

课程设计报告

学 生 姓 名：陈烁　崔越　王逸文

学　　号：2016000201017 2016000202020 2016080201002

设 计 题 目：MIPS 指令多周期 CPU 设计

指 导 老 师：马上、陈学英

时　　间：2018.7.13

摘要

利用有限状态机的原理，设计了一种 MIPS 多周期指令 CPU。包括 ALU 模块，控制模块，D 触发器，带使能端的 D 触发器，寄存器文件模块，符号扩展模块等。实现了加减乘、与或、逻辑/算数移位等基本运算，扩展运算功能为立即数与和立即数或。用汇编语言实现了一种 32 阶 FIR 低通滤波器，将其作为 CPU 的测试程序，CPU 运行结果的信噪比达到 48.126dB。在 Basys3 开发板上完成了所设计 CPU 的板级测试。

目录

第 1 章　引言

1.1　项目研究目的与意义

通过本设计，掌握多周期 CPU 的工作原理，控制器、运算器等部件设计的基本方法和技能，并通过用与、或、非、异或等门级电路实现的结构化描述编写部分部件，加深对基本原理的理解和掌握。

通过在所设计的 CPU 上实现 FFT,FIR 等基本算法，学习高级语言、汇编语言以及机器语言的转换，加深对 MIPS 指令集的理解和掌握。

通过使用硬件描述语言 Verilog、EDA 工具软件进行软件设计与仿真，并在 FPGA 上实现，以掌握时序电路的工作原理，培养调试，动手操作的能力。

在了解设计的流程和框架的基础上，通过开放性设计，培养创新思维。利用多种优化方法优化 CPU 性能，提高解决问题的能力。

1.2　项目研究的主要内容及关键技术

- 位宽为 32 位的 ALU 模块的设计与实现

使用与、或、非、异或等门级电路实现的结构化描述编写硬件描述语言，以门级独立完成了设计和实现并且仿真测试了 ALU，支持 MCU 实现。进行了综合，仿真波形和调试。根据仿真波形验证了计算的正确性。

- 控制模块（Control Unit）的设计与实现

包括主控制单元和 ALU 译码器。根据控制器的接口要求，在有限状态机的基础上，对于当前的 PC 的 opcodee 实（和 funct）字段，产生指令执行的控制码和运算码，实现了译码，执行，储存器寄存器读写等状态转换，并进行了综合，仿真波形和调试。根据仿真波形验证了状态转换的正确性。

- 其他模块的设计与实现

使用硬件描述语言，搭建了数据通路中需要的触发器，多路选择器等单元，搭建寄存器文件，设计数据通路，并进行了综合，仿真波形和调试。根据仿

真波形验证了设计的正确性。

- 32 阶 FIR 滤波器的设计与实现

  用 matlab 下的数字滤波器设计工具设计了一个 32 阶的低通滤波器，将系数以 16 比特二进制形式输出，其中整数位 3 位，符号位 1 位，小数位 12 位。所使用检测序列的输入输出位宽也为整数位 3 位，符号位 1 位，小数位 12 位。用基于 MIPS 指令集的汇编或者机器码编写了滤波器程序，以支持 CPU 实现。CPU 实现 FIR 滤波器时调用了非 DSP 的 IP Core。进行了综合，仿真，检验了所设计 FIR 滤波器的有效性，与正确结果比对性能达到了 48.162dB。

- 上板验证

  下载程序到开发板，通过添加 ILA 检测了实际信号的结果。

第 2 章　项目系统方案设计

2.1　项目系统设计原理

本设计的多周期 CPU 工作原理图如图 1 所示。

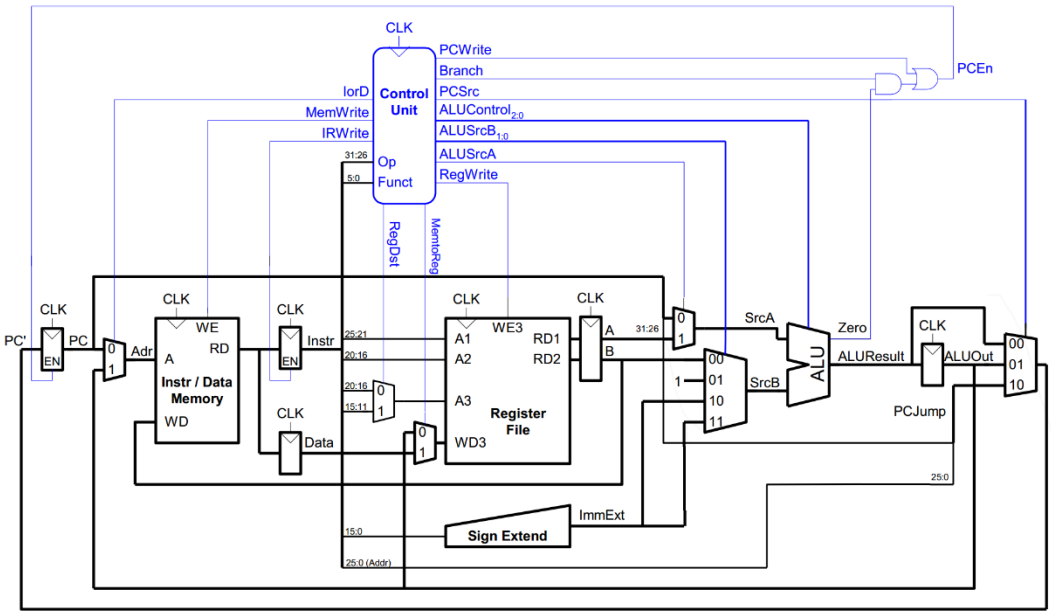


图 1　所设计 CPU 原理图

多周期 CPU 的特点是把指令执行分为多个阶段，每个阶段在一个时钟周期内完成，该时钟周期由最复杂的阶段所花时间决定。由于在阶段与阶段之间需要实现硬件间的数据传递，因此每一步都设置存储元件，执行结果在下个时钟周期到来

前保存到相应存储元件中，并在下一个时钟沿到来时取出。本设计所用到的主要指令及其功能如表 1 所示。

表 1 本设计所涉及的 MIPS 指令

R 型指令							
指令	[31:26]	[25:21]	[20:16]	[15:11]	[10: 6]	[5:0]	功能
Add	000000	rs	rt	rd	000000	100000	寄存器加
Sub	000000	rs	rt	rd	000000	100010	寄存器减
And	000000	rs	rt	rd	000000	100100	寄存器与
Or	000000	rs	rt	rd	000000	100101	寄存器或
Xor	000000	rs	rt	rd	000000	100110	寄存器异或
Sll	000000	00000	rt	rd	sa	000000	左移
Srl	000000	00000	rt	rd	sa	000010	逻辑右移
Sra	000000	00000	rt	rd	sa	000011	算术右移
Jr	000000	rs	rt	rd	000000	001000	寄存器跳
I 型指令							
Addi	001000	rs	rt	immediate			立即数加
Andi	001100	rs	rt	immediate			立即数与
Ori	001101	rs	rt	immediate			立即数或
Lw	100011	rs	rt	offset			取数据
Sw	101011	rs	rt	offset			存数据
Beq	000100	rs	rt	offset			相等转移
Lui	001111	00000	rt	immediate			设置高位
J 型指令							
J	000010	address					跳转

1）系统的初始状态为 S0 状态，每次上电或 reset 信号为高电平时，均进入 S0 状态。在该状态，ALU 完成 PC+1 的运算得到 PC’，PC 从指令寄存器中取出，进入存储器读取指令。

2）读出的指令将进入译码阶段，从寄存器文件中读取源操作数，译出指令类型，为后续执行做准备。

3）对于不同的指令，其执行阶段又有所不同，例如寄存器文件读，符号扩展，寄存器文件写，储存器写等，通过主控制器控制实现。

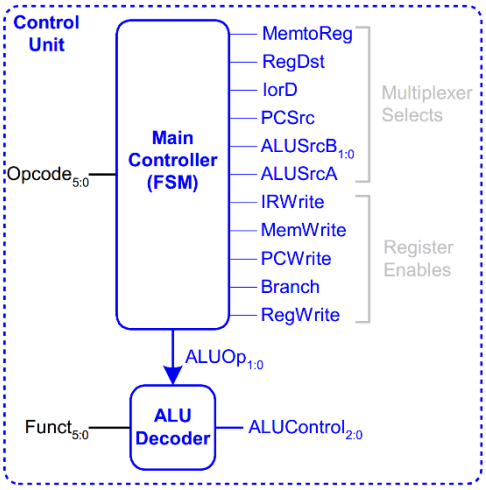


图 3 控制单元

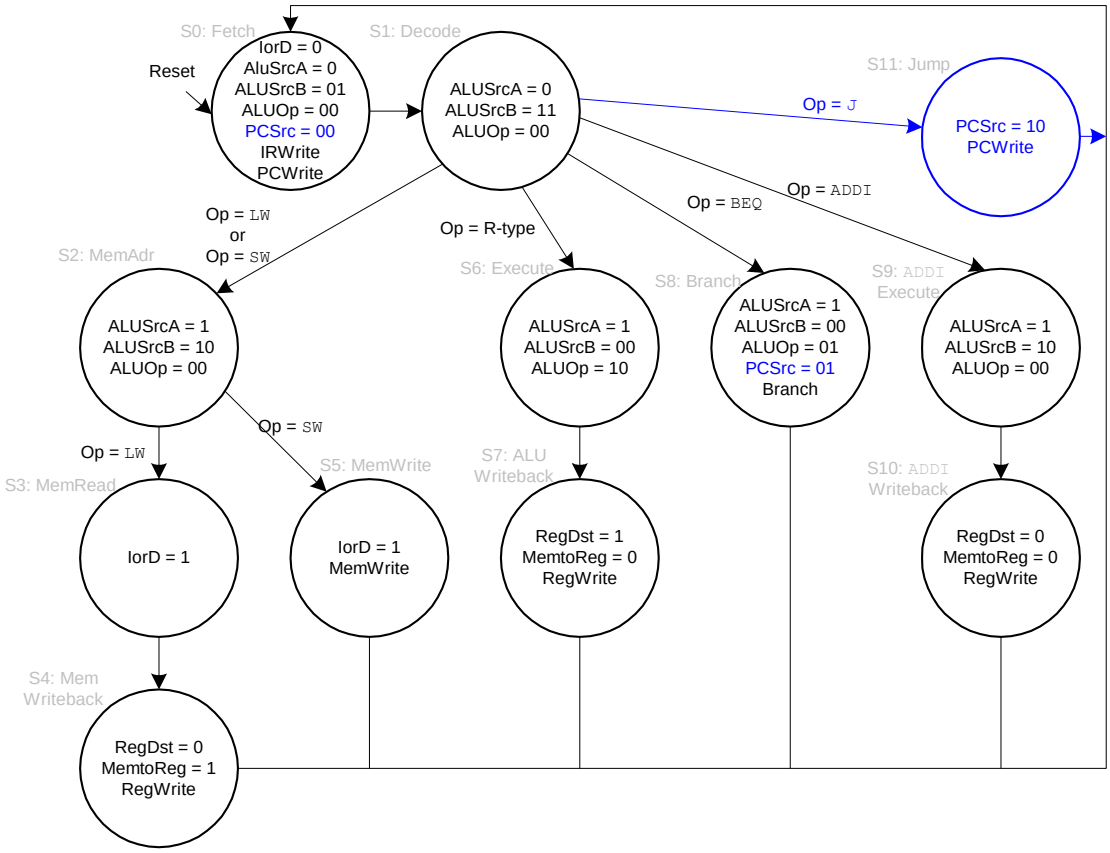


图 2 状态转换图

多周期处理器的控制单元分为一个主控制器和一个 ALU 译码器如图 3 所示，主控制器产生数据路径的复用器选择和使能信号，选择信号包括 MemtoReg、RegDst、IorD、PCSrc、ALUSrcB 和 ALUSrcA；使能信号包括 IRWrite、MemWrite、PCWrite、Branch 和 RegWrite。状态转换图如图 2 所示。根据此状态转换关系，通过合理设置数据流，即可让 CPU 运转起来。

2.2 项目系统设计方案及模块组成

本节介绍系统设计的概况，各模块的详细实现方案将并仿真测试一起在第四章中予以详细介绍。

系统的模块组成有：

（1） 多路选择器

a) 32 位 2 选 1 多路选择器
共有三处用到了 32 位 2 选 1 的多路选择器，分别用于选择 PC 指向的地址是指令/数据，以完成取值/读出存储器数据；选择写入寄存器的数据来源于 ALUOut/存储器读，对应与 I 指令/R 指令有关的数据写入；选择 ALUSrcA 来源于寄存器读/PC，以完成 R/I 指令的运算或完成 PC+1。

b) 32 位 4 选 1 多路选择器
用于选择不同的数据源完成 R 指令运算，I 指令运算或 PC+1.

c) 32 位 3 选一多路选择器
用于选择 PC'的来源。

d) 5 位 2 选 1 多路选择器
用于选择寄存器写的目的地址是来源于 R 指令直接译码还是由 I 指令计算得到。

（2） D 触发器

a) 32 位 D 触发器
有两处用到了 32 位 D 触发器。

b) 32 位带使能端的 D 触发器
有两处用到了有使能端的 32 位 D 触发器，分别用于在使能信号为高电平时写入要跳转到的 PC，和读入指令使其进入译码阶段。

（3） 符号/逻辑扩展器
用于扩展 15 的立即数到 32 位以进入 ALU 进行运算。

（4） 运算器

a) 主运算器
引入运算符 LE (Logic Extender),AE (Arithmetic extender)和 CE (Carry Extender)，用真值表化简得到逻辑表达式，以实现 Pass，AND，OR，NOT，加法，减法，+1，-1 等功能。

b) 移位器（可实现算术/逻辑右移，算术/逻辑左移）
利用逻辑右移和逻辑/算术左移的对称性，可以很方便的完成移位；对于引入符号位而较复杂的算术右移，利用真值表实现符号位的选择。

c) 前缀加法器
加快运算速度的扩展功能。

d) 比较器
可以完成 32 位数小于，等于，大于的判断。

（5） 控制模块

a) 主控制单元
利用有限状态机的原理，采用三段式编写硬件描述语言程序，完成如图 2 所示的状态转移。

b) ALU 译码器

根据 opcpde（和 funct，如果有），译码得到相应的 ALU Control 字段，选择相应的 ALU 功能。

（6） 寄存器文件

（7） 指令/数据存储器

通过调用 IP 核，导入存有 32 位机器码和 FIR 滤波器系数，输入序列的.coe 文件，创建存储器文件。

（8） 数据流模块

用于控制数据流在各个模块中的流向。

第 3 章 项目设计资源简介（软件平台与硬件平台）

3.1 VIVADO 流程简单介绍

3.1.1 创建工程

一、双击 Vivado  Vivado 2018.1 应用程序图标打开程序进入初始界面

二、单击 Create Project 新建工程->输入项目名称，选择项目位置，注意选择的保存路径中不可有中文->选择 RTL Project（其为默认类型）->设置如图 4 所示搜索条件，选择 xc7a35tcpg236-1， 即第一个型号->浏览弹出的工程 Summary，确认无误后单击 Finish 完成创建。

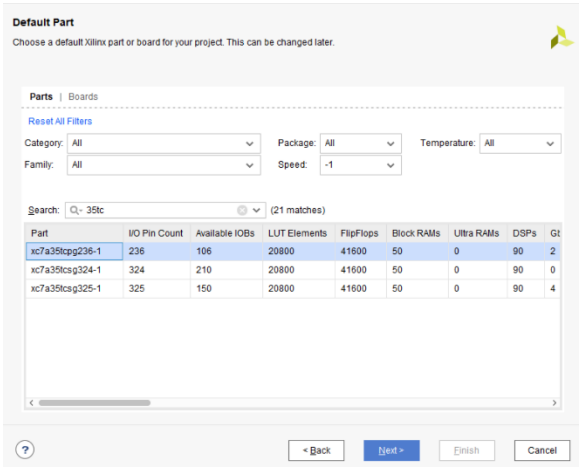


图 4 开发板型号选择

3.1.2 添加源文件

添加设计源文件：

单击界面左侧 Project Manager 中的 Add Source->Add or Create Design Source -> 如有写好的程序的单击 Add Files，若要自己编写新程序则单击 Create File->输入程序名，选择程序保存路径，可按照其所默认的路径设置->Finish 完成创建->可

在随后弹出的 Define Module 弹窗中设置 Module 的输入输出端口，也可直接在程序中手动设置。
创建完成的 design source 文件位于 Source 中的 design source 目录，双击即可打开进行编写。
仿真源文件和约束文件的添加同理。其中添加约束文件前一般要事先编写好约束。

3.1.3 仿真与综合

Vivado 下可以进行四种类型的仿真，如图 5 所示，

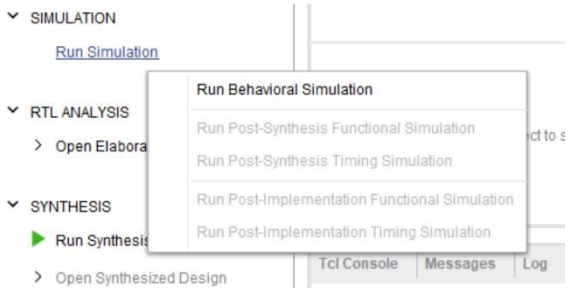


图 5 Vivado 仿真功能

其中 Behavioral Simulation 对功能进行仿真，常用于验证代码功能是否正确，而不考虑硬件延时。设计流程中的第二个仿真是综合后门级功能仿真，即图 5 中的 Post-Synthesis Functional Simulation。绝大多数的综合工具除了可以输出一个标准网表文件以外，还可以输出 Verilog 或者 VHDL 网表。最后进行时序的仿真，以验证电路是否存在时序问题。

Post-Synthesis Simulation 和 Post-Implementation Simulation 需要分别在 Synthesis Implementation 之后才能进行

3.2 BASYS3 组成结构及 I/O 配置简单介绍

Basys3 是围绕着一个 Xilinx+Artix-7+FPGA 芯片 XC7A35T-1CPG236C 搭建的，它提供了完整、随时可以使用的硬件平台，并且它适合于从基本逻辑器件到复杂控制器件的各种主机电路。Basys3 上集成了大量的 I/O 设备和 FPGA 所需的电路，由此可以构建无数设备而不需其他器件。
其主要外围设备如图 6 所示，对应说明如表 2 所示

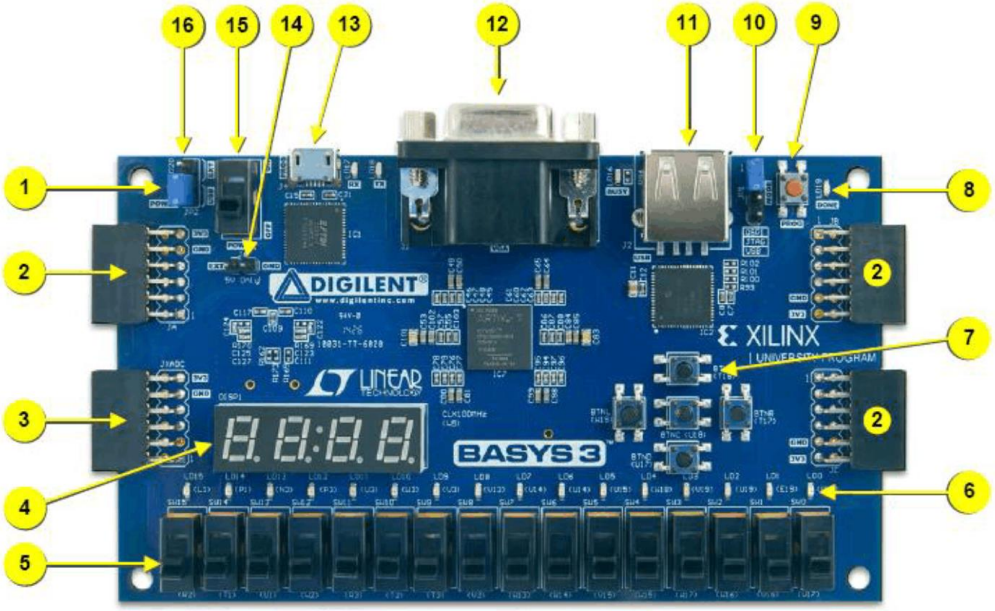


图 6 Basys3 的主要外围设备

表 2 Basys3 主要外围设备说明

序号	描述	序号	描述
1	电源指示灯	9	FPGA配置复位按键
2	Pmod连接口	10	编程模式跳线柱
3	专用模拟信号Pmod连接口	11	USB连接口
4	4位7段数码管	12	VGA连接口
5	16个拨键开关	13	UART/JTAG共用USB接口
6	16个LED	14	外部电源接口
7	5个按键开关	15	电源开关
8	FPGA编程指示灯	16	电源选择跳线柱

3.3 设计语言 Verilog 程序结构及常用描述方法简单说明

Verilog HDL 是一种用于数字逻辑电路设计的语言。用 Verilog HDL 描述的电路设计就是该电路的 Verilog HDL 模型。Verilog HDL 既是一种行为描述的语言也是一种结构描述的语言。这也就是说，既可以用电路的功能描述也可以用元器件和它们之间的连接来建立所设计电路的 Verilog HDL 模型。Verilog 模型可以是实际电路的不同级别的抽象。这些抽象的级别和它们对应的模型类型共有以下五种：

一、 系统级(system):用高级语言结构实现设计模块的外部性能的模型。
二、 算法级(algorithm):用高级语言结构实现设计算法的模型。
三、 RTL 级(Register Transfer Level):描述数据在寄存器之间流动和如何处理这些数据的模型。
四、 门级(gate-level):描述逻辑门以及逻辑门之间的连接的模型。
五、 开关级(switch-level):描述器件中三极管和储存节点以及它们之间连接的模型。

一个复杂电路系统的完整 Verilog HDL 模型是由若干个 Verilog HDL 模块构成的，每一个模块又可以由若干个子模块构成。其中有些模块需要综合成具体电路，而有些模块只是与用户所设计的模块交互的现存电路或激励信号源。利用 Verilog HDL 语言结构所提供的这种功能就可以构造一个模块间的清晰层次结构来描述极其复杂的大型设计，并对所作设计的逻辑电路进行严格的验证。

一个 Verilog 程序的基本构架如下：

```
module module_name(
input,
output);
    //变量声明
    reg
    wire
    parameter
    //程序代码段
    …//常用命令有 assign、 generate、always…
    //代码段结束
endmodule
```

第 4 章 项目单元主模块设计

4.1 ALU 模块设计

4.1.1 模块描述

- 模块功能说明

ALU 模块全称算术逻辑单元模块，实现 32 位加法，减法，乘法和 SLT 及基本与或非功能。功能概念图见图 4.1，功能编码见表 4.1。

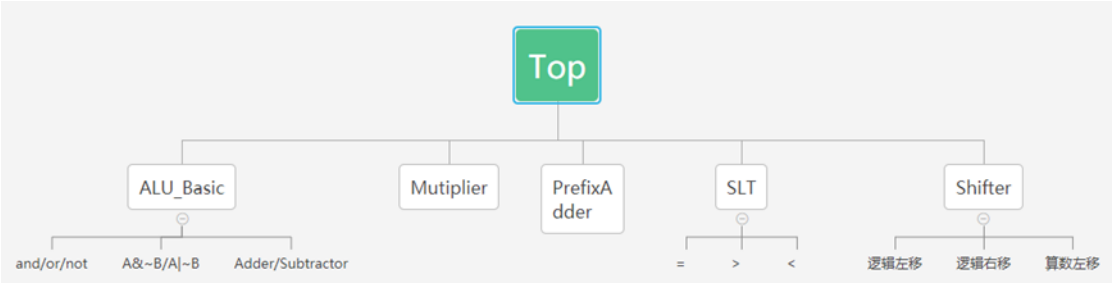


图 4.1：ALU 功能图

功能	$F_{3:0}$	$Fshifter_{2:0}$

A AND B	0000	xxx
A OR B	0001	xxx
NOT A	0011	xxx
A add B	0010	xxx
A subtract B	0110	xxx
SLT	1000	xxx
S multiplication B	0111	xxx
Shifter_算数左移	1010	001
Shifter_算数右移	1010	100
Shifter_逻辑右移	1010	010
A AND B’	0100	xxx
A OR B’	0101	xxx

表 4.1: ALU 功能编码

- 模块符号及输入输出端口说明

```
module Top(
    input [31:0] A,          //输入 A
    input [31:0] B,          //输入 B
    input [3:0] F,           //功能编码 F
    input [2:0] Shifter_F,   //移位数
    output [31:0] Y          //输出 Y
    output Zero);            //相等输出 Zero
```

- 模块原理及数据流程说明

本模块实现功能由门级电路搭建，模块原理图见图 4.2。其中加法器使用行波进位加法器，乘法器原理图见图 4.3。

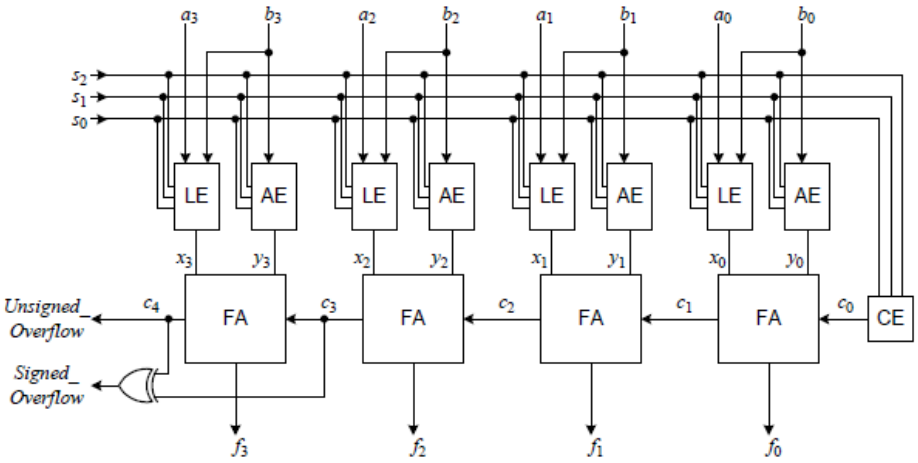


图 4.2: ALU 原理图

图 4.3：乘法器原理图

- 源程序附录引用说明

见附录 1。

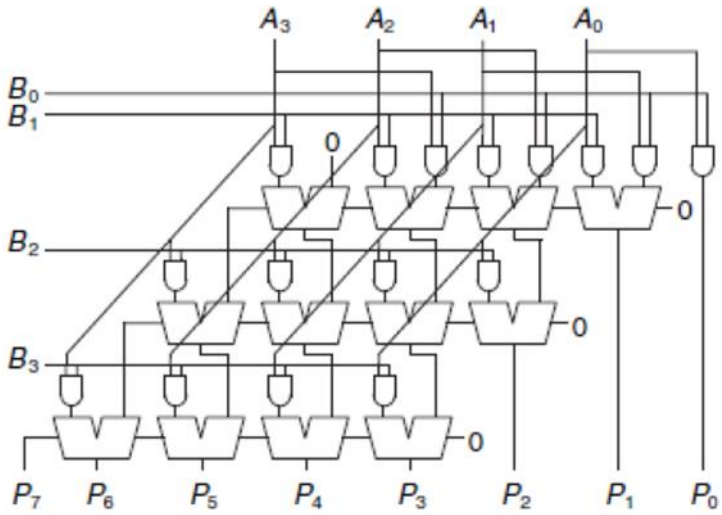
32 位通用移位器

- 模块功能说明

实现逻辑/数字左移，逻辑/数字右移

- 模块符号及输入输出端口说明

```
module left_shift_top(
```




```
    input [31:0] A,//  待移位数
    input [31:0] B,//  移位大小 0-31
    input [2:0] F,//移位方式  001 逻辑/算术左移，010 逻辑右移，100 算术右移
    output [31:0] C);//移位结果
```

- 模块原理及数据流程说明

本模块的所有操作都基于逻辑/算术左移，对于其他移位方式，依托在多路选择器后添加的门电路实现。

  - 逻辑/算术左移

    逻辑左移与算术左移没有本质上的区别，直接利用 32 个 32 选 1 的多路选择器实现。

  - 逻辑右移

    由于逻辑右移后高位填补的是 0，因此考虑先将待移位数翻转，翻转后 MSB 变为 LSB，LSB 变为 MSB，将翻转后的 A’进行逻辑/算术左

移，后再将移位后的得到 C’翻转，即可得到 A 逻辑右移的结果。

- 算术右移

  算术右移与逻辑右移区分最大处在于算术右移后高位填补符号位，因此首先标记符号位的值，在进行逻辑右移的基础上，将逻辑右移后所填补的 0 与符号位进行或运算，即可得到正确地结果。

- 源程序附录引用说明

  见附录 1。

4.1.2 模块仿真

（1）ALU 基本模块和乘法器

- 输入信号波形设置

```
initial begin
A = 32'b0100_0000_1000_1111_0000_1111_0101_0000;
B = 32'b0010_0100_0000_0111_0000_0111_1101_0110;
F = 4'b0000; //AND
#10;
F = 4'b0001; //OR
#10;
F = 4'b0011; //NOT
#10;
F = 4'b0010; //add
#10;
F = 4'b0110; //sub
#10;
F = 4'b1000; //SLT
#10;
F = 4'b0111; //mul
end
```

- 仿真波形

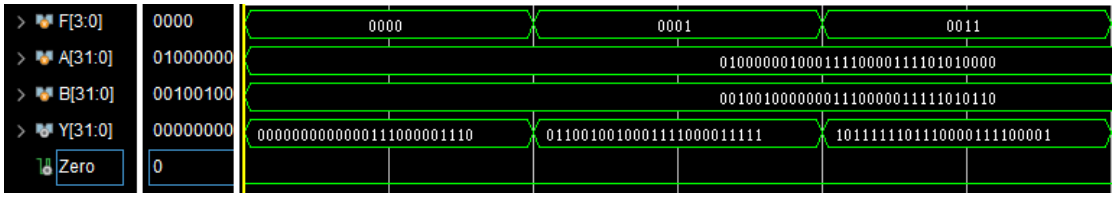


图 4.4：AND，OR，NOT

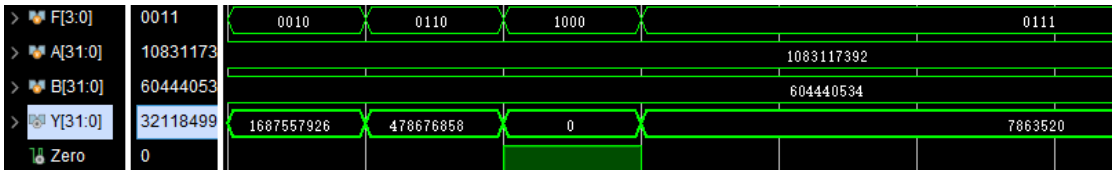


图 4.5: add，sub，SLT，mul

- 波形数据分析与结论

输入信号 A= 32’b0100_0000_1000_1111_0000_1111_0101_0000 =(1083117392)$_{10}$，B=32’b0010_0100_0000_0111_0000_0111_1101_0110=(604440534)$_{10}$；得到 AND 结果为 00000000000001110000011101010000，正确；OR 结果为 01100100100011110000111111010110，正确；NOT 结果为 10111111011100001111000010101111，正确；add 结果为(1687557926)$_{10}$，正确；sub 结果为(478676858)$_{10}$；正确；SLT 结果为(1)$_{10}$，正确。mul 取前 16 位计算，结果为 7863520，正确。

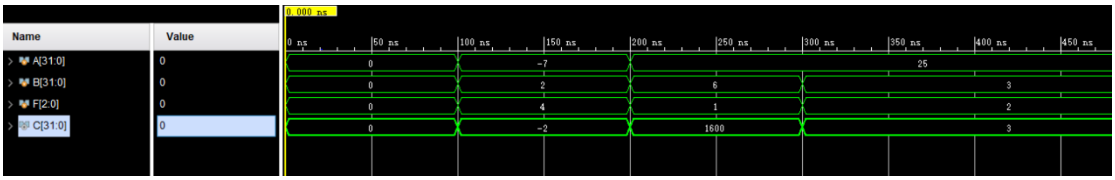
（2）通用移位器

- 输入信号波形设置，

```
module left_shift_sim1(

    );
    reg [31:0] A=32'b0;
    reg [31:0] B=32'b0;
    reg [2:0] F=3'b0;
    wire [31:0] C;

    left_shift_top uut
    (
        .A(A),
        .B(B),
        .C(C),
        .F(F)
     );

    initial
     begin
     #100;
     A=32'b11111111111111111111111111111001;
     B=32'b10;//移 2 位
     F=3'b100; //算术右移
     #100;
     A=32'b11001;
```

```
        B=32'b110;//移 6 位
        F=3'b001;//逻辑左移
        #100;
        A=32'b11001;
        B=32'b011;//移 3 位
        F=3'b010;//逻辑右移
        end
endmodule
```

- 仿真波形



- 波形数据分析与结论
  - 算术右移的原数据为 11111111111111111111111111111001，即-7，右移 2 位后的结果为-2。结果正确。
  - 逻辑左移的原数据为 00000000000000000000000000011001，即 25，左移 6 位后的结果为$1600 = 25 \times 2^6$。结果正确。
  - 逻辑右移的原数据为 00000000000000000000000000011001，即 25，右移 3 位后的结果为 3。结果正确。

4.2 控制模块设计

4.2.1 模块描述

- 模块功能说明

控制模块控制 CPU 的整体功能，每个周期控制模块将送出不同的控制指令，控制 CPU 其他模块完成设计的功能。

- 模块符号及输入输出端口说明

```
module main_decoder_FSM(
        input [5:0] Opcode,    //指令高 6 位
        input clr,              //异步复位
        input clk,              //时钟
        output reg MemtoReg,
                reg RegDst,
```

```
                reg IorD,
                reg [1:0] PCSrc,
                reg [1:0] ALUSrcB,
                reg ALUSrcA,
                reg IRWrite,
                reg MemWrite,
                reg PCWrite,
                reg Branch,
                reg RegWrite,
                reg [2:0] ALUOp
        );                                  //主控制模块

    module ALUDecoder(
        input [2:0] ALUOp,
        input [5:0] funct,
        output reg [3:0] ALUcontrol
        );                                  //ALU 控制模块
```

- 模块原理及数据流程说明

本模块分为两部分，主模块使用状态机实现控制信号周期性改变，状态转换图见图 4.6，ALU 控制模块使用组合逻辑实现，输出 ALU 控制信号。

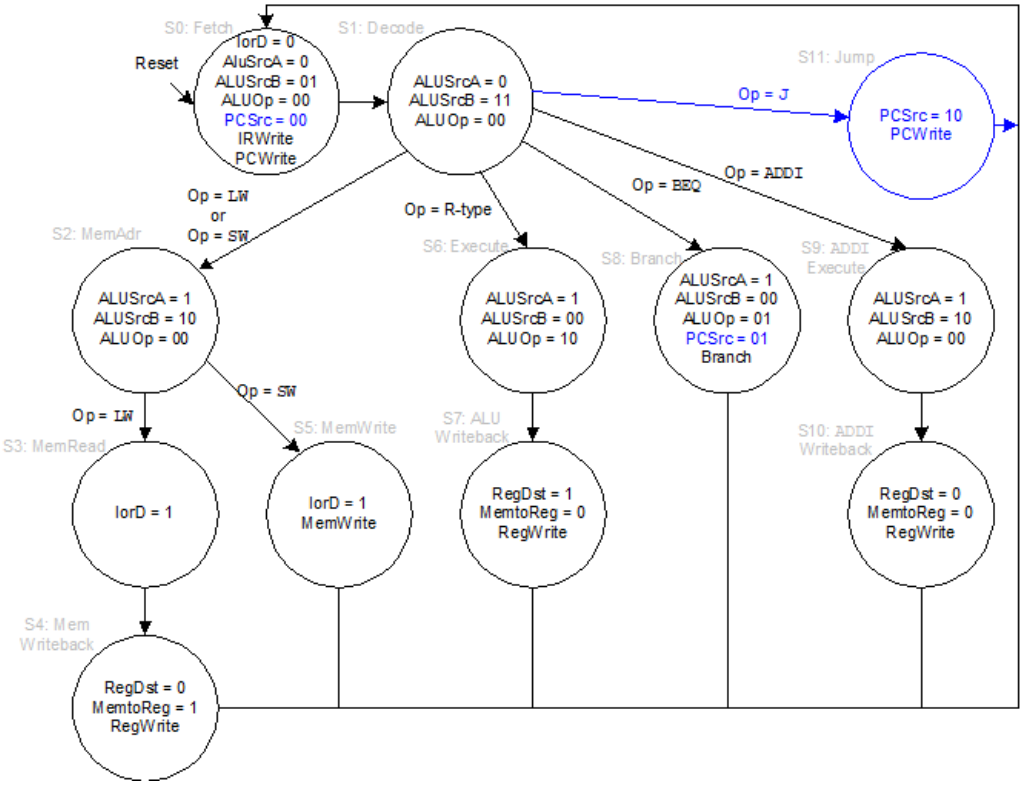


图 4.6 控制模块状态转换图

- 源程序附录引用说明
  见附录 2。

4.2.2 控制模块仿真

- 输入信号波形设置

```
 initial begin
clr = 1'b1;
opcode = 6'b000000;
funct = 6'b100000;      //add
#10;
opcode = 6'b100011;    //lw
#10;
opcode = 6'b101011;      //sw
#10;
opcode = 6'b000100;      //beq
#10;
opcode = 6'b001000;      //addi
#10;
opcode = 6'b000010;      // j
#10;
opcode = 6'b001100;       //andi
#10;
opcode = 6'b001101;       //ori
end
```

- 仿真波形

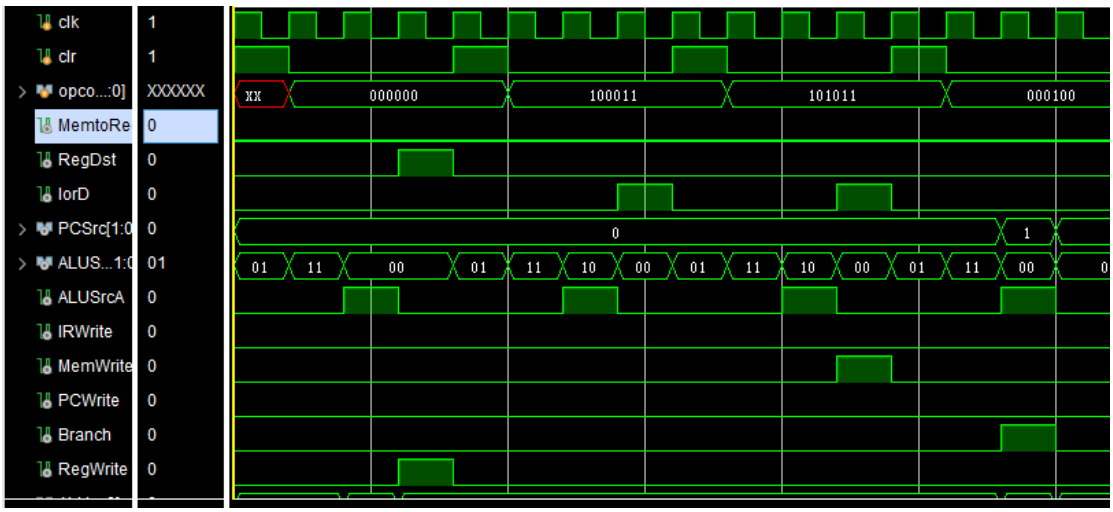


图 4.7：add，lw，sw，beq

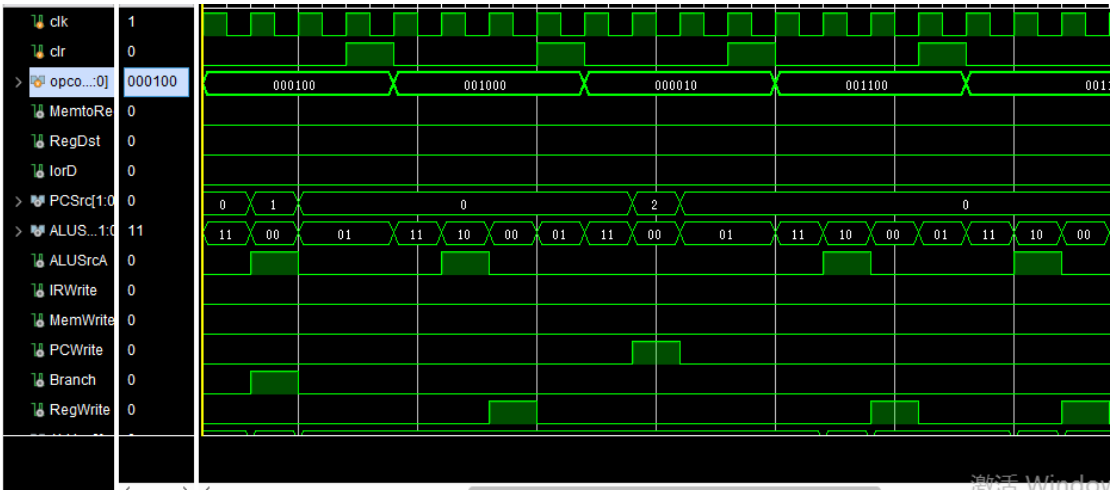


图 4.8: addi，j，andi，ori

- 波形数据分析与结论

如图 4.7 和图 4.8 所示，add, sw, addi, andi, ori 都经历四个周期，信号输出经核对也正确，beq 和 j 指令都经历三个周期，lw 经历五个周期，输出都与设计符合。模块实现正确。

4.3 复用器和触发器模块设计

4.3.1 模块描述

- 模块功能说明

根据 CPU 的整体设计要求，需要实现 4 种多路选择器和两种触发器。

1) 多路选择器

a) 32 位 2 选 1 多路选择器

共有三处用到了 32 位 2 选 1 的多路选择器，分别用于选择 PC 指向的地址是指令/数据，以完成取值/读出存储器数据；选择写入寄存器的数据来源于 ALUOut/存储器读，对应与 I 指令/R 指令有关的数据写入；选择 ALUSrcA 来源于寄存器读/PC，以完成 R/I 指令的运算或完成 PC+1。

b）32 位 4 选 1 多路选择器

用于选择不同的数据源完成 R 指令运算，I 指令运算或 PC+1.

c）32 位 3 选一多路选择器

用于选择 PC’的来源。

d）5 位 2 选 1 多路选择器

用于选择寄存器写的目的地址是来源于 R 指令直接译码还是由 I 指令计算得到。

（2）D 触发器

a）32 位 D 触发器

有两处用到了 32 位 D 触发器。

b）32 位带使能端的 D 触发器

有两处用到了有使能端的 32 位 D 触发器，分别用于在使能信号为高电平时写入要跳转到的 PC，和读入指令使其进入译码阶段。

- 模块符号及输入输出端口说明

由于模块有较多重复，这里仅举一个选择器和一个带使能的 D 触发器为例。

```
module MemtoReg_mux(
    input s,
    input [31:0] i1,
    input [31:0] i2,
    output [31:0] o
    );

module d_flipflow_enable(
    input [31:0] A,
    output reg [31:0] B,
    input en,
    input CLK
    );
```

- 模块原理及数据流程说明

本模块分为两部分，选择器使用组合逻辑实现，关键语句 assign o = s ? i2 : i1; 触发器使用 always 沿触发实现。

- 源程序附录引用说明

见附录 3。

4.3.2 选择器模块仿真

因模块大量重复，仅测试 IorD 选择器

- 输入信号波形设置

```
initial begin
s = 1'b0;
#10;
i1 = 12'b00111;
i2 = 12'b00000;
end     //IorD2-1 选择器
```

- 仿真波形

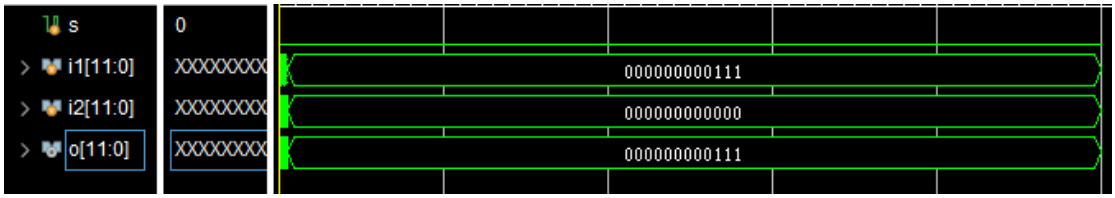


图 4.9：IorD_Mux

- 波形数据分析与结论

如图 4.9 所示，选择信号为 0，应选择 i1 作为输出，结果正确。

4.3 储存模块设计

4.2.1 模块描述

- 模块功能说明

本模块需要实现一个单口 RAM 作为设计中的指令/数据储存器和一个 32*32 的可读写寄存器文件。如图 4.10 所示，其中 RAM 调用 IP 核实现，寄存器文件使用 32 个 32 位寄存器实现。

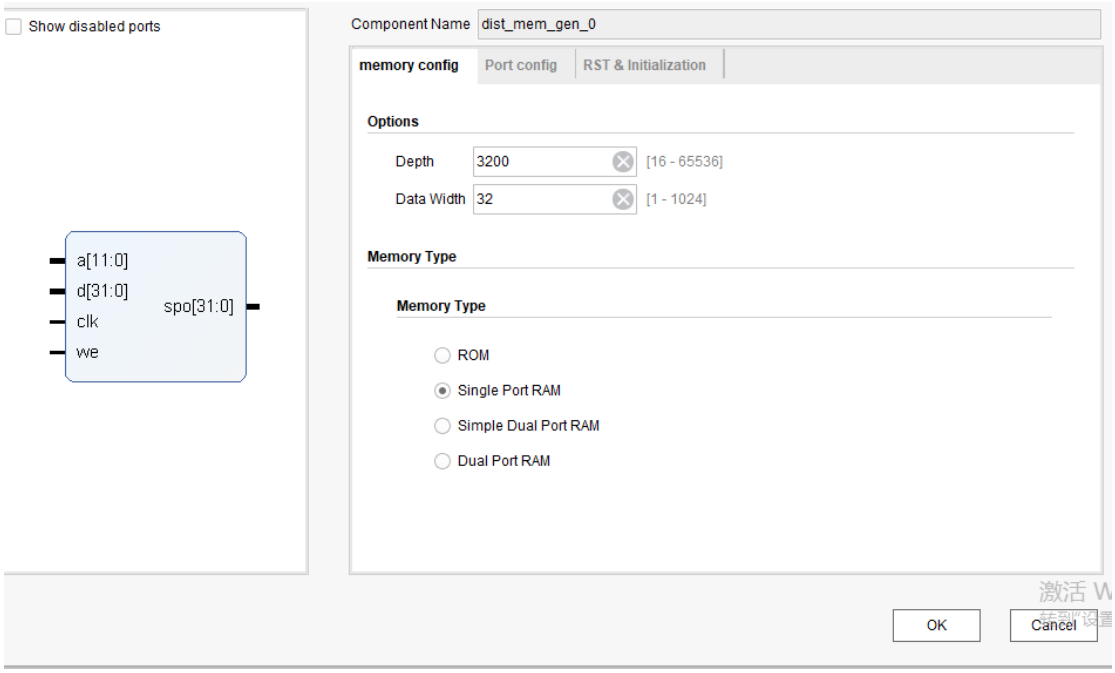


图 4.10：指令/数据储存器

- 模块符号及输入输出端口说明

```
dist_mem_gen_0 (
    input wire [11 : 0] a
    input wire [31 : 0] d
```

```
    input wire clk
    input wire we
    output wire [31 : 0] spo
);

module Register_File(
    input clk,
    input we3,
    input [4:0] ra1, ra2, wa3,
    input [31:0] wd3,
    output [31:0] rd1, rd2
);
```

- 模块原理及数据流程说明

本模块分为两部分，RAM 使用 IP 核实现， 寄存器文件使用 32 个 32 位寄存器实现，即 reg [31:0] rf [31:0]。

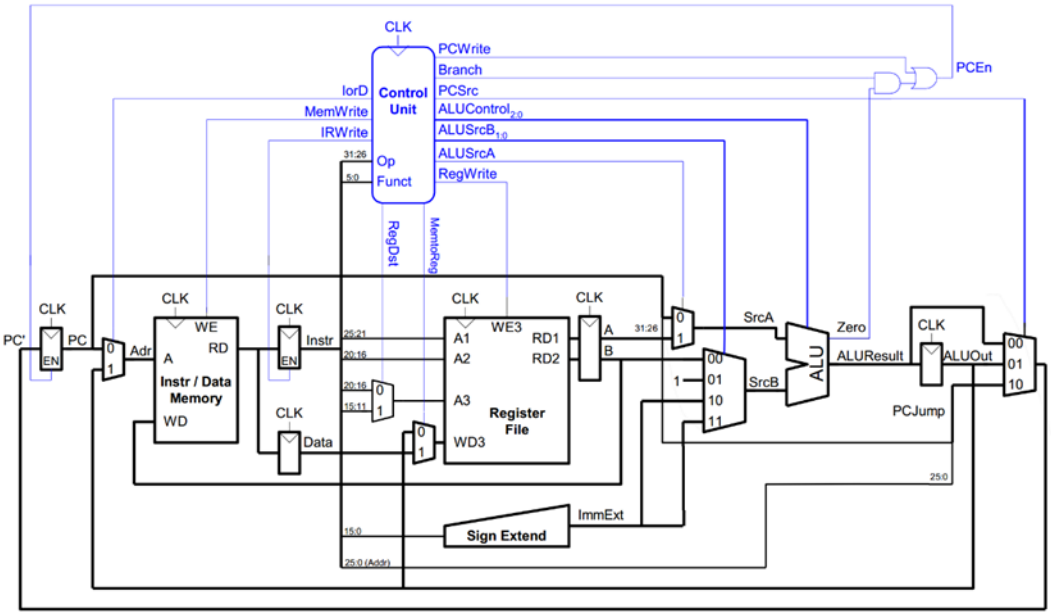
- 源程序附录引用说明

寄存器文件代码见附录 4。

第 5 章 项目系统电路设计

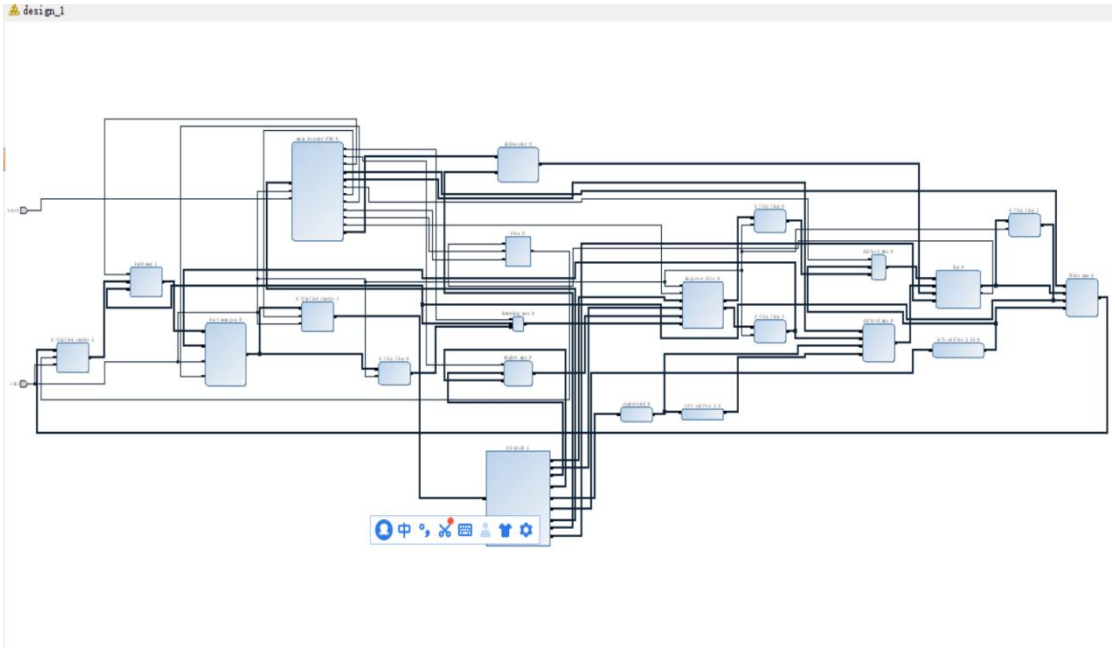
5.1 系统设计描述

- 系统组成结构与电路原理



- 系统封装模型及 I/O 端口说明

利用 block design 实现的整个系统设计模型。如下图所示整个 CPU 仅有一个 I/O 端口——clr，当 clr 输入为 1 时，整个 CPU 被重置。



- 系统组成模块说明（重点对第四章未介绍的模块进行描述）

  见第四章
- 系统源程序附录引用说明

  系统源程序见附录。
- 5.2　系统仿真（参见第 4.1.2）

第 6 章 项目系统设计实现与测试

6.1 测试文件设计

6.1.1　测试文件功能与参数指标

测试程序实现的功能为：一个 32 阶低通 FIR 滤波器，滤波器用 Matlab 数字滤波器设计工具设计得到，其各参数如图 7 所示。处理数据后使滤波器的输入输出都为二进制形式，位宽为整数位 3 位，符号位 1 位，小数位 12 位，共有 1000 个输入。

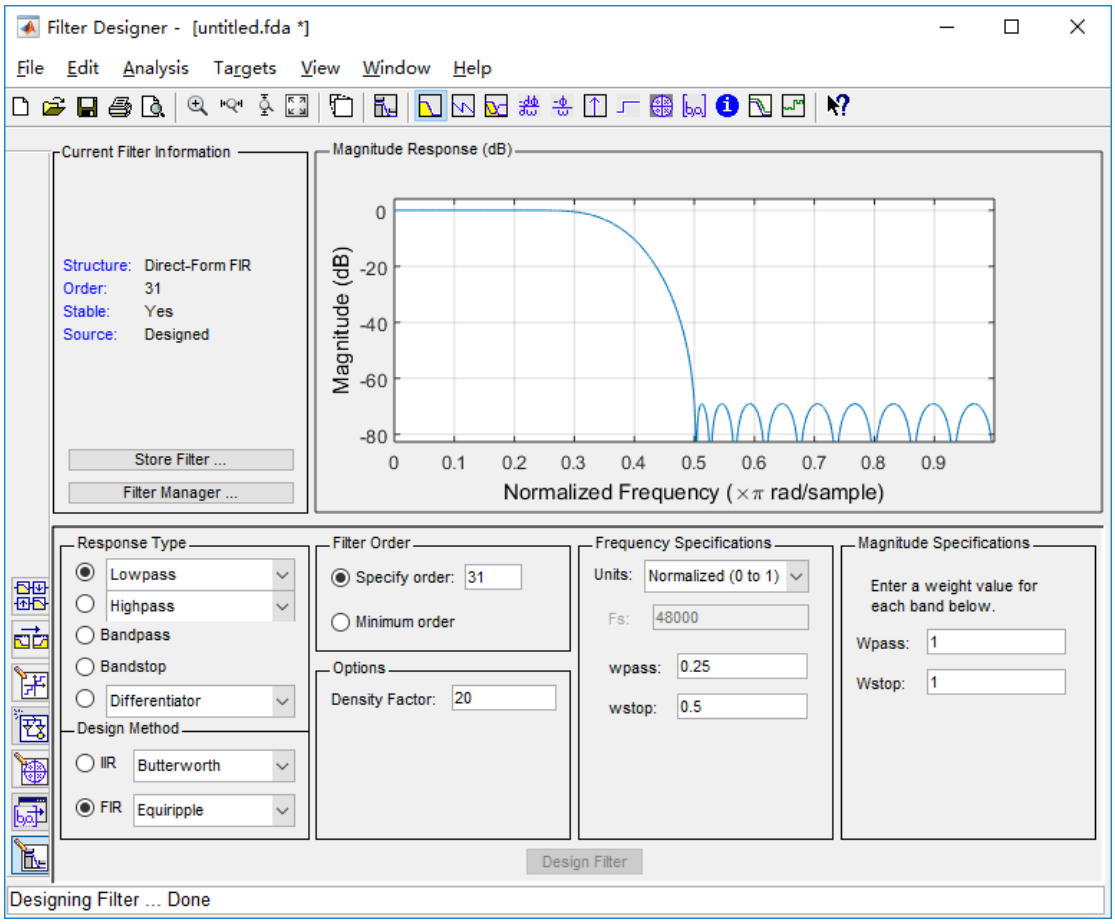


图 7 Malab FDATool 滤波器设计参数

用信噪比验证测试程序的性能，计算方式为

$$20 \times \log \frac{\sum 正确结果的幅度}{\sum |测试结果 - 正确结果|}$$

6.1.2 测试文件原理

FIR(Finite Impulse Response)滤波器是有限长单位冲激响应滤波器的简称，又称为非递归型滤波器，是数字信号处理系统中最基本的元件，它可以在保证任意幅频特性的同时具有严格的线性相频特性，同时其单位抽样响应是有限长的，因而滤波器是稳定的系统。

在真实的信号处理环境中，在进入 FIR 滤波器前，首先要将信号通过 A/D 器件进行模数转换，把模拟信号转化为数字信号；为了使信号处理能够不发生失真，信号的采样速度必须满足香农采样定理，一般取信号频率上限的 4-5 倍做为采样频率；一般可用速度较高的逐次逼进式 A/D 转换器，不论采用乘累加方法还是分布式算法设计 FIR 滤波器，滤波器输出的数据都是一串序列，要使它能直观地反应出来，还需经过数模转换，因此由 FPGA 构成的 FIR 滤波器的输出须外接 D/A 模块。本测试的目是测试 CPU 的性能，因此至今对数字信号进行处理，输出数字信号。

FIR 滤波器的原理的数学表示为，

$$y[n] = \sum_{r=0}^{31} h[r] \times x[n-r]$$

不难看出，它即为线性时不变系统的卷积和公式。

6.1.3 测试文件设计思路

- 有 1000 个x输入必有 1000 个y输出，即写入存储器。则在分配存储器空间时，可以将 1000 个x与 1000 个y紧邻放置，则地址指针只需要储存当前计算的y的地址，如此第一个x（认为与$h[0]$相乘的$x[n]$为第一个x，$x[n-31]$为最后一个）的地址即为$y+1000$，后续x的地址顺次-1。如此可减少寻址的指令数。
- 对于$n>0$ 时的每一个y，计算时都要用到累加，因此使用循环不失为一个省力的办法。
- 设置两层循环，外层循环 1000 次，代表计算 1000 个y，内层循环 32 次（如果可以算到 32 次），代表对相应的$x[n-r]$与$h[r]$乘积求和。显然内层循环的循环条件比外层复杂，需要满足 1）$r \le 31$，2）$n-r \ge 0$。由于 MIPS 指令中只有小于置数而没有小于等于置数，因此将第一个条件设为$r<32$，尽管这样会要求在程序初始时将 32 写寄存器 32，但这种开销是无法避免的。对于第二个条件，个很好的不引入立即数 1 的办法是：第一层循环用 1-1000 计数，第二层循环用 0-31 计数，满足$n-r>0$即可，而 0 又可以直接使用用寄存器变量\$0，可以节省一条指令。

6.1.4 测试程序附录引用说明

以下对附录中的程序进行逐行说明。

```
addi$s0, $0, 0              #store initial address of h
addi$s2, $0, 1032           #store initial address of y
addi$t8, $0, 1001               #n=1000 (for comperasion only)
addi$t0, $0, 1              #i=1 initialization for i
addi$s6, $0, 32         #parameter 32
loopt:    slt $t1, $t0, $t8         # i<1001?    2037
    beq $t1, $0, done # .... 20       # nope: done
    addi $t2, $0, 0         # r=0 start from 0, for benefit of 'slt' in loopb
    addi $s3, $s0, 0        #initial address(h[0]) of h
    addi $s5, $s2, -1000        #initial address(x[n-0]) of x
    addi $s4, $0, 0         #initialization of y[i] y[i]=0
    loopb:   slt $t3, $t2, $t0              # r<i?(essentially r<=ith)
        slt $s7, $t2, $s6           # r<32? #
        beq $t3, $0, prepnxtloopt       #...        #nope: go to prepare for next loopt
        beq $s7, $0, prepnxtloopt     #...
        lw $t4, 0($s3)              #load h[r]
        lw $t5, 0($s5)              #load x[n-r]
```

```
        mul $t6, $t5, $t4           #x[n-r]*h[r]
        add $s4, $t6, $s4           #y[n]+=x[n-r]*h[r]
        addi $s3, $s3, 1            #update address for h
        addi $s5, $s5, -1           #updata address for x
        addi $t2, $t2, 1            #r+=1
        j loopb                         #j
prepnxtloopt:    sw $s4, 0($s2)             #memwrite y
            addi $s2, $s2, 1            #update address for y
            addi $t0, $t0, 1            #i+=1
            j loopt # j 2037
 done:  j done
```

6.2 系统硬件测试

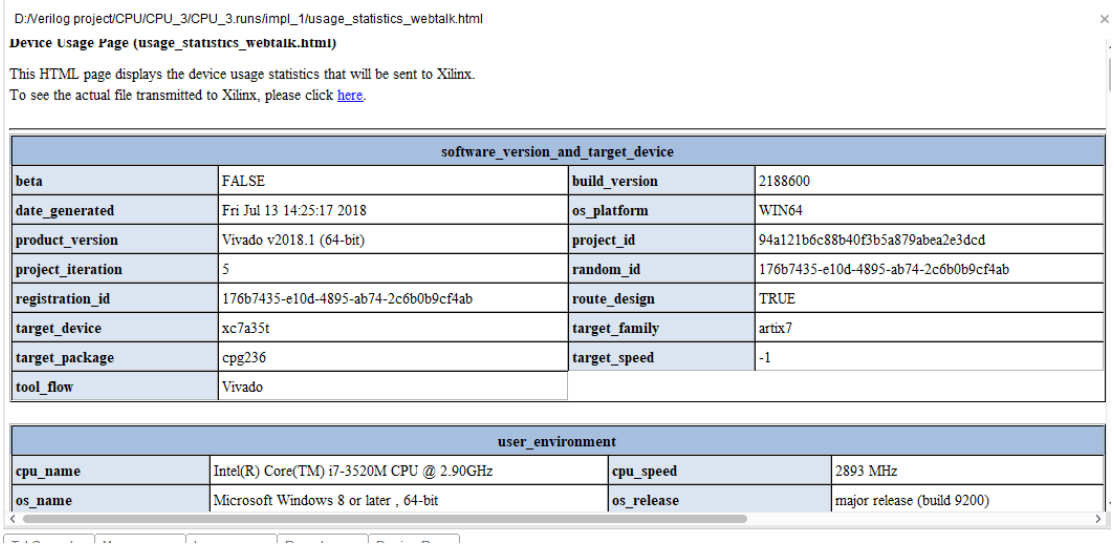
6.2.1 管脚适配

整个 CPU 仅有一个 I/O 端口——clr，当 clr 输入为 1 时，整个 CPU 被重置。

```
set_property PACKAGE_PIN W19 [get_ports clr]


create_clock -period 100.000 -name clk -waveform {0.000 50.000} [get_ports clk]
```

6.2.2 系统编程及下载（截图）

比特流文件生成报告



D:/Verilog project/CPU/CPU_3/CPU_3.runs/impl_1/usage_statistics_webtalk.html

Device Usage Page (usage_statistics_webtalk.html)

This HTML page displays the device usage statistics that will be sent to Xilinx.
To see the actual file transmitted to Xilinx, please click here.

software_version_and_target_device			
beta	FALSE	build_version	2188600
date_generated	Fri Jul 13 14:25:17 2018	os_platform	WIN64
product_version	Vivado v2018.1 (64-bit)	project_id	94a121b6c88b40f3b5a879abea2e3dcd
project_iteration	5	random_id	176b7435-e10d-4895-ab74-2c6b0b9cf4ab
registration_id	176b7435-e10d-4895-ab74-2c6b0b9cf4ab	route_design	TRUE
target_device	xc7a35t	target_family	artix7
target_package	cpg236	target_speed	-1
tool_flow	Vivado		

user_environment			
cpu_name	Intel(R) Core(TM) i7-3520M CPU @ 2.90GHz	cpu_speed	2893 MHz
os_name	Microsoft Windows 8 or later , 64-bit	os_release	major release (build 9200)

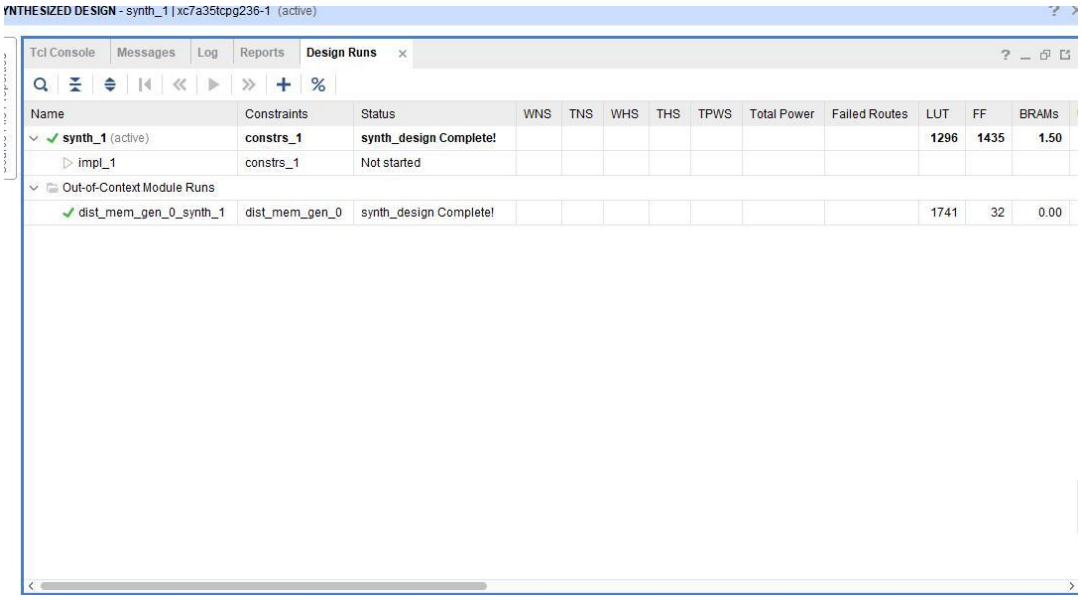
6.2.3 系统数据测试、分析与结论

测试 1000 条数据，经计算，本测试程序的信噪比为 48.126dB，大于 35Db 达到要求。总指

令数为 38500。

6.2.4 系统运行资源及运行速度

系统综合报告（资源数包含了用于 debug 的 ILA 模块）



本多周期 CPU 时钟频率为 20MHz，单个周期时间为 50ns。

第 7 章　结束语（设计总结）

7.1 项目设计过程的重点与难点

本次设计中，重点设计主要在 ALU 模块的设计，整体模块组合和时序控制优化部分。具体包括如何进行加法器，乘法器和移位器的门级设计， 如何将各模块准确进行数据和控制通路连接，如何调整修改设计保证时序仿真成功通过。

在设计过程中，我们遇到的设计难点包括如何将各模块成功组合在一起，以及在组合后出现的代码逻辑问题导致功能出错，还有综合报错以及实现报错等问题。通过调试和讨论，最后依次克服了这些难题。

7.2 项目设计过程中遇到的主要问题及处理方法

设计过程中，遇到的主要问题有模块连接出错的问题。我们开始时使用 Vivado 的 Block Design 功能，将各模块打包成 IP 核，通过 Block Design 直接连接。但是功能仿真过程中出现了很多问题，而 Block Design 不方便进行各模块细致的 debug。最后我们改用一个 TopCPU.v 文件直接连接各模块，调试至功能仿真成功。当然改变方法浪费了一定的时间。

还有一个问题在于综合优化后，因为没有总输出，导致模块全部被优化掉，这个问题在我们查阅资料后，给模块加上{keep}锁定命令后解决。当然，在后期

使用 ILA 加入输出后，不再需要{keep}。

7.3 收获与改进意见

经过这次大型的 MCU 课程设计，我们成功实现了一个由 FIR 程序测试的多周期通用 CPU 硬件电路设计。在一学期的设计过程中，我们首先学会了如何高效的进行团队合作，其次我们熟悉了 Vivado 软件使用和硬件设计的基本流程。

通过使用 Vivado 平台，我们了解了硬件设计的基本流程，学会了 Verilog 的基本语法，仿真文件的写法，综合和实现的方法以及如何将线上设计下载到开发板中。

通过进行单元模块设计，我们重温了数字电路课程学习的设计知识，用实践强化了对数字模块设计的理解，学习了如何在硬件设计中进行模块化设计，如何设计 ALU，移位器，寄存器集和如何调用 IP 核。

通过控制模块设计和各模块连接的过程，我们加强了对计算机体系结构的理解和对 CPU 设计原理的掌握，细致研究了 CPU 的数据通路和控制通路，和每条指令每周期的具体实现过程。

通过 FIR 程序汇编语言的编写，学习了汇编语言的语法，如何用汇编语言转机器码等过程。

当然，在设计中还有需要许多可以改进的地方。如在 ALU 模块，还可以对加法器和乘法器进行优化，在 CPU 的速率方面，还可以尝试单周期流水线的设计，或者考虑多核多线程方面的优化。在硬件面积方面，还可以通过优化乘法器，进一步降低所需面积。因为时间原因，这些工作在本次设计中没有尝试，在未来的学习中，可以尝试进行更多优化方面的探索。

附录 1:

```
module Top(
    input [31:0] A,
    input [31:0] B,
    input [3:0] F,
    input [2:0] Shifter_F,
    output [31:0] Y
    );
    wire [31:0] Y0, Y1, Y2, Y3, Y4;
    wire [4:0] H;     //one-hot
    wire [3:0] NH;
    wire [3:0] NF;
    wire [31:0] t [4:0];

    // get NF
    genvar i;
```

```
generate
   for (i=0;i<4;i=i+1)
      begin
         not(NF[i], F[i]);
      end

// get H
and(H[0], NF[3], F[2], F[1], F[0]);
and(H[1], F[3], NF[2], NF[1], NF[0]);
and(H[2], F[3], NF[2], NF[1], F[0]);
and(H[3], F[3], NF[2], F[1], NF[0]);
not(NH[0], H[0]);
not(NH[1], H[1]);
not(NH[2], H[2]);
not(NH[3], H[3]);
and(H[4], NH[0], NH[1], NH[2], NH[3]);

/*
generate
      case(F)

         4'b0000: ALU_AND uut(.A(A), .B(B), .Y(Y));
         4'b0001: ALU_OR uut(.A(A), .B(B), .Y(Y));
         4'b0010: ALU_Adder uut(.A(A), .B(B), .Y(Y));
         4'b0011: ALU_Multiplier uut(.A(A), .B(B), .Y(Y));
         4'b0100: ALU_NAND uut(.A(A), .B(B), .Y(Y));
         4'b0101: ALU_NOR uut(.A(A), .B(B), .Y(Y));
         4'b0110: ALU_Subtractor uut(.A(A), .B(B), .Y(Y));
         4'b0111: ALU_SLT uut(.A(A), .B(B), .Y(Y));
         4'b1000: ALU_LSL uut(.A(A), .B(B), .Y(Y));
         4'b1001: ALU_LSR uut(.A(A), .B(B), .Y(Y));
         4'b1010: ALU_ASR uut(.A(A), .B(B), .Y(Y));
         4'b1011: ALU_PrefixAdder uut(.A(A), .B(B), .Y(Y));
         4'b1100: ALU_AND uut(.A(A), .B(B), .Y(Y));
         4'b1101: ALU_AND uut(.A(A), .B(B), .Y(Y));
         4'b1110: ALU_AND uut(.A(A), .B(B), .Y(Y));
         4'b1111: ALU_AND uut(.A(A), .B(B), .Y(Y));
         */
```

```
            ALU_Multiplier uut1(.A(A[15:0]), .B(B[15:0]), .Y(Y0));
            ALU_SLT uut2(.a(A), .b(B), .Y(Y1));
            //ALU_PrefixAdder uut3(.A(A), .B(B), .Y(Y2));

            ALU_Shifter uut5(.F(Shifter_F), .A(A), .B(B), .C(Y3));
            ALU_Basic uu4(.F(F), .A(A), .B(B), .Y(Y4));

// sure output

module ALU_Basic(
        input [3:0] F,
        input [31:0] A,
        input [31:0] B,
        output [31:0] Y
        );
        wire Cin;
        wire [31:0] W, Z;
        LE uut1 (.F(F), .A(A), .B(B), .X(W));
        AE uut2 (.F(F), .A(A), .B(B), .Y(Z));
        CE uut3 (.F(F), .Cin(Cin));
        FA uut4 (.x(W), .y(Z), .c0(Cin), .s(Y));
endmodule
    assign Y2 = 32'b0;
    for(i=0;i<32;i=i+1)
        begin
            and(t[0][i], Y0[i], H[0]);
            and(t[1][i], Y1[i], H[1]);
            and(t[2][i], Y2[i], H[2]);
            and(t[3][i], Y3[i], H[3]);
            and(t[4][i], Y4[i], H[4]);
            or(Y[i], t[0][i], t[1][i], t[2][i], t[3][i], t[4][i]);
        end
endgenerate
endmodule

module AE(
        input [3:0] F,
        input [31:0] A,
```

```
        input [31:0] B,
        output [31:0] Y
        );
genvar i;
        generate
            for (i=0;i<32;i=i+1)
                begin
                    AE_basic uut_AE_basic(.F(F), .a(A[i]), .b(B[i]), .y(Y[i]));
                end
        endgenerate
endmodule

module AE_basic(
        input [3:0] F,
        input a,
        input b,
        output y
        );
wire [3:0] NF;
wire Na, Nb, a_or_b, a_or_Nb;

// get ~F, ~a, ~b , a+b, a+~b
not(Na, a);
not(Nb, b);
or(a_or_b, a, b);
or(a_or_Nb, a, Nb);

genvar i;
generate
    for (i=0;i<4;i=i+1)
        begin
            not(NF[i], F[i]);
        end
endgenerate

// Intermediate variable t
wire [1:0] t;
and(t[0], NF[3], NF[2], F[1], NF[0], b);
```

```
and(t[1], NF[3], F[2], F[1], NF[0], Nb);

// get y
or(y, t[0], t[1]);
endmodule

module CE(
        input [3:0] F,
        output Cin
        );

wire [3:0] NF;
// get ~F
genvar i;
generate
    for (i=0;i<4;i=i+1)
        begin
            not(NF[i], F[i]);
        end
endgenerate

// get Cin
and(Cin, NF[3], F[2], F[1], NF[0]);
endmodule

module LE(
        input [3:0] F,
        input [31:0] A,
        input [31:0] B,
        output [31:0] X
        );

genvar i;
generate
    for (i=0;i<32;i=i+1)
        begin
            LE_basic uut_LE_basic(.F(F), .a(A[i]), .b(B[i]), .x(X[i]));
        end
```

```
endgenerate
endmodule

module LE_basic(
        input [3:0] F,
        input a,
        input b,
        output x
        );
wire [3:0] NF;
wire Na, Nb, a_or_b, a_or_Nb;

// get ~F, ~a, ~b , a+b, a+~b
not(Na, a);
not(Nb, b);
or(a_or_b, a, b);
or(a_or_Nb, a, Nb);

genvar i;
generate
    for (i=0;i<4;i=i+1)
        begin
            not(NF[i], F[i]);
        end
endgenerate

// Intermediate variable t
wire [6:0] t;
and(t[0], NF[3], NF[2], NF[1], NF[0], a, b);
and(t[1], NF[3], NF[2], NF[1], F[0], a_or_b);
and(t[2], NF[3], NF[2], F[1], NF[0], a);
and(t[3], NF[3], NF[2], F[1], F[0], Na);
and(t[4], NF[3], F[2], NF[1], NF[0], a, Nb);
and(t[5], NF[3], F[2], NF[1], F[0], a_or_Nb);
and(t[6], NF[3], F[2], F[1], NF[0], a);

// get x
or(x, t[0], t[1], t[2], t[3], t[4], t[5], t[6]);
```

```
endmodule

module FA(input [31:0] x,
                       [31:0] y,
                              c0,
              output [31:0] s
      );
      wire [31:0] c;
    FA_basic uut(
                    .Xi(x[0]),
                    .Yi(y[0]),
                    .Cin(c0),
                    .Cout(c[1]),
                    .Si(s[0]));
      genvar i;
      generate
        for (i=1;i<32;i=i+1)
          begin
          FA_basic uut(
            .Xi(x[i]),
            .Yi(y[i]),
            .Cin(c[i]),
            .Cout(c[i+1]),
            .Si(s[i]));
          end
        endgenerate
endmodule

module FA_basic (
      input Cin, Xi, Yi,
      output Cout, Si
      );
      wire Pi, Ni, Gi, Mi;
      xor(Pi,Xi,Yi);
      xor(Si,Cin,Pi);
      and(Ni,Pi,Cin);
      and(Mi,Xi,Yi);
```

```
        or(Cout,Mi,Ni);
endmodule

module ALU_Multiplier(
        input [15:0] A,
        input [15:0] B,
        output [31:0] Y
        );
wire [15:0] t1 [15:0];
wire [16:0] t2 [16:0];
wire [16:0] Cin [15:0];

genvar i, j;
generate
    for (i=0;i<17;i=i+1)
    begin
        assign t2[0][i] = 1'b0;
    end
    for (i=0;i<16;i=i+1)
    begin
        assign Cin[i][0] = 1'b0;
        for (j=0;j<16;j=j+1)
            begin
                and(t1[i][j], A[j], B[i]);
                FA_basic uut(.Xi(t1[i][j]), .Yi(t2[i][j+1]),
                                .Cin(Cin[i][j]), .Cout(Cin[i][j+1]), .Si(t2[i+1][j]));
            end
        assign Y[i] = t2[i+1][0];
        assign t2[i+1][16] = Cin[i][16];
    end
    for (i=16;i<32;i=i+1)
        begin
            assign Y[i] = t2[16][i-15];
        end
endgenerate

endmodule
```

```
module ALU_SLT(
        input [31:0] a,
        input [31:0] b,
        output [31:0] Y
        );
        wire equal, greater;
        wire nequal, ngreater;
        wire m;
        wire [31:0] g,e;
        not(m,a[31]);
        and(g[31],a[31],m);
        xnor(e[31],a[31],b[31]);
        genvar i;
        generate
              for (i=30;i>-1;i=i-1)
                    begin
                          one_bit_comparator uut(
                          .gin(g[i+1]),
                          .ein(e[i+1]),
                          .xin(a[i]),
                          .yin(b[i]),
                          .gout(g[i]),
                          .eout(e[i]));
                    end
        endgenerate
        assign equal=e[0];
        assign greater=g[0];
        for (i=3; i<32; i=i+1)
           begin
              assign Y[i] = 1'b0;
           end
        not(nequal, equal);
        not(ngreater, greater);
        and(Y[0], equal, ngreater);
        and(Y[1], nequal, greater);
        and(Y[2], nequal, ngreater);
endmodule
```

```
module one_bit_comparator(
        input gin,
        input ein,
        input xin,
        input yin,
        output gout,
        output eout
        );
        wire l1,l2,l3,l4,l5,l6;
        not (l1,xin);
        not (l2,yin);
        and (l3,ein,xin,l2);
        or (gout,gin,l3);
        xnor (l4,xin,yin);
        and(l5,ein,l4);
        and(l6,gin,ein);
        or(eout,l6,l5);
endmodule

module ALU_Shifter(
                input [31:0] A,
                input [31:0] B,
                input [2:0] F,
                output [31:0] C
                );

                wire [31:0] C1;//????¡Á¨®??????
                wire [31:0] A2;//????¡¤?¡Á??¨®??A
                wire [31:0] C2_i;//A2??????¡Á¨®??????
                wire [31:0] C2;//??C2_i??¡¤?¡Á??¨®?????????????
                wire [31:0] s;//??????¡À¨º
                wire [31:0] ci;//????¡¤?????????
                wire [31:0] C3_i;//??¡¤?????????C2_i?¨®??????
                wire [31:0] C3;//????????????

                //????¡Á¨®
                left_shift U0(
                        .A(A),
```

```
        .B(B),
        .C(C1)
        );
//??????
 genvar i;
 generate
 for(i=0;i<32;i=i+1)
 begin
       assign A2[i]=A[31-i];
 end
 endgenerate
 left_shift U1(
        .A(A2),
        .B(B),
        .C(C2_i),
        .s(s)
        );

  generate
  for(i=0;i<32;i=i+1)
     begin
          assign C2[i]=C2_i[31-i];
     end
  endgenerate
 //??¡Á???

 generate
 for(i=0;i<32;i=i+1)
 begin
       and(ci[i],A2[0],s[i]);
 end
 endgenerate

 or (C3_i[31],C2_i[31]);
 or (C3_i[30],C2_i[30],ci[31]);
 or (C3_i[29],C2_i[29],ci[31],ci[30]);
 or (C3_i[28],C2_i[28],ci[31],ci[30],ci[29]);
 or (C3_i[27],C2_i[27],ci[31],ci[30],ci[29],ci[28]);
```

```
                or (C3_i[26],C2_i[26],ci[31],ci[30],ci[29],ci[28],ci[27]);
                or (C3_i[25],C2_i[25],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26]);
                or (C3_i[24],C2_i[24],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25]);
                or
(C3_i[23],C2_i[23],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24]);
                or
(C3_i[22],C2_i[22],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23]);
                or
(C3_i[21],C2_i[21],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22])
;
                or
(C3_i[20],C2_i[20],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21]);
                or
(C3_i[19],C2_i[19],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21],ci[20]);
                or
(C3_i[18],C2_i[18],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21],ci[20],ci[19]);
                or
(C3_i[17],C2_i[17],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21],ci[20],ci[19],ci[18]);
                or
(C3_i[16],C2_i[16],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21],ci[20],ci[19],ci[18],ci[17]);
                or
(C3_i[15],C2_i[15],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21],ci[20],ci[19],ci[18],ci[17],ci[16]);
                or
(C3_i[14],C2_i[14],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15]);
                or
(C3_i[13],C2_i[13],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14]);
                or
(C3_i[12],C2_i[12],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],
ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13]);
                or
```

(C3_i[11],C2_i[11],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12]);

or

(C3_i[10],C2_i[10],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11]);

or

(C3_i[9],C2_i[9],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10]);

or

(C3_i[8],C2_i[8],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9]);

or

(C3_i[7],C2_i[7],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9],ci[8]);

or

(C3_i[6],C2_i[6],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9],ci[8],ci[7]);

or

(C3_i[5],C2_i[5],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9],ci[8],ci[7],ci[6]);

or

(C3_i[4],C2_i[4],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9],ci[8],ci[7],ci[6],ci[5]);

or

(C3_i[3],C2_i[3],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9],ci[8],ci[7],ci[6],ci[5],ci[4]);

or

(C3_i[2],C2_i[2],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9],ci[8],ci[7],ci[6],ci[5],ci[4],ci[3]);

or

(C3_i[1],C2_i[1],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9],ci[8]

```
,ci[7],ci[6],ci[5],ci[4],ci[3],ci[2]);
                or
(C3_i[0],C2_i[0],ci[31],ci[30],ci[29],ci[28],ci[27],ci[26],ci[25],ci[24],ci[23],ci[22],ci[
21],ci[20],ci[19],ci[18],ci[17],ci[16],ci[15],ci[14],ci[13],ci[12],ci[11],ci[10],ci[9],ci[8]
,ci[7],ci[6],ci[5],ci[4],ci[3],ci[2],ci[1]);

                generate
                for(i=0;i<32;i=i+1)
                   begin
                       assign C3[i]=C3_i[31-i];
                   end
                endgenerate

                //?¨®?? ?¡¤?¡§C
                wire [31:0] C3C;
                wire [31:0] C2C;
                wire [31:0] C1C;
                generate
                for(i=0;i<32;i=i+1)
                   begin
                       and(C1C[i],F[0],C1[i]);
                       and(C2C[i],F[1],C2[i]);
                       and(C3C[i],F[2],C3[i]);
                       or(C[i],C1C[i],C2C[i],C3C[i]);
                   end
                endgenerate
    endmodule

    module left_shift(
        input [31:0] A,
        input [31:0] B,
        output [31:0] C,
        output [31:0] s
        );
        //c0-c31 ??????¡Á?????32:1 mux????????????
        wire [31:0] c0;
        wire [31:0] c1;
        wire [31:0] c2;
```

```
wire [31:0] c3;
wire [31:0] c4;
wire [31:0] c5;
wire [31:0] c6;
wire [31:0] c7;
wire [31:0] c8;
wire [31:0] c9;
wire [31:0] c10;
wire [31:0] c11;
wire [31:0] c12;
wire [31:0] c13;
wire [31:0] c14;
wire [31:0] c15;
wire [31:0] c16;
wire [31:0] c17;
wire [31:0] c18;
wire [31:0] c19;
wire [31:0] c20;
wire [31:0] c21;
wire [31:0] c22;
wire [31:0] c23;
wire [31:0] c24;
wire [31:0] c25;
wire [31:0] c26;
wire [31:0] c27;
wire [31:0] c28;
wire [31:0] c29;
wire [31:0] c30;
wire [31:0] c31;

wire low;//??
//32????????
parameter s0 = 32'b0;
parameter s1 = 32'b1;
parameter s2 = 32'b10;
parameter s3 = 32'b11;
parameter s4 = 32'b100;
parameter s5 = 32'b101;
```

```
parameter s6 = 32'b110;
parameter s7 = 32'b111;
parameter s8 = 32'b1000;
parameter s9 = 32'b1001;
parameter s10 = 32'b1010;
parameter s11 = 32'b1011;
parameter s12 = 32'b1100;
parameter s13 = 32'b1101;
parameter s14 = 32'b1110;
parameter s15 = 32'b1111;
parameter s16 = 32'b10000;
parameter s17 = 32'b10001;
parameter s18 = 32'b10010;
parameter s19 = 32'b10011;
parameter s20 = 32'b10100;
parameter s21 = 32'b10101;
parameter s22 = 32'b10110;
parameter s23 = 32'b10111;
parameter s24 = 32'b11000;
parameter s25 = 32'b11001;
parameter s26 = 32'b11010;
parameter s27 = 32'b11011;
parameter s28 = 32'b11100;
parameter s29 = 32'b11101;
parameter s30 = 32'b11110;
parameter s31 = 32'b11111;
//?????¨¤??????????s
AND_f U0(
        .a(s0),
        .b(B),
        .co(s[0])
);
AND_f       U1(
        .a(s1),
        .b(B),
        .co(s[1])
);
AND_f       U2(
```

```
        .a(s2),
        .b(B),
        .co(s[2])
    );
    AND_f       U3(
        .a(s3),
        .b(B),
        .co(s[3])
    );
    AND_f       U4(
        .a(s4),
        .b(B),
        .co(s[4])
    );
    AND_f       U5(
        .a(s5),
        .b(B),
        .co(s[5])
    );
    AND_f       U6(
        .a(s6),
        .b(B),
        .co(s[6])
    );
    AND_f       U7(
        .a(s7),
        .b(B),
        .co(s[7])
    );
    AND_f       U8(
        .a(s8),
        .b(B),
        .co(s[8])
    );
    AND_f       U9(
        .a(s9),
        .b(B),
        .co(s[9])
```

```
);
AND_f       U10(
        .a(s10),
        .b(B),
        .co(s[10])
);
AND_f       U11(
        .a(s11),
        .b(B),
        .co(s[11])
);
AND_f       U12(
        .a(s12),
        .b(B),
        .co(s[12])
);
AND_f       U13(
        .a(s13),
        .b(B),
        .co(s[13])
);
AND_f       U14(
        .a(s14),
        .b(B),
        .co(s[14])
);
AND_f       U15(
        .a(s15),
        .b(B),
        .co(s[15])
);
AND_f       U16(
        .a(s16),
        .b(B),
        .co(s[16])
);
AND_f       U17(
        .a(s17),
```

```
        .b(B),
        .co(s[17])
    );
    AND_f       U18(
        .a(s18),
        .b(B),
        .co(s[18])
    );
    AND_f       U19(
        .a(s19),
        .b(B),
        .co(s[19])
    );
    AND_f       U20(
        .a(s20),
        .b(B),
        .co(s[20])
    );
    AND_f       U21(
        .a(s21),
        .b(B),
        .co(s[21])
    );
    AND_f       U22(
        .a(s22),
        .b(B),
        .co(s[22])
    );
    AND_f       U23(
        .a(s23),
        .b(B),
        .co(s[23])
    );
    AND_f       U24(
        .a(s24),
        .b(B),
        .co(s[24])
    );
```

```
AND_f       U25(
        .a(s25),
        .b(B),
        .co(s[25])
);
AND_f       U26(
        .a(s26),
        .b(B),
        .co(s[26])
);
AND_f       U27(
        .a(s27),
        .b(B),
        .co(s[27])
);
AND_f       U28(
        .a(s28),
        .b(B),
        .co(s[28])
);
AND_f       U29(
        .a(s29),
        .b(B),
        .co(s[29])
);
AND_f       U30(
        .a(s30),
        .b(B),
        .co(s[30])
);
AND_f       U31(
        .a(s31),
        .b(B),
        .co(s[31])
);
//????32??mux ????????????????
genvar i;
generate
```

```
//32:1mux *32
for (i=0;i<32;i=i+1)
begin
	and (c31[i], s[i], A[31 - i]);
end
for (i=0;i<31;i=i+1)
begin
	and (c30[i], s[i], A[30 - i]);
end
for (i=0;i<30;i=i+1)
begin
	and (c29[i], s[i], A[29 - i]);
end
for (i=0;i<29;i=i+1)
begin
	and (c28[i], s[i], A[28 - i]);
end
for (i=0;i<28;i=i+1)
begin
	and (c27[i], s[i], A[27 - i]);
end
for (i=0;i<27;i=i+1)
begin
	and (c26[i], s[i], A[26 - i]);
end
for (i=0;i<26;i=i+1)
begin
	and (c25[i], s[i], A[25 - i]);
end
for (i=0;i<25;i=i+1)
begin
	and (c24[i], s[i], A[24 - i]);
end
for (i=0;i<24;i=i+1)
begin
	and (c23[i], s[i], A[23 - i]);
end
for (i=0;i<23;i=i+1)
```

```
begin
        and (c22[i], s[i], A[22 - i]);
end
for (i=0;i<22;i=i+1)
begin
        and (c21[i], s[i], A[21 - i]);
end
for (i=0;i<21;i=i+1)
begin
        and (c20[i], s[i], A[20 - i]);
end
for (i=0;i<20;i=i+1)
begin
        and (c19[i], s[i], A[19 - i]);
end
for (i=0;i<19;i=i+1)
begin
        and (c18[i], s[i], A[18 - i]);
end
for (i=0;i<18;i=i+1)
begin
        and (c17[i], s[i], A[17 - i]);
end
for (i=0;i<17;i=i+1)
begin
        and (c16[i], s[i], A[16 - i]);
end
for (i=0;i<16;i=i+1)
begin
        and (c15[i], s[i], A[15 - i]);
end
for (i=0;i<15;i=i+1)
begin
        and (c14[i], s[i], A[14 - i]);
end
for (i=0;i<14;i=i+1)
begin
        and (c13[i], s[i], A[13 - i]);
```

```
end
for (i=0;i<13;i=i+1)
begin
        and (c12[i], s[i], A[12 - i]);
end
for (i=0;i<12;i=i+1)
begin
        and (c11[i], s[i], A[11 - i]);
end
for (i=0;i<11;i=i+1)
begin
        and (c10[i], s[i], A[10 - i]);
end
for (i=0;i<10;i=i+1)
begin
        and (c9[i], s[i], A[9 - i]);
end
for (i=0;i<9;i=i+1)
begin
        and (c8[i], s[i], A[8 - i]);
end
for (i=0;i<8;i=i+1)
begin
        and (c7[i], s[i], A[7 - i]);
end
for (i=0;i<7;i=i+1)
begin
        and (c6[i], s[i], A[6 - i]);
end
for (i=0;i<6;i=i+1)
begin
        and (c5[i], s[i], A[5 - i]);
end
for (i=0;i<5;i=i+1)
begin
        and (c4[i], s[i], A[4 - i]);
end
for (i=0;i<4;i=i+1)
```

```
                begin
                        and (c3[i], s[i], A[3 - i]);
                end
                for (i=0;i<3;i=i+1)
                begin
                        and (c2[i], s[i], A[2 - i]);
                end
                for (i=0;i<2;i=i+1)
                begin
                        and (c1[i], s[i], A[1 - i]);
                end
                for (i=0;i<1;i=i+1)
                begin
                        and (c0[i], s[i], A[0 - i]);
                end
        endgenerate

        assign low=0;//??
        //????????¡Á¨®??
        or (C[0],c0[0],low);
        or (C[1],c1[0],c1[1]);
        or (C[2],c2[0],c2[1],c2[2]);
        or (C[3],c3[0],c3[1],c3[2],c3[3]);
        or (C[4],c4[0],c4[1],c4[2],c4[3],c4[4]);
        or (C[5],c5[0],c5[1],c5[2],c5[3],c5[4],c5[5]);
        or (C[6],c6[0],c6[1],c6[2],c6[3],c6[4],c6[5],c6[6]);
        or (C[7],c7[0],c7[1],c7[2],c7[3],c7[4],c7[5],c7[6],c7[7]);
        or (C[8],c8[0],c8[1],c8[2],c8[3],c8[4],c8[5],c8[6],c8[7],c8[8]);
        or (C[9],c9[0],c9[1],c9[2],c9[3],c9[4],c9[5],c9[6],c9[7],c9[8],c9[9]);
        or
(C[10],c10[0],c10[1],c10[2],c10[3],c10[4],c10[5],c10[6],c10[7],c10[8],c10[9],c10[10
]);
        or
(C[11],c11[0],c11[1],c11[2],c11[3],c11[4],c11[5],c11[6],c11[7],c11[8],c11[9],c11[10],
c11[11]);
        or
(C[12],c12[0],c12[1],c12[2],c12[3],c12[4],c12[5],c12[6],c12[7],c12[8],c12[9],c12[10
],c12[11],c12[12]);
```

```
        or
(C[13],c13[0],c13[1],c13[2],c13[3],c13[4],c13[5],c13[6],c13[7],c13[8],c13[9],c13[10
],c13[11],c13[12],c13[13]);
        or
(C[14],c14[0],c14[1],c14[2],c14[3],c14[4],c14[5],c14[6],c14[7],c14[8],c14[9],c14[10
],c14[11],c14[12],c14[13],c14[14]);
        or
(C[15],c15[0],c15[1],c15[2],c15[3],c15[4],c15[5],c15[6],c15[7],c15[8],c15[9],c15[10
],c15[11],c15[12],c15[13],c15[14],c15[15]);
        or
(C[16],c16[0],c16[1],c16[2],c16[3],c16[4],c16[5],c16[6],c16[7],c16[8],c16[9],c16[10
],c16[11],c16[12],c16[13],c16[14],c16[15],c16[16]);
        or
(C[17],c17[0],c17[1],c17[2],c17[3],c17[4],c17[5],c17[6],c17[7],c17[8],c17[9],c17[10
],c17[11],c17[12],c17[13],c17[14],c17[15],c17[16],c17[17]);
        or
(C[18],c18[0],c18[1],c18[2],c18[3],c18[4],c18[5],c18[6],c18[7],c18[8],c18[9],c18[10
],c18[11],c18[12],c18[13],c18[14],c18[15],c18[16],c18[17],c18[18]);
        or
(C[19],c19[0],c19[1],c19[2],c19[3],c19[4],c19[5],c19[6],c19[7],c19[8],c19[9],c19[10
],c19[11],c19[12],c19[13],c19[14],c19[15],c19[16],c19[17],c19[18],c19[19]);
        or
(C[20],c20[0],c20[1],c20[2],c20[3],c20[4],c20[5],c20[6],c20[7],c20[8],c20[9],c20[10
],c20[11],c20[12],c20[13],c20[14],c20[15],c20[16],c20[17],c20[18],c20[19],c20[20]);
        or
(C[21],c21[0],c21[1],c21[2],c21[3],c21[4],c21[5],c21[6],c21[7],c21[8],c21[9],c21[10
],c21[11],c21[12],c21[13],c21[14],c21[15],c21[16],c21[17],c21[18],c21[19],c21[20],c
21[21]);
        or
(C[22],c22[0],c22[1],c22[2],c22[3],c22[4],c22[5],c22[6],c22[7],c22[8],c22[9],c22[10
],c22[11],c22[12],c22[13],c22[14],c22[15],c22[16],c22[17],c22[18],c22[19],c22[20],c
22[21],c22[22]);
        or
(C[23],c23[0],c23[1],c23[2],c23[3],c23[4],c23[5],c23[6],c23[7],c23[8],c23[9],c23[10
],c23[11],c23[12],c23[13],c23[14],c23[15],c23[16],c23[17],c23[18],c23[19],c23[20],c
23[21],c23[22],c23[23]);
        or
(C[24],c24[0],c24[1],c24[2],c24[3],c24[4],c24[5],c24[6],c24[7],c24[8],c24[9],c24[10
```

```
],c24[11],c24[12],c24[13],c24[14],c24[15],c24[16],c24[17],c24[18],c24[19],c24[20],c
24[21],c24[22],c24[23],c24[24]);
        or
(C[25],c25[0],c25[1],c25[2],c25[3],c25[4],c25[5],c25[6],c25[7],c25[8],c25[9],c25[10
],c25[11],c25[12],c25[13],c25[14],c25[15],c25[16],c25[17],c25[18],c25[19],c25[20],c
25[21],c25[22],c25[23],c25[24],c25[25]);
        or
(C[26],c26[0],c26[1],c26[2],c26[3],c26[4],c26[5],c26[6],c26[7],c26[8],c26[9],c26[10
],c26[11],c26[12],c26[13],c26[14],c26[15],c26[16],c26[17],c26[18],c26[19],c26[20],c
26[21],c26[22],c26[23],c26[24],c26[25],c26[26]);
        or
(C[27],c27[0],c27[1],c27[2],c27[3],c27[4],c27[5],c27[6],c27[7],c27[8],c27[9],c27[10
],c27[11],c27[12],c27[13],c27[14],c27[15],c27[16],c27[17],c27[18],c27[19],c27[20],c
27[21],c27[22],c27[23],c27[24],c27[25],c27[26],c27[27]);
        or
(C[28],c28[0],c28[1],c28[2],c28[3],c28[4],c28[5],c28[6],c28[7],c28[8],c28[9],c28[10
],c28[11],c28[12],c28[13],c28[14],c28[15],c28[16],c28[17],c28[18],c28[19],c28[20],c
28[21],c28[22],c28[23],c28[24],c28[25],c28[26],c28[27],c28[28]);
        or
(C[29],c29[0],c29[1],c29[2],c29[3],c29[4],c29[5],c29[6],c29[7],c29[8],c29[9],c29[10
],c29[11],c29[12],c29[13],c29[14],c29[15],c29[16],c29[17],c29[18],c29[19],c29[20],c
29[21],c29[22],c29[23],c29[24],c29[25],c29[26],c29[27],c29[28],c29[29]);
        or
(C[30],c30[0],c30[1],c30[2],c30[3],c30[4],c30[5],c30[6],c30[7],c30[8],c30[9],c30[10
],c30[11],c30[12],c30[13],c30[14],c30[15],c30[16],c30[17],c30[18],c30[19],c30[20],c
30[21],c30[22],c30[23],c30[24],c30[25],c30[26],c30[27],c30[28],c30[29],c30[30]);
        or
(C[31],c31[0],c31[1],c31[2],c31[3],c31[4],c31[5],c31[6],c31[7],c31[8],c31[9],c31[10
],c31[11],c31[12],c31[13],c31[14],c31[15],c31[16],c31[17],c31[18],c31[19],c31[20],c
31[21],c31[22],c31[23],c31[24],c31[25],c31[26],c31[27],c31[28],c31[29],c31[30],c31
[31]);

    endmodule
```

附录 2：

```
    module main_decoder_FSM(
```

```
        input [5:0] Opcode,
        input clr,
        input clk,
        output reg MemtoReg,
                reg RegDst,
                reg IorD,
                reg [1:0] PCSrc,
                reg [1:0] ALUSrcB,
                reg ALUSrcA,
                reg IRWrite,
                reg MemWrite,
                reg PCWrite,
                reg Branch,
                reg RegWrite,
                reg [2:0] ALUOp
        );
    reg [3:0] current_state, next_state;
    parameter S0 = 4'b0000, S1=4'b0001, S2=4'b0010, S3=4'b0011,S4=4'b0100,
S5=4'b0101, S6=4'b0110,
                    S7=4'b0111, S8=4'b1000, S9=4'b1001, S10=4'b1010, S11=4'b1011,
S12=4'b1100, S13=4'b1101;
    parameter lw = 6'b100011, sw = 6'b101011, r_type = 6'b000000, beq = 6'b000100,
addi = 6'b001000, j = 6'b000010, andi = 6'b001100, ori = 6'b001101;

    always @(posedge clk or posedge clr)
      begin
        if (clr == 1'b1)
          current_state <= S0;
        else
          current_state <= next_state;
      end

    always @(*)
    begin
       case(current_state)
         S0: next_state <= S1;
         S1: case (Opcode)
                lw: next_state <= S2;
```

```
                    sw: next_state <= S2;
                    r_type: next_state <= S6;
                    beq: next_state <= S8;
                    addi: next_state <= S9;
                    j: next_state <= S11;
                    andi: next_state <= S12;
                    ori: next_state <= S13;
                endcase
            S2: if(Opcode == lw)
                    next_state <= S3;
                else if(Opcode == sw)
                    next_state <= S5;
            S3: next_state <= S4;
            S4: next_state <= S0;
            S5: next_state <= S0;
            S6: next_state <= S7;
            S7: next_state <= S0;
            S8: next_state <= S0;
            S9: next_state <= S10;
            S10: next_state <= S0;
            S11: next_state <= S0;
            S12: next_state <= S10;
            S13: next_state <= S10;
        endcase
    end

    always @(*)
        begin
            case (current_state)
                S0: begin MemtoReg=0; IorD =
1'b0;MemWrite=1'b0;IRWrite=1'b1;PCWrite=1'b1;Branch=1'b0;
RegDst=1'b0;PCSrc=2'b00;ALUOp=3'b000;ALUSrcB=2'b01;ALUSrcA=1'b0;RegWr
ite=1'b0; end
                S1: begin ALUSrcB=2'b11; IRWrite = 0; PCWrite = 0; end
                S2: begin ALUSrcA=1; ALUSrcB=2'b10; end
                S3: begin IorD=1; ALUSrcA=0; ALUSrcB=2'b00; end
                S4: begin RegDst=0;MemtoReg=1;RegWrite=1; IorD=0; end
                S5: begin IorD=1;MemWrite=1; ALUSrcA=0; ALUSrcB=2'b00; end
```

```
            S6: begin ALUSrcA=1;ALUSrcB=2'b00;ALUOp=3'b010; end
            S7: begin RegDst=1;MemtoReg=0;RegWrite=1;
ALUSrcA=0;ALUSrcB=2'b00;ALUOp=3'b000; end
            S8: begin
ALUSrcA=1;ALUSrcB=2'b00;ALUOp=3'b001;PCSrc=2'b01;Branch=1; end
            S9: begin ALUSrcA=1;ALUSrcB=2'b10;ALUOp=3'b000; end
            S10: begin RegDst=0;MemtoReg=0;RegWrite=1;
ALUSrcA=0;ALUSrcB=2'b00;ALUOp=3'b000; end
            S11: begin PCSrc=2'b10;PCWrite=1;ALUSrcB=2'b00; end
            S12: begin ALUSrcA=1;ALUSrcB=2'b10;ALUOp=3'b100; end
            S13: begin ALUSrcA=1;ALUSrcB=2'b10;ALUOp=3'b101; end
          endcase
        end
    endmodule

    module ALUDecoder(
          input [2:0] ALUOp,
          input [5:0] funct,
          output reg [3:0] ALUcontrol
          );
        always@(*)
        begin

ALUcontrol[3]=~ALUOp[2]&ALUOp[0]&funct[5]&~funct[4]&funct[3]&~funct[2]
&funct[1]&~funct[0];
        ALUcontrol[2]=~ALUOp[2]&ALUOp[0]|~ALUOp[2] &
ALUOp[1]&funct[5]&~funct[4]&~funct[3]&~funct[2]&funct[1]&~funct[0];
        ALUcontrol[1]=~ALUOp[2]&ALUOp[0]|~ALUOp[2] &
ALUOp[1]&funct[5]&~funct[4]&~funct[2]&~funct[0]&~funct[3]|~ALUOp[2]&~AL
UOp[1]&~ALUOp[0];
        ALUcontrol[0]=ALUOp[2]&~ALUOp[1]&ALUOp[0]|~ALUOp[2] &
ALUOp[1]&funct[5]&~funct[4]&~funct[3]&funct[2]&~funct[1]&funct[0];
        end
    endmodule
```

附录 3：

```
module IorD_mux(
        input s,
        input [31:0] i1,
        input [31:0] i2,
        output reg [31:0] o
        );
        always@(*)
           begin
              case (s)
                 1'b0: o = i1;
                 1'b1: o = i2;
              endcase
           end
endmodule

module d_flipflow_enable(
        input [31:0] A,
        output reg [31:0] B,
        input en,
        input CLK
        );
                always @ (posedge CLK)//我们用正的时钟沿做它的敏感信号
              begin
              if(en)
                 B = A;//上升沿有效的时候，把 d 捕获到 q
              else
                 ;
              end
endmodule
```

附录 4：

```
module Register_File(
   input clk,
   input we3,
   input [4:0] ra1, ra2, wa3,
   input [31:0] wd3,
   output [31:0] rd1, rd2
```

```
);
reg [31:0] rf [31:0];

always @(posedge clk)
   begin
      if(we3) rf[wa3] <= wd3;
   end
      assign rd1 = (ra1 != 0) ? rf[ra1] : 0;
      assign rd2 = (ra2 != 0) ? rf[ra2] : 0;
endmodule
```