|  |
| --- |
| tigerwang202@gmail.com |
| rAVR内核在Xilnx FPGA上的实现 |
| 8位处理器内核设计实验 |

|  |
| --- |
| WangMengyin  2011-10-12 |

1. 背景

rAVR项目并不准备实现完整功能的AVR兼容内核，相反它实现了最小功能的AVR内核。最初的设计目标是实现一个适合非常小的CPLD（例如EPM240）的处理器，同时仍然留下一些空间给其他逻辑。开发平台选用“Marsohod”的开源开发板。关于该开发板信息参考以下连接（[http://www.marsohod.org/index.php/howtostart/plata](http://www.marsohod.org/index.php/howtostart/plata" \t "_blank)），是俄语网站。

开发板提供4个按键、8盏LED灯、2个步进电机接口。

该AVR项目实现下述内容：

1. 仅实现4个通用寄存器r16~r19。
2. 通用寄存器r20中的位输出连接8盏LED灯输出引脚。
3. 通用寄存器r21中的位连接至步进电机的6个输出引脚
4. 通用寄存器r22中的低4位连接至4个按键状态输入引脚。

没有实现I/O端口、定时器、中断及其他AVR功能。但不管怎样该内核同真的AVR单片机是兼容的。Altera CPLD内置UFM用户Flash存储器，它以512个16位长度的字组成，AVR程序存放其中。

项目实现下列AVR指令：

ADD 0000 11rd dddd rrrr

SUB 0001 10rd dddd rrrr

AND 0010 00rd dddd rrrr

EOR 0010 01rd dddd rrrr

OR 0010 10rd dddd rrrr

MOV 0010 11rd dddd rrrr

CP 0001 01rd dddd rrrr

LSR 1001 010d dddd 0110

SUBI 0101 KKKK dddd KKKK

ANDI 0111 KKKK dddd KKKK

ORI 0110 KKKK dddd KKKK

CPI 0011 KKKK dddd KKKK

LDI 1110 KKKK dddd KKKK

BREQ 1111 00kk kkkk k001

BRNE 1111 01kk kkkk k001

BRCS 1111 00kk kkkk k000

BRCC 1111 01kk kkkk k000

“d”代表目的寄存器，“r”代表源寄存器，“k”代表立即数。

内核只有2个标志位“Z”和“C”，设计实现于此对应的下列条件跳转指令（BREQ、BRNE、BRCS、BRCC），没有实现与子程序调用相关的实跳转指令，没有实现数据存储器。由于资源及其有限，实现上述设计似乎是不可行的，但通过运行ATMEL公司AVRStudio编译生成的汇编程序，测试应用板上按键控制对应LED指示灯状态，表明该内核确实能正常工作。

测试程序如下：

.include "1200def.inc"

.device AT90S1200

.cseg

.org 0

start:

;initial one bit in register

ldi r16,$80

rd\_port:

;read port (key status)

mov r17,r22

cpi r17,$0f

;go and blink one LED if no key pressed

breq do\_xor

cpi r17,$0e

;go and right shift LEDs if key[0] pressed

breq do\_rshift

cpi r17,$0d

;go and left shift LEDs if key[1] pressed

breq do\_lshift

;jump to read keys

or r16,r16

brne rd\_port

do\_rshift:

cpi r16,1

breq set80

lsr r16

mov r20,r16

brne pause

set80:

ldi r16,$80

mov r20,r16

or r16,r16

brne pause

do\_lshift:

cpi r16,$80

breq set1

lsl r16

mov r20,r16

brne pause

set1:

ldi r16,$01

mov r20,r16

or r16,r16

brne pause

do\_xor:

eor r20,r16

pause:

ldi r18,$10

cycle2:

ldi r19,$FF

cycle1:

or r19,r19

or r19,r19

subi r19,1

brne cycle1

subi r18,1

brne cycle2

or r16,r16

brne rd\_port

1. 目的
2. 熟悉8位处理器组成结构，为今后实现更为复杂处理器设计打下基础。
3. 评估Xilinx FPGA上实现8位处理器所需资源、速度。
4. 了解AVR指令系统。
5. rAVR源代码分析

rAVR采用同步设计，程序指令存放在16位字宽的UFM Flash中，共512个字的容量。UFM与rAVR内核采用串行接口，通过地址数据移位时钟信号将指令读入rAVR内核。

注：下列若非特殊说明，信号均是高电平有效。

模块rAVR描述包括下列接口：

表 1

|  |  |
| --- | --- |
| 名称 | 作用 |
| clk | 时钟 |
| reset | 异步复位信号 |
| UFM接口地址 | |
| arclkena | 地址串行移入时钟使能 |
| arclkshift | 地址串行移入操作使能 |
| ardout | 地址串行移入数据（地址输出给UFM） |
| UFM接口数据 | |
| drdin | 数据串行移出（数据输入给rAVR） |
| drshift | 数据移入操作使能 |
| 并行I/O接口 | |
| port0 | r20（输出连接LED指示灯） |
| port1 | r21（输出连接步进电机） |
| port2 | r22（输入连接按钮和其他输入） |

模块中定义一些信号和寄存器，比较重要的有下列几个：

reg [15:0]opcode 操作符寄存器，用于存放UFM中取得指令。

reg [8:0]ip 指令指针指向下一条执行指令在UFM中的地址，需要说明下，UFM中一个字长16位，这里的地址是16位字的地址，当转换为8位字节字长地址须将ip乘以2。

reg [7:0]alu\_operand0;

reg [7:0]alu\_operand1;

reg [2:0]alu\_cmd;

alu\_operand储存送入ALU参与运算的两个操作数，alu\_operand0对应操作数Rd、alu\_operand1对应操作数Rs或立即数。

alu\_cmd是指令操作码部分。

reg flag\_z;

reg flag\_c;

reg flag是Z、C两个程序状态字。

程序使用counter计数器驱动UFM读状态机，产生适合UFM读取时序，从中取出指令。这部分主要操作EPM240内置的UFM，不作为重点关注。

UFM读状态机还产生与rAVR内核运行相关控制信号，分别是：

wire opcode\_ready 信号，在读取1个指令操作完成时，该信号置高，通知其他进程在下一时钟周期完成指令译码操作。

wire fix\_result 信号，控制将ALU运算结果写回通用寄存器及程序状态字。

wire addr\_inc 信号，控制PC自增，PC <= PC + 1。

rAVR运行流程如下：

UFM读状态机完成一个读操作，即完成去指令操作，将opcode\_ready = 1，指令保存在opcode寄存器中。

指令译码由若干组合逻辑组成，首先有3个组合逻辑，分别从opcode中取出立即数immediate、源寄存器地址src\_reg\_idx、目的寄存器地址dest\_reg\_idx。src\_reg\_idx、dest\_reg\_idx分别作为2个多路选择器地址，从通用寄存器组中选择匹配地址的寄存器并将其分别输出至source\_val和dest\_val信号。这里并不是说1条指令中会**同时包含**上述3个地址，对于跳转指令可能不含有任何一个，而对于SUBI等I结尾的指令才包含立即数。在后续的**指令译码**中会根据指令中不同操作码做相应处理。

指令译码根据操作码不同选择相应的sel\_cmd信号，这里采用一种简化设计，考虑到带有立即数的算术运算（或逻辑运算）与不带立即数的运算，操作过程相同，仅是送入ALU的操作数会有不同。例如：SUB 与SUBI指令，其中SUB指令操作数据是R(r)和R(d)寄存器，而SUBI指令中的R(r)寄存器被立即数代替，显然不需要为SUB和SUBI两个相似的指令创建2个不同的ALU控制信号，这里使用如果sel\_imm信号指示带有立即数的指令，指令中含有立即数将sel\_imm = 1 。对于跳转指令不译码给ALU，而在程序计数器中对IP地址计算。根据sel\_imm选择source\_val（R(s)寄存器中值）或立即数送入alu\_operand1。

指令译码的最后环节是将译码产生的信号在时钟上升沿送入锁存器，在下个时钟周期执行ALU运算。

ALU输入信号，包括alu\_operand0、alu\_operand1、alu\_cmd（ALU命令）运算逻辑是一个组合逻辑，case语句根据不同alu\_cmd对操作数执行相应操作，结果输出alu\_result及程序状态字flag\_c、flag\_z。

**注意：**代码中flag\_c、flag\_z是信号，flag\_c\_fixed、flag\_z\_fixed综合成寄存器。

在下一个clk上升沿ALU输出信号被写入相应寄存器（R(?)和flag\_?\_fixed）。

ALU运算完成同时程序计数器也将更新后程序指针ip写入。基本上ip指针的运算与ALU运算是并行的，对于算术或逻辑指令，新的ip值为ip+1，正常要执行下一条指令。如果遇到跳转指令，在指令译码过程中need\_jump = 1，新的ip值为ip <= ip + {opcode[9],opcode[9],opcode[9:3]}，即跳转后执行的指令地址。

**注意：**根据AVR指令集描述，跳转后PC指针地址PC = PC + **1** + k （k是跳转偏移量），这样就和上面ip的赋值就少加1，仔细查看代码，发现代码在addr\_inc == 1时，已经对ip值进行自增操作（ip <= ip + 1'b1），因此此后跳转中就不用重复加1。

至此新的ip产生跳转到状态（1）执行新一轮取指操作。