mips ejtag原理和实现

乔崇

November 2, 2011

Contents

Abstract

ejtag是mips的onchip debug调试标准。现在龙芯1号和龙芯3号系列都支持ejtag调试。通过ejtag可以大大方便软件调试,这里讲讲ejtag原理和ejtag-debug软件。
ejtag-debug是我编写的一个ejtag调试调试工具，支持读写寄存器、内存、反汇编、执行用户编写的小程序、gdb远程调试和脚本语言。

1 ejtag基础：jtag

ejtag和jtag的关系:
ejtag利用了jtag pin,通过jtag tap状态机器,访问jtag寄存器。通过jtag寄存器来控制处理器进入ejtag异常，在异常里面访问dmseg(0xff2000000-0xff2fffff0)来实现和pc主机进行交互。这一章的内容是讲jtag信号、状态机和寄存器。[1]

1.1 jtag信号

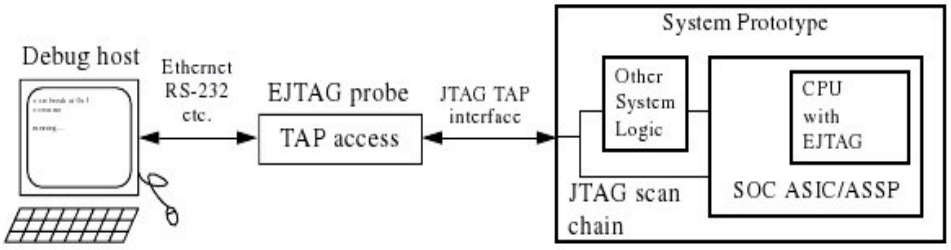

Figure 1: ejtag连接图

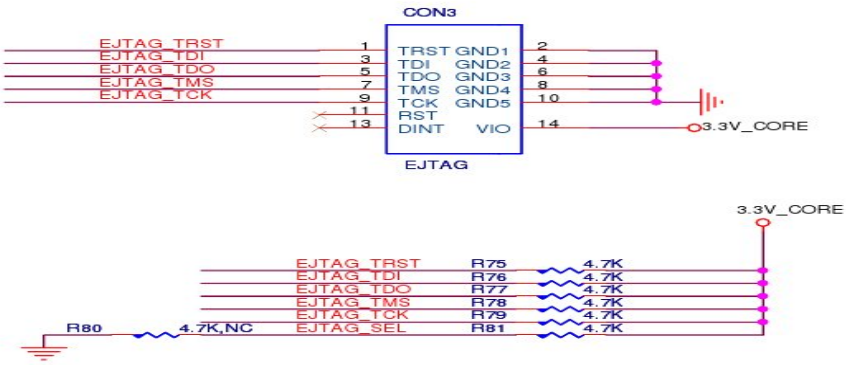

Figure 2: ejtag座信号定义

[1] jtag参考ieee1149.1,ejtag参考MD00047-2B-EJTAG-SPC-03.10.pdf

TRST	test reset 复位任选
TDI	test data in 串行输入到cpu
TDO	test data out 从cpu串行输出
TCLK	test clock 时钟
TMS	test mode select 状态机控制

在tclk的上升沿采集tms,tdi,tdo的信号。

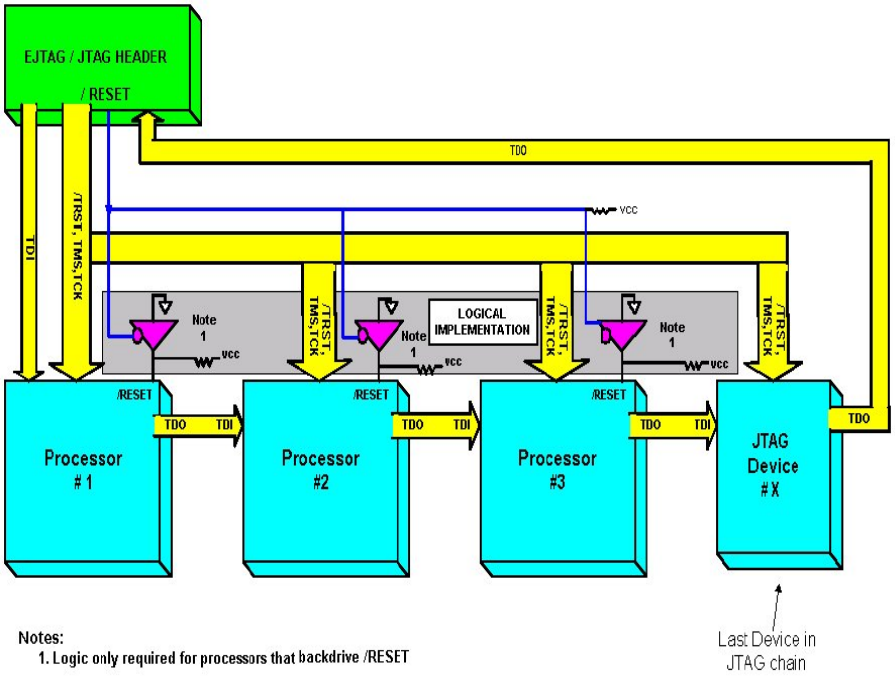

Figure 3: jtag chain

当有多个设备通过jtag连接起来的时候，将设备的tdo连接接到另一个设备的tdi上，成为一个jtag链接。
tms,tclk,trst信号是直接连到每个设备上的。

1.3 jtag状态机

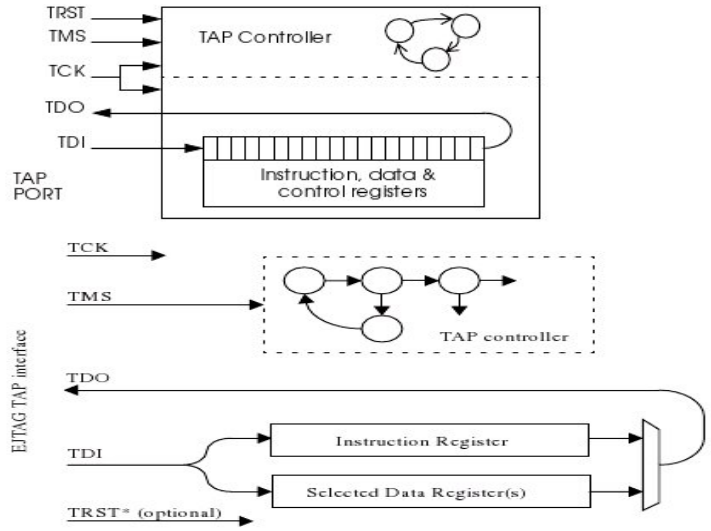

Figure 4: ejtag连接图

上面的两个图形象的画出ejtag tap controller工作的过程，一个是移位，另一个是ir,dr寄存器。通过状态机设置ir，访问dr，就实现访问ir指向的寄存器。ejtag ir是5位，dr是32-64位。

通过jtag状态机读写ir,dr寄存器,来访问jtag/ejtag的寄存器。

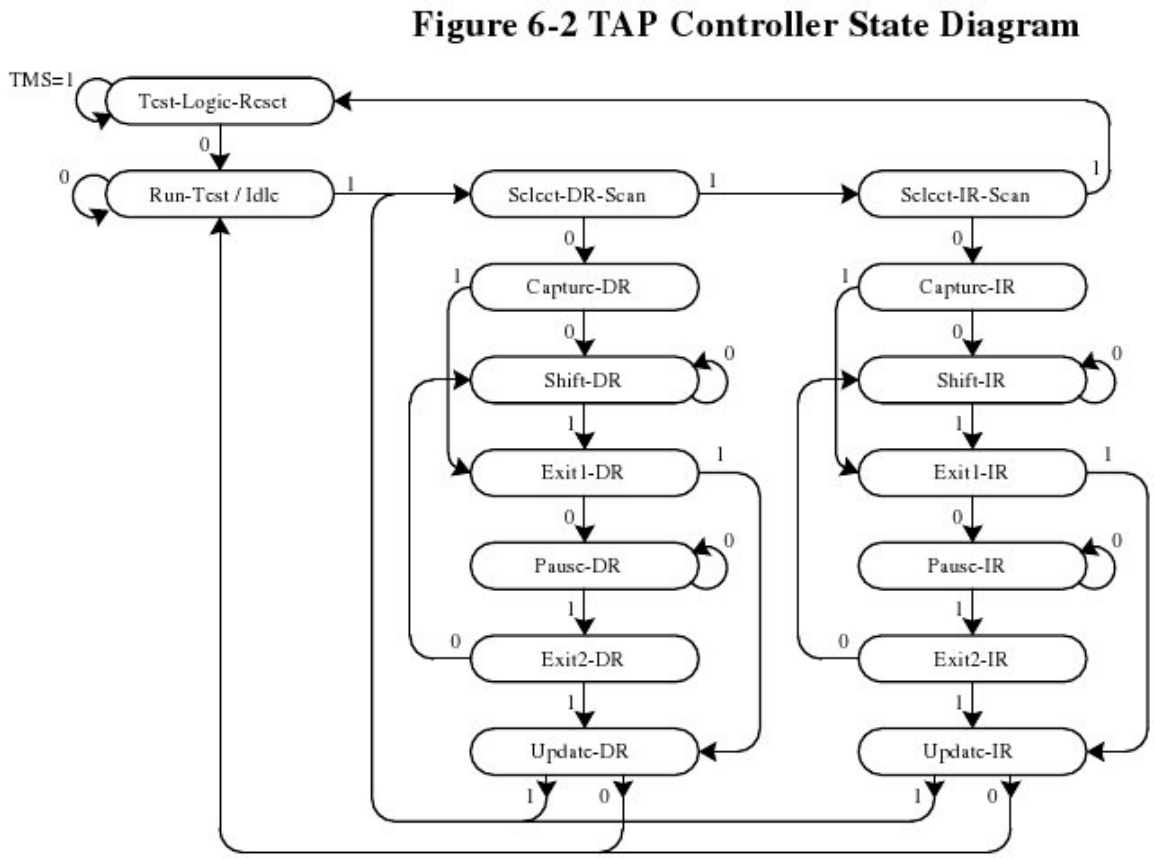

Figure 5: jtag状态机

1.4 jtag的寄存器

ejtag标准扩展了jtag寄存器，如增加了address,data,control寄存器等。要实现ejtag调试主要是反复访问这3个寄存器。

Table 1: Table 6-1 TAP Instruction Overview

Code	Instruction	Function
All 0’s	(Free for other use)	Free for other use, such as JTAG boundary scan
0x01	IDCODE	Selects Device Identi?cation (ID) register
0x02	(Free for other use)	Free for other use, such as JTAG boundary scan
0x03	IMPCODE	Selects Implementation register
0x04 - 0x07	(Free for other use)	Free for other use, such as JTAG boundary scan
0x08	ADDRESS	Selects Address register
0x09	DATA	Selects Data register
0x0A	CONTROL	Selects EJTAG Control register
0x0B	ALL	Selects the Address, Data and EJTAG Control registers
0x0C	EJTAGBOOT	Makes the processor take a debug exception after reset
0x0D	NORMALBOOT	Makes the processor execute the reset handler after reset
0x0E	FASTDATA	Selects the Data and Fastdata registers
0x0F	(EJTAG reserved)	Reserved for future EJTAG use
0x10	TCBCONTROLA	Selects the control register TCBTraceControl in the Trace Control Block
0x11	TCBCONTROLB	Selects another trace control block register,Used to access the registers specified by the TCBCONTROLBREG field
0x12	TCBDATA	and transfers data between the TAP and the TCB control register
0x13	TCBCONTROLC	Selects another trace control block register
0x14	PCSAMPLE	Selects the PCsample register
0x15 - 0x1B	(EJTAG reserved)	Reserved for future EJTAG use
0x1C - All 1’s	(Free for other use)	Free for other use, such as JTAG boundary scan
All 1’s	BYPASS	Select Bypass register

2 ejtag-debug的架构

由很多小命令组成，支持脚本语言。

2.1 编译

```
1 git clone 10.2.5.27:/home2/qiaochong/ejtag-debug
2 cd ejtag-debug
3 make
```

默认的交叉工具链是mipsel-linux-gcc
如果不是需要
make CROSS_COMPILE=mipsel-linux-
make的时候采用mycc,mycpp.pl来实现将c先进行一次预编译处理将tgt_exec,tgt_compile转换成二进制数组。
编译生成ejtag_debug_pp和ejtag_debug_usb
ejtag_debug_pp是用在并口电缆

ejtag_debug_usb是用usb电缆

接着编译bin目录下的程序:
bin目录下的内容是一些小程序用来烧flash,md5sum等.
bin目录下的程序可以编译成elf32或者elf64格式的。

```
cd bin
make CROSS_COMPILE=mipsel-linux- test.elf
make CROSS_COMPILE=mipsel-linux- test.bin
make CROSS_COMPILE=mipsel-linux- test.elf64
make CROSS_COMPILE=mipsel-linux- test.bin64
```

test.bin是abi o32,test.bin64是abi n64.

2.2 运行方法

需要超级用户权限
`sudo ./ejtag_debug_usb`
配置文件是`ejtag.cfg`
程序会自动打开`./ejtag.cfg`并执行里面的内容。

2.3 ejtag-debug参数

```
./ejtag_debug_usb [-dlStch] [-e cmd] [-T n]
-d: verbose on,show debug messags
-e 'cmd': run cmd
-l: do not use read line
-S: log disassemble info
-s: run cmdserver
-t: disable timer
-T n: set timer n ms
-c: do not load cfg file
-h: show this help
```

2.4 ejtag-debug的命令

ejtag-debug由很多小命令组成，支持名字自动补齐。在行首加#表示注释。

- h [cmd] 查看帮助
- setconfig [configname] [value]
  重要的设置:
  helpaddr: 帮助地址，帮助程序执行的地址，地址应该指向ddr,cache/uncache都可以put,get,callbin,call等命令会将帮助程序先上载到helpaddr上，然后执行helpaddr
  usb_ejtag.put_speed: 设置usb上传速度，16bit,越大越慢
  putelf.uncached: 设置putelf为uncached,1在put前刷cache,2不刷cache.
  core.cpucount：设置cpu数目
  core.cpuno：设置当前调试的cpu号
  core.cpuwidth：设置cpu的数据宽度
  还有很到配置用setconfig命令可以列出来

- set [regname|regno] [value] 读写寄存器

```
1         set 读出所有寄存器
2         set pc 读pc
3         set at 读at
4         set pc 0xffffffffbfc00000
5 #设置的数值是pc0xffffffffbfc00000
```

- setenv [envname] [value] 设置环境变量
- d1-d8,d1q-d8q,m1-m8 1-8字节dump/modify的意思，实现的时候是函数指针,具体功能有先进行选择，默认是读内存。相关的命令有mems,cp0s,regs,jtagregs,spiroms
- mems:select access mem
- cp0s:select access cp0
  读cp0_config1:

```
1         cp0s 1
2         d8 16 1
```

- source file 顺序执行file中的命令，如果命令返回错误停止执行后面的命令
- Source file和source一样但fork一个进程来执行
- loop count cmd args.. 循环执行count此cmd args...
- echo args 打印args
- echo_on 回显示执行的命令
- echo_off 不回显执行的命令
- verbose [on|off|fdno] 打开或关闭debug调试，或存到fd中。
  下面命令实现打印出ejtag上执行的汇编代码

```
1         timer 0
2         verbose on
3         setconfig log.level 15
4         setconfug log.disas 1
```

- waitreg reg data 等待ejtag reg为data
- waitfacc [startaddr] [endaddr] 等代cpu访问dmseg且地址在startaddr-endaddr间或者为0xff200200
- goback [addr] 跳到dmseg addr
- run binfile 运行bin文件，如run gzrom.bin
- msleep n 等待n ms
- shell cmd 执行shell命令，或者调用脚本程序
- timer n nms检查一次ejtag状态,n==0的时候不检查

- b addr,设置软件断点，在addr上插入sdbbp
- unb addr 删除软件断点,恢复原来的指令
- hb addr [ibm] 设在硬件断点,ibm为1的位不比较地址
- unhb addr 删除硬件断点
- s单步执行
- uns取消单步执行
- cont 退出ejtag模式，继续执行
- cpu cpuno 切换cpu到cpuno
- map filemame start [size] 映射文件到dmseg start大小为size
- unmap filemame start [size] 取消映射文件到dmseg start大小为size，dmseg内容写代文件中
- memset/fmemset/smemset addr value size 在处理器上运行memset
- memcpy/fmemcpy/smemcpy saddr dassr size 在处理器上运行memcpy
- put/fput/sput file address ,上传文件到板子上,相关的配置usb_ejtag.put_speed,put.pack_size,put.md5sum
- get/fget/sget filename address size,从板子上下载文件,相关配置get.pack_size
- putelf/fputelf/sputelf elffile, 上传elf文件
- initrd initrdfile ,设置initrd
- karg "kernel args" ,设置内存参数
- disas address [count] ,反汇编
- callbin/fcallbin/scallbin binfile [arg0 arg1 ...],执行bin文件
- call/fcall/scall address [arg0 arg1 ...],调用函数
- erase 檫除增个flash芯片
- erase_area start end sectorsize 檫除部分flash芯片
- program ramaddr flashaddr size flash编程
  这些命令用来烧写flash，使用方法如下:(以后准备用callbin直接调用c语言来烧写flash)

```
1 setconfig flash.type st25vf080
2 erase_area 0 0x7ffff 0x10000
3 program 0x81000000 0 0x70000
```

- server [port] 启动命令server,另外一个ejtag-debug程序可以通过网络发命令过来执行,默认端口为8880
- shell program [args] 启动命令server同时，运行shell命令program

- gdbserver [port] 启动gdbserver,可以远程gdb remote链接调试,默认端口为50010
- gdb elffile 一个命令运行scripts/gdb.sh,实现后台启动gdbserver,前台运行gdb来调试elffile
- expr [expression|#regname]
  expr命令执行简单的+-*/\和进制转换，从左向右计算，支持括号.
  expr能读寄存器内容，主要为了方便调试用的
- newcmd cmdname oldcmd note,增加一个新的命令
- cache op addr size 发cache op从addr到addr+size
- cacheflush addr size 刷cache使得内存和cache一致，根据cache配置刷icache和dcache或者二级cache
- cache_config perl脚本读cpu的prid和config寄存器来设置cache参数，cache，cacheflush会用到这些参数
- selectcore coreno 龙芯3b选择调试的cpu,0xf是8个cpu串起来

2.4.1 callbin的实现

callbin调用的bin文件是pic位置无关代码，通过got段进行重定位。c语言的入口函数是mymain，callbin的参数直接作为mymain的参数。如果参数是字符串则callbin会将字符串地址拷贝到内存里面，并将内存地址作为参数。callbin中包含叫做tinyc的c库,实现printf,内存，字符串操作等。

实现memset的bin程序test.c

```
1 int mymain(char *buf,int len)
2 {
3         printf("this␣is␣a␣test\n");
4         memset(buf,c,len);
5         return 0;
6 }
```

编译:

```
1 make CROSS_COMPILE=mipsel-linux- test.bin
```

调用:

```
1 callbin bin/test.bin 0xa0000000 0x12
```

默认打印是通过ejtag打印到pc机端，如果要打印到串口，可以定义putchar函数:

实现memset的bin程序test.c

```
1 int putchar(c)
2 {
3  *(volatile char *)0xb4000000=c;
4 }
```

```
5  int mymain(char *buf,int len)
6  {
7          printf("this␣is␣a␣test\n");
8          memset(buf,c,len);
9          return 0;
10 }
```

callbin的从定位和clear bss在start.S和start64.S中做的。got section存的是函数的地址，要用当前的的实际地址修正函数的地址和gp的值，然后才能正确执行。编译的时候的起始地址是0，运行的时候的起始地址是helpaddr。callbin源码位于lib/callbin.c中。

2.5 ejtag-debug的脚本语言

2.5.1 source命令

- ejtag-debug的source file和Source file命令可以file文件里面的命令依次执行.
- loop命令可以实现简单的固定循环
- expr命令可以作简单计算和寄存器值作为参数

```
1  d4 ‘0xbfc00000+0x10‘ 10
2  loop 10 d4 0xbfc00000 10
```

2.5.2 外部perl脚本

本来想在内部直接集成一种脚本语言，如awk，但后来发现比较麻烦。因此采用了shell命令方法来实现脚本语言。跟busybox原理差不多. 因为ejtag-debug刚启动的时候要打开设备作初始化，如果执行一个命令打开关闭一遍设备会性能很差。这里采用server,clint的方法来解决。要执行脚本语言前现启动server,会打开端口8880来接收命令。在另一个终端象这样运行

```
1          ejtag-debug set
```

则会telnet 8880，输入命令，打印输出也重定向到网络。上面的过程可以简化成1个shell命令

```
1          shell ejtag-debug set
2          shell perl scipts/test.pl
```

perl脚本分析
scripts/io.pm是将命令封装成perl的函数

- inb就是调用d1q
- outb就是调用m1
- inb/inh/inw/inq
- outb/outh/outw/outq
- do_cmd执行命令,直接输出到终端

- do_cmd1执行命令,结果返回

要写一个perl脚本test.pl访问ejtag，只要写一个perl程序，前面包含

```
#!/usr/bin/perl
use bignum;
push @INC,qq(./scripts);
require qq(io.pm);
```

后面就可以调用inb/outb/do_cmd等函数访问ejtag了。调用的时候，运行:

```
        shell perl scipts/test.pl
```

test.pl

```
#!/usr/bin/perl
use bignum;
push @INC,qq(./scripts);
require qq(io.pm);
outb(0xffffffffbfe001e3,0x80);
outb(0xffffffffbfe001e0,0xd);
outb(0xffffffffbfe001e3,0x3);
printf "value is %x",inb(0xffffffffbfe001e3);
do_cmd("cont");
```

2.6 ejtag和gdb

为了实现通过通过gdb利用ejtag来调试程序,ejtag-debug中增加了gdbserver命令，可以直接执行gdb命令，内部调用gdbserver

2.6.1 gdb remote协议

运行info gdb命令可以查到相信的gdb remote标准。

```
        * Remote Protocol::             GDB Remote
           Serial Protocol
        * Packets:
```

运行gdb进行调试的时候，可以打开remote调试功能，能够打印出remote packet的内容

```
gdb) set debug remote -1
```

2.6.2 gdbserver的实现

- 打开tcp端口50010,设置成非阻塞方式,等待端口上的数据
- 解析packet,根据packet内容进行内存读写，设置断点等
- 查询ejtag状态,如过从正常状态进入ejtag状态，发送信号给gdb端

2.6.3 gdbserver的使用

运行gdb命令:

```
cpu0 -gdb /mnt/sdd3/work/3afirewall/linux-loongson-release/vmlinux
+ EJTAGEXE=/mnt/sdd2/work/bioscfg/ejtag_debug_pp
++ /mnt/sdd2/work/bioscfg/ejtag_debug_pp setconfig core.abisize
+ abisize=00000040
+ /mnt/sdd2/work/bioscfg/ejtag_debug_pp gdbserver 50010 1
++ mktemp /tmp/gdbXXX
+ tmpfile=/tmp/gdbDeq
+ '[' 00000040 = 00000040 ']'
+ echo 'set␣mips␣abi␣n64␣'
+ echo 'target␣remote␣:50010'
+ mipsel-gdb -q -x /tmp/gdbDeq /mnt/sdd3/work/3afirewall/linux-loongson-release/vmlinux
(no debugging symbols found)
[New Thread 1]
0xffffffff802053d4 in cpu_idle ()
(gdb) info threads
[New Thread 2]
[New Thread 3]
[New Thread 4]
  4 Thread 4  0xffffffff80205414 in cpu_idle ()
  3 Thread 3  0xffffffff802053d4 in cpu_idle ()
  2 Thread 2  0xffffffff80205414 in cpu_idle ()
* 1 Thread 1  0xffffffff802053d4 in cpu_idle ()
```

- 可以利用gdb的monitor命令从gdb端在ejtag-debug上执行命令

```
gdb)monitor setconfig
gdb)monitor d1 0xffffffffbfc00000 10
```

- 每个处理器是一个线程，info threads列出处理器信息
- detach 断开gdb remote连接
- 目前对于多cpu，进入gdb的时候只停止当前的cpu,info threads会停止所有的cpu。
- b,hb,watch,rwatch,awatch等设置断点

3 ejtag硬件

3.1 并口电缆

访问并口可以直接读取pc的io地址，或者打开并口设置/dev/parport0,通过ioctl来访问端口。
并口的pin2-pin9对应数据0-7，作为输出，11,13,15可以作为输入。

两个config和并口配置有关:

```
pp.pins                         : 0x00421730:pp pins {23..0}={rst,tclk,tms,tdo,tdi,trst}: 0x
pp.ppdev                        : 0x00000000:pp use /dev/parport0
```

3.2 usb电缆

- usb电缆包括:ezusb芯片,FPGA
- FPGA内部有一个mips处理器和一个完成一次ejtag寄存器访问的状态机,一个使用4096的fifo.
- mips处理器是李伟写的mips789 http://opencores.org/project,mips789
- usb采用bulk传输
- 除了实现对ejtag的tap寄存器访问外，还对上载和下载作了特别的优化,速度可以达到1MB/s。