

龙芯ejtag仿真器手册

龙芯中科技术有限公司

September 3, 2017

Contents

1 安装	1
1.1 windows下安装	1
1.2 Linux下安装	1
1.3 配置文件和启动参数	1
1.4 命令行参数	2
1.5 ejtag-debug参数	2
1.6 ejtag使用顺序	2
1.7 ejtag连接和速度问题	2
2 寄存器读写	3
2.1 通用寄存器读写	3
2.2 协处理器读写	3
3 ejtag寄存器读写	3
4 内存读写	4
4.1 读写内存	4
4.2 内存测试	4
4.3 保存内存到文件	4
4.4 上传文件到内存	4
4.5 上传启动elf文件	5
5 cache相关命令	5
6 flash烧写	6
6.1 配置内存参数	6
6.2 烧写flash	6
6.3 cache锁定烧flash	6
7 调试功能	6
7.1 软件指令断点	6
7.2 硬件指令断点	6
7.3 数据断点	7
7.4 单步调试	7
7.5 软单步调试	7
8 gdb调试功能	7
8.1 gdbserver功能	7
8.2 内嵌gdb功能	7
8.3 内嵌gdb模块调试功能	8
8.4 内嵌gdb扩展命令mycmd功能	8
8.5 gdb中内存访问控制	8
9 其他命令	8
10 二进制扩展功能	8

11 脚本扩展功能	9
11.1 集成简单脚本功能	9
11.2 调用外部perl脚本	11
12 eclipse安装和设置	12
12.1 linux下安装	12
12.2 windows下安装	12
13 ejtag自动调用eclipse调试	12
14 手动设置eclipse调试	12
15 利用vxworks开发环境workbench通过ejtag来调试	15

Abstract

EJTAG是mips的onchip debug调试标准。现在龙芯1号、龙芯2号(龙芯2F和以前版本不支持)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 安装

软件下载地址为<http://ftp.loongnix.org/embedd/ls1b/ejtag/>。

其中ejtag-debug-xxx.tar.gz是linux版本。ejtag-debug-cygwin-xxx.rar为windows版本。请下载安装最新版本使用。

1.1 windows下安装

- 龙芯ejtag在windows下使用需要安装驱动，如果windows下没有安装驱动执行ejtag_debug_usb.exe会提示缺少libusb库，或者找不到设备。驱动按照过程如下：
 1. 首先将usb电缆插入pc机的usb口，这个时候windows会检测出未知的usb设备插入提示按照驱动
 2. 选择手动安装，并指定安装目录为ejtag-debug/driver目录，然后下一步来自动安装驱动
 3. 还可以在设备管理器里面找为未安装驱动的usb设备，vid: 2961,pid: 6688，然后指定按照ejtag-debug/driver目录里面的驱动即可。

驱动安装完后，直接双击ejtag_debug_usb.exe执行就可以了。

- ejtag程序的一些命令会调到脚本，需要安装perl。可以下载active perl或者strawberry perl都可以<http://www.perl.org/get.html#win32>。
- 如果在cygwin下执行，需要将ejtag-debug里面的cygwin1.dll删除，否则程序会自动退出去。

1.2 Linux下安装

在linux下不需要安装驱动，直接以超级用户权限执行ejtag_debug_usb.exe即可。

1.3 配置文件和启动参数

配置文件是ejtag.cfg 程序会自动打开./ejtag.cfg并执行里面的内容。因为本软件同时支持龙芯1、2、3三个系列的处理器，在运行之前需要根据你要调试的处理器类型对ejtag.cfg进行配置。

- setconfig core.cpubcount 1 设置处理器核的个数，如龙芯3A有4个处理器核，应该设置成4，龙芯3B设置成8，龙芯2H、龙芯1A、1B、1C、1D设置成1。
- setconfig core.cpuwidth 32 设置处理器寄存器宽度，如龙芯3A是64位处理器，应该设置成64，龙芯3B、龙芯2H设置成64，龙芯1A、1B、1C、1D设置成32。
- usblooptest设置usb ejtag的分频和位相选择，当其他都配置正确ejtag还不正常工作时，可以尝试对ejtag时钟降频

```

1 usblooptest help
2 usblooptest 0x81000070 {0x10000|divison} #TCLK freq divided by division, divison must only one bit
   set
3 usblooptest 0x81000070 {0x20000|samplesel} #sample tdi output at the samplesel clk after rising
   edge of tck, samplesel value from 0 to 3

```

简单写法jtag_clk divison [samplesel]

- selectcore coreid
selectcore命令用于龙芯3b/3c选择多核ejtag的串接方式0-7表示调试单个核，-1表示8个核串到一起。

其他和功能相关的一些设置：

- 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
- usb_ejtag.put_speed 范围0 0xffff，调节ejtag的usb读取速度，设置越大速度越慢，当主机速度较慢的时候需要配置延迟大一些
- usb.maxtimeout usb访问的最大超时时间，如果usb_ejtag.put_speed设置较大，usb.maxtimeout也要设置更大一些

- `putelf.uncached`: 设置0 `putelf`、`program`命令上传数据到`cache`地址, 然后刷`cache`, 1在`put`、`program`先刷`cache`然后上传到`uncache`地址, 2`put`、`program`上传到`uncache`地址, 不刷`cache`. 如果设置为0, 1一定要保证`cache`被初始化, 如果`cache`没被初始化运行`cache_config`和`cache_int`命令
- `timer 1000`: 设置`timer 1000ms`检查一次`ejtag`状态, 如果`usb`速度慢设置更大一些, `timer 0`关闭`timer`检测`ejtag`状态功能

1.4 命令行参数

`./ejtag_debug_usb`

1.5 ejtag-debug参数

```
./ejtag_debug_usb [-d|Stch] [-e cmd] [-T n]
-d: verbose on, show debug messages
-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
```

1.6 ejtag使用顺序

1. `ejtag.cfg`里面要设置正确`core.cpuscount`, `core.cpuwidth`
2. 主板先上电
3. `ejtag usb`端插入
4. `ejtag` 插头插到主板的`ejtag`座上, 注意三角形(1脚)对主板上的三脚形(1脚)
5. 运行`sudo ./ejtag_debug_usb -t`来执行`ejtag`软件
6. 如果`ejtag`已经插在座上, 处理器下电了, 处理器再上电后需要在`ejtag`软件里面允许

```
jtagled trst:0 trst:1
```

来对处理器发出一个`trst`复位操作

1.7 ejtag连接和速度问题

- 将`ejtag`插入`usb`口可以观察到`usb ejtag`的两个灯开始都亮, 1s后只有一个灯亮。这说明`usb ejtag`硬件没有问题。
- `sudo ./ejtag_debug_usb -t`运行`ejtag`软件, `jtagled 1`命令灯亮, `jtagled 0`命令灯灭。说明`usb`驱动安装正常。
- `usbver` 看返回是否是一个逻辑日期, 日期是2013年后方可使用
- 然后将`ejtag`插头插到主办上, 注意1脚对1脚, `ejtag.cfg`里面的`cache_config`注释掉
- 运行`sudo ./ejtag_debug_usb -t`, 运行`jtagregs d8 1 1`来读处理器的`ejtag id`寄存器, 如果是`0x20010819`或者是`0x5a5a5a5a`都说明连接正确
- 运行`set`命令读处理器的通用寄存器, 如果能读出来且非全0, 则说明处理器运行起来了, `ejtag`也连接上了
- 如果读不出来, 按`ctrl-c`退出。可能是处理器在无程序的情况下运行到地址空洞, 设备没响应, 总线卡住了。可以运行`resetcpu`命令来复位`cpu`, 然后按`set`就能读出通用寄存器内容了。

- 在进行put, get, program命令的时候, 如果是虚拟机, 虚拟usb速度很慢, 响应的龙芯处理器要等更长时间接usb数据, setconfig usb_ejtag.put_speed要设置更大一下, 同setconfig usb.maxtimeout也要设置更大一些。
- 在进行put, get, program命令的时候, 可能会刷cache, 一定要保证cache已经初始化过了, 或者设置setconfig putelf.uncached 2, 或者运行cache_config, cache_init先对cache进行初始化。

2 寄存器读写

2.1 通用寄存器读写

- 读通用寄存器: set [寄存器名]
- 写通用寄存器: set [寄存器名] [数值]
- save [file]: 保存通用寄存器内容到文件/临时内存中
- restore [file]: 恢复通用寄存器内容来自于文件/临时内存中

为了方面脚本软件调用, 也可以用下面的方法访问

```
1 cpuregs
2 d4 1 2
3 m4 1 0x100或者
4
5 cpuregs d4 1 2
6 cpuregs m4 1 0x100
```

cpuregs表示设置d1、d2、d4、d8为1、2、4、8字节寄存器读功能, m1、m2、m4、m8为1、2、4、8字节寄存器写功能。

d4 1 2表示读寄存器1开始的两个寄存器也就是at和v0寄存器。m4 1 0x100描述写寄存器1为0x100, 也就是设置at寄存器为0x100。

2.2 协处理器读写

- 选择协处理器组为sel, 默认为0: cp0s [sel]
- 协处理器读: d4 regno [count] 或者 cp0s sel d4 regno [count]
- 写处理器写: m4 regno value 或者 cp0s sel m4 regno value

例子

```
1 读cp0_config0: cp0s 0 d4 16 1
2 读cp0_config1: cp0s 1 d4 16 1
3 写为cp0_config02: cp0s 0 m4 16 2
```

3 ejtag寄存器读写

- 选择ejtag寄存器读写功能: jtagregs
- ejtag寄存器读: d4 regno [count] 或者 jtagregs d4 regno [count]
- ejtag寄存器写: m4 regno value 或者 jtagregs d4 regno value

例子

```
1 读ejtag id寄存器: jtagregs d4 1 1
2 写ejtag control寄存器为0x1000: jtagregs m4 10 0x1000
```

4 内存读写

4.1 读写内存

- 选择内存读写功能: `mems`
- 读内存: `d4 addr [count]` 或者 `mems d4 addr [count]`
- 写内存: `m4 addr value` 或者 `mems m4 addr value`
- 反汇编: `disas addr [count]`: 反汇编`addr`开始`count`个指令
- `[s]memcpy[1/2/4] src dst size`: 拷贝内存从`src`到`dst`, 命令包括`smemcpy1`, `fmemcpy1`, `memcpy1`等等。`smemcpy[1/2/4]`不用`helpaddr`速度慢, `memcpy[1/2/4]`跳到`helpaddr`中执行速度快。
- `[s]memset[1/2/4] addr c size`: 设置`size`大小内存内容为`c`, 命令包括 `smemset1`, `fmemset1`, `memset1`等等。`smemset[1/2/4]`不用`helpaddr`速度慢, `memset[1/2/4]`跳到`helpaddr`中执行速度快。

例子

- 1 读内存地址`0xbf000000` `0x100`个4字节: `mems d4 0xbf000000 0x100`
- 2 写内存地址`0xa0000000`为`0x1000`: `mems m4 0xa0000000 0x1000`

4.2 内存测试

- 测试内存从`startaddr`到`endaddr`读写正确性, 方式是写0, 写-1然后读回看是否相等: `memtest startaddr endaddr`
- 测试内存从`startaddr`到`endaddr`读写正确性, 方式是从`startaddr`到`endaddr`写入数值, 开始是`initval`, 每次地址数据增加`incval`, 全写完一遍后, 再读出比较正确性: `memtest1 startaddr endaddr initval incval`

4.3 保存内存到文件

- 直接下载`address`地址开始`size`大小内存到`filename`中: `sget filename address size`
- 通过`helpaddr`里面的下载程序, 下载`address`地址开始`size`大小到`filename`中: `fget filename address size`
- 通过`helpaddr`里面的下载程序, 快速下载`address`地址开始`size`大小到`filename`中: `get filename address size`

`helpaddr`是一端可写内存, `fget`和`get`命令会先将一小段下载程序上载到`helpaddr`中, 然后执行这段小程序来帮助下载。

4.4 上传文件到内存

- 直接上载文件到`address`地址开始内存中: `put filename address`
- 通过`helpaddr`里面的下载程序, 上载文件到`address`地址开始内存中: `fput filename address size`
- 通过`helpaddr`里面的下载程序, 快速上载文件到`address`地址开始内存中: `put filename address`

`helpaddr`是一端可写内存, `fput`、`put`和`putelf`命令会先将一小段下载程序上载到`helpaddr`中, 然后执行这段小程序来帮助下载。

4.5 上传启动elf文件

另外ejtag还支持上传elf文件要加载的内容到内存对应的命令是sputelf/fputelf/putelf elffile。

- putelf elffile
上传elf文件到内存，地址和入口信息都从elf文件获得并设置pc, sputelf/fputelf/putelf底层分别调用sput/fput/put
- initrd initrdfile [addr]
initrd命令上载ramdisk文件到addr, addr默认是0x84000000, initrd的内存地址和大小被保存在config设置karg.rd_start, karg.rd_size文件中。
initrd可以这样生成

```
1 cd ramdisk
2 find . |cpio -o -H newc|gzip -c > initrd.gz
```

- karg arg0 arg1 arg2 ...
karg命令将内核参数arg0 arg1 ... 和所有ENV_开始的环境变量转化成内核参数和环境变量格式写入setconfig karg.bootparam_addr设置的地址里面(默认是0xa4000000)。karg会自动根据initrd的信息设置rd_start, rd_size内核参数。因此initrd命令要在karg命令前运行。
- pmon的环境变量xxx的数值yyy通过程序的ENV_xxx环境变量传递到内核里面，程序的环境变量可以通过setenv命令设置，也就是说通过setenv ENV_xxx yyy设置，比较重要的环境有cpuclock, memsize, highmemsize, 设置方法如下

```
1 setenv ENV_memsize 256
2 setenv ENV_highmemsize 0
3 setenv ENV_cpuclock 266000000
```

分别是设置内核内存大小和cpu的时钟。

龙芯1号系列一般上传内核命令如下:

```
1 putelf vmlinux
2 karg console=ttyS0,115200 rdinit=/sbin/init initcall_debug=1
3 cont
```

龙芯2, 3号系列一般上传内核命令如下:

```
1 put vmlinux 0xffffffff84000000
2 #上pmon
3 load /dev/ram@0x84000000
4 g console=ttyS0,115200 rdinit=/sbin/init initcall_debug=1
5 cont
```

5 cache相关命令

- cache_config : cache大小自动获取
- cache_init : cache初始化
- cacheflush addr size : 刷新数据和指令cache从地址addr开始, 大小为size。当size大于config cacheflush.nohelp_size用helpaddr
- cache op addr size : 刷cache op操作到开始于addr, size大小内存。当size大于config cacheflush.nohelp_size用helpaddr
- cachel op addr size : 刷cache op操作到开始于addr, size大小内存, 在helpaddr上执行速度快

6 flash烧写

6.1 配置内存参数

- 龙芯1A: `source configs/config.lsla;call configddr;`
- 龙芯1B: `source configs/config.lslb;call configddr;`
- 龙芯3A: `source configs/config.ls3a;call configddr;`
- 龙芯2H: `source configs/config.ls2h;call set_ddr_pll;call configddr;`

6.2 烧写flash

- 龙芯1A: `source configs/config.lsla;call configddr;call erase;call program [gzrom.bin]`
- 龙芯1B: `source configs/config.lslb;call configddr;call erase;call program [gzrom.bin]`
- 龙芯3A: `source configs/config.ls3a;call configddr;call erase;call program [gzrom.bin]`
- 龙芯2H: `source configs/config.ls2h;call set_ddr_pll;call configddr;call erase;call program [gzrom.bin]`

6.3 cache锁定烧flash

龙芯232, 264, 464 ip都支持cachelock功能, 因此可以利用cachelock来做flash烧写而不使用内存。

- 龙芯1A: `source configs/config.lsla;call program_cachelock [gzrom.bin]`
- 龙芯1B: `source configs/config.lslb;call program_cachelock [gzrom.bin]`
- 龙芯3A: `source configs/config.ls3a;call program_cachelock [gzrom.bin]`
- 龙芯3A2000: `source configs/config.ls3a2000;call program_cachelock [gzrom.bin]`
- 龙芯2H: `source configs/config.ls2h;call program_cachelock [gzrom.bin]`

7 调试功能

7.1 软件指令断点

- 设置软件指令断点到addr: `b addr`
- 删除addr上的指令断点: `unb addr`

软件断点触发的时候, 处理器会停止进入ejtag调试状态。这个时候, 当timer非0的时候, ejtag软件会自动打印出断点信息和指令内容。

7.2 硬件指令断点

- 设置软件指令断点到addr: `hb addr [ibm]`
- 删除addr上的指令断点: `unhb addr [ibm]`

硬件断点触发的时候, 处理器会停止进入ejtag调试状态。这个时候, 当timer非0的时候, ejtag软件会自动打印出断点信息和指令内容。

其中ibm是instruct address bit mask, 硬件判断地址相等的方法是`pc&~ibm == addr&~ibm`, 也就是ibm为1的位不比较地址。

7.3 数据断点

- 设置写数据到addr触发断点: `watch addr [dbm]`
- 设置读数据到addr触发断点: `rwatch addr [dbm]`
- 设置读写数据addr均触发断点: `awatch addr [dbm]`
- 删除addr的数据断点: `unwatch addr`

硬件断点触发的时候, 处理器会停止进入ejtag调试状态。这个时候, 当timer非0的时候, ejtag软件会自动打印出断点信息和指令内容。

其中dbm是data address bit mask, 硬件判断地址相等的方法是`load store address & ~dbm == addr & ~dbm`, 也就是ibm为1的位不比较地址。

7.4 单步调试

- 单步count次: `si [count]`
- 取消单步: `unsi`

当timer非0的时候, ejtag软件会自动打印出单步发生的时候的信息和指令内容。

7.5 软单步调试

龙芯2h, 3a不支持单步。可以通过设置硬件断点, 软件断点进行单步。

- 硬件断点软单步: `si.h [count]`
- 软件断点软单步: `si.s [count]`

以上命令都不需要删除断点。

8 gdb调试功能

8.1 gdbserver功能

- 启动gdbserver: `gdbserver [port]`
- 启动gdbserver并自动配置: `call gdbserver`

调试32位处理器

```
1 mipsel-gdb elffile
2 gdb) set architecture mips:isa32
3 gdb) set mips abi o32
4 gdb) target remote :port
```

调试64位处理器

```
1 mipsel-gdb elffile
2 gdb) set architecture mips:isa64
3 gdb) set mips abi n64
4 gdb) target remote :port
```

8.2 内嵌gdb功能

- 内嵌gdb调试程序:

```
1 gdb elffile
```

需要设置好core.abisize

内嵌gdb将gdbserver和gdb功能集成到一起, 操作方式和普通gdb完全一样。执行的gdb命令默认运行`./mipsel-gdb`, 可以设置环境变量GDB来设置其他gdb, 建议使用`gdb-multiarch`。

8.3 内嵌gdb模块调试功能

将模块ko文件拷贝到ejtag的modules目录中。

```
1 gdbmod elffile
2 gdb)mycmd modules
```

gdbmod执行gdb调试，gdb中执行modules命令自动load模块调试信息。gdbmod命令会新开启一个gdb-server，将mmaped address通过页表转换成unmaped地址访问。

8.4 内嵌gdb扩展命令mycmd功能

gdbmod命令启动gdb后会，或者gdb启动gdb，然后运行source scripts/gdb1.py后。会在gdb中增加mycmd一组命令。这组命令是用来帮助调试linux用户程序，分析进行，vm行为的。输入mycmd后按tab键可以提示出命令。

该组命令需要内核有完整的调试信息。

8.5 gdb中内存访问控制

- 通过tlb访问mmaped address, 默认关闭，通过gdbserver.helpaccess配置打开

```
1 setconfig gdbserver.helpaccess 1
```

- gdbmap命令进行地址转换和加快访问

```
1 gdbmap add addr len [file|transaddr] [rw] |gdbmap del addr len |gdbmap:set remote address map for
gdb
```

9 其他命令

- cont
cont命令让程序推出ejtag debug状态，继续执行
- resetcpu [arg0] ...
无参数的时候，通过写0x11000到ejtag控制寄存器来复位处理器并进入debug状态, 带参数的时候将参数直接写到control 寄存器里面，如resetcpu 0x10000 0可以使处理器复位后继续执行。
- cpus [count] [file] 扫描每个处理器的asid和pc，格式是低32位是pc，高8位asid。当存文件时，只保存pc数值。这个命令不影响处理器的执行，但需要处理器支持pc sample
- sample [count] [file] 通过触发ejtag异常来获得pc。这个命令会不断中断处理器执行，处理器执行会变慢。

10 二进制扩展功能

- 在ejtag地址空间执行bin程序: scallbin bin/xxx.bin
- 在内存地址中执行bin程序，普通加载，程序放在helpaddr开始的一段区域中: fcallbin bin/xxx.bin
- 在内存地址中执行bin程序，快速加载，程序放在helpaddr开始的一段区域中: callbin bin/xxx.bin

在ejtag调试程序的bin目录下，集成了一个ejtag小系统，可以直接调用c库函数，printf会直接在ejtag界面打印信息。

实现memset的bin程序test.c

```
1 int mymain(char *buf, int len)
2 {
3     int argc = *(int *)ARGC_REG;
4     printf(" this is a test\n");
5     if(argc>=2)
6         memset(buf, c, len);
7     return 0;
8 }
```

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 }
```

11 脚本扩展功能

11.1 集成简单脚本功能

- local定义局部变量

定义3个局部变量abc

```
1 local a b c
```

- let对变量赋值, 支持同时对多个变量复制, 如果首先在函数内找局部变量, 否则找全局变量, 否这设置环境变量

定义变量abc并分别设置为123

```
1 local a b c
2 let a b c 1 2 3
```

- unset删除变量, 可以同时删除多个变量
- letl对局部变量赋值, 如果局部变量不存在创建这个局部变量, 支持同时对多个变量复制
- unsetl删除局部变量, 可以同时删除多个局部变量

定义局部变量abc并分别设置为123

```
1 letl a b c 1 2 3
```

- expr 表达式计算
- expr1 表达式计算, 同shell里面的expr命令
- test 判断文件和表达式, 和shell命令test一样
- \$(cmd) 命令结果替换
- \${+cmd} 命令结果替换, 但fork子进程
- {cmd} 等价于\$(expr cmd)
- if value cmd
- while value cmd

- while循环

```

1 do while value
2 cmd1
3 cmd2
4 loop_break [n]
5 loop_continue [n]
6 end

```

- for循环

```

1 for letl i 1;$(expr $i<100);letl i $(expr $i+1)
2 cmd1
3 loop_break [n]
4 loop_continue [n]
5 end

```

- if, elsif, else

```

1 do if value
2 cmd1
3 elsif vall
4 cmd2
5 else
6 cmd3
7 loop_break [n]
8 loop_continue [n]
9 end

```

- function 函数

```

1 function fname
2 cmd1
3 cmd2
4 cmd3 $1 $2 $3
5 ret [val]

```

- call fname [arg1] [arg2...] :调用function fname
- source命令调用命令脚本: source cmdfile
- 添加脚本成为新命令: newcmd cmdname oldcmd

简单脚本类子

```

1 function sum
2 local a i
3 let a i 0 0
4 do while $(expr $i<=$1)
5 let a $(expr $a+$i)
6 let i $(expr $i+1)
7 end
8 echo $a
9 ret $a
10
11 function sum1
12 for letl a i 0 0;$(expr $i<=$1);letl i $(expr $i+1)
13 let a $(expr $a+$i)
14 end
15 echo $a
16 ret $a
17
18 call sum 100
19 call sum1 100
20 echo $?

```

简单脚本类子

```

1 function fib
2 local i r
3 let i $1
4 do if $(expr $i==0)
5 let r 0
6 elsif $(expr $i==1)
7 let r 1
8 else
9 let r $(expr $(call fib $(expr $i-1))+$(call fib $(expr $i-2)))
10 end
11 echo $r
12 ret $r
13
14 function fibl
15 local i n r0 r1
16 letl i r0 r $1 0 1
17
18 do if $(expr $i==0)
19 let r 0
20 elsif $(expr $i==1)
21 let r 1
22 else
23 for letl n 2;$(expr $n<=$i);letl n $(expr $n+1)
24 letl r0 r $r $(expr $r+$r0)
25 end
26 end
27 expr %lld $r
28 ret $r
29
30 call fib 5
31 echo $?

```

11.2 调用外部perl脚本

调用外部perl脚本，执行复杂命令：shell perl scripts/xxx.pl

perl脚本里面包含scripts/io.pm，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

```

1 #!/usr/bin/perl
2 use bignum;
3 push @INC, qq(./scripts);
4 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：

```

1 shell perl scripts/test.pl

```

test.pl

```

1 #!/usr/bin/perl
2 use bignum;
3 push @INC, qq(./scripts);

```

```

4 require qq(io.pm);
5 outb(0xffffffffbfe001e3, 0x80);
6 outb(0xffffffffbfe001e0, 0xd);
7 outb(0xffffffffbfe001e3, 0x3);
8 printf " value is %x", inb(0xffffffffbfe001e3);
9 do_cmd(" cont");

```

ejtag还支持python, shell脚本等, 参考scripts/testpython.py, scripts/testshell.sh.

12 eclipse安装和设置

12.1 linux下安装

1. 解压eclipse-cpp-juno-SR2-linux-gtk.tar.gz, 并运行eclipse即可 如果eclipse提示, 没有安装gdk, 继续步骤 2
2. jdk-7u21-linux-i586.tar.gz

12.2 windows下安装

到eclipse官网下载对应的eclipse-cdt发行版解压运行即可。

13 ejtag自动调用eclipse调试

ejtag目录下workspace目录是预先配置好eclipse配置。下面命令自动将workspace拷贝到workspace1并自动修改配置, 启动workspace调试。

1. eclipse file
该命令自动启动gdbserver, 并执行eclipse。通过预先设置的debug和attach配置都可以连接到正在硬件, 调试file文件。
2. eclipse_remote file ip:port 该执行eclipse。通过预先设置的debug和attach配置都可以连接到gdbserver ip:port, 调试file文件。

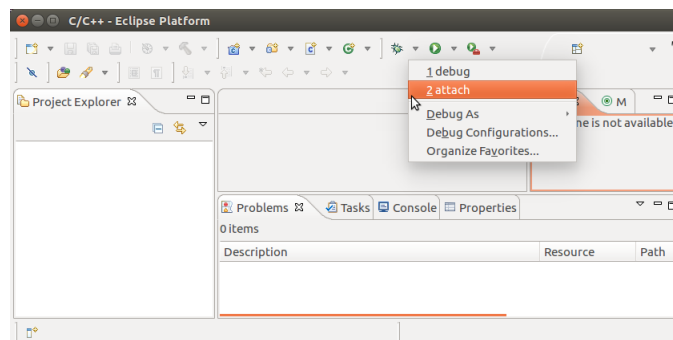


Figure 1: ejtag自动启动eclipse

14 手动设置eclipse调试

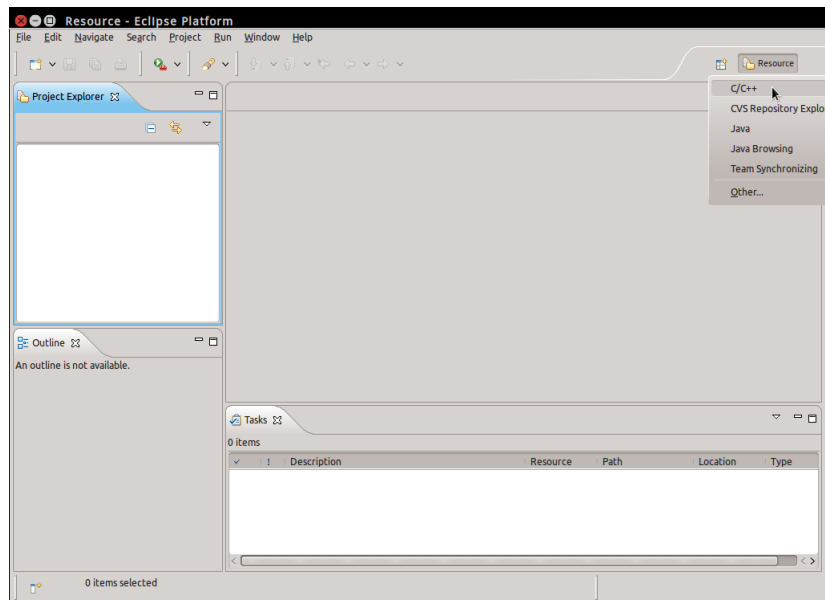


Figure 2: 选择调试c, c++

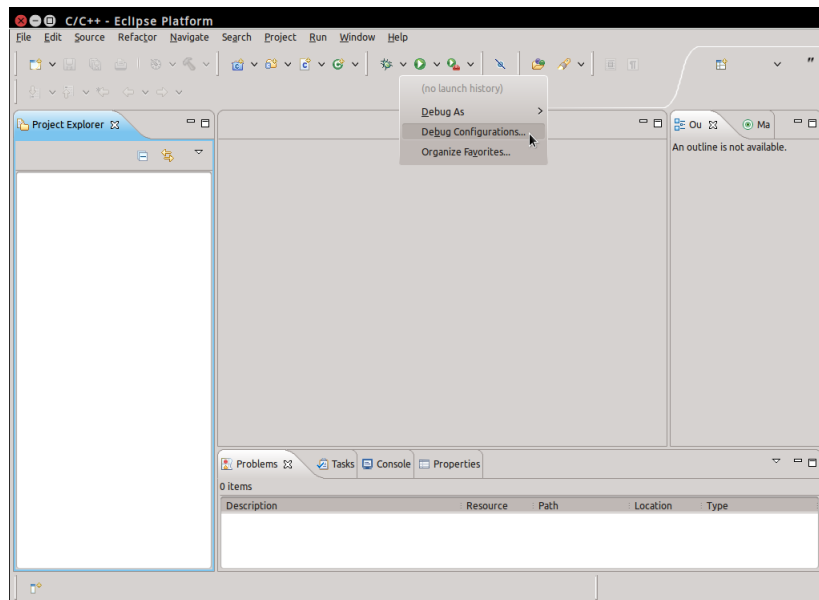


Figure 3: 选择调试c++ debug configure

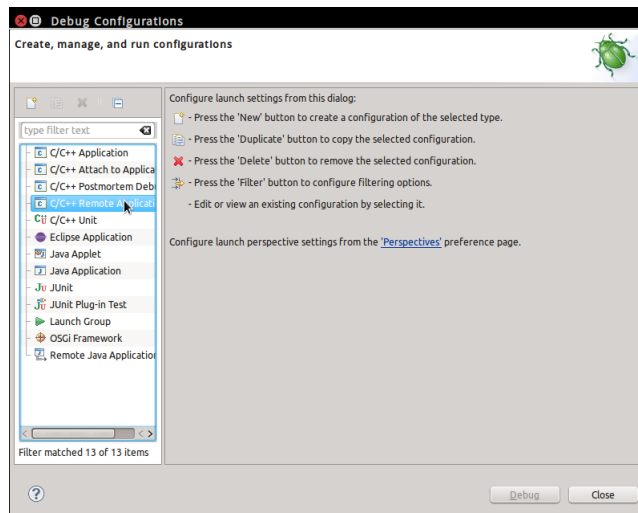


Figure 4: 选择调试c++ remote debug configure

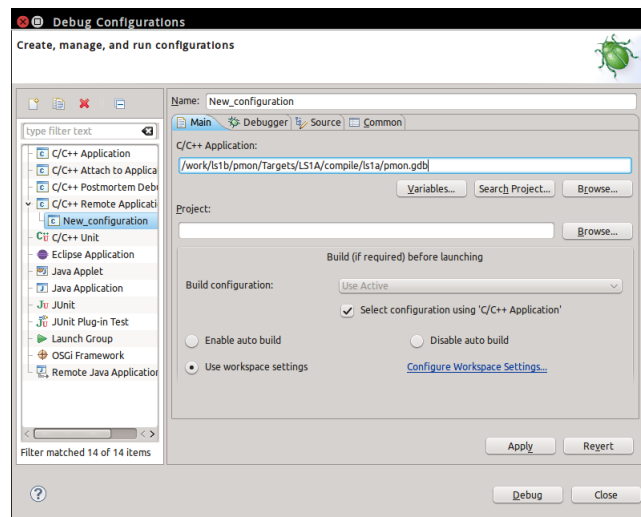


Figure 5: 选择调试c++ debug configure main

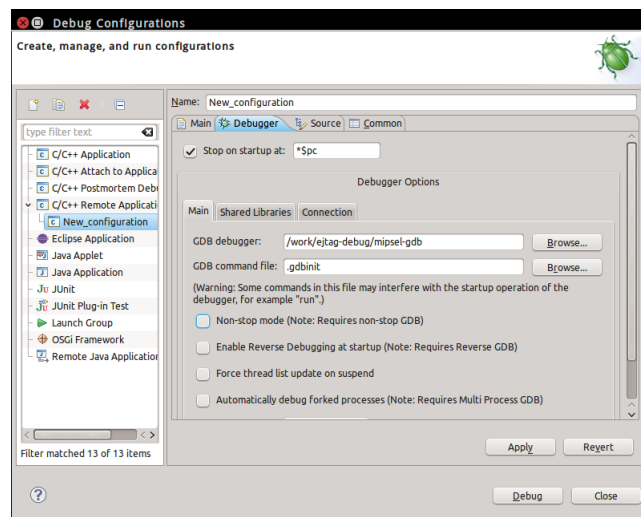


Figure 6: 选择调试c++ debugger为mipsel-gdb
break at 可以设置成*\$pc, 这样可以停在调试开始的地方。

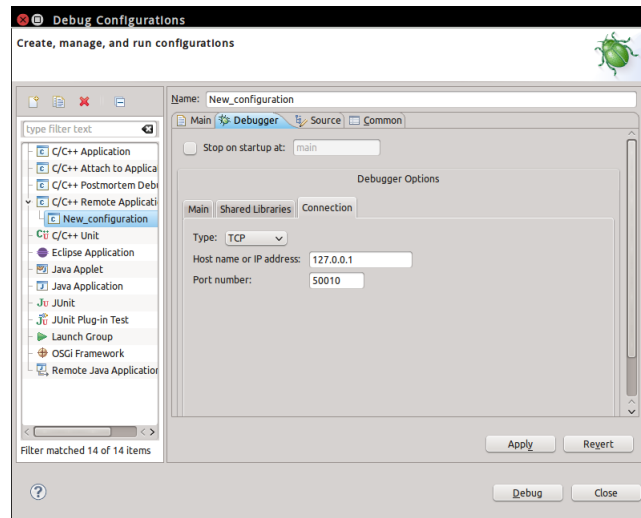


Figure 7: 选择调试c++ debug connection 127.0.0.1:50010

15 利用vxworks开发环境workbench通过jtag来调试

- 下拉debug图标，选择debug configure

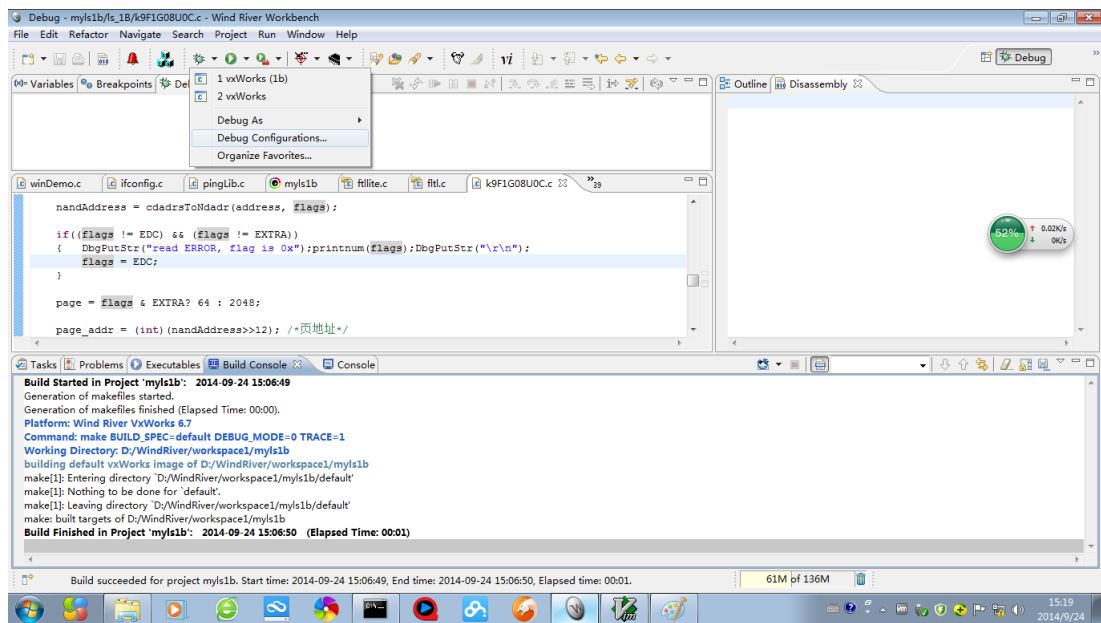


Figure 8: 选择调试c++ debug configure

- 双击 C++ local Application, 新建一项, 并设置project和调试的程序

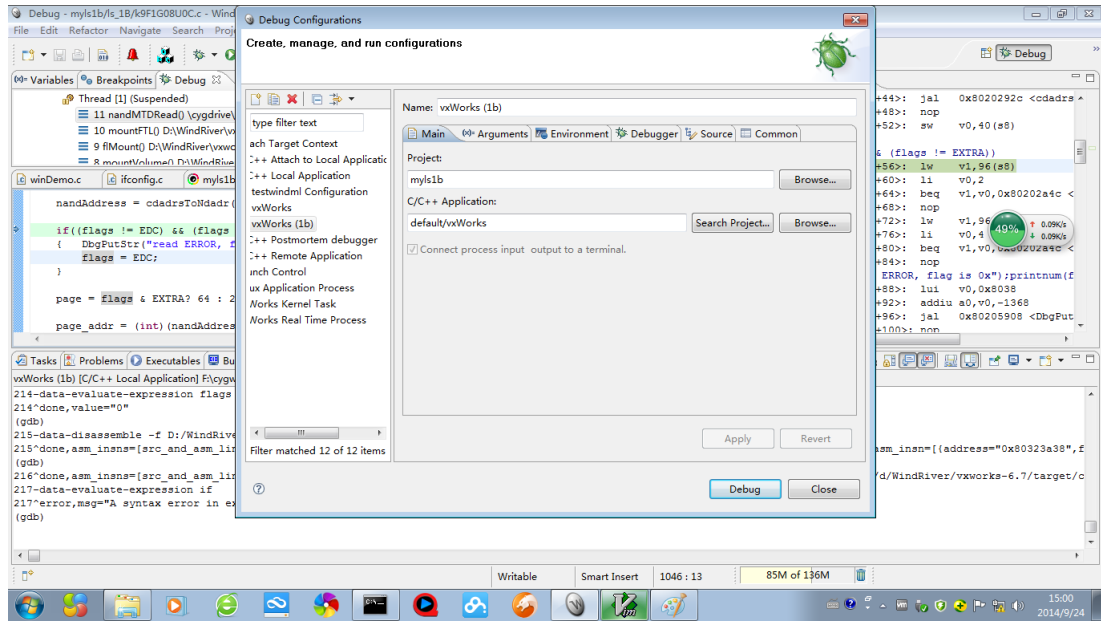


Figure 9: 双击local Application

- 设置debuger, 选则gdbserver debugger, 交叉gdb程序可以用ejtag程序里面的mipsel-gdb.exe

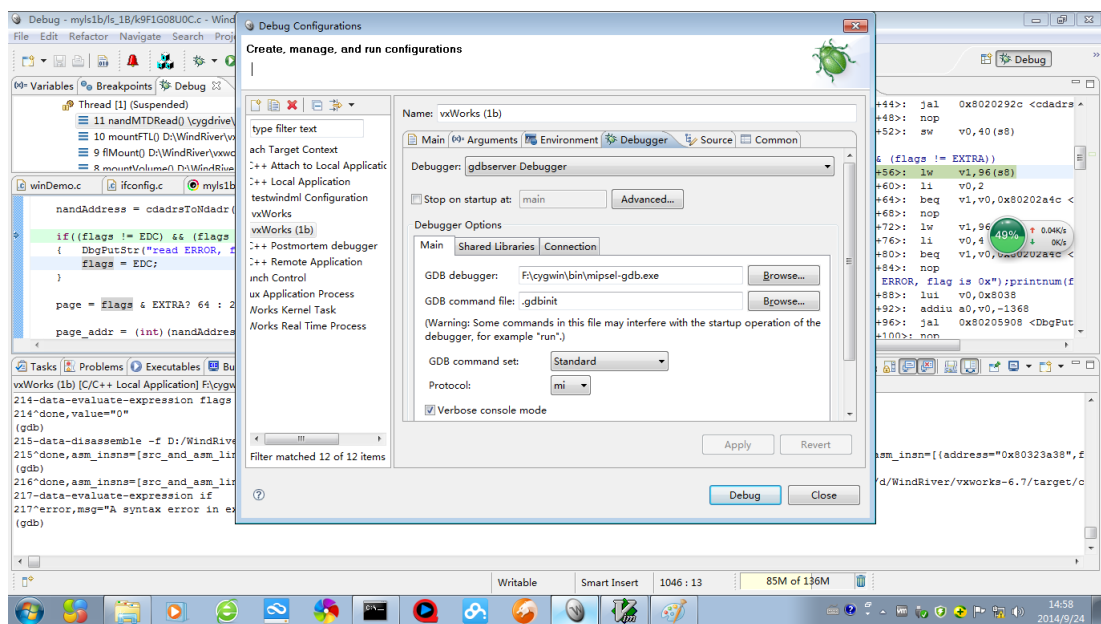


Figure 10: 设置debuger, 选择gdbserver debugger

- 设置connection, server ip设置成ejtag-debug程序运行机器的ip, 端口号默认设置成50010

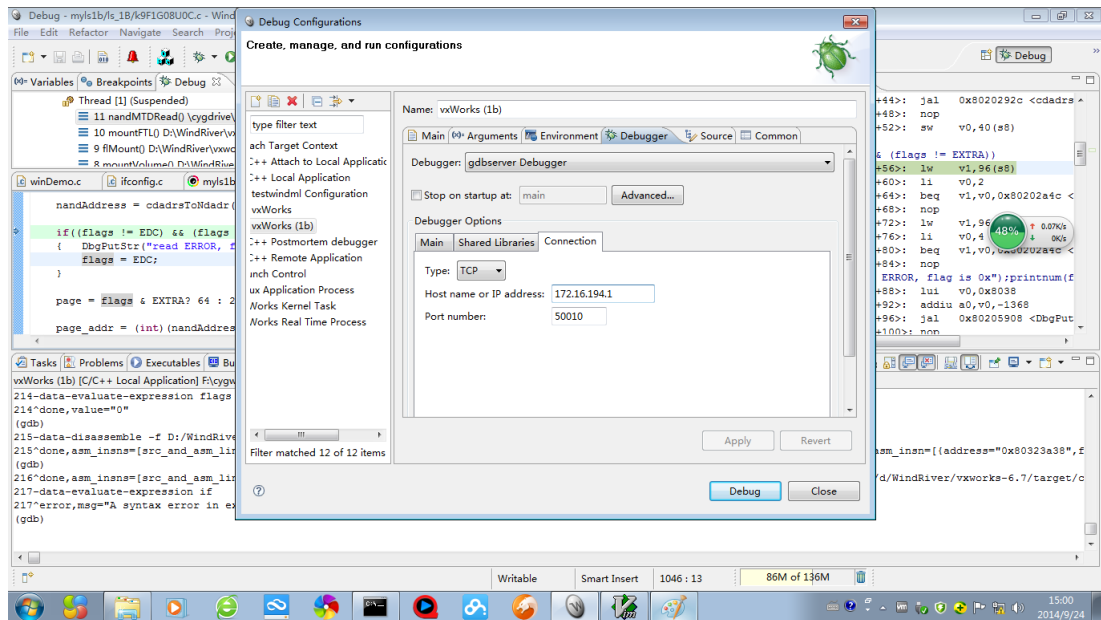


Figure 11: 设置gdbserver ip, port