

# JLink调试指导说明

---

## JLink仿真器简介

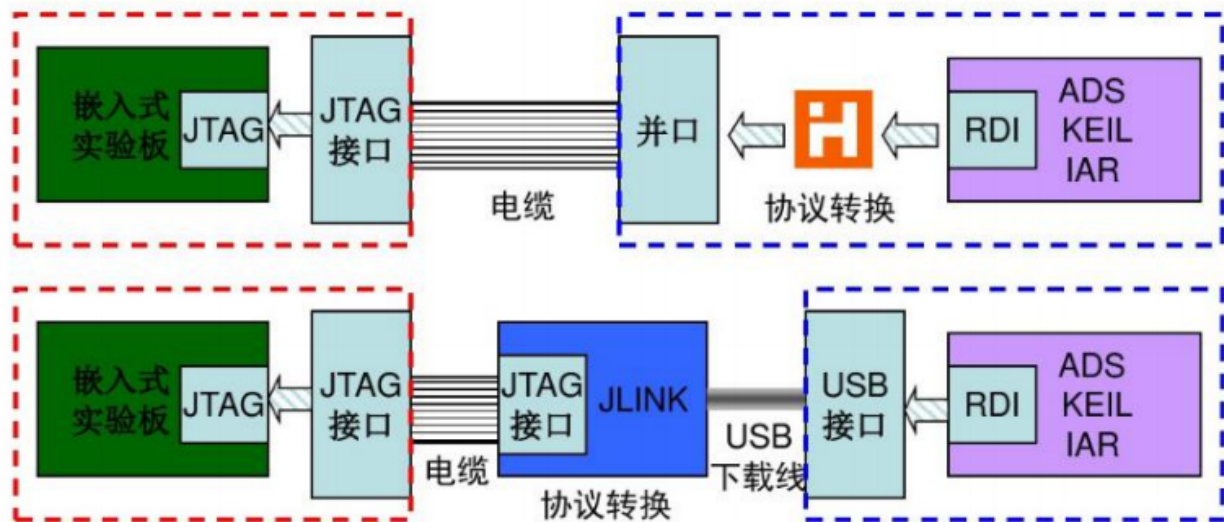
---

J-Link是德国SEGGER公司推出基于JTAG的仿真器。简单地说，J-Link是一个小型USB到JTAG协议转换盒。其连接到计算机用的是USB接口，而到目标板内部用的还是jtag协议。它完成了一个从软件到硬件转换的工作。



JLink原理

# This is j-link



JTAG: 国际标准测试协议  
RDI: ARM公司提出的调试接口标准

## JLink工作环境介绍

需要连接以下引脚

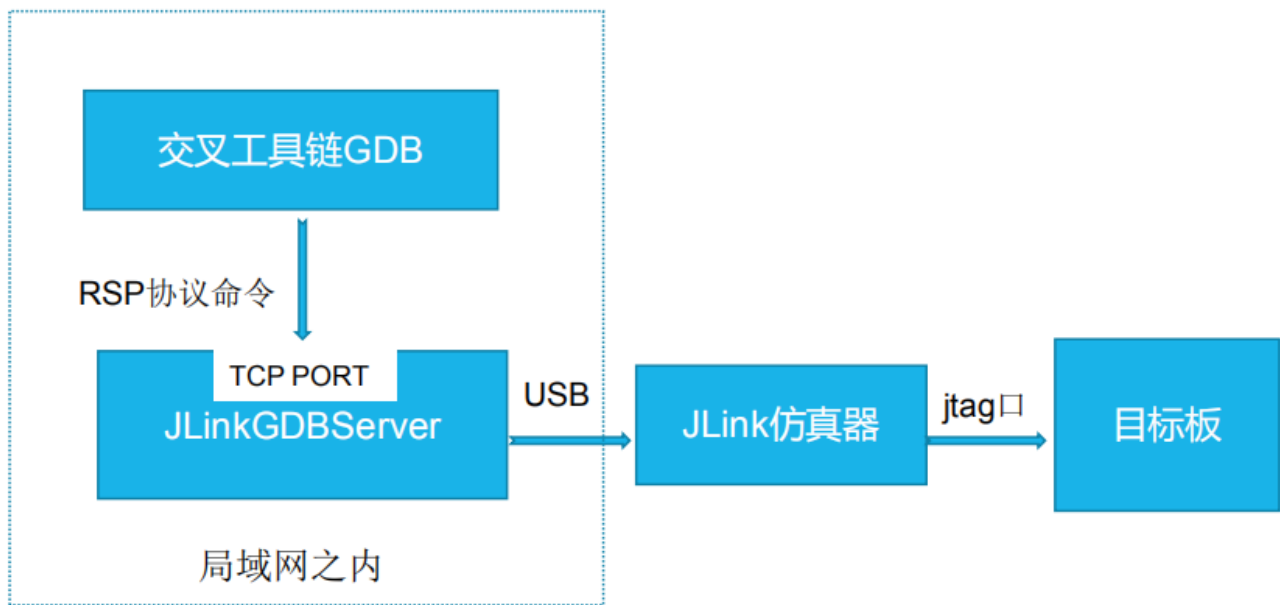
信号名	方向	作用
TCK	输入	为测试逻辑提供时钟
TDI	输入	接收测试指令和数据
TMS	输入	用于TAP控制器控制测试和操作
TDO	输出	串行输出到测试逻辑
GND	输入	地
nTRST	输入 ( 可选 )	用于TAP控制器的异步初始化

VTref	1 ●	● 2	NC
nTRST	3 ●	● 4	GND
TDI	5 ●	● 6	GND
TMS	7 ●	● 8	GND
TCK	9 ●	● 10	GND
RTCK	11 ●	● 12	GND
TDO	13 ●	● 14	GND
RESET	15 ●	● 16	GND
DBGREQ	17 ●	● 18	GND
5V-Supply	19 ●	● 20	GND

## JLinkGDBServer介绍

JLinkGDBServer作用：

- 1.解析GDB发送的Remote Serial Protocol ( RSP ) 协议命令
- 2.输出JLink仿真器控制信号



JLinkGDBServer运行在host-PC上，它并不是gdbserver。target-board上也没有运行gdbserver。所以gdbserver的很多功能，JLinkGDBServer是没有的，不要把JLinkGDBServer当做gdbserver使用。

## 搭建 JLink 调试环境

### Ubuntu环境

#### step 1

前往 [https://www.segger.com/downloads/jlink/JLink\\_Linux\\_x86\\_64.deb](https://www.segger.com/downloads/jlink/JLink_Linux_x86_64.deb) 下载jlink for linux的软件包，如在ubuntu上使用，则下载deb软件包，然后使用 `sudo dpkg -i JLink_Linux_V664b_x86_64.deb` 命令来安装

#### step 2

在终端执行 `JLinkGDBServer -device Cortex-A7` 命令，启动JLinkGDBServer服务，等待GDB远程连接

```
wuhuating@PCwuhuating:~$ JLinkGDBServer -device Cortex-A7
SEGGER J-Link GDB Server V6.64b Command Line Version

JLinkARM.dll V6.64b (DLL compiled Mar 20 2020 10:08:28)

Command line: -device Cortex-A7
-----GDB Server start settings-----
GDBInit file:                none
GDB Server Listening port:    2331
SWO raw output listening port: 2332
Terminal I/O port:          2333
Accept remote connection:    yes
Generate logfile:            off
Verify download:             off
Init regs on start:          off
Silent mode:                 off
Single run mode:             off
Target connection timeout:    0 ms
-----J-Link related settings-----
J-Link Host interface:       USB
J-Link script:               none
J-Link settings file:        none
-----Target related settings-----
Target device:                Cortex-A7
Target interface:             JTAG
Target interface speed:       4000kHz
Target endian:                little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link V9 compiled Dec 13 2019 11:14:50
Hardware: V9.40
S/N: 59425868
Feature(s): RDI, GDB, FlashDL, FlashBP, JFlash, RDDI
Checking target voltage...
Target voltage: 3.31 V
Listening on TCP/IP port 2331
Connecting to target...

J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x5BA00477 (Cortex-A7)
Connected to target
Waiting for GDB connection...█
```

### step3

另开一个终端，进入melis源码source目录下，执行  
sudo /工具链绝对路径/arm-melis-eabi-gdb XXX.elf，将会加载指定的符号表，  
进入gdb命令行

### step4

进入gdb命令行之后，执行  
target remote 127.0.0.1:2331，则会顺利连接成功，即可进行远程调试目标板

```
wuhuati@PCwuhuati:~/workspace/sshserver/workspace/nnn/melis-v3.0/source$ sudo /home/wuhuati/workspace/sshserver/workspace/nnn/melis-v3.0/toolchain/bin/arm-melis-eabi-gdb /home/wuhuati/workspace/sshserver/workspace/nnn/melis-v3.0/source/ekernel/melis30.elf
GNU gdb (GNU Tools for Arm Embedded Processors 9-2020-q1-update) 8.3.0.20190709-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=arm-melis-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
/home/wuhuati/.gdbinit:1: Error in sourced command file:
~/GdbPlugins/gef/gef.py:57: Error in sourced command file:
Undefined command: "from". Try "help".
Reading symbols from /home/wuhuati/workspace/sshserver/workspace/nnn/melis-v3.0/source/ekernel/melis30.elf...
(gdb) target remote 127.0.0.1:2331
Remote debugging using 127.0.0.1:2331
cpu_do_idle () at ekernel/arch/arm/armv7a/cortex-a7/vector_s.S:490
490      mov     pc, lr
(gdb)
```

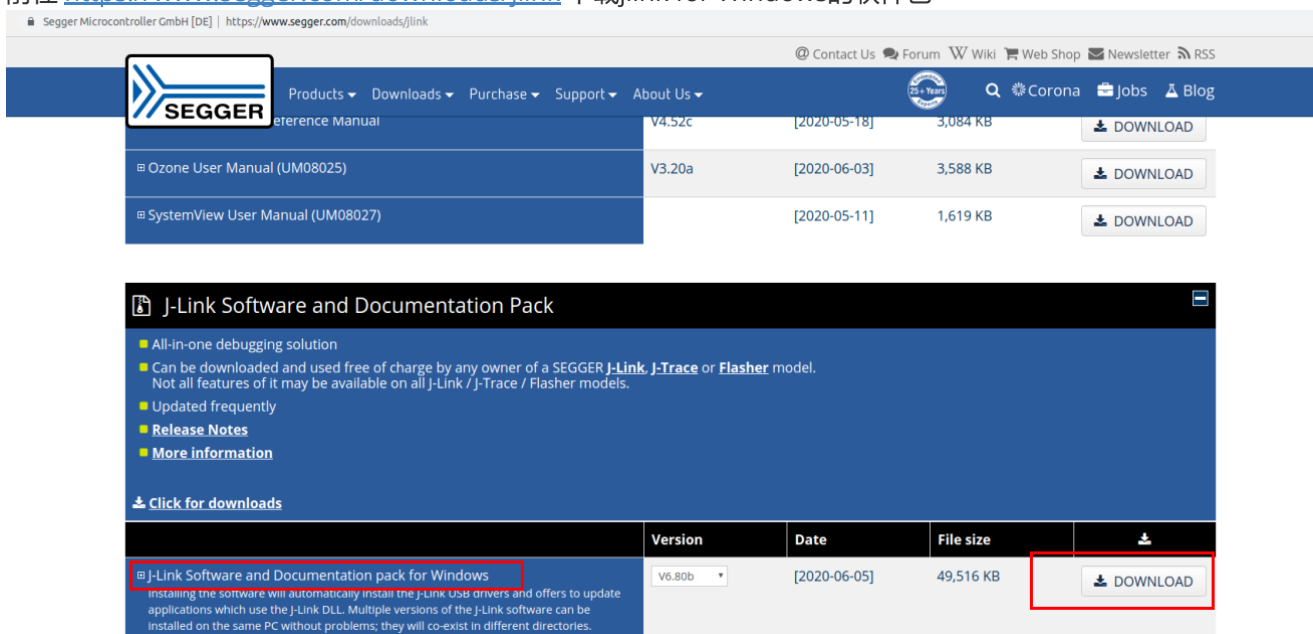
## 注意

如遇到异常或者错误，请拔出转接板，重启目标板，再插入转接板，因为转接板会供电

## Windows环境

### step1

前往 <https://www.segger.com/downloads/jlink> 下载jlink for Windows的软件包



SEGGER Microcontroller GmbH [DE] | <https://www.segger.com/downloads/jlink>

Products ▾ Downloads ▾ Purchase ▾ Support ▾ About Us ▾

Reference Manual V4.52c [2020-05-18] 3,084 KB [DOWNLOAD](#)

▢ Ozone User Manual (UM08025) V3.20a [2020-06-03] 3,588 KB [DOWNLOAD](#)

▢ SystemView User Manual (UM08027) [2020-05-11] 1,619 KB [DOWNLOAD](#)

### J-Link Software and Documentation Pack

- All-in-one debugging solution
- Can be downloaded and used free of charge by any owner of a SEGGER J-Link, J-Trace or Flasher model. Not all features of it may be available on all J-Link / J-Trace / Flasher models.
- Updated frequently
- [Release Notes](#)
- [More information](#)

[Click for downloads](#)

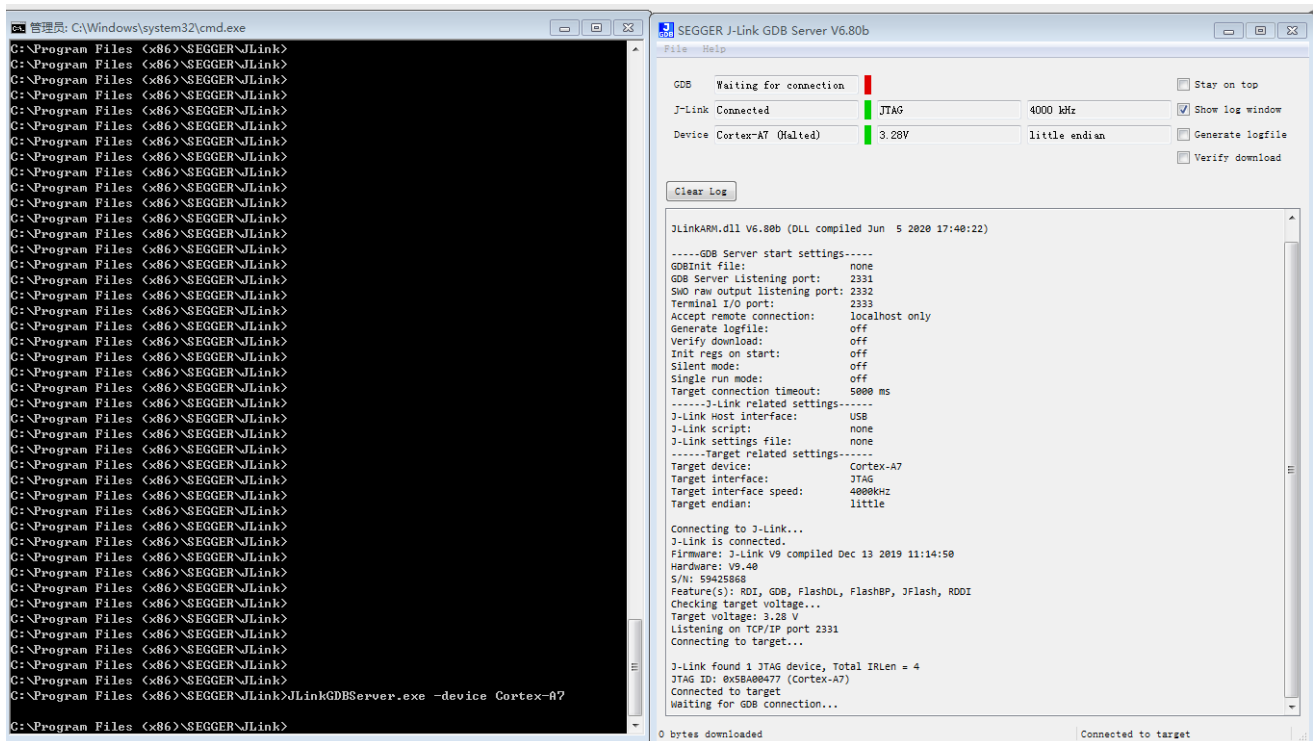
	Version	Date	File size	
▢ J-Link Software and Documentation pack for Windows	V6.80b	[2020-06-05]	49,516 KB	<a href="#">DOWNLOAD</a>

Installing the software will automatically install the J-Link user drivers and offers to update applications which use the J-Link DLL. Multiple versions of the J-Link software can be installed on the same PC without problems; they will co-exist in different directories.

一路点击next到底即可。

### step2

在命令行进入刚才jlink的安装路径,执行 JLinkGDBServer.exe -device Cortex-A7 命令，启动JLinkGDBServer服务，等待GDB的连接

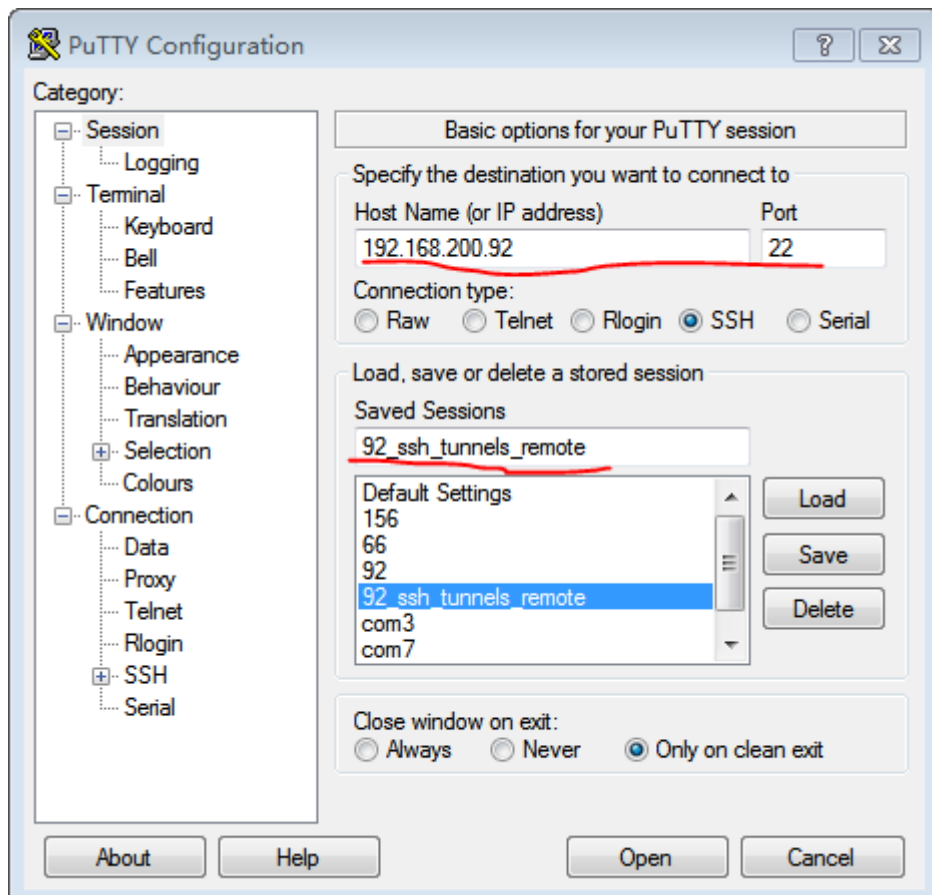


### step3

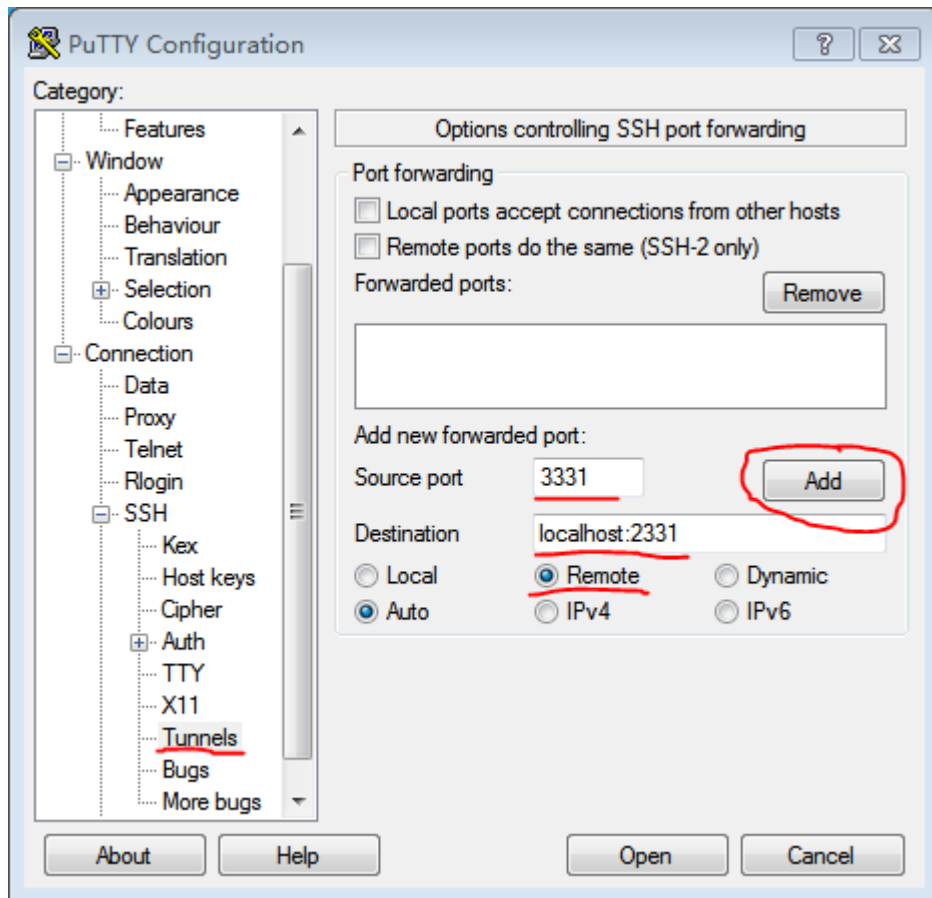
利用putty建立ssh隧道 (技术细节参考[ssh-tunnel的技术原理](#))

putty配置：

(1)仍旧填写要连接的编译服务器地址和端口，session名字可随意命名，这里命名为92\_ssh\_tunnels\_remote。

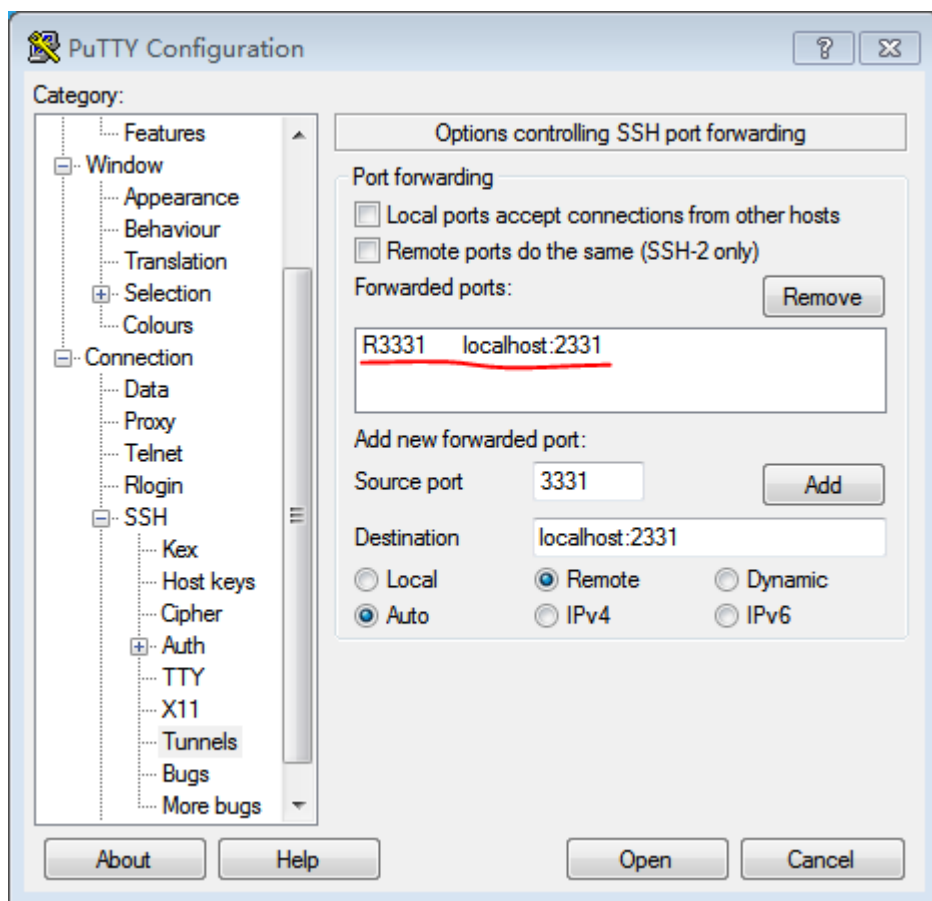


(2)进入Connection->SSH->Tunnels配置页面。putty运行于本地PC，从putty的角度来看，编译服务器为远程remote。因为我们希望编译服务器运行gdb，向本地PC发送控制命令，通过PC转发给JLink，所以隧道模式为Remote。Source port = 3331是指remote端（即编译服务器）的端口（也可指定其他端口号，这里以3331为例），Destination = localhost:2331是指本地PC连接的JLinkGDBServer的IP地址和端口号。填好后，点击Add。

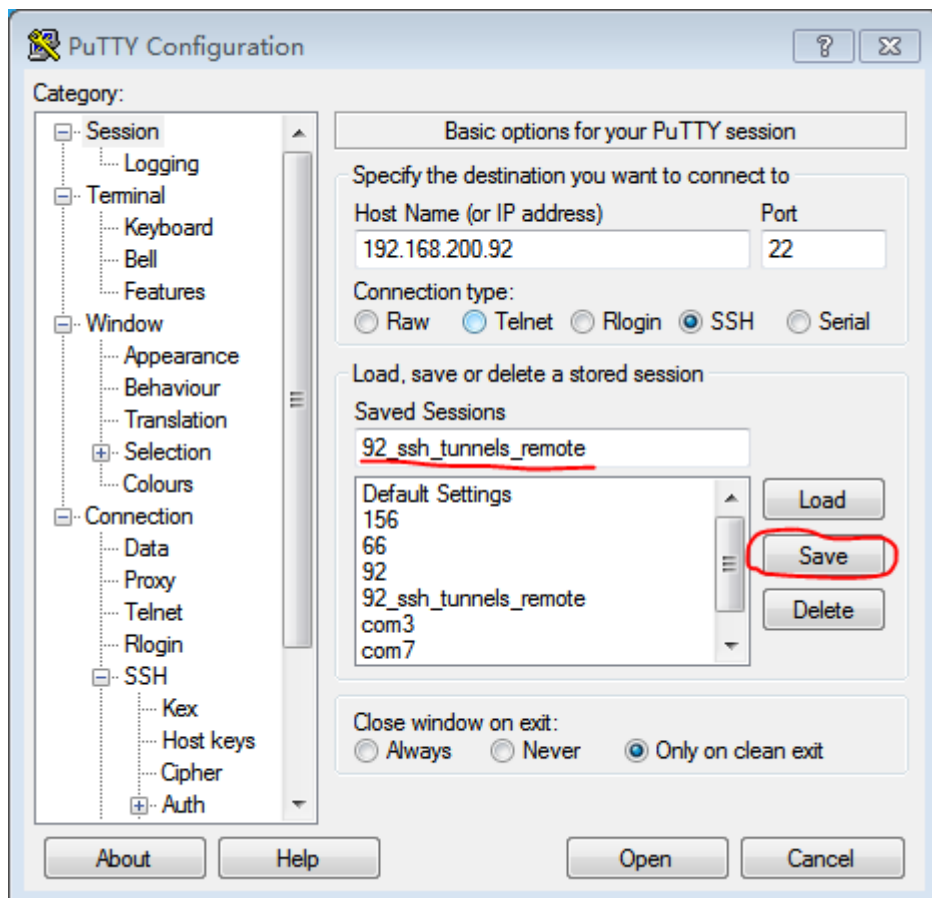


点击Add后，效果如下：





(3)回到Session配置页，点击Save，就把刚才的配置保存在92\_ssh\_tunnels\_remote的session里了。



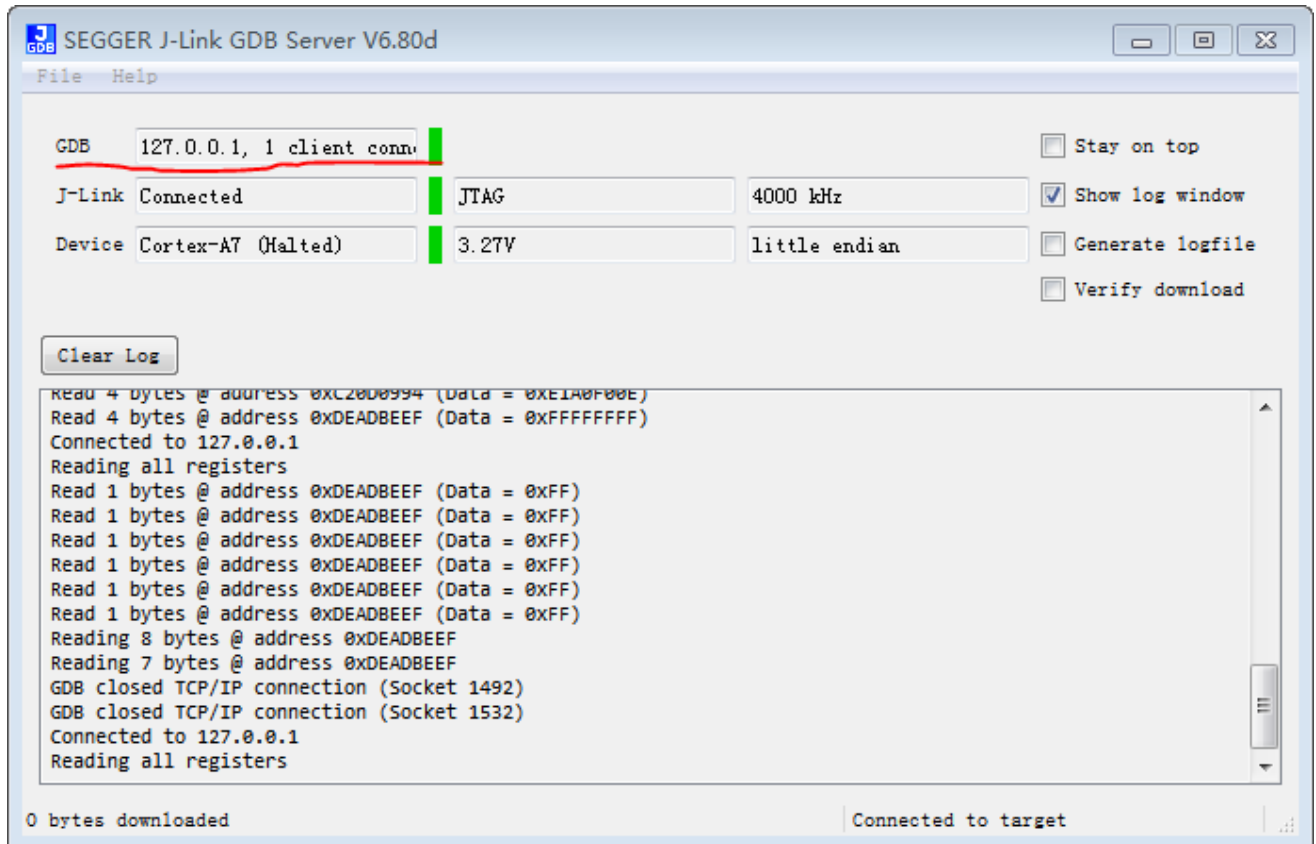


(4)open 92\_ssh\_tunnels\_remote这个session，连接上92编译服务器，这时这个连接的session就自动建立了remote ssh tunnel，编译服务器的端口3331被打开了。

## step4

在编译服务器上运行交叉编译工具链的gdb，利用ssh隧道转发技术，通过本地PC中转，连接JLinkGDBServer。进入melis-v459 SDK，运行：`toolchain/bin/arm-melis-eabi-gdb source/ekernel/melis30.elf`。启动GDB 并加载符号表。

进入gdb命令行之后，执行`target remote :3331`，则会连接成功。这里3331是编译服务器自身的端口，所以省略了hostname。编译服务器的:3331通过ssh tunnel转发到本地PC的localhost:2331，从而连接上了JLinkGDBServer。



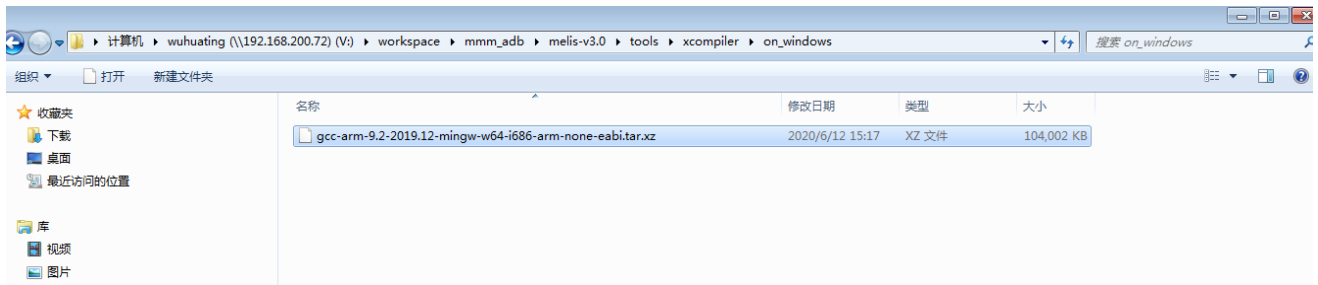
上图J-Link GDB Server显示GDB成功连接。随后即可使用编译服务器的gdb，进行远程调试目标板。

一般情况下利用ssh隧道转发就可以gdb调试了。但是有些客户公司，禁用了开发电脑的USB端口，故需自带笔记本电脑，在不接入客户编译服务器的情况下，连接开发板调试。这时在笔记本电脑上按照下面的步骤进行本地PC调试。

## step5

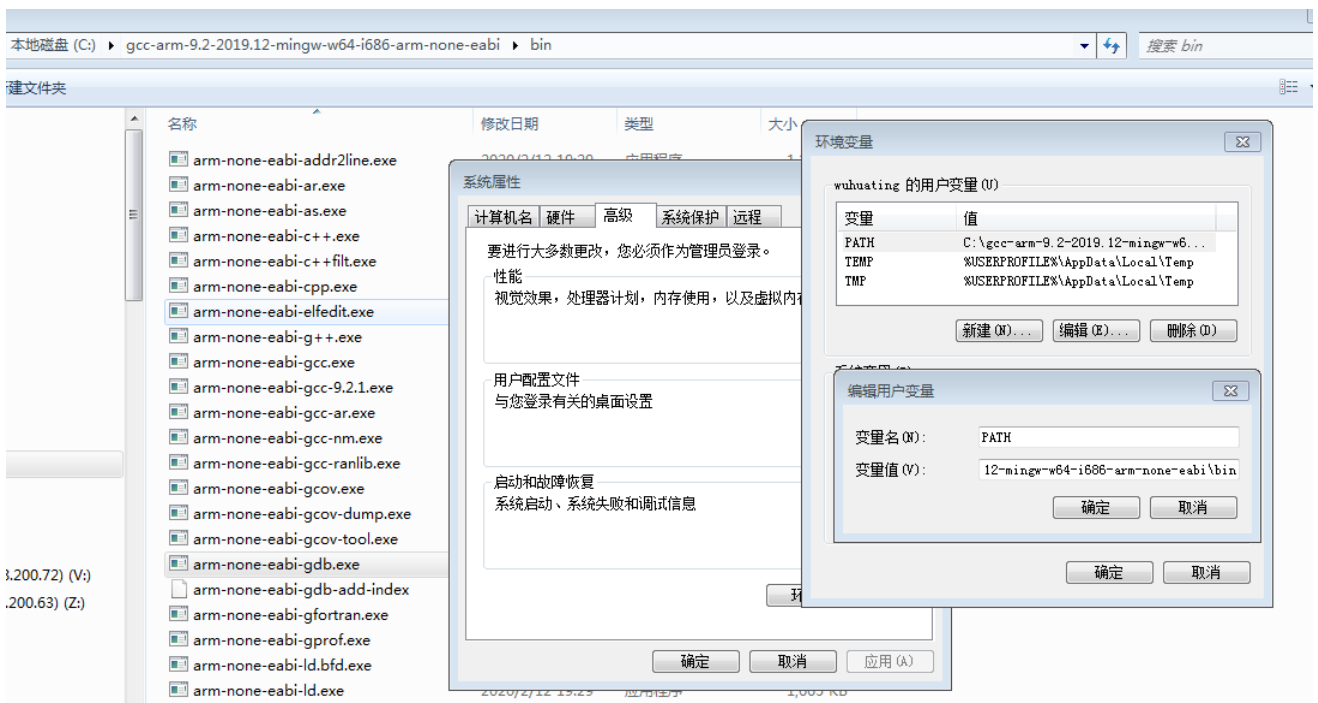
gdb的安装

- 1.解压melis-v3.0/tools/xcompiler/on\_windows/目录下的gcc-arm-9.2-2019.12-mingw-w64-i686-arm-none-eabi.tar.xz



## 2. 配置gdb

在环境变量Path变量值中加入C:\gcc-arm-9.2-2019.12-mingw-w64-i686-arm-none-eabi\bin  
(此路径是作者的安装(解压)路径,读者可以根据自己的情况,自行修改)



## 3.

命令行执行 arm-none-eabi-gdb \xxx\melis30.elf ,启动GDB 并加载符号表

```

C:\Users\wuhuating>
C:\Users\wuhuating>arm-none-eabi-gdb U:\workspace\mmm_adb\melis-v3.0\source\kernel\melis30.elf
C:\gcc-arm-9.2-2019.12-mingw-w64-i686-arm-none-eabi\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Toolchain for the A-profile Architecture 9.2-2020.02 (arm-9.10)) 8.3.0.20190709-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.linaro.org/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from U:\workspace\mmm_adb\melis-v3.0\source\kernel\melis30.elf
..
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
cpu_do_idle (<) at ekernel/arch/arm/armv7a/cortex-a7/vector_s.S:493
493      ekernel/arch/arm/armv7a/cortex-a7/vector_s.S: No such file or directory

(gdb)

```

## step6

进入gdb命令行之后，执行

target remote localhost:2331，则会顺利连接成功，即可进行远程调试目标板

```

C:\Users\wuhuating>arm-none-eabi-gdb U:\workspace\mmm_adb\melis-v3.0\source\ekernel\melis30.elf
C:\gcc-arm-9.2-2019.12-mingw-w64-i686-arm-none-eabi\bin\arm-none-eabi-gdb.exe: warning: Couldn't determine a path for the index cache directory.
GNU gdb (GNU Toolchain for the A-profile Architecture 9.2-2020.02 (arm-9.10)) 8.3.0.20190709-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.linaro.org/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from U:\workspace\mmm_adb\melis-v3.0\source\ekernel\melis30.elf.
..
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
cpu_do_idle (<) at ekernel/arch/arm/armv7a/cortex-a7/vector_s.S:493
493      ekernel/arch/arm/armv7a/cortex-a7/vector_s.S: No such file or directory.

(gdb) b rt_mq_create
Breakpoint 1 at 0xc20d6900: file ekernel/core/rt-thread/ipc.c, line 1982.
(gdb) c
Continuing.

Breakpoint 1, rt_mq_create (name=0xc2448190 "adb-queue", msg_size=4,
    max_msgs=3, flag=0 '\000') at ekernel/core/rt-thread/ipc.c:1982
1982      ekernel/core/rt-thread/ipc.c: No such file or directory.
(gdb)

```

## melis调试实例

### 设置断点

以在 rt\_free 函数入口处设置断点为例, 执行 b rt\_free

```

(gdb) b rt_free
Breakpoint 1 at 0xc203c208: file ekernel/core/rt-thread/wrapper/epos.c, line 169.
(gdb) c
Continuing.

Breakpoint 1, rt_free (ptr=0xc2633100) at ekernel/core/rt-thread/wrapper/epos.c:169
169         if (ptr == RT_NULL)
(gdb) bt
#0  rt_free (ptr=0xc2633100) at ekernel/core/rt-thread/wrapper/epos.c:169
#1  0xc21587bc in dfs_file_open (fd=0xc26670d0, path=0xc2a9b700 "/", flags=2097152) at ekernel/subsys/thirdparty/dfs/src/dfs_file.c:78
#2  0xc215a1f4 in opendir (name=0xc2a9b700 "/") at ekernel/subsys/thirdparty/dfs/src/dfs_posix.c:658
#3  0xc2125cc8 in msh_auto_complete_path (path=0xc2bca76c <incomplete sequence \303>) at ekernel/subsys/finsh_cli/msh.c:508
#4  0xc2125fa4 in msh_auto_complete (prefix=0xc2bca76c <incomplete sequence \303>) at ekernel/subsys/finsh_cli/msh.c:624
#5  0xc2126d80 in shell_auto_complete (prefix=0xc2bca76c <incomplete sequence \303>) at ekernel/subsys/finsh_cli/shell_entry.c:389
#6  0xc21274a8 in finsh_thread_entry (parameter=0x0) at ekernel/subsys/finsh_cli/shell_entry.c:674
#7  0xc28202cc in ret_from_create_c () at ekernel/arch/arm/armv7a/cortex-a7/port.c:68
#8  0xc2030fa4 in rt_thread_delete_sethook (hook=0xc212703c <finsh_thread_entry>) at ekernel/core/rt-thread/thread.c:84
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
(gdb) info reg
r0          0xc2633100      3261280512
r1          0xc263310a      3261280522
r2          0xc2633108      3261280520
r3          0xc26670d0      3261493456
r4          0x0            0
r5          0xc212703c      3255988284
r6          0xdeadbeef      3735928559
r7          0xdeadbeef      3735928559
r8          0xdeadbeef      3735928559
r9          0xdeadbeef      3735928559
r10         0xdeadbeef      3735928559
r11         0xc2c10f04      3267432196
r12         0xffffffff      4294967295
sp          0xc2c10ed8      0xc2c10ed8
lr          0xc21587bc      3256190908
pc          0xc203c208      0xc203c208 <rt_free+16>
cpsr       0x60000053      1610612819
fpscr       0x0            0
r8_usr      0xdeadbeef      3735928559
r9_usr      0xdeadbeef      3735928559
r10_usr     0xdeadbeef      3735928559
r11_usr     0xc2c10f04      3267432196
r12_usr     0xffffffff      4294967295

```

## 观察变量

进入断点后，可以观察变量或者结构体内容

```

(gdb) p *rt_current_thread
$1 = (name = "tshell", '\000' <repeats 25 times>, type = 1 '\001', flags = 0 '\000', module_id = 0x0, list = {next = 0xc26172f0, prev = 0xc2529408 <rt_object_container+4>}, tlist = {
  next = 0xc25c9998 <rt_thread_priority_table+160>, prev = 0xc25c9998 <rt_thread_priority_table+160>}, sp = 0xc2c10cd8, entry = 0xc212703c <finsh_thread_entry>, parameter = 0x0, stack_addr = 0xc2bd1000,
  stack_size = 262144, error = 0, stat = 1 '\001', current_priority = 20 '\024', init_priority = 20 '\024', number_mask = 1048576, event_set = 0, event_info = 0 '\000', init_tick = 10, remaining_tick = 10,
  thread_timer = {parent = {name = "tshell", '\000' <repeats 25 times>, type = 138 '\212', flag = 0 '\000', module_id = 0x0, list = {next = 0xc2617250, prev = 0xc2529408 <rt_object_container+132>}}, row = {{
    next = 0xc2617440, prev = 0xc2617440}}, timeout_func = 0xc2031d00 <rt_thread_timeout>, parameter = 0xc26173a8, init_tick = 0, timeout_tick = 0}, cleanup = 0x0, user_data = {0, 0, 0, 0}, cputime = 0,
  montime = 0, sched_l = 2, sched_o = 1, born = 39, preempt_count = 0)
(gdb)

```

```

Breakpoint 2, rt_interrupt_enter () at ekernel/core/rt-thread/irq.c:65
65         level = rt_hw_interrupt_disable();
(gdb) n
66         preempt_count_add(HARDIRQ_OFFSET);
(gdb) n
67         RT_OBJECT_HOOK_CALL(rt_interrupt_enter_hook, ());
(gdb) n
68         rt_hw_interrupt_enable(level);
(gdb) n
69     }
(gdb) p rt_interrupt_enter_hook
$2 = (void (*)(void)) 0x0
(gdb)

```

## linux内核调试实例

### 调试vmlinux

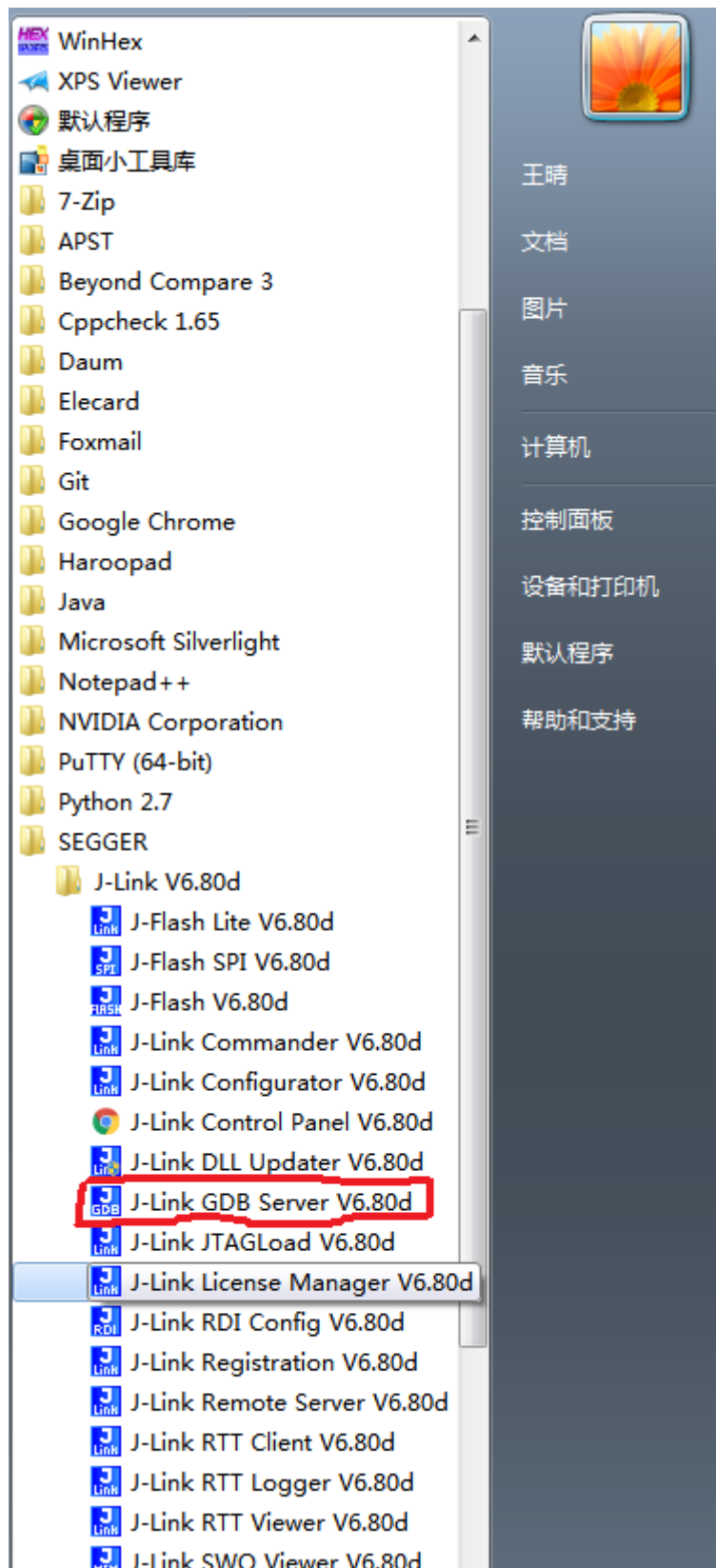
以调试tina-v459的linux内核为例。在host-server上调试，使用host-server-gdb（即tina-v459-sdk自带的交叉编译工具链的工具arm-openwrt-linux-muslgnueabi-gdb），利用putty的ssh-tunnel功能，转发控制命令给host-PC的J-Link GDB Server，再通过J-Link仿真器发送jtag命令给target-board完成控制。

make kernel\_menuconfig，打开CONFIG\_DEBUG\_INFO，  
make menuconfig，打开CONFIG\_KERNEL\_DEBUG\_INFO，该配置会覆盖kernel\_menuconfig的CONFIG\_DEBUG\_INFO，所以是必须的。

通过上述配置，编译出的内核镜像vmlinux将携带调试信息。

编译固件，烧入tina-v459 perf1板，上电启动。

host-PC：运行J-Link GDB Server，默认配置即可。



J-LinkGDBServer运行后，会暂停住target-board系统。

host-server：在tina-sdk内启动host-server-gdb。

在gdb内执行：

```
file vmlinux ,
```

设置断点，

然后 `target remote :2331` 连接JLinkGDBServer，

continue 即可进行调试了。

```
ericwang@Exdroid92:~/workspace/exdroid/tina-v459/lichee/linux-4.9$ ../../prebuilt/gcc/linux-x86/arm/toolchain-sunxi-musl/toolchain/bin/arm-openwrt-linux-muslgnueabi-gdb
GNU gdb (GDB) 8.0
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=arm-openwrt-linux-muslgnueabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file vmlinux
Reading symbols from vmlinux...done.
(gdb) break v4l_s_input
Breakpoint 1 at 0xc037b4c0: file drivers/media/v4l2-core/v4l2-ioct1.c, line 1048
.
(gdb) target remote :2331
Remote debugging using :2331
cpu_v7_do_idle () at arch/arm/mm/proc-v7.S:76
76      ret     lr
(gdb) continue
Continuing.

Breakpoint 1, v4l_s_input (ops=0xbf029c24, file=0xde056240, fh=0xde063c00,
    arg=0xdcc37e54) at drivers/media/v4l2-core/v4l2-ioct1.c:1048
1048     {
(gdb)
```

[debug] 0:tina-v459\* "Exdroid92" 17:24 17-7月-20

## 调试内核模块(ko)

内核模块ko是动态加载的，host-server-gdb不知道动态加载的ko的section的基地址，需要用户自己设置。一般可以设置.text, .bss, .data三个section的基地址。

以调试vin\_v4l2.ko为例，建议.text, .bss, .data的基地址都从target-board运行后的状态文件获取。方法是在target-board上的目录/sys/module/<module\_name>/sections/下的文件，记录了该内核模块的各个section的基地址，读取即可。

在target-board的终端里执行：

```
cd /sys/module/vin_v4l2/sections
```

```
cat .text
```

```
cat .bss
```



```

cat .data
root@(none):/# cd /sys/module/vin_v4l2/sections/
root@(none):/sys/module/vin_v4l2/sections# ll
drwxr-xr-x    2 root    root          0 Jan  1 01:56 .
drwxr-xr-x    6 root    root          0 Jan  1 00:00 ..
-r--r--r--    1 root    root        4096 Jan  1 01:57 .ARM.exidx
-r--r--r--    1 root    root        4096 Jan  1 01:57 .ARM.exidx.exit.text
-r--r--r--    1 root    root        4096 Jan  1 01:57 .ARM.exidx.init.text
-r--r--r--    1 root    root        4096 Jan  1 01:57 .bss
-r--r--r--    1 root    root        4096 Jan  1 01:57 .data
-r--r--r--    1 root    root        4096 Jan  1 01:57 .exit.text
-r--r--r--    1 root    root        4096 Jan  1 01:57 .gnu.linkonce.this_module
-r--r--r--    1 root    root        4096 Jan  1 01:57 .init.text
-r--r--r--    1 root    root        4096 Jan  1 01:57 .rodata
-r--r--r--    1 root    root        4096 Jan  1 01:57 .rodata.str
-r--r--r--    1 root    root        4096 Jan  1 01:57 .rodata.str1.4
-r--r--r--    1 root    root        4096 Jan  1 01:57 .strtab
-r--r--r--    1 root    root        4096 Jan  1 01:57 .symtab
-r--r--r--    1 root    root        4096 Jan  1 01:57 .text
-r--r--r--    1 root    root        4096 Jan  1 01:57 __bug_table
-r--r--r--    1 root    root        4096 Jan  1 01:57 __param
root@(none):/sys/module/vin_v4l2/sections# cat .bss
0xbf030080
root@(none):/sys/module/vin_v4l2/sections# cat .data
0xbf02e018
root@(none):/sys/module/vin_v4l2/sections# cat .text
0xbf016000
root@(none):/sys/module/vin_v4l2/sections# █

```

获取了.text, .bss, .data三个section的基地址后。再启动J-Link GDB Server，暂停住target-board的内核。然后回到host-server的ssh终端，在host-server-gdb内通过add-symbol-file指令增加vin\_v4l2.ko的符号表：

```

add-symbol-file drivers/media/platform/sunxi-vin/vin_v4l2.ko 0xbf016000 -s .bss 0xbf030080 -s .data
0xbf02e018
(gdb) add-symbol-file drivers/media/platform/sunxi-vin/vin_v4l2.ko 0xbf016000 -s .bss 0xbf030080 -s .data 0xbf02e018
add symbol table from file "drivers/media/platform/sunxi-vin/vin_v4l2.ko" at
      .text_addr = 0xbf016000
      .bss_addr = 0xbf030080
      .data_addr = 0xbf02e018
(y or n) y
Reading symbols from drivers/media/platform/sunxi-vin/vin_v4l2.ko...done.
(gdb) █

```

还有一种方式：通过objdump获取ko库内的各section的偏移地址，然后根据ko库在target-board上的实际地址，加上偏移量，来计算各section的基地址，但这种方式个人认为不准确。故不推荐使用。

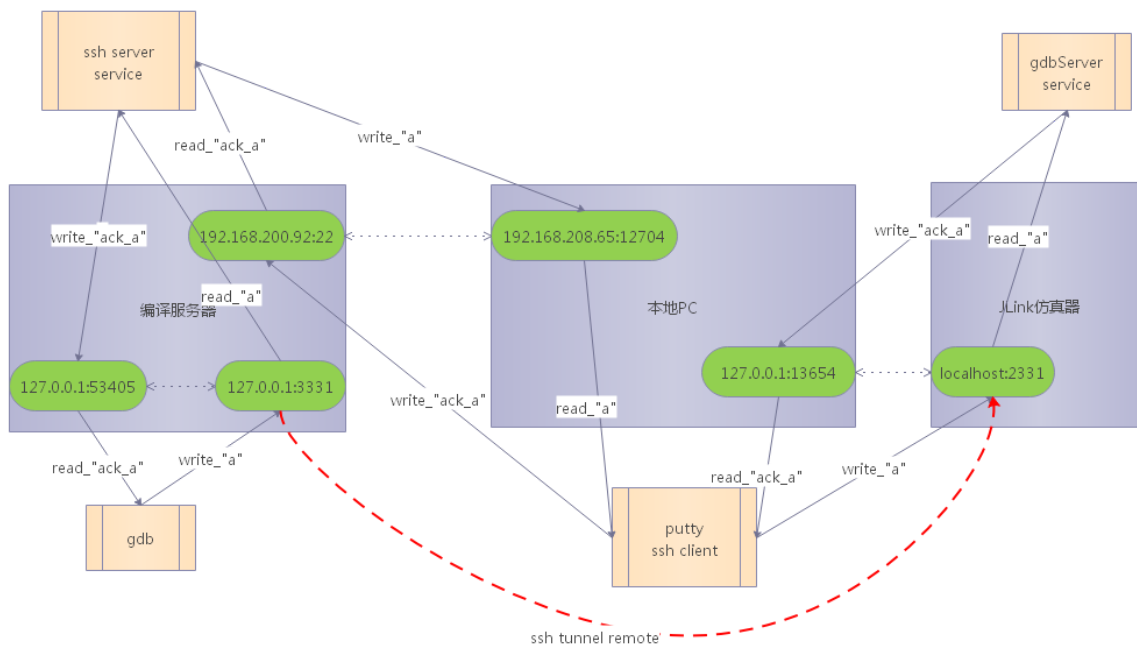
增加了ko库的符号表后，即可在ko库的源码中设置断点，

然后 `target remote :2331` 连接JLinkGDBServer

然后 `continue` 让内核恢复运行，进行debug。

## ssh tunnel的技术原理

( 本描述为个人推测，如有错误请指正。 )



编译服务器（IP地址192.168.200.92）的ssh server和本地PC（IP地址192.168.208.65）的putty，建立网络连接：192.168.200.92:22 <==> 192.168.208.65:12704

本地PC的putty打开本地PC端口13654和JLink仿真器的gdbServerService的端口localhost:2331进行网络连接：  
127.0.0.1:13654 <==> localhost:2331

J-Link GDB Server实际上是运行在本地PC的程序，这里为了直观，画在了J-Link仿真器一侧。所以IP地址使用localhost表示本机。127.0.0.1也是表示本机的IP地址。

putty的配置了ssh tunnel的session登录编译服务器时，编译服务器的ssh server service打开端口3331进行监听。编译服务器端的gdb运行时，执行target remote :3331。连接本地端口3331。这时gdb和端口3331建立了网络连接：127.0.0.1:53405 <==> 127.0.0.1:3331。

数据流如上图所示，gdb向端口3331发送命令"a"，ssh server service读取后向本地PC的端口12704发送，putty从12704读取数据，再写到localhost:2331。从gdb看来，发送到端口3331的数据，就被转发到了J-Link GDB server的localhost:2331。这就是ssh隧道连接。

ssh隧道连接有2种模式：

remote：数据流从远程编译服务器发到本地PC。

local：数据流从本地PC发送到远程编译服务器。

命令"a"的数据流向：编译服务器端口3331 -> 本地PC端口12704 -> JLink GDB Server端口localhost:2331。

应答"ack\_a"的数据流向：本地PC端口13654 -> 编译服务器端口22 -> 编译服务器端口53405。

