《嵌入式系统》

（第五讲）

厦门大学信息学院软件工程系 曾文华

2023年10月10日

第5章　开发环境和调试技术

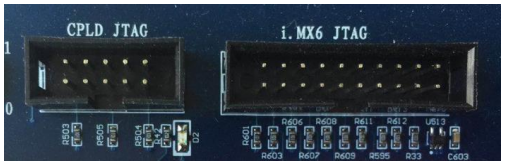
- 5.1　交叉开发模式概述
- 5.2　宿主机环境
- 5.3　目标板环境
- 5.4　交叉编译工具链
- 5.5　本地调试（gdb）
- 5.6　远程调试（gdb + gdbserver）
- 5.7　内核调试（gdb + kgdb）
- 5.8　网络调试

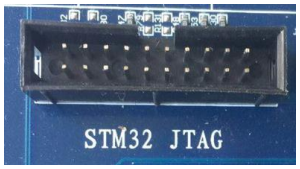
5.1 交叉开发模式概述

- 交叉开发模式：宿主机（PC机：VMware下的Ubuntu）-目标板（IMX6实验箱：超级终端Xshell）

- GNU软件：
  ① Shell：Shell基本上是一个命令解释器，类似于DOS下的command
  ② glibc：glibc是GNU发布的libc库，即c运行库
  ③ GCC：GCC（GNU Compiler Collection，GNU编译器套件）是由 GNU 开发的编程语言编译器
  ④ gdb： UNIX及UNIX-like下的调试工具
  ⑤ vim：vim是一个类似于vi的著名的功能强大、高度可定制的文本编辑器，在vi的基础上改进和增加了很多特性
  ⑥ Emacs： Emacs是著名的集成开发环境和文本编辑器

- 宿主机与目标板的连接方式：
  ① 串口（COM1 TO USB）
  ② 以太网接口（RJ45）
  ③ USB接口
  ④ JTAG接口（Joint Test Action Group）

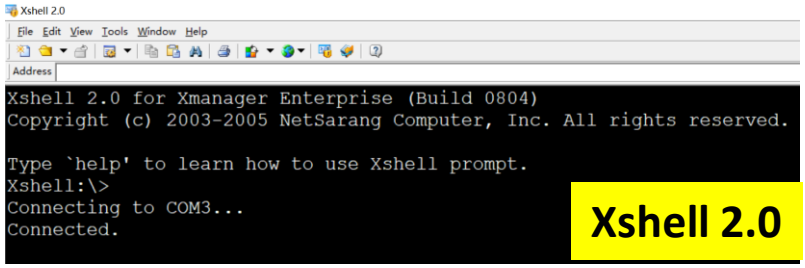




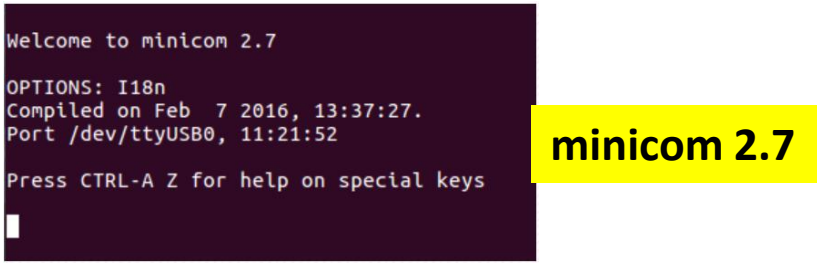
实验箱上的JTAG接口

5.2 宿主机环境

- 5.2.1 串口终端



  - Windows下的超级终端
    - 超级终端是Windows自带的一个串口调试工具，其使用较为简单，被广泛使用在串口设备的初级调试上，如Xshell 2.0。



  - Linux下的minicom
    - minicom是一个串口通信工具，就像Windows下的超级终端。可用来与串口设备通信，如调试交换机和Modem等。它的Debian软件包的名称就叫minicom，用apt-get install minicom命令即可下载安装。如果宿主机是纯Linux环境，则需要使用minicom作为串口终端。

• 5.2.2 BOOTP

– BOOTP（Bootstrap Protocol，引导程序协议）是一种引导协议，基于IP/UDP协议，也称自举协议，是DHCP协议的前身。BOOTP用于无盘工作站的局域网中，可以让无盘工作站从一个中心服务器上获得IP地址。通过BOOTP协议可以为局域网中的无盘工作站分配动态IP地址，这样就不需要管理员去为每个用户去设置静态IP地址。

– BOOTP的一般工作流程就是BOOTP客户端（目标板，实验箱）和BOOTP服务器（宿主机，PC机，Ubuntu）之间的交互，其流程如下：

① 由BOOTP启动代码来启动BOOTP客户端，这个时候BOOTP客户端还没有IP地址。

② BOOTP客户端使用广播形式的IP地址255.255.255.255向网络中发出IP地址查询要求。

③ 运行BOOTP协议的服务器接收到这个请求，会根据请求中提供的MAC地址找到BOOTP客户端，并发送一个含有IP地址、服务器IP地址、网关等信息的回应帧。

④ BOOTP客户端会根据该回应帧来获得自己的IP地址并通过专用文件服务器（如TFTP服务器）下载启动镜像文件，模拟成磁盘来完成启动。

• 5.2.3 TFTP

– TFTP（Trivial File Transfer Protocol，简单文件传输协议）是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。

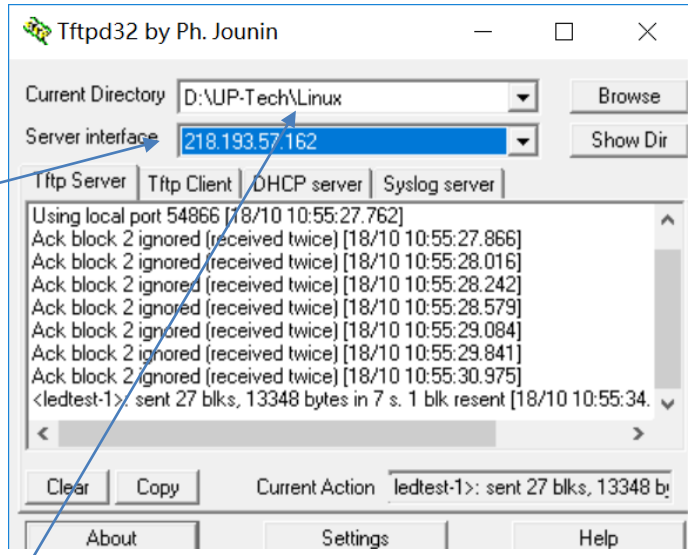
– TFTP是简化了的FTP，TFTP没有用户权限管理的功能。

– 例如：在Windows系统下运行“tftpd32.exe”

– 在实验箱的“超级终端”下执行：

get request 向windows请求资源

- tftp -gr led 218.193.57.162
- 218.193.57.162：Windows系统的IP地址
- 注意：实验箱要与Windows在同一个网段



– 则将Windows系统下的文件led（位于D:\UP-Tech\Linux目录中），通过TFTP协议，传送到实验箱中。注意：这里是通过网线进行文件的传送。

• 5.2.4 交叉编译

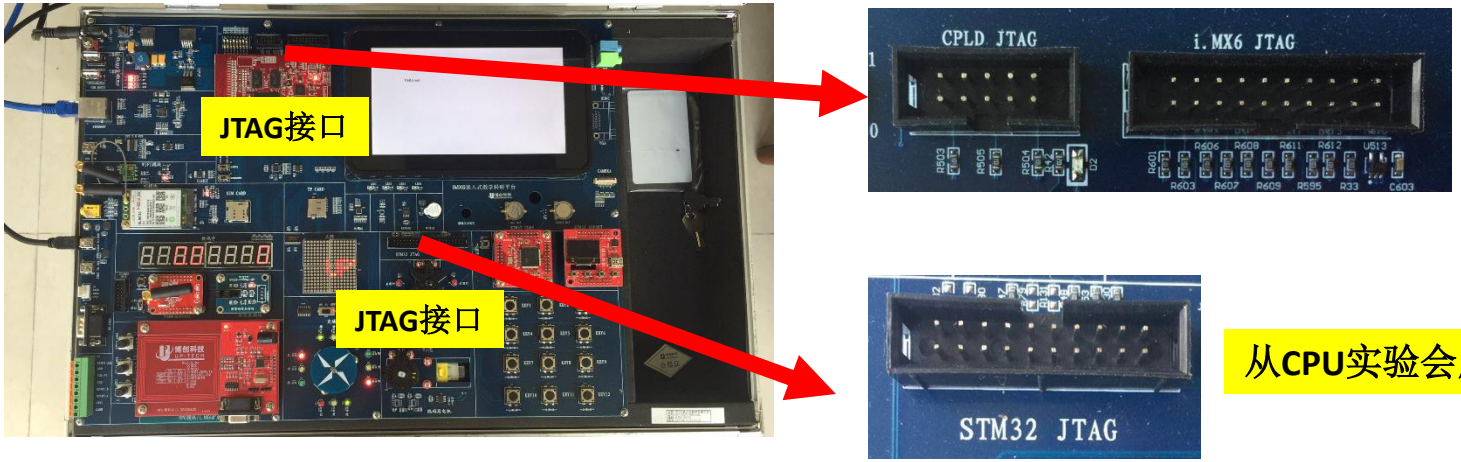
– 交叉编译：在x86架构的宿主机（PC机，Ubuntu）上编译生成适用于ARM架构（IMX6实验箱）的ELF格式的可执行代码。

- 交叉编译：arm-poky-linux-gnueabi-gcc
- 本地编译：gcc
- ELF：Executable and Linkable Format，可执行与可链接格式，是一种用于二进制文件、可执行文件、目标代码、共享库和核心转储格式文件。

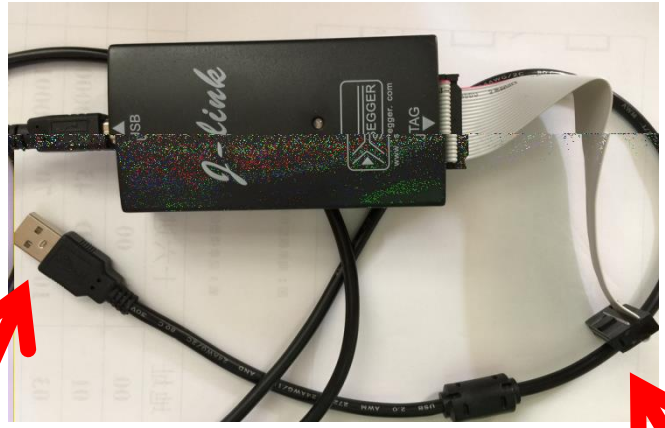
```
root@uptech:/imx6/whzeng/hello# make
arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mf
pu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/co
rtexa9hf-neon-poky-linux-gnueabi    -O2 -pipe -g -feliminate-unused-debug-types
   -c -o hello.o hello.c
arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mf
pu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/co
rtexa9hf-neon-poky-linux-gnueabi    -o hello hello.o
root@uptech:/imx6/whzeng/hello#
```

5.3 目标板环境

- **5.3.1 JTAG接口简介**
  - JTAG（Joint Test Action Group，联合测试工作组）是一种国际标准测试协议（IEEE 1149.1兼容），主要用于芯片内部测试。现在多数的高级器件都支持JTAG协议，如DSP、FPGA器件等。标准的JTAG接口是5线：TMS、TCK、TDI、TDO、nTRST，分别为模式选择、时钟、数据输入、数据输出线、系统复位。

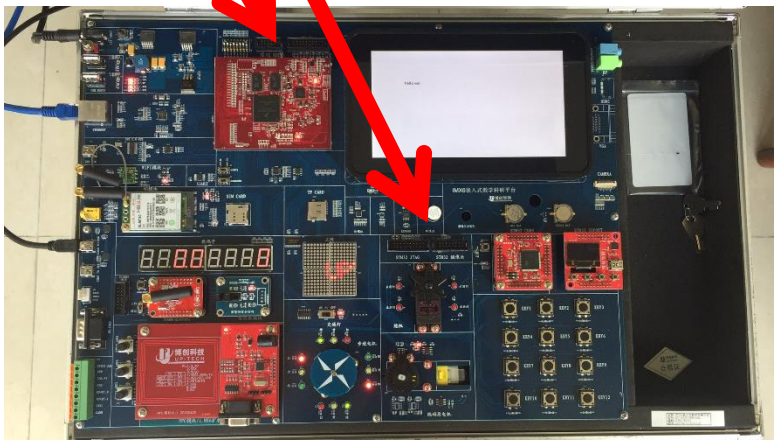


i.MX6 JTAG 20 / STM32 JTAG 20

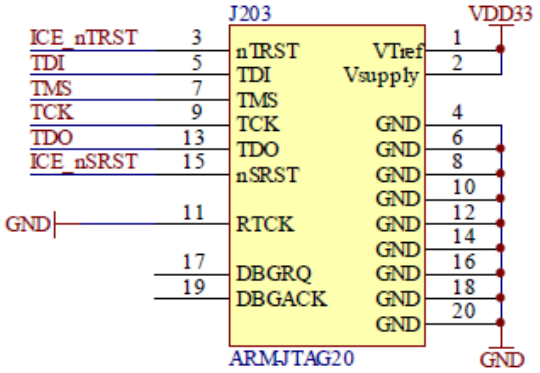


USB接口

20芯扁线



JTAG 20



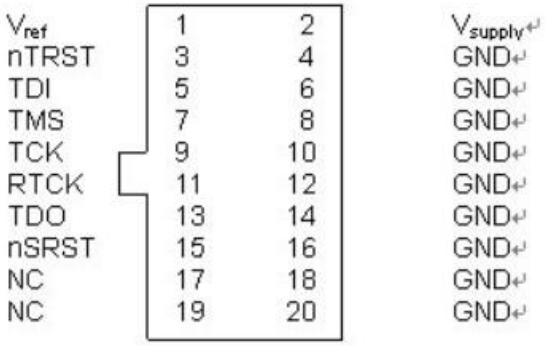


图 1　JTAG 口的信号排列图

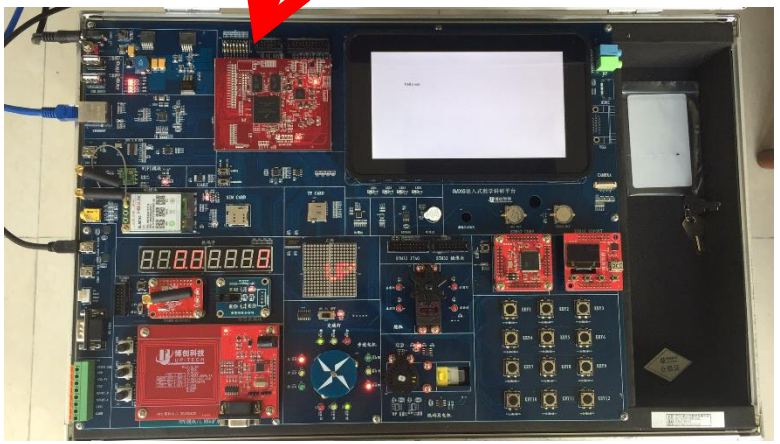
表 1 JTAG引脚说明

序号	信号名	方向	说 明
1	Vref	Input	接口电平参考电压，通常可直接接电源
2	Vsupply	Input	电源
3	nTRST	Output	(可选项) JTAG复位。在目标端应加适当的上拉电阻以防止误触发。
4	GND	--	接地
5	TDI	Output	Test Data In from Dragon-ICE to target.
6	GND	--	接地
7	TMS	Output	Test Mode Select
8	GND	--	接地
9	TCK	Output	Test Clock output from Dragon-ICE to the target
10	GND	--	接地
11	RTCK	Input	(可选项) Return Test Clock。由目标端反馈给Dragon-ICE的时钟信号，用来同步TCK信号的产生。不使用时可以直接接地。
12	GND	--	接地
13	TDO	Input	Test Data Out from target to Dragon-ICE.
14	GND	--	接地
15	nSRST	Input/Output	(可选项) System Reset，与目标板上的系统复位信号相连。可以直接对目标系统复位，同时可以检测目标系统的复位情况。为了防止误触发，应在目标端加上适当的上拉电阻。
16	GND	--	接地
17	NC		保留
18	GND	--	接地
19	NC	--	保留
20	GND	--	接地

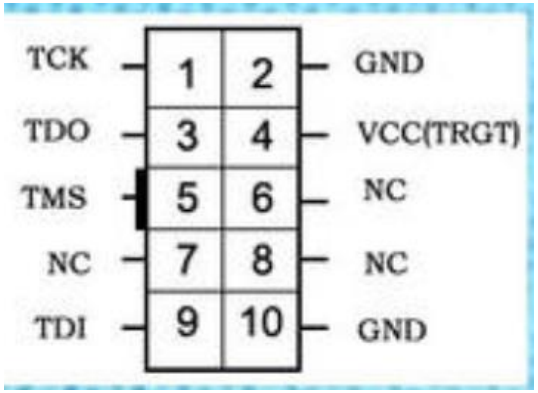
CPLD JTAG 10

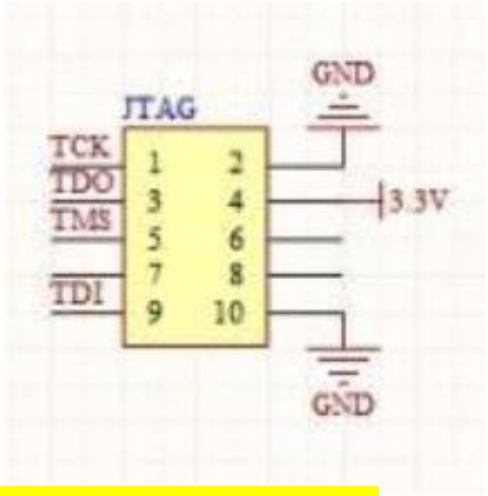
USB接口

10芯扁线



JTAG 10





TCK——测试时钟输入

TDI——测试数据输入，数据通过TDI输入JTAG口

TDO——测试数据输出，数据通过TDO从JTAG口输出

TMS——测试模式选择，TMS用来设置JTAG口处于某种特定的测试模式

NC——未用的管脚

VCC(TRGT)——电源（+5V）

GND——地线

• 5.3.2 Boot Loader简介

相当于ROM-BIOS

– 嵌入式Linux系统启动后，先执行Bootloader，进行硬件和内存的初始化工作，然后加载Linux内核和根文件系统映像，完成Linux系统的启动。

– Bootloader：引导加载程序，是嵌入式目标板（实验箱）加电后运行的第一段软件代码；是在操作系统内核运行之前用来初始化硬件设备、建立内存空间的映射图的小程序。

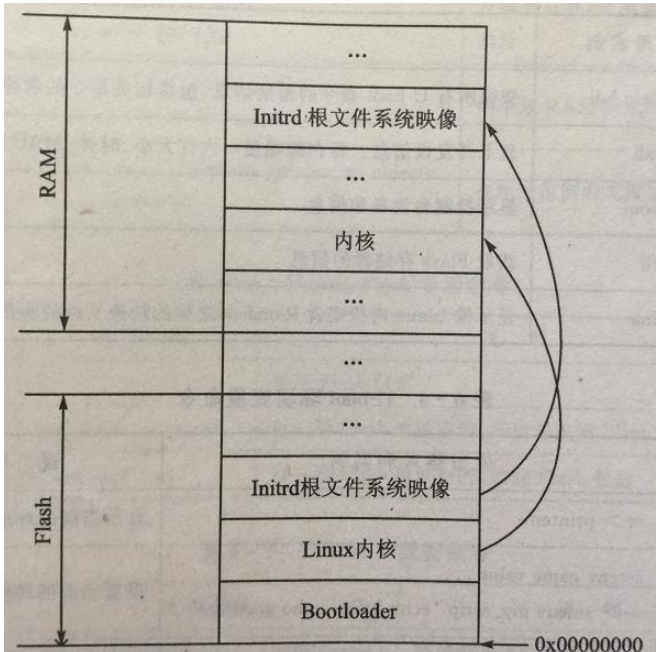


图 6－7 Bootloader 加载内核和根文件系统映像

Bootloader加载Linux内核和根文件系统映像

– 常见的Bootloader：

- **u-boot**：Universal Boot Loader，是遵循GPL条款的开放源码项目，u-boot的作用是系统引导（IMX6实验箱用的）。

- **vivi**：韩国Mizi公司开发的Bootloader引导程序。

IMX6实验箱开机后显示的内容

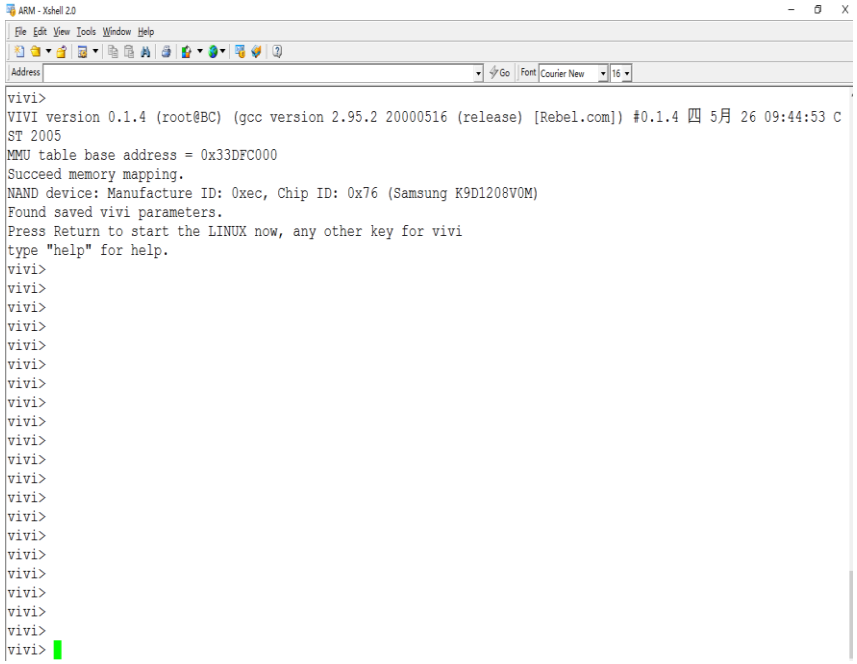
```
U-Boot 2014.04 (Nov 12 2015 - 14:30:52)

CPU:   Freescale i.MX6DL rev1.3 at 792 MHz
CPU:   Temperature 42 C, calibration data: 0x57e50a5f
Reset cause: POR
Board: MX6-SabreSD
I2C:   ready
DRAM:  1 GiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1, FSL_SDHC: 2
*** Warning - bad CRC, using default environment

mx6sabresd.c:Hannstar-XGA
No panel detected: default to Hannstar-XGA
Display: Hannstar-XGA (1024x768)
In:    serial
Out:   serial
Err:   serial
Found PFUZE100 deviceid=10,revid=21
mmc2(part 0) is current device
Net:   Phy not found
PHY reset timed out
FEC [PRIME]
Warning: failed to set MAC address

Normal Boot
Hit any key to stop autoboot:  0
```

u-boot



vivi

5.4 交叉编译工具链

- 本地编译：在PC机上，编译生成PC平台运行的程序。
  - cd /imx6/whzeng/hello-pc
  - make
    - gcc -O2 -pipe -g -feliminate-unused-debug-types -c -o hello-pc.o hello-pc.c
    - gcc -o hello-pc hello-pc.o

- 交叉编译：在PC机上，编译生成目标机（ARM，实验箱）平台运行的程序。
  - cd /imx6/whzeng/hello
  - make
    - arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi -O2 -pipe -g -feliminate-unused-debug-types -c -o hello.o hello.c
    - arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi -o hello hello.o

- 交叉编译工具链：是一个由编译器、链接器、解释器组成的集成开发环境。

实验箱的交叉编译工具链

位于/opt/fsl-imx-wayland/4.9.88-2.0.0/environment-setup-cortexa9hf-neon-poky-linux-gnueabi文件中

```
# Check for LD_LIBRARY_PATH being set, which can break SDK and generally is a bad practice
# http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html#AEN80
# http://xahlee.info/UnixResource_dir/_/ldpath.html
# Only disable this check if you are absolutely know what you are doing!
if [ ! -z "$LD_LIBRARY_PATH" ]; then
    echo "Your environment is misconfigured, you probably need to 'unset LD_LIBRARY_PATH'"
    echo "but please check why this was set in the first place and that it's safe to unset."
    echo "The SDK will not operate correctly in most cases when LD_LIBRARY_PATH is set."
    echo "For more references see:"
    echo "  http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html#AEN80"
    echo "  http://xahlee.info/UnixResource_dir/_/ldpath.html"
    return 1
fi
export SDKTARGETSYSROOT=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi
export PATH=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/x86_64-pokysdk-linux/usr/bin:/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/x86_64-pokysdk-linux/usr/sbin:/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/x86_64-pokysdk-
linux/bin:/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/x86_64-pokysdk-linux/sbin:/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/x86_64-pokysdk-linux/usr/bin/../x86_64-pokysdk-linux/bin:/opt/fsl-imx-wayland/4.9.88-2.0.0/
sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi:/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-musl:$PATH
export PKG_CONFIG_SYSROOT_DIR=$SDKTARGETSYSROOT
export PKG_CONFIG_PATH=$SDKTARGETSYSROOT/usr/lib/pkgconfig:$SDKTARGETSYSROOT/usr/share/pkgconfig
export CONFIG_SITE=/opt/fsl-imx-wayland/4.9.88-2.0.0/site-config-cortexa9hf-neon-poky-linux-gnueabi
export OECORE_NATIVE_SYSROOT="/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/x86_64-pokysdk-linux"
export OECORE_TARGET_SYSROOT="$SDKTARGETSYSROOT"
export OECORE_ACLOCAL_OPTS="-I /opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/x86_64-pokysdk-linux/usr/share/aclocal"
unset command_not_found_handle
export CC="arm-poky-linux-gnueabi-gcc  -march=armv7-a -mfpu=neon -mfloat-abi=hard -mcpu=cortex-a9 --sysroot=$SDKTARGETSYSROOT"
export CXX="arm-poky-linux-gnueabi-g++  -march=armv7-a -mfpu=neon -mfloat-abi=hard -mcpu=cortex-a9 --sysroot=$SDKTARGETSYSROOT"
export CPP="arm-poky-linux-gnueabi-gcc -E  -march=armv7-a -mfpu=neon -mfloat-abi=hard -mcpu=cortex-a9 --sysroot=$SDKTARGETSYSROOT"
export AS="arm-poky-linux-gnueabi-as "
export LD="arm-poky-linux-gnueabi-ld  --sysroot=$SDKTARGETSYSROOT"
export GDB=arm-poky-linux-gnueabi-gdb
export STRIP=arm-poky-linux-gnueabi-strip
export RANLIB=arm-poky-linux-gnueabi-ranlib
export OBJCOPY=arm-poky-linux-gnueabi-objcopy
export OBJDUMP=arm-poky-linux-gnueabi-objdump
export AR=arm-poky-linux-gnueabi-ar
export NM=arm-poky-linux-gnueabi-nm
export M4=m4
export TARGET_PREFIX=arm-poky-linux-gnueabi-
export CONFIGURE_FLAGS="--target=arm-poky-linux-gnueabi --host=arm-poky-linux-gnueabi --build=x86_64-linux --with-libtool-sysroot=$SDKTARGETSYSROOT"
export CFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types "
export CXXFLAGS=" -O2 -pipe -g -feliminate-unused-debug-types "
export LDFLAGS="-Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed"
export CPPFLAGS=""
export KCFLAGS="--sysroot=$SDKTARGETSYSROOT"
export OECORE_DISTRO_VERSION="4.9.88-2.0.0"
export OECORE_SDK_VERSION="4.9.88-2.0.0"
export ARCH=arm
export CROSS_COMPILE=arm-poky-linux-gnueabi-

# Append environment subscripts
if [ -d "$OECORE_TARGET_SYSROOT/environment-setup.d" ]; then
    for envfile in $OECORE_TARGET_SYSROOT/environment-setup.d/*.sh; do
            . $envfile
    done
fi
if [ -d "$OECORE_NATIVE_SYSROOT/environment-setup.d" ]; then
    for envfile in $OECORE_NATIVE_SYSROOT/environment-setup.d/*.sh; do
            . $envfile
    done
fi
```

执行make前，先要执行：

source /opt/fsl-imx-wayland/4.9.88-2.0.0/environment-setup-cortexa9hf-neon-poky-linux-gnueabi

• 5.4.1 交叉编译的构建

– 交叉编译工具链是一个由标准库、编译器、链接器、汇编器、调试器组成的集成开发环境。

– ARM平台的交叉编译工具：arm-poky-linux-gnueabi-gcc

```
arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi    -O2 -pipe -g -feliminate-unused-debug-types    -c -o hello.o hello.c
```

– 制作交叉编译工具链的方法：

① 从头编译

② 脚本编译

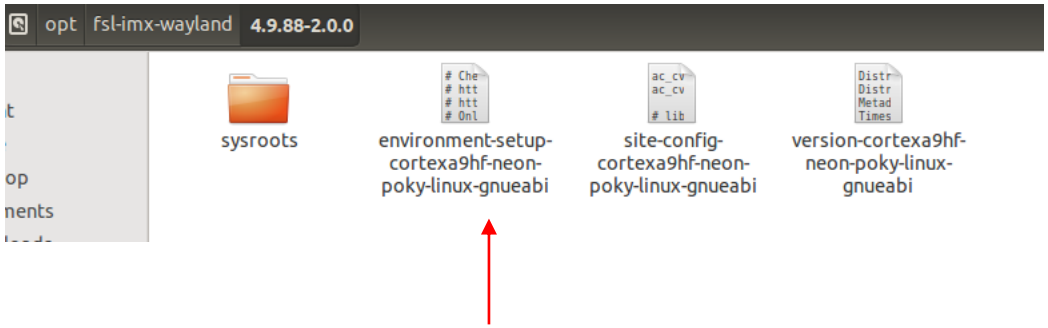
③ 下载使用

IMX6实验箱交叉编译工具链的制作

- 解压fsl-6dl-source.tar.gz文件：
  - tar -zxvf fsl-6dl-source.tar.gz -C /home/uptech/

- 进入sdk目录，执行：
  - cd /home/uptech/fsl-6dl-source/sdk
  - ./fsl-imx-wayland-glibc-x86_64-meta-toolchain-qt5-cortexa9hf-neon-toolchain-4.9.88-2.0.0.sh

- 则将交叉编译工具链安装到/opt/fsl-imx-wayland/4.9.88-2.0.0目录下：



source /opt/fsl-imx-wayland/4.9.88-2.0.0/environment-setup-cortexa9hf-neon-poky-linux-gnueabi

- **5.4.2 相关工具**

  - **1、glibc（标准库）**
    - **glibc是GNU发布的libc库，即C运行库（GNU C Library）**
    - **glibc是Linux系统中最底层的API**

  - **2、gcc（编译器）**
    - **gcc（GNU Compiler Collection，GNU编译器套件），是由 GNU 开发的编程语言编译器**
    - **gcc编译过程的4个阶段：**
      ① 预处理：生成“*.i”文件
      ② 汇编：用as命令，编译源文件，生成汇编文件（“*.s”文件）
      ③ 编译：用cc命令，生成目标文件（“*.o”文件）
      ④ 链接：用ld命令，生成可执行文件

① 预处理（-E）：

- cd /imx6/whzeng/hello-pc
- gcc -E -o hello-pc.i hello-pc.c
- 生成“hello-pc.i”文件

② 汇编（-S）：

- cd /imx6/whzeng/hello-pc
- gcc -S -o hello-pc.s hello-pc.i
- 生成“hello-pc.s”文件（汇编文件，即汇编语言程序）

③ 编译（-c）：

- cd /imx6/whzeng/hello-pc
- gcc -c -o hello-pc.o hello-pc.s
- 生成“hello-pc.o”文件（目标文件）

④ 链接（-o）：

- cd /imx6/whzeng/hello-pc
- gcc -o hello-pc hello-pc.o
- 生成“hello-pc”文件（可执行文件）

– 执行可执行文件：

- cd /imx6/whzeng/hello-pc
- ./hello-pc
  - Hello, world!

hello-pc.i

hello.i

```
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 374 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
# 385 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 386 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
# 375 "/usr/include/features.h" 2 3 4
# 398 "/usr/include/features.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4
# 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4
# 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4
# 399 "/usr/include/features.h" 2 3 4
# 28 "/usr/include/stdio.h" 2 3 4




# 1 "/usr/lib/gcc/x86_64-linux-gnu/4.8/include/stddef.h" 1 3 4
# 212 "/usr/lib/gcc/x86_64-linux-gnu/4.8/include/stddef.h" 3 4
typedef long unsigned int size_t;
# 34 "/usr/include/stdio.h" 2 3 4

# 1 "/usr/include/x86_64-linux-gnu/bits/types.h" 1 3 4
# 27 "/usr/include/x86_64-linux-gnu/bits/types.h" 3 4
# 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
# 28 "/usr/include/x86_64-linux-gnu/bits/types.h" 2 3 4


typedef unsigned char __u_char;
typedef unsigned short int __u_short;
typedef unsigned int __u_int;
typedef unsigned long int __u_long;


typedef signed char __int8_t;
typedef unsigned char __uint8_t;
typedef signed short int __int16_t;
```

预处理文件

hello-pc.s

hello.i × hello.s ×

```
        .file   "hello.c"
        .section        .rodata
.LC0:
        .string "hello world! "
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4"
        .section        .note.GNU-stack,"",@progbits
```

汇编语言程序

- gcc常用编译选项：

✓ -E：只进行预处理，不编译　　第一步：预处理

✓ -S：只汇编，不编译　　第二步：汇编

✓ -c：只编译、汇编，不链接　　第三步：编译，“-c”表示编译

✓ -g：包含调试信息

✓ -I：指定include包含文件的搜索目录

✓ -o：输出成指定文件名　　第四步：链接，“-o”表示输出

✓ -v：详细输出编译过程中所采用的每一个选项

✓ -C：预处理时保留注释信息

✓ -ggdb：在可执行文件中包含可供GDB使用的调试信息

✓ -fverbose-asm：在编译成汇编语言时，把C变量的名称作为汇编语言中的注释

✓ -save-temps：自动输出预处理文件、汇编文件、对象文件，编译正常进行

✓ -fsyntax-only：只测试源文件语法是否正确，不会进行任何编译操作

✓ -ffreestanding：编译成独立程序，而非宿主程序

gcc 使用入门教程

- 例1：一个源文件（hello-pc.c）

  - ✓ cd /imx6/whzeng/hello-pc
  - ✓ gcc -g -Wall -o hello-pc hello-pc.c
  - ✓ ./hello-pc
    - ✓ Hello, world!

  - ✓ -g：表示在生成的目标文件中带调试信息。
  - ✓ -Wall：选项 -Wall 开启编译器几乎所有常用的警告—强烈建议你始终使用该选项。
  - ✓ -o：机器码的文件名是通过 -o 选项指定的。该选项通常作为命令行中的最后一个参数。如果被省略，输出文件默认为“a.out”。
  - ✓ ./：路径 ./ 指代当前目录，因此 ./hello-pc 载入并执行当前目录下的可执行文件“hello-pc”。

- 例2：多个源文件（test_main.c，test_sum.c）

  - ✓ cd /imx6/whzeng/test
  - ✓ gcc -g -Wall -o test-pc test_main.c test_sum.c

同学们可以在自己电脑的Ubuntu环境下测试一下！

test_main.c

```
#include <stdio.h>
#include <stdlib.h>

extern int sum(int value);

struct inout {
   int value;
   int result;
};


int main(int argc, char * argv[])
{
   struct inout * io = (struct inout * ) malloc(sizeof(struct inout));

   if (NULL == io) {
      printf("Malloc failed!\n");
      return -1;
   }

   if (argc != 2) {
      printf("Wrong parameter!\n");
      return -1;
   }

   io -> value = *argv[1] - '0';
   io -> result = sum(io -> value);

   printf("Your enter: %d, result:%d\n", io -> value, io -> result);

   return 0;
}
```

```
root@uptech-virtual-machine:/imx6/whzeng/test# ./test-pc
Wrong parameter!

root@uptech-virtual-machine:/imx6/whzeng/test# ./test-pc 2
Your enter: 2, result:3

root@uptech-virtual-machine:/imx6/whzeng/test# ./test-pc 5
Your enter: 5, result:15

root@uptech-virtual-machine:/imx6/whzeng/test# ./test-pc 58
Your enter: 5, result:15

root@uptech-virtual-machine:/imx6/whzeng/test# ./test-pc 8
Your enter: 8, result:36


root@uptech-virtual-machine:/imx6/whzeng/test# ./test-pc 12
Your enter: 1, result:1
```

1+2=3

1+2+3+
4+5=15

1+2+3+
4+5=15

1+2+3+
4+5+6+
7+8=36

test_sum.c

```
int sum(int value)
{
   int result = 0;
   int i = 0;

   for (i = 0; i < value; i++)
      result += (i + 1);

   return result;
}
```

– 3、binutils（链接器+汇编器）

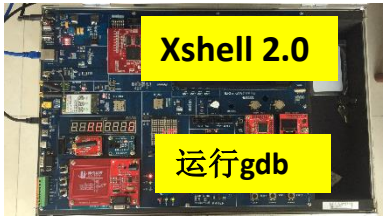
- binutils是与gcc配套的工具集，binutils工具集中的部分工具除了被gcc在后台使用为我们创建程序文件之外，其他则有助于方便开发和调试。

- binutils主要包括：
  ① addr2line：指令地址翻译器，用于得到程序指令地址所对应的函数，以及函数所在的源文件名和行号。
  ② ar：静态库生成器，用于创建和修改档案文件，以及从档案文件中抽取文件。静态库（.a文件）就是一种档案文件，需要用它生成和管理。
  ③ as：汇编编译器，用于将汇编代码转换为目标文件。
  ④ ld：链接器。
  ⑤ nm：符号显示器，用于列出程序文件中的符号及符号在内存中（开始）地址，符号包含C程序中的函数名和变量名。
  ⑥ objdump：信息查看器，能显示程序文件的相关信息和对程序文件进行反汇编。
  ⑦ objcopy：段剪辑器，可以用来从程序文件中拷贝出我们所指定的段；在将引导加载器烧至闪存中时，有时需要通过从程序中抽取段的方式生成烧写文件，这时objcopy工具就能派上用场。
  ⑧ ranlib：库索引生成器，用于生成一个档案文件的内容索引，以加快对档案文件的查找速度；将该工具运用与静态库能提高库参与链接的效率。
  ⑨ readelf：显示有关 ELF二进制文件的信息，通过readelf -h *.exe进行查看。

5.5 本地调试（gdb）

- gdb：GNU Debugger，GNU调试器

- 本地调试：
  - 调试ARM可执行文件（实验箱上运行的可执行文件）
    - 在“串口超级终端Xshell 2.0”上运行：gdb
    - 此时“本地”是指实验箱



  - 调试x86可执行文件（Ubuntu上运行的可执行文件）
    - 在Ubuntu的“终端”上运行：gdb
    - 此时“本地”是指PC机（Ubuntu）

- **1、调试ARM可执行文件：**

  - 生成带调试信息（**-g**）的可执行文件，在**Ubuntu**的“终端”上执行：
    - **cd /imx6/whzeng/test**
    - **make**
      - **arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi -O2 -pipe -g -feliminate-unused-debug-types -c -o test_main.o test_main.c**
      - **arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi -O2 -pipe -g -feliminate-unused-debug-types -c -o test_sum.o test_sum.c**
      - **arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi -o gdbtest test_main.o test_sum.o**
    - 将生成“**gdbtest**”可执行文件

test_main.c

```c
#include <stdio.h>
#include <stdlib.h>

extern int sum(int value);

struct inout {
    int value;
    int result;
};

int main(int argc, char * argv[])
{
    struct inout * io = (struct inout * ) malloc(sizeof(struct inout));

    if (NULL == io) {
        printf("Malloc failed!\n");
        return -1;
    }

    if (argc != 2) {
        printf("Wrong parameter!\n");
        return -1;
    }

    io -> value = *argv[1] - '0';
    io -> result = sum(io -> value);

    printf("Your enter: %d, result:%d\n", io -> value, io -> result);

    return 0;
}
```

test_sum.c

```
int sum(int value)
{
    int result = 0;
    int i = 0;

    for (i = 0; i < value; i++)
        result += (i + 1);

    return result;
}
```

Makefile

```
CC=arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/fsl-imx-wayland/4.9.88-2.0.0/sysroots/cortexa9hf-neon-poky-linux-gnueabi

EXP_INSTALL = install -m 755

INSTALL_DIR = ../bin

EXEC = ./gdbtest

OBJS = test_main.o test_sum.o

all: $(EXEC)

$(EXEC): $(OBJS)
	$(CC) -o $@ $(OBJS)

install:
	$(EXP_INSTALL) $(EXEC) $(INSTALL_DIR)

clean:
	-rm -f $(EXEC) *.elf *.gdb *.o
```

– 在实验箱的“超级终端（Xshell 2.0）”下运行gdb调试工具：

- mount -t nfs 192.168.33.129:/imx6 /mnt
  - 192.168.33.129：虚拟机的IP地址
- cd /mnt/whzeng/test
- 方法一：gdb gdbtest
- 方法二：gdb
  - GNU gdb (GDB) 7.7.1
  - Copyright (C) 2014 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 "arm-poky-linux-gnueabi".
  - 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 gdbtest
  - Reading symbols from hello...(no debugging symbols found)...done.
  - (gdb) quit

file命令：装入调试的可执行文件

quit命令：退出gdb

list命令（或者l命令）：查看源程序

```
(gdb) list
3
4       extern int sum(int value);
5
6       struct inout {
7           int value;
8           int result;
9       };
10
11      int main(int argc, char * argv[])
12      {
(gdb)
13          struct inout * io = (struct inout * ) malloc(sizeof(struct inout));
14
15          if (NULL == io) {
16              printf("Malloc failed!\n");
17              return -1;
```

```
(gdb) l
3
4           extern int sum(int value);
5
6           struct inout {
7               int value;
```

```
(gdb) list sum
1       int sum(int value)
2       {
3           int result = 0;
4           int i = 0;
5
6           for (i = 0; i < value; i++)
7               result += (i + 1);
8
9           return result;
10      }
(gdb) l sum
1       int sum(int value)
2       {
3           int result = 0;
4           int i = 0;
5
6           for (i = 0; i < value; i++)
7               result += (i + 1);
8
9           return result;
10      }
(gdb)
```

查看函数sum：list sum 或者 l sum

break命令（或者b命令）：设置断点

break main

```
(gdb) break main
Breakpoint 1 at 0x8358: file test_main.c, line 12.
(gdb)
```

b main

b test_sum.c:sum

```
(gdb) b main
Breakpoint 1 at 0x8358: file test_main.c, line 12.
(gdb) b test_sum.c:sum
Breakpoint 2 at 0x84f8: f                    line 6.
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x00008358 in main at test_main.c:12
2       breakpoint     keep y   0x000084f8 in sum at test_sum.c:6
(gdb)
```

info break　　查看设置的断点

delete 1　　删除第1个断点

```
(gdb) delete 1
(gdb) info break
Num     Type           Disp Enb Address    What
2       breakpoint     keep y   0x000084f8 in sum at test_sum.c:6
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) info break
No breakpoints or watchpoints.
(gdb)
```

delete　　删除所有断点

```
(gdb)
14          struct inout * io = (struct inout * ) malloc(sizeof(struct inout));
15
16          if (NULL == io) {
17              printf("Malloc failed!\n");
18              return -1;
19          }
20
21          if (argc != 2) {
22              printf("Wrong parameter!\n");
23              return -1;
(gdb) b 16
Breakpoint 1 at 0x836c: file test_main.c, line 16.
(gdb) b 21
Breakpoint 2 at 0x8[illegible] test_main.c, line 21.
(gdb) info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x0000836c in main at test_main.c:16
2       breakpoint     keep y   [illegible] in main at test_main.c:21
(gdb)
```

break 16

break 17

info b 查看设置的断点

run命令（或者r命令）：运行程序

```
(gdb) run
Starting program: /mnt/whzeng/gdbtest/gdbtest
Cannot access memory at address 0x0
Wrong parameter!
[Inferior 1 (process 875) exited with code 0377]
(gdb) r
Starting program: /mnt/whzeng/gdbtest/gdbtest
Cannot access memory at address 0x0
Wrong parameter!
[Inferior 1 (process 878) exited with code 0377]
(gdb) run 3
Starting program: /mnt/whzeng/gdbtest/gdbtest 3
Cannot access memory at address 0x0
Your enter: 3, result:6
[Inferior 1 (process 879) exited normally]
(gdb) r 3
Starting program: /mnt/whzeng/gdbtest/gdbtest 3
Cannot access memory at address 0x0
Your enter: 3, result:6
[Inferior 1 (process 882) exited normally]
(gdb)
```

该程序的执行需要带参数

continue （c命令）：从断点处继续运行程序

```
(gdb) r 3
Starting program: /mnt/whzeng/gdbtest/gdbtest 3
Cannot access memory at address 0x0

Breakpoint 1, main (argc=2, argv=0x7efffdb4) at test_main.c:12
12      {
(gdb) c
Continuing.

Breakpoint 2, sum (value=value@entry=3) at test_sum.c:6
6           for (i = 0; i < value; i++)
(gdb) c
Continuing.
Your enter: 3, result:6
[Inferior 1 (process 911) exited normally]
(gdb)
```

next命令（或者n命令）：单步执行

```
(gdb) r 5
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /mnt/whzeng/gdbtest/gdbtest 5
Cannot access memory at address 0x0

Breakpoint 4, main (argc=2, argv=0x7efffdb4) at test_main.c:14
14          struct inout * io = (struct inout * ) malloc(sizeof(struct inout));
(gdb) next
13      {
(gdb) next
14          struct inout * io = (struct inout * ) malloc(sizeof(struct inout));
(gdb) next
16          if (NULL == io) {
(gdb) next
21          if (argc != 2) {
(gdb) next
26          io -> value = *argv[1] - '0';
(gdb) next
27          io -> result = sum(io -> value);
(gdb) next
29          printf("Your enter: %d, result:%d\n", io -> value, io -> result);
(gdb) next
Your enter: 5, result:15
31          return 0;
(gdb)
```

```
(gdb) r 5
Starting program: /mnt/whzeng/gdbtest/gdbtest 5
Cannot access memory at address 0x0

Breakpoint 4, main (argc=2, argv=0x7efffdb4) at test_main.c:14
14          struct inout * io = (struct inout * ) malloc(sizeof(struct inout));
(gdb) n
13      {
(gdb) n
14          struct inout * io = (struct inout * ) malloc(sizeof(struct inout));
(gdb) n
16          if (NULL == io) {
(gdb) n
21          if (argc != 2) {
(gdb) n
26          io -> value = *argv[1] - '0';
(gdb) n
27          io -> result = sum(io -> value);
(gdb) n
29          printf("Your enter: %d, result:%d\n", io -> value, io -> result);
(gdb) n
Your enter: 5, result:15
31          return 0;
(gdb)
```

print（或者p）命令：查看变量的值

```
(gdb) print result
$18 = 0
(gdb) n
7               result += (i + 1);
(gdb) n
6           for (i = 0; i < value; i++)
(gdb) n
7               result += (i + 1);
(gdb) n
6           for (i = 0; i < value; i++)
(gdb) n
7               result += (i + 1);
(gdb) n
6           for (i = 0; i < value; i++)
(gdb) print result
$19 = 3
```

```
(gdb) p result
$15 = 3
(gdb) n
7               result += (i + 1);
(gdb) n
6           for (i = 0; i < value; i++)
(gdb) n
7               result += (i + 1);
(gdb) n
6           for (i = 0; i < value; i++)
(gdb) p result
$16 = 6
(gdb) n
7               result += (i + 1);
(gdb) n
6           for (i = 0; i < value; i++)
(gdb) n
7               result += (i + 1);
(gdb) n
6           for (i = 0; i < value; i++)
(gdb) p result
$17 = 10
```

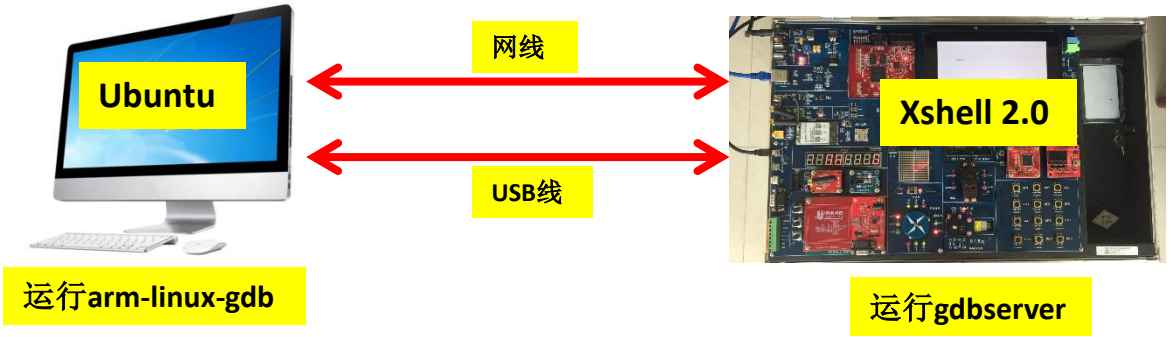
gdb的常用命令

命令	命令缩写	说明
list	l	显示多行源代码
break	b	设置断点,程序运行到断点的位置会停下来
info	i	描述程序的状态
run	r	开始运行程序
display	disp	跟踪查看某个变量,每次停下来都显示它的值
step	s	执行下一条语句,如果该语句为函数调用,则进入函数执行其中的第一条语句
next	n	执行下一条语句,如果该语句为函数调用,不会进入函数内部执行(即不会一步步地调试函数内部语句)
print	p	打印内部变量值
continue	c	继续程序的运行,直到遇到下一个断点
set var name=v		设置变量的值
start	st	开始执行程序,在main函数的第一条语句前面停下来
file		装入需要调试的程序
kill	k	终止正在调试的程序
watch		监视变量值的变化
backtrace	bt	产看函数调用信息(堆栈)
frame	f	查看栈帧
quit	q	退出GDB环境

- **2、调试x86可执行文件：**
  - 在Ubuntu的“终端”下执行（-g表示包含调试信息）：
    - cd /imx6/whzeng/test
    - gcc -g -Wall -o test-pc test_main.c test_sum.c
    - 将生成“test-pc”可执行文件
    - 方法一：gdb test-pc
    - 方法二：gdb
      - GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
      - Copyright (C) 2014 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 "x86_64-linux-gnu".
      - 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"...
      - Reading symbols from gdbtest-pc...done.
      - (gdb) file test-pc
      - Reading symbols from hello...(no debugging symbols found)...done.
      - (gdb) quit
  - 其他命令同实验箱的“超级终端”环境下

5.6 远程调试（gdb + gdbserver）

- 远程调试：
  - 用于调试ARM程序，即在Ubuntu（宿主机）上调试运行在实验箱（目标机）上的程序
  - 在实验箱的“Xshell 2.0超级终端”上运行：gdbserver
  - 在Ubuntu的“终端”上运行：arm-linux-gdb



- # 第一步：准备arm-linux-gdb工具

  - ## 1、下载gdb-7.12.tar.gz
    - 将下载的gdb-7.12.tar.gz拷贝到Ubuntu的/imx6/whzeng目录下

  - ## 2、解压gdb-7.12.tar.gz
    - cd /imx6/whzeng
    - tar zxvf gdb-7.12.tar.gz

  - ## 3、修改/imx6/whzeng/gdb-7.12/gdb/remote.c文件的内容
    - chmod 777 -R /imx6/whzeng
    - 点击/imx6/whzeng/gdb-7.12/gdb/remote.c文件，打开该文件，按照以下方式进行修改：
    - 屏蔽process_g_packet函数中的下列两行：

```
// if (buf_len > 2 * rsa->sizeof_g_packet)
//    error (_("Remote 'g' packet reply is too long: %s"), rs->buf);
```

    - 在其后添加：

```
if (buf_len > 2 * rsa->sizeof_g_packet) {
    rsa->sizeof_g_packet = buf_len ;
    for (i = 0; i < gdbarch_num_regs (gdbarch); i++)
    {
      if (rsa->regs[i].pnum == -1)
      continue;

      if (rsa->regs[i].offset >= rsa->sizeof_g_packet)
      rsa->regs[i].in_g_packet = 0;
      else
      rsa->regs[i].in_g_packet = 1;
    }
  }
```

- **4、编译：**
  - **cd /imx6/whzeng/gdb-7.12**
  - **./configure --target=arm-linux --prefix=$PWD/__install**
  - **make**
  - **make install**

- **5、将编译好的工具（arm-linux-gdb）拷贝到/usr/bin目录中：**
  - **cp /imx6/whzeng/gdb-7.12/__install/bin/arm-linux-gdb /usr/bin/**

- 第二步：在实验箱上运行gdbserver

  - 在实验箱的“超级终端”上执行：
    - mount -t nfs 192.168.33.129:/imx6 /mnt
    - cd /mnt/whzeng/test
    - gdbserver 192.168.33.129:6666 gdbtest
      - Process gdbtest created; pid = 805
      - Listening on port 6666
    - “gdbtest”为可执行文件

- 第三步：在Ubuntu上运行arm-linux-gdb
  - 在Ubuntu的“终端”上执行：
    - cd /imx6/whzeng/test
    - arm-linux-gdb gdbtest

“gdbtest”为可执行文件

```
GNU gdb (GDB) 7.12
Copyright (C) 2016 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-pc-linux-gnu --target=arm-linux".
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"...
Reading symbols from test...done.
(gdb) r
Don't know how to run.  Try "help target".
(gdb) target remote 192.168.33.155:6666
Remote debugging using 192.168.0.138:6666
warning: Can not parse XML target description; XML support was disabled at compile time
Reading /lib/ld-linux-armhf.so.3 from remote target...
warning: File transfers from remote targets can be slow. Use "set sysroot" to access files locally instead.
Reading /lib/ld-linux-armhf.so.3 from remote target...
Reading symbols from target:/lib/ld-linux-armhf.so.3...Reading /lib/ld-2.20.so from remote target...
Reading /lib/.debug/ld-2.20.so from remote target...
(no debugging symbols found)...done.
0x76fd7b00 in ?? () from target:/lib/ld-linux-armhf.so.3
(gdb)
```

192.168.33.155：实验箱的IP地址

  - 接下去可以使用gdb的有关命令调试可执行文件“gdbtest”

– 此时，“Xshell超级终端”上将显示以下的内容：

```
[root@zx64352 /]# gdbserver 192.168.0.139:6666 /test
Process /test created; pid = 927
Listening on port 6666
Remote debugging from host 192.168.0.139
```

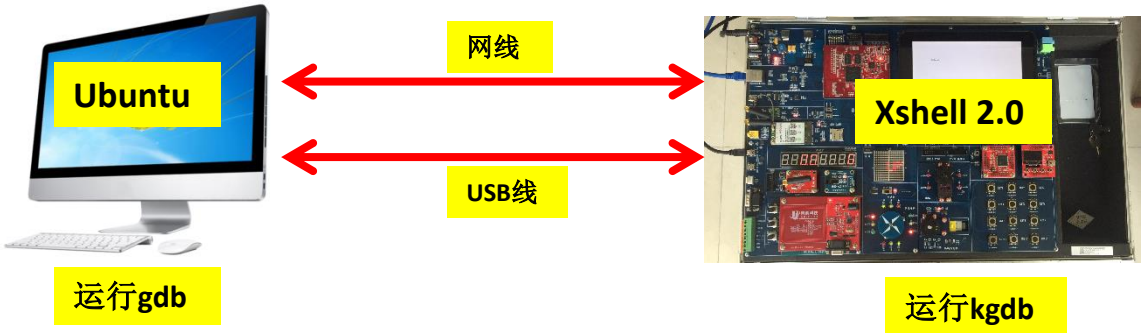
– 此时，Ubuntu“终端”上将显示以下的内容：

```
(gdb) target remote 192.168.0.138:6666
Remote debugging using 192.168.0.138:6666
warning: Can not parse XML target description; XML support was disabled at compile time
Reading /lib/ld-linux.so.3 from remote target...
warning: File transfers from remote targets can be slow. Use "set sysroot" to access files
 locally instead.
Reading /lib/ld-linux.so.3 from remote target...
Reading symbols from target:/lib/ld-linux.so.3...(no debugging symbols found)...done.
0x40000b00 in ?? () from target:/lib/ld-linux.so.3
(gdb)
```

5.7 内核调试（gdb + kgdb）

- 使用**printk**函数：调试应用程序时，可以使用printf函数显示有关信息。调试Linux内核时，则是使用printk函数显示有关信息。

- 使用**kgdb**内核调试工具：
  - 在目标机（实验箱，Xshell 2.0）上运行kgdb
  - 在宿主机（PC机，Ubuntu）上运行gdb



5.8 网络调试

- 如果嵌入式系统（目标机，实验箱）上有网络通信程序，则需要网络调试工具。
- 在传统的网络分析和调试技术中，嗅探器（Sniffer）是最常见也是最重要的一种技术。
- tcpdump是一款功能强大、截取灵活的开源嗅探器工具。
- 除了tcpdump外，还有arp、ping、route、netstat等网络调试与诊断工具。

小结

- 嵌入式系统交叉开发环境：
  - 宿主机环境（PC电脑，Vmware + Ubuntu）
  - 目标板环境（实验箱，Xshell 2.0）
  - 交叉编译环境（x86环境编译ARM程序）

- 嵌入式系统调试技术：
  - gdb：本地调试
  - gdb + gdbserver：远程调试
  - gdb + kgdb：内核调试
  - tcpdump：网络调试（网络通信程序调试）

进一步探索

- 了解GNU工具的具体使用方法。

Thanks