SEED-QUARK_X1000 软件用户手册

2015 INTEL QUARK Solution

SEED-QUARK_X1000 User Guide

概述：

文档介绍了基于 Intel Quark 平台的 Linux 服务器下的开发套件的安装配置与使用，套件下各个部件的使用介绍，以及系统下各个启动方式的配置等操作。

名称：

SEED-QUARK_X1000 软件用户指南

版本：

Version A

修改：

版本	修改时间	修正作者	修正说明
Version A	2015/1/20		文件创建

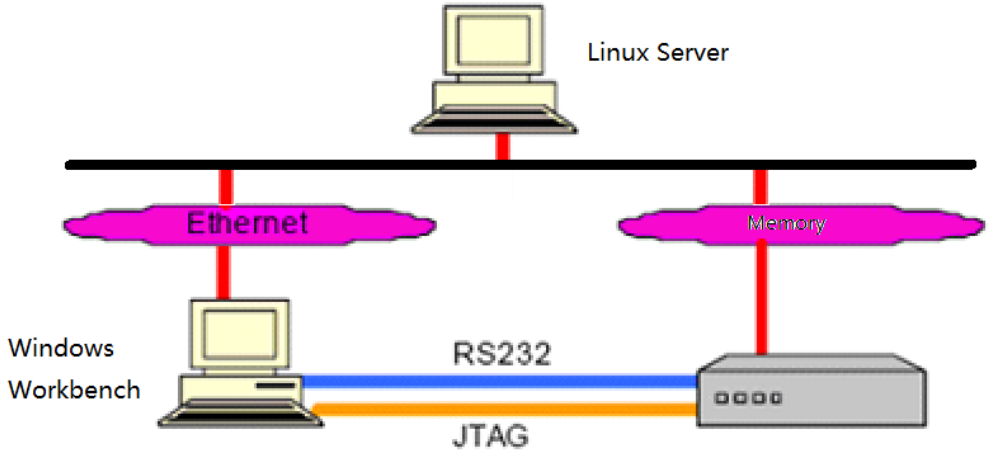
Note：任何修改操作请在上述表格备注说明。

www.seeddsp.com

目　录

第 1 章 SEED-QUARK_X1000 开发环境

SEED-QUARK_X1000 Linux 开发环境通常包括 Linux 服务器、Windows 工作台及 SEED-QUARK_X1000 平台，如下图示：



开发工程师在 Linux 服务器上建立交叉编译环境，Windows 工作台通过串口和 JTAG 与 SEED-QUARK_X1000 开发平台连接，开发人员可以在 Windows 工作台进行程序开发或者远程登陆到 Linux 服务器进行开发。

Linux 服务器搭建建议选择常用的 Linux 发行版本，便于各种资源的搜集，建议采用以下版本的 Linux 发行版：**64 位的 Ubuntu 12.04**。

Linux 服务器搭建建议选择常用的 Linux 发行版本，便于各种资源的搜集，本文档中所有测试均采用 Ubuntu 12.04。并需要一个网络连接来下载第三方的源码，需要至少 70GB 的磁盘空间去构建软件开发环境。

Linux 系统 PC 机端的安装，在此不做详细介绍，用户可以很方便的从网络上获取丰富的资源。

Note：

- **安装 Linux 系统过程中，请勿选择安装防火墙；**
- **在安装该版本之前，请删除所有已安装的其他版本；**
- **个别的组件编译时需要不同的编译选项或者其他配置，为了避免交叉影响，在编译不同部分时请每次使用不同的终端。**

第 2 章 SEED-QUARK_X1000 环境构建

SEED-QUARK_X1000 所使用的软件开发套件版本为：

bsp4_QuarkDIY_v1.0.0

开发环境构建建议完全按照以下步骤与路径进行配置，以简化后续各种配置的繁琐，除了在环境变量设置过程以 root 账号登陆 Linux 服务器，并进行配置，其他过程请**不要**在 root 账户下运行命令。

约定：

Host # 表示 Linux 开发机（服务器）控制台提示符

Target # 表示 SEED-QUARK_X1000 平台的串口控制台提示符

Note：

- **除了“apt-get install”之外的其他命令请不要在 root 账户下运行；**
- **环境搭建内容的详细文档请参考 Intel Quark 文档《Quark_BSP_BuildandSWUserGuide_329687_005.pdf》。**

2.1 环境变量设置

- 根据软件所依赖的开发工具，先安装以下工具。

```
Host # apt-get install build-essential gcc-multilib vim-common
Host # apt-get install uuid-dev iasl subversion gcc
Host # apt-get install git diffstat texinfo gawk chrpath file bitbake libssl-dev
```

- 建立相关文件路径。

```
Host # mkdir ~/quark
Host # mkdir ~/tools
```

- 拷贝光盘目录 04. Software 下的 bsp4_QuarkDIY_v1.0.0.tar.bz2 到新建的 tools 目录中，用以下命令解压。

```
Host # tar jxvf bsp4_QuarkDIY_v1.0.0.tar.bz2
```

- 解压后得到 meta-clanton-adcdiy_v1.0.0.tar.bz2、Quark_ADCDIY_EDKII_V1.0.0.tgz、spi-flash-tools_v1.0.0.tar.bz2、sysimage_v1.0.0.tar.gz。

2.2 构建 EDKII

- EDKII 是构建 QUARK 必须的开源固件。具体操作命令如下：

```
Host # cd ~/tools
Host # tar zxvf Quark_ADCDIY_EDKII_V1.0.0.tgz -C ../quark
Host # cd ~/quark/Quark_EDKII_v1.0.0
Host # ./svn_setup.py
Host # svn update
Host # ./buildallconfigs.sh GCC46 QuarkPlatform
```

Note：

- **在执行上述命令时请打开新的终端；**
- **请不要在 root 账户下运行以上命令；**
- **命令 svn update 的执行过程需要消耗几分钟的时间，具体时间的长短取决于**

您的网络连接的速度；

- **GCC46 为 GCC 版本号，本文使用 GCC46 版本的 GCC。请设置时与您主机已安装的 GCC 版本号保持一致。**

2.3 使用 Yocto 创建内核和文件系统

- 打开一个新的终端，提取出 Yocto。

```
Host # cd ~/tools
Host # tar jxvf meta-clanton-adcdiy_v1.0.0.tar.bz2 -C ../quark
Host # cd ~/quark/meta-clanton_v1.0.0
```

- 修改 setup 目录下 meta-intel.cfg 文件中的下载地址，修改地址如下。

```
url = http://git.yoctoproject.org/git/meta-intel
```

- 修改 setup 目录下 meta-oe.cfg 文件中的下载地址，修改地址如下。

```
url = https://github.com/openembedded/meta-oe
```

- 修改 setup 目录下 poky.cfg 文件中的下载地址，修改地址如下。

```
url = http://git.yoctoproject.org/git/poky
```

- 运行 setup.sh 脚本去下载构建 Yocto 所需要的外部源。

```
Host # ./setup.sh
```

- 拷贝光盘目录 04. Software 下的 downloads.tar.gz 到 ~/quark/meta-clanton_v1.0.0/yocto_build 目录中，可以新建一个终端（此终端仅用于解压 downloads.tar.gz 文件），用以下命令解压。

```
Host # cd ~/quark/meta-clanton_v1.0.0/yocto_build
Host # tar zxvf downloads.tar.gz
```

- 创建内核及文件系统等每次都必须进行初始化编译环境，初始化编译环境，使用如下命令。

```
Host #source poky/oe-init-build-env yocto_build
```

- 构建一个全功能的 linux 文件系统命令如下：

```
Host #bitbake image-full-galileo
```

- 编译生成文件所在目录为：./tmp/deploy/images/；

所需生成文件包括：

- ✓ boot (directory)
- ✓ bzImage
- ✓ core-image-minimal-initramfs-clanton.cpio.gz
- ✓ image-full-galileo-clanton.ext3
- ✓ grub.efi

- 如果文件 grub.conf 存在于 U 盘或者 SD 卡的/boot/grub/路径下，可以将生成的内核和文件系统复制到 U 盘或者 SD 中，通过 grub 来引导启动。

Note：

- **在执行上述命令时请打开新的终端；**
- **请不要在 root 账户下运行以上命令；**
- **编译全功能的 Yocto 的执行过程需要消耗几小时的时间，具体时间的长短取决于您的网络连接的速度和电脑的性能；**
- **编译完成后相应文件名会包含时间等信息，部分文件属于 linux 系统下的 link**

文件，其不能在 fat32 格式下使用，请自行修改文件名。

2.4 给相关文件进行签名

签名不是所有核心板开发的必须步骤，但是 Intel 的部分核心板卡需要对相关文件进行签名才能启动系统。具体需要签名的文件有 grub.conf(在 boot/grub 目录)、bzImage、core-image-minimal-initramfs-clanton.cpio.gz。

- 构建签名工具 sign 命令如下：

```
Host # tar -jxvf spi-flash-tools_v1.0.0.tar.bz2 -C ~/quark
Host # cd ~/quark/spi-flash-tools_v1.0.0
Host # make asset-signing-tool/sign
```

- 解压签名必备 key 文件有两个，均在在目录/sysimage_v1.0.0/config 中，分别为 key.pem 和 privkey.pem，具体需要使用哪个 key 文件，请参考核心板上的说明。本文以 key.pem 为例进行说明。解压命令如下：

```
Host # tar -zxvf sysimage_v1.0.0.tar.gz -C ~/quark
```

- 给 grub.conf 文件签名，创建 grub.conf.csbh 文件，命令如下；

```
Host # cd ~/quark/meta-clanton_v1.0.0/yocto_build/tmp/deploy/images/boot/grub
Host # ~/quark/spi-flash-tools_v1.0.0/asset-signing-tool/sign -i grub.conf -c -s 0 -x 5 -k
~/quark/sysimage_v1.0.0/config/key.pem
```

- 给 bzImage 文件签名，创建 bzImage.csbh 文件，命令如下；

```
Host # cd ~/quark/meta-clanton_v1.0.0/yocto_build/tmp/deploy/images
Host # ~/quark/spi-flash-tools_v1.0.0/asset-signing-tool/sign -i bzImage -c -s 0 -x 6 -k
~/quark/sysimage_v1.0.0/config/key.pem
```

- 给 core-image-minimal-initramfs-clanton.cpio.gz 文件签名，创建 core-image-minimal-initramfs-clanton.cpio.gz.csbh 文件，命令如下。

```
Host # cd ~/quark/meta-clanton_v1.0.0/yocto_build/tmp/deploy/images
Host # ~/quark/spi-flash-tools_v1.0.0/asset-signing-tool/sign -i
core-image-minimal-initramfs-clanton.cpio.gz -c -s 0 -x 7 -k
~/quark/sysimage_v1.0.0/config/key.pem
```

- 为保证系统的正常启动，所有的签名文件都需要拷贝到 SD 卡中，并遵循以下规范：
  - 每一个.csbh 后缀的文件均需要和相应的未签名文件放在一起；
  - grub.conf 必须放在目录/boot/grub 下；

Note:

- **在执行上述命令时请打开新的终端；**
- **请不要在 root 账户下运行以上命令；**
- **如果按照上述步骤签名后，系统仍然无法从 micro SD 卡启动，请联系技术支持人员。**

2.5 构建 Linux 交叉编译工具链

- 构建交叉工具链命令如下：

```
Host # source poky/oe-init-build-env yocto_build
Host # bitbake image-full –c populate_sdk
```

```
Host # ./tmp/deploy/sdk/clanton-tiny-uclibc-x86_64-i586-toolchain-1.4.2.sh
```

- 生成文件为“/opt/clanton-tiny/1.4.2/environment-setup-i586-poky-linux-uclibc”；

Note:

- **在执行上述命令时请打开新的终端；**
- **请不要在 root 账户下运行以上命令；**
- **每次编译交叉编译工具链时，打开新的超级终端必须先运行 “source poky/oe-init-build-env yocto_build”。**

2.6 应用程序编译

光盘中提供 ARROW SEED 编写的应用程序源码，在光盘文件的目录 04. Software\seed-quark 下。包括 seed-can-api.c、seed-can-api.h、seed-quark.c、seed-quark-fpga.c、seed-quark-fpga.h、seed-spi-api.c、seed-spi-api.h、seed-uart-api.c、seed-uart-api.h、seed-zigbee-api.h 文件。

- 交叉编译工具链的使用：

- 当您编译应用程序之前，首先要定义 CC，CONFIGURE_FLAGS 和其它环境变量的默认值，命令如下：

```
Host # source /opt/clanton-tiny/1.4.2/environment-setup-i586-poky-linux-uclibc
```

- 编译应用程序，命令如下：

```
Host # $CC myfile.c -o myfile
```

- 使用光盘中提供的源码，编译应用程序，命令如下：

```
Host # $CC seed-can-api.c seed-can-api.h seed-quark.c seed-quark-fpga.c
seed-quark-fpga.h seed-spi-api.c seed-spi-api.h seed-uart-api.c seed-uart-api.h
seed-zigbee-api.h -o seed-quark
```

Note:

- **在执行上述命令时请打开新的终端；**
- **请不要在 root 账户下运行以上命令。**

第 3 章 SD 卡启动开发板

从 SD 卡启动开发板与从 U 盘启动开发板的具体步骤都可以参考以下命令。

3.1 SD 卡要满足以下条件

SD 卡必须是 FAT32 格式。
SD 卡的大小必须<=32GB。

3.2 启动步骤

- 复制相应的文件到 SD 卡中，包括：
  - ✓ boot (directory)
  - ✓ bzImage
  - ✓ core-image-minimal-initramfs-clanton.cpio.gz
  - ✓ image-full-galileo-clanton.ext3
  - ✓ grub.efi

最终 SD 卡中包含如下的结构和文件：

Note:

- **需要将对应文件的签名文件一同拷贝到 SD 卡中；**
- **包括 core-image-minimal-initramfs-clanton.cpio.gz.csbh、grub.conf.csbh、bzImage.csbh 三个文件。**
- **拷贝章节 2.6 中编译的应用程序 seed-quark 到 Micro SD 卡根目录下；**
- **将 SD 卡插入开发板，给开发板上电。**

第 4 章 SEED-QUARK_X1000 程序测试

SEED-QUARK_X1000 平台测试程序提供了网络、USB、WIFI 等测试例程，实现对 SEED-QUARK_X1000 平台各个外设的测试功能。

4.1 串口控制台搭建

用户通过主机端串口同 SEED-QUARK_X1000 进行交互，对 SEED-QUARK_X1000 进行控制并运行程序等。主机端串口使用 PC 机 windows 系统自带的超级终端即可。配置如下：

- 点击 PC 机左下角开始-->程序-->附件-->通讯-->超级终端；
- 在“您的区号（或城市号）是什么（C）？”下键入 010 后点击确定；
- 点击确定，在新弹出的对话框中输入你喜欢的名称，如 quark；
- 在新的对话框中的“连接时使用”下选择你希望使用的串口设备，点击确定；

在端口设置选项中配置波特率 115200，数据位 8，奇偶校验无，停止位 1，数据流控制无。

4.2 硬件连接

SEED-QUARK_X1000 平台软件测试时，硬件统一连接如下：

- SEED-QUARK_X1000_IO 的 Micro SD 卡座 CON6 插入烧写好的 Micro SD 卡；
- SEED-QUARK_X1000_IO 的 9 针串口 CON3 通过串口线与测试主机相连；
- SEED-QUARK_X1000_IO 的电源座 CON12 连接好 5V 电源。

4.3 开发板启动准备

- 通过按压底板上的 SW1 POWER 键给核心板上电。等待 1 分钟左右，待串口终端中打印如下内容后，启动过程结束；

```
Poky 9.0.2 (Yocto Project 1.4 Reference Distro) 1.4.2 clanton /dev/ttyS1

clanton login:
```

- 在串口终端中输入 root，点击回车，登陆开发板。

```
clanton login: root
root@clanton:~#
```

4.4 软件程序测试

本测试中网络、USB 采用单独测试方法，其他外设模块与接口则通过应用程序 seed-quark 进行测试。

Note：

- **SEED-Quark_X1000 应用测试中的 Can 接口、PWM 接口暂不提供，如需支持请联系各地 Sales 咨询。**

4.4.1 网络测试

4.4.1.1 eth0 测试

A）测试准备：

- 按照本章中 4.2 内容连接好硬件；
- 使用网线将底板的 CON1 网络接口与测试主机的网络接口相连；
- 按照本章中 4.3 内容启动开发板；
- 在终端中输入如下命令进入网络配置文件；

```
Target # vi /etc/network/interfaces
```

- 本例程使用静态 IP“192.168.1.160”进行测试；
- 在打开的配置文件中找到如下内容：

```
# Wired or wireless interfaces
auto eth0
iface eth0 inet dhcp
iface eth1 inet dhcp
```

- 屏蔽以上提到的内容，如下：

```
# Wired or wireless interfaces
#auto eth0
#iface eth0 inet dhcp
#iface eth1 inet dhcp
```

- 设置 eth0 的静态 IP：

```
auto eth0
iface eth0 inet static
address 192.168.1.160
gateway 192.168.1.1
netmask 255.255.255.0
```

Note：

- **以上所设置的 IP 地址保持与测试主机的 IP 地址处于同一网段即可。**

- 保存，退出该配置文件；
- 执行如下命令。

```
Target # /etc/init.d/networking restart
```

B）测试过程

本次例程中，当前测试主机的 IP 地址为 192.168.1.137，在终端中输入 ping 命令，如果能够 ping 通，网络 eth0 测试通过。

```
root@clanton:~# ping 192.168.1.137
PING 192.168.1.137 (192.168.1.137): 56 data bytes
64 bytes from 192.168.1.137: seq=0 ttl=64 time=2.224 ms
64 bytes from 192.168.1.137: seq=1 ttl=64 time=1.193 ms
64 bytes from 192.168.1.137: seq=2 ttl=64 time=1.214 ms
```

4.4.1.2 eth1 测试

A）测试准备：

- 按照本章中 4.2 内容连接好硬件；
- 使用网线将底板的 CON2 网络接口与测试主机的网络接口相连；

- 按照本章中 4.3 内容启动开发板；
- 在终端中输入如下命令进入网络配置文件；

```
Target # vi /etc/network/interfaces
```

- 本例程使用静态 IP“10.115.140.74”进行测试；
- 在打开的配置文件中找到如下内容：

```
# Wired or wireless interfaces
auto eth0
iface eth0 inet dhcp
iface eth1 inet dhcp
```

- 屏蔽以上提到的内容，如下：

```
# Wired or wireless interfaces
#auto eth0
#iface eth0 inet dhcp
#iface eth1 inet dhcp
```

- 设置 eth1 的静态 IP：

```
auto eth1
iface eth1 inet static
address 10.115.140.180
gateway 10.115.140.1
netmask 255.255.252.0
```

Note：

- **以上所设置的 IP 地址保持与测试主机的 IP 地址处于同一网段即可。**

- 保存，退出该配置文件；
- 执行如下命令。

```
Target # /etc/init.d/networking restart
```

B）测试过程

本次例程中，当前测试主机的 IP 地址为 10.115.140.74，在终端中输入 ping 命令，如果能够 ping 通，网络 eth1 测试通过。

```
root@clanton:~# ping 10.115.140.74
PING 10.115.140.74 (10.115.140.74): 56 data bytes
64 bytes from 10.115.140.74: seq=0 ttl=128 time=3.852 ms
64 bytes from 10.115.140.74: seq=1 ttl=128 time=2.241 ms
64 bytes from 10.115.140.74: seq=2 ttl=128 time=2.269 ms
```

4.4.1.3 eth0 eth1 同时测试

- 两个网络接口可同时使用，但要确保两个网络接口的 IP 设置处于不同的网段，并且与相对应的测试主机保持在同一个网段内。配置文件修改如下：

```
# Wired or wireless interfaces
#auto eth0
#iface eth0 inet dhcp
#iface eth1 inet dhcp
auto eth0
iface eth0 inet static
address 192.168.1.160
gateway 192.168.1.1
```

```
netmask 255.255.255.0
auto eth1
iface eth1 inet static
address 10.115.140.180
gateway 10.115.140.1
netmask 255.255.252.0
```

- 配置完 eth0 与 eth1 的 IP 地址等信息后，如果要在首次启动板卡时使用双网络，则需要在启动板卡后运行如下命令：

```
Target # /etc/init.d/networking restart
```

4.4.2 USB 测试

4.4.2.1 USB2.0 主设备端测试

A）测试准备：

- 准备 USB 设备一个；
- 按照本章中 4.2 内容连接好硬件；
- 按照本章中 4.3 内容启动开发板。

B）测试过程

- 将 USB 设备插入 USB 接口 CON4 中；
- 串口终端打印如下信息，证明 USB-CON4 测试通过。

```
root@clanton:~# [  119.290167] usb 2-1: new high-speed USB device number 2 using ehci-pci
[  119.461228] scsi0 : usb-storage 2-1:1.0
[  120.897958] scsi 0:0:0:0: Direct-Access     SMI      USB DISK         1100 PQ: 0 ANSI: 0
 CCS
[  120.926634] sd 0:0:0:0: [sda] 8184480 512-byte logical blocks: (4.19 GB/3.90 GiB)
[  120.941516] sd 0:0:0:0: [sda] Write Protect is off
[  120.951889] sd 0:0:0:0: Attached scsi generic sg0 type 0
[  120.971657] sd 0:0:0:0: [sda] No Caching mode page present
[  120.977240] sd 0:0:0:0: [sda] Assuming drive cache: write through
[  121.003597] sd 0:0:0:0: [sda] No Caching mode page present
[  121.009179] sd 0:0:0:0: [sda] Assuming drive cache: write through
[  121.023220]  sda: sda1
[  121.036769] sd 0:0:0:0: [sda] No Caching mode page present
[  121.042419] sd 0:0:0:0: [sda] Assuming drive cache: write through
[  121.048623] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

4.4.2.2 USB2.0 从设备端测试

A）测试准备：

- 准备 Mini B 型 USB 线缆一根。

B）测试过程：

- 按照本章中 4.2 内容连接好硬件；
- 按照本章中 4.3 内容启动开发板；
- 在测试主机上安装好 USB 驱动程序。安装过程如下：

1）将 USB 线缆的一端接入测试 PC 主机，一端接入开发的 USB 的 CON5 接口。等待 Windows 的设备驱动安装过程，数分钟后，显示程序安装失败；

2）点击 PC 机上的开始菜单打开控制面板,点击进入系统和安全界面，在该界面中找到系统选项点击进入，在左侧的列表中点击进入设备管理器；

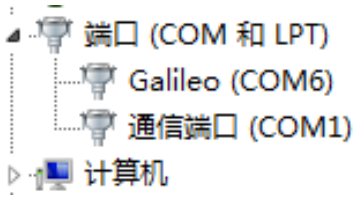
3）向下打开端口（COM 和 LPT）,可以看见名为 Gadget Serial v2.4 的端口；

4）右键单击 Gadget Serial v2.4 端口，并选择更新驱动程序软件；

5）在弹出的对话框中选择浏览计算机以查找驱动程序软件（R）；

6) 在弹出的对话框中点击浏览，找到 drivers 目录，点击下一步，正确的驱动文件 linux-cdc-acm.inf 会被安装；

7）当驱动正确安装后，在设备管理器中向下打开端口（COM 和 LPT）,可以看见名为 Galileo（COMx）的设备；



此时，SEED-QUARK_X1000 作为 USB 从设备已经被 PC 机识别；

串口终端打印如下信息，证明 USB-CON5 测试通过。

```
root@clanton:~# [  112.646057] g_serial gadget: high-speed config #1: CDC ACM config
```

4.4.3 SEED-Quark_X1000 应用测试

在进行 SEED-Quark_X1000 一系列测试之前，需要按照本章中 4.2、4.3 内容连接好硬件，启动开发板并按照如下命令进入指定目录，执行相应的可执行文件。

```
Target # cd /media/mmcblk0p1
Target # ./seed-quark
```

命令执行后，终端显示如下：

```
clanton login: root
root@clanton:~# cd /media/mmcblk0p1/
root@clanton:/media/mmcblk0p1# ./seed-quark
SPI Device (/dev/spidev1.0) open success!
SPI mode: 0
SPI bits pTransferBufer word: 8
SPI max speed: 10000KHz (10MHz)

/***********************************/
/*     SEED-Quark_X1000 Tests      */
/***********************************/

=========
Main Menu
=========

0: GPIO
1: Zigbee
2: PWM
3: DA
4: AD
5: RS485
6: CAN
7: Blue Tooth
8: Isolate IO

Enter Choice:
```

键入不同的数字选项，选择测试内容。

4.4.3.1 GPIO 测试

测试过程：

- 在 Enter Choice：后输入 0，终端显示 GPIO test begin；
- 开发板的 LED1, LED2, LED3, LED4 通用指示灯将闪烁 3 次；
- 通用指示灯停止闪烁后，终端显示如下内容，GPIO 测试完成。

```
Enter Choice:

GPIO test begin.
GPIO test over.


Menu:

r: Return Main Menu
e: Exit

Enter Choice:
```

4.4.3.2 Zigbee 测试

A）测试准备：

- 准备开发板两块，Zigbee 模块两块，公头短胶棒天线两个，短路子两个，RS-232/RS-485 转换器一个，串口转 USB 线一根；
- 将 Zigbee 模块安插到底板上。一侧插针插入底板的 SZ05_CON1，一侧插针插入底板的 SZ05_CON2，Zigbee 模块与天线连接端 SMA 接口朝向开发板外缘；
- 连接超级终端：

  ①将 RS-232/RS-485 转换器的一端连接到开发的 CON14,一端接入串口转 USB 线 ，将串口转 USB 线的 USB 接口端连接到测试主机。

  ②打开电脑的超级终端（开始->所有程序->附件->通讯->超级终端），选择正确的串口，属性设置为：波特率 38400、数据位 8、校验 NONE、停止位 1、流控：无；
- 启动开发板，并运行应用程序 seed-quark；
- Zigbee 模块配置过程：

  ①在终端中输入数字 1，选择 Zigbee 测试，终端打印如下内容：

```
Enter Choice:

Please configurate zigbee at first!
Please enter any key when Zigbee config is done!
```

  ②终端中打印如上内容后，在开发板底板上的 JP4 上带电插入短路子，此时开发板底板上的 LED8（告警灯）与 LED10（运行灯）同时闪烁，系统进入配置状态。此时超级终端中显示如下：

```
Ver:4.4--00

请选择语言(Please Select Language)
1.中文    2.English
```

  ③在终端中输入 1，超级终端显示如下：

```
请输入安全码：SHUNCOM
```

④此时输入安全码 SHUNCOM,超级终端显示配置界面如下：

```
请选择设置参数:
 1.节点地址    2.节点名称    3.节点类型    4.网络类型    5.网络ID
 6.无线频点    7.地址编码    8.发送模式    9.波特率      A.校 验
 B.数据位      D.设备重启    E.配置显示    F.数据源址
SHUNCOM>
```

⑤按照 Zigbee 文档配置相关参数。在本例程中，两块 Zigbee 的参考配置参数如下：

```
SHUNCOM Z-BEE CONFIG:
 节点地址: 0000
 节点名称: 000111
 节点类型: 中心节点
 网络类型: 星型网
 网 络 ID: 47
 无线频点: 04
 地址编码: HEX
 发送模式: 广播
 波特率:   38400
 校  验:   None
 数据位:   8+0+1
 数据源址: 不输出

请选择设置参数:
 1.节点地址    2.节点名称    3.节点类型    4.网络类型    5.网络ID
 6.无线频点    7.地址编码    8.发送模式    9.波特率      A.校 验
 B.数据位      D.设备重启    E.配置显示    F.数据源址
SHUNCOM>
```

```
SHUNCOM Z-BEE CONFIG:
 节点地址: 4477
 节点名称: 111000
 节点类型: 终端节点
 网络类型: 星型网
 网 络 ID: 47
 无线频点: 04
 地址编码: HEX
 发送模式: 广播
 波特率:   38400
 校  验:   None
 数据位:   8+0+1
 数据源址: 不输出

请选择设置参数:
 1.节点地址    2.节点名称    3.节点类型    4.网络类型    5.网络ID
 6.无线频点    7.地址编码    8.发送模式    9.波特率      A.校 验
 B.数据位      D.设备重启    E.配置显示    F.数据源址
SHUNCOM>
```

Note：

- **配置为中心节点的为主设备，配置为终端节点的为从设备。参数配置结束后拔掉短路子。**
- **两块 Zigbee 的配置过程完全相同，可在一块开发板上先后进行。**

- 测试前的硬件连接：
  - 断电，拔掉之前连接在开发板上的 RS-232/RS-485 转换器；
  - 在两块开发上分别安装两块已配置好的 Zigbee 模块，安装方法与配置时相同；
  - 在 Zigbee 模块上分别连接天线；
  - 将开发板的两个串口分别接入测试主机，超级终端的设置参数参见第一章的串口控制台搭建。

B）测试过程：

- 将两块开发板上电，相关过程参见 4.4.3.2 所述，直到两个终端都显示如下内容：

```
Enter Choice:

Please configurate zigbee at first!
Please enter any key when Zigbee config is done!
```

- 在两个终端中按任意键，两个终端都显示如下内容：

```
Zigbee configuration complete!

=========
Zigbee Menu
=========
Set Zigbee mode:
m: Master
s: Slave
```

- 在安装了配置为从模式的 Zigbee 模块的开发板所连接的终端内输入 s，该终端打印信息如下：

```
Zigbee test begin.
```

- 在安装了配置为主模式的 Zigbee 模块开发板所连接的终端内输入 m，该终端打印信息如下：

```
Zigbee test begin.

Zigbee master write: www.seeddsp.com
Zigbee master read: www.arrowasia.com

Zigbee test over.
```

- 此时安装了配置为从模式的 Zigbee 模块开发板所连接的终端打印如下信息：

```
Zigbee test begin.

Zigbee slaver read: www.seeddsp.com
Zigbee slaver write: www.arrowasia.com

Zigbee test over.
```

- 出现以上信息代表 Zigbee 测试通过。

4.4.3.3 DA 测试

A）测试准备：

- 开发板一块，示波器一台。

B）测试过程：

- 启动开发板，并运行应用程序 seed-quark；
- 连接示波器，一脚接开发板的 CON9 的 GND，一脚连接 CON9 的标记为 A、B、C、D 中的任何一个的插针（GND 和 A、B、C、D 在开发板背面标记）；
- 在终端中输入数字 3 的同时观察示波器输出，示波器显示 2kHz 方波，幅值 2.5V。

Note：

- **DA 输出会自动停止，如果没有完全测试所有 DA 端口，在终端中输入 r 后回到主菜单，重新选择 3 进行测试。**

4.4.3.4 AD 测试

测试过程：

- 启动开发板，并运行应用程序 seed-quark；
- 在终端中输入数字 4，串口终端打印如下信息，代表 AD 测试成功。

```
Enter Choice:
AD test begin.
SPI Device (/dev/spidev0.0) open success!
SPI mode: 0
SPI bits pTransferBufer word: 16
SPI max speed: 10000KHz (10MHz)
SEED-Quark_X1000 AD single-channel channel4 value is 0x4000

SEED-Quark_X1000 AD multi-channels channel0 value is 0x0
SEED-Quark_X1000 AD multi-channels channel1 value is 0x1001
SEED-Quark_X1000 AD multi-channels channel2 value is 0x2001
SEED-Quark_X1000 AD multi-channels channel3 value is 0x3000
SEED-Quark_X1000 AD multi-channels channel4 value is 0x4000
SEED-Quark_X1000 AD multi-channels channel5 value is 0x5000

SEED-Quark_X1000 AD numerous channels test is over
AD test over.
```

Note：

- **打印信息中 channel0-changel5 通道值的高 4 位分别对应各自的 AD 通道数。**

4.4.3.5 RS485 测试

A）测试准备：

- 将 RS-232/RS-485 转换器的的一端连接到开发板的 CON14，一端接入串口转 USB 线 ，将串口转 USB 线的 USB 接口端连接到测试主机。

B）测试过程：

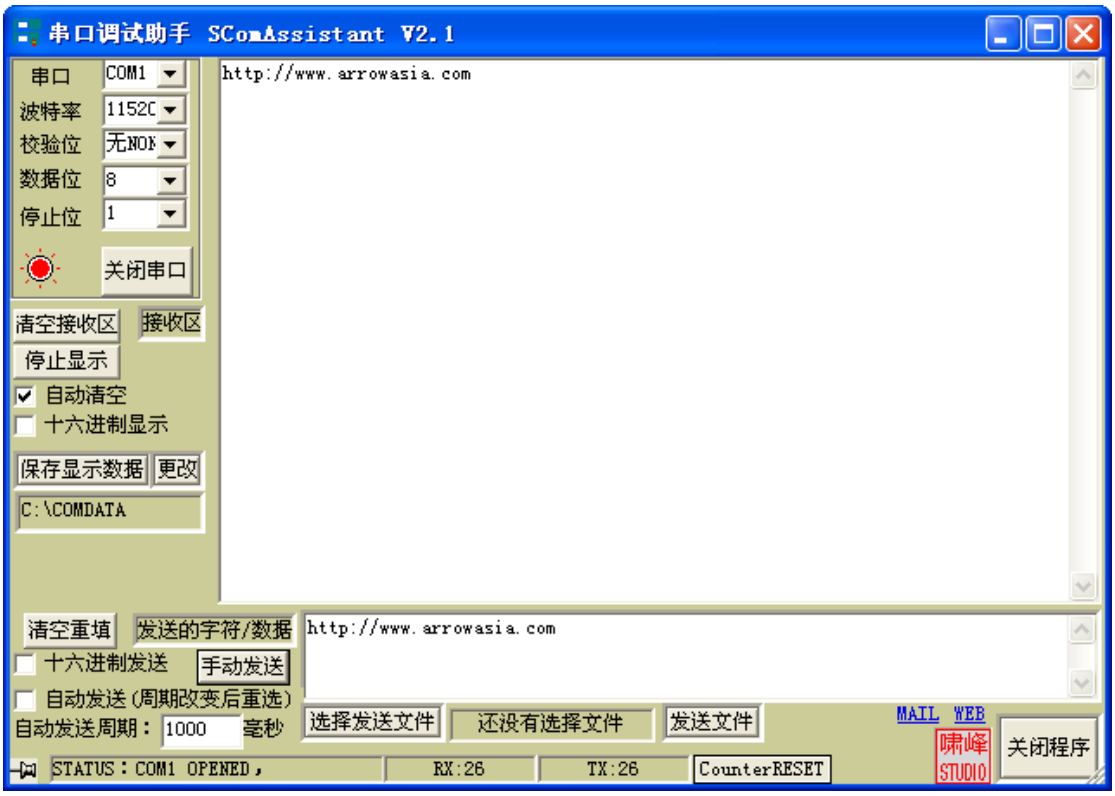
- 启动开发板，并运行应用程序 seed-quark；
- 打开串口调试助手，设置好波特率，校验位等参数；
- 在终端中输入 5，终端中打印如下信息：

```
Enter Choice:

RS485 test begin.
```

- 在串口调试助手中输入要传输的字符并以 Enter 回车键结束。按手动发送键，串口终端和串口调试助手中打印如下信息，代表 RS485 测试通过。

```
RS485 test begin.
RS485 test over.
```



Note：

- **在串口调试助手中输入的发送内容必须要以 Enter 回车键结束。输入的发送内容的有效字符数不能超过 255 个字符。**

4.4.3.6 Blue Tooth 测试

A）测试准备：

- Intel Centrino Wireless-N 135 芯片一块，2.4GHz 天线一根，带蓝牙功能的电子设备一个；
- 将 Intel Centrino Wireless-N 135 芯片安装到底板的 PCIE2 接口中，将天线与 Intel Centrino Wireless-N 135 芯片的 A 口相连接。

B）测试过程：

- 在 Micro SD 卡中拷入 Intel Centrino Wireless-N 135 芯片的驱动 iwlwifi-135-ucode-18.168.6.1.tgz，将之解压缩于 Micro SD 卡的根目录下；
- 启动开发板，并运行应用程序 seed-quark；
- 在终端中输入数字 7，终端显示如下信息：

```
Blue Tooth test begin.
Blue Tooth test over.
```

- 在终端中键入 e，退出程序，回到文件目录下；
- 打开一个蓝牙设备；
- 在终端中输入如下命令扫描周围的蓝牙设备，该过程需等待几秒钟；

```
Target # hcitool scan --flush
```

- 终端打印信息如下：

```
root@clanton:/media/mmcblk0p1# hcitool scan --flush
Scanning ...
        D0:2D:B3:06:9B:2C       Android
        E8:2A:EA:8C:BB:CF       CNLW7BJNICKLI
root@clanton:/media/mmcblk0p1#
```

- 在终端中输入如下命令连接扫描到的设备，其中 D0:2D:B3:06:9B:2C 为扫描设备的设备号；

```
Target # hcitool cc D0:2D:B3:06:9B:2C
```

- 在终端中输入如下命令去 ping 扫描到的设备；

```
Target # l2ping -c 5 D0:2D:B3:06:9B:2C
```

- 终端打印如下信息，Blue Tooth 测试通过。

```
root@clanton:/media/mmcblk0p1# hcitool cc D0:2D:B3:06:9B:2C
root@clanton:/media/mmcblk0p1# l2ping -c 5 D0:2D:B3:06:9B:2C
Ping: D0:2D:B3:06:9B:2C from 97:0C:16:00:00:00 (data size 44) ...
44 bytes from D0:2D:B3:06:9B:2C id 0 time 81.58ms
44 bytes from D0:2D:B3:06:9B:2C id 1 time 40.60ms
44 bytes from D0:2D:B3:06:9B:2C id 2 time 26.67ms
44 bytes from D0:2D:B3:06:9B:2C id 3 time 25.67ms
44 bytes from D0:2D:B3:06:9B:2C id 4 time 113.66ms
5 sent, 5 received, 0% loss
root@clanton:/media/mmcblk0p1#
```

4.4.4 WIFI 测试

A）测试准备：

- Intel Centrino Wireless-N 135 芯片一块，2.4GHz 天线一根，带 wifi 功能的电子设备一个，无线路由器一台；
- 将 Intel Centrino Wireless-N 135 芯片安装到底板的 PCIE2 接口中，将天线与 Intel Centrino Wireless-N 135 芯片的 A 口相连接；
- 将无线路由器上电，设置好路由器的名字以及密码。本测试中路由器的设置为：账号 test123，密码 12345678。

B）测试过程：

- 在 Micro SD 卡中拷入 Intel Centrino Wireless-N 135 芯片的驱动 iwlwifi-135-ucode-18.168.6.1.tgz,将之解压缩于 Micro SD 卡的根目录下；
- 打开具有 wifi 功能的电子设备，搜索到名为”test123”的 wifi，输入密码“12345678”，连接到路由器。找到电子设备的 IP 地址。本测试中设备的 IP 地址为：”192.168.1.2”；
- 开发板上电，启动开发板（参考：4.3 内容），终端打印如下信息：

```
Poky 9.0.2 (Yocto Project 1.4 Reference Distro) 1.4.2 clanton /dev/t

clanton login: root
root@clanton:~#
```

- 在终端中输入如下命令，进入文件 wpa_supplicant.conf；

```
Target # vi /etc/wpa_supplicant.conf
```

- 修改 wpa_supplicant.conf 文件，修改后内容如下：

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
        ssid="test123"
        key_mgmt=WPA-PSK
        psk="12345678"
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
- /etc/wpa_supplicant.conf 1/9 11%
```

其中，ssid 对应无线路由账号，psk 对应无线路由密码。

- 保存，退出该配置文件；
- 在终端中输入如下命令；

```
Target # /usr/sbin/wpa_supplicant -Dnl80211 -i wlan0 -B -c /etc/wpa_supplicant.conf
```

- 终端显示如下内容：

```
Successfully initialized wpa_supplicant
[  799.428461] iwlwifi 0000:01:00.0: L1 Disabled; Enabling L0S
[  799.442461] iwlwifi 0000:01:00.0: Radio type=0x0-0x0-0x0
[  799.652229] iwlwifi 0000:01:00.0: L1 Disabled; Enabling L0S
[  799.666095] iwlwifi 0000:01:00.0: Radio type=0x0-0x0-0x0
[  799.895783] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
root@clanton:~# [  800.619943] wlan0: authenticate with 08:10:77:57:13:e9
[  800.680122] wlan0: send auth to 08:10:77:57:13:e9 (try 1/3)
[  800.688243] wlan0: authenticated
[  800.700137] wlan0: associate with 08:10:77:57:13:e9 (try 1/3)
[  800.712006] wlan0: RX AssocResp from 08:10:77:57:13:e9 (capab=0x411 status=0 aid=2)
[  800.734794] wlan0: associated
[  800.737863] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
```

- 在终端中输入 Ctrl+c，终端显示如下内容：

```
Successfully initialized wpa_supplicant
[  799.428461] iwlwifi 0000:01:00.0: L1 Disabled; Enabling L0S
[  799.442461] iwlwifi 0000:01:00.0: Radio type=0x0-0x0-0x0
[  799.652229] iwlwifi 0000:01:00.0: L1 Disabled; Enabling L0S
[  799.666095] iwlwifi 0000:01:00.0: Radio type=0x0-0x0-0x0
[  799.895783] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
root@clanton:~# [  800.619943] wlan0: authenticate with 08:10:77:57:13:e9
[  800.680122] wlan0: send auth to 08:10:77:57:13:e9 (try 1/3)
[  800.688243] wlan0: authenticated
[  800.700137] wlan0: associate with 08:10:77:57:13:e9 (try 1/3)
[  800.712006] wlan0: RX AssocResp from 08:10:77:57:13:e9 (capab=0x411 status=0 aid=2)
[  800.734794] wlan0: associated
[  800.737863] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
^C
root@clanton:~#
```

- 在终端中输入如下命令，获取板卡动态 IP：

```
Target # /sbin/udhcpc -i wlan0
```

- 终端显示如下内容，即板卡 IP 为 192.168.1.3；

```
root@clanton:~# /sbin/udhcpc -i wlan0
udhcpc (v1.20.2) started
Sending discover...
Sending select for 192.168.1.3...
Lease of 192.168.1.3 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
root@clanton:~#
```

- 在终端中输入如下命令，获取 wlan0 状态；

```
Target # ifconfig wlan0 192.168.1.3 192.168.1.1
```

其中“192.168.1.3”为板卡 IP, ”192.168.1.1”为网关。

- 在终端中输入如下命令，终端显示如下内容，WIFI 测试通过。

```
Target # ping 192.168.1.2
```

其中，“192.168.1.2”为本例程中所使用设备的 IP 地址。

```
root@clanton:~# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: seq=0 ttl=64 time=332.897 ms
64 bytes from 192.168.1.2: seq=1 ttl=64 time=13.140 ms
64 bytes from 192.168.1.2: seq=2 ttl=64 time=16.272 ms
64 bytes from 192.168.1.2: seq=3 ttl=64 time=91.126 ms
64 bytes from 192.168.1.2: seq=4 ttl=64 time=36.863 ms
64 bytes from 192.168.1.2: seq=5 ttl=64 time=35.406 ms
64 bytes from 192.168.1.2: seq=6 ttl=64 time=11.930 ms
64 bytes from 192.168.1.2: seq=7 ttl=64 time=84.197 ms
64 bytes from 192.168.1.2: seq=8 ttl=64 time=22.000 ms
64 bytes from 192.168.1.2: seq=9 ttl=64 time=20.749 ms
64 bytes from 192.168.1.2: seq=10 ttl=64 time=55.678 ms
64 bytes from 192.168.1.2: seq=11 ttl=64 time=3.854 ms
64 bytes from 192.168.1.2: seq=12 ttl=64 time=13.121 ms
64 bytes from 192.168.1.2: seq=13 ttl=64 time=122.979 ms
```