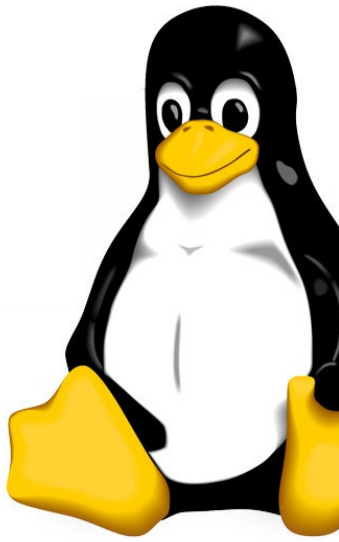

MYD-AM437X Series Linux 4.1.18 Development Guide

MYiR™ Make Your Idea Real

MYD-C437X

MYD-C437X-PRU

Table of Contents

MYD-AM437X Series Linux 4.1.18 Development Guide

Introduction

This document quickly provides the information you need most while evaluating and developing an embedded Linux system on MYD-AM437X series development board,it includes the content of setup development environment, cross compiling bootloader, kernel and linux applications. Finally, it covers how to update the embedded linux system.

This document is suitable for embedded Linux development engineers with a certain development experience

Version History:

Version	Description	Time
V1.0	Initial Version	2017.06.19

Hardware Version:

This document applies to MYD-C437X and MYD-C437X-PRU development board of MYIR.

MYD-C437X	MYD-C437X-PRU
am437x general development board	am437x PRU development board

> Note: The default password for root of the embedded Linux system is not set.

1. Software Resources

Along with the MYD-AM437X series development board, we provide a SDK for developing an embedded Linux system on MYD-AM437X series development board, it includes the fundamental cross compile toolchains,source code of U-boot, Kernel, drivers and test applications of all the peripheral modules.

The contents of this section describe the software resources of the MYD-C437X and MYD-C437X-PRU in form.

Table 1-1 Software Resouces List

Category	Name	Description	Source	MYD-C437X	MYD-C437X-PRU
Bootloader	U-boot201605	First and second stage bootloader SPL and U-Boot，responsible for system initialization and boot Linux kernel	YES	√	√
Kernel	Linux 4.1.18	Kernel designed for MYD-AM437X	YES	√	√
Driver	USB Host	USB host driver	YES	√	√
Driver	USB Device	USB device driver（Gadget）	YES	√	√
Driver	Ethernet	Ethernet driver	YES	√	√
Driver	PRU Ethernet	Industrial ethernet driver	YES	×	√
Driver	MMC/SD	MMC/SD card driver	YES	√	√
Driver	EMMC	EMMC driver	YES	√	√
Driver	I2C	I2C driver	YES	√	√
Driver	SPI	SPI driver	YES	√	√
Driver	LCD	LCD framebuffer driver, use 7 inches 800*480 LCD as default	YES	√	√
Driver	RTC	Internal RTC driver	YES	√	√
Driver	RX-8025T	External RTC driver	YES	×	√
Driver	RS485	RS485 driver	YES	√	√
Driver	ADC	ADC driver	YES	√	√
Driver	Resistive TouchScreen	Resistive touchscreen driver	YES	√	√
Driver	Capacitive TouchScreen	Capacitive touchscreen driver with FT5X06 IC	YES	√	√
Driver	UART	Uart driver	YES	√	√
Driver	CAN	CAN driver	YES	√	√
Driver	PMU	Power manager unit driver	YES	√	√
Driver	LED	LED driver	YES	√	√
Driver	Button	GPIO Button driver	YES	√	√
Driver	Camera	camera driver with ov2659 IC	YES	√	√
Driver	Audio	Audio driver	YES	√	×
Driver	HDMI	HDMI driver	YES	√	×
Filesystem	RAMDISK	Ramdisk filesystem built with Buildroot	Binary compression package	√	√

Filesystem	UBIFS	UBIFS filesystem built with Buildroot	Binary compression package	√	√
Application	CAN	CAN test application	YES	√	√
Application	EEPROM	EEPROM test application	YES	√	√
Application	FrameBuffer	Framebuffer test applicaton	YES	√	√
Application	Keypad	Keypad test application	YES	√	√
Application	RTC	RTC test application	YES	√	√
Application	GPIO	GPIO test application	YES	√	√
Application	LED	LED test application	YES	√	√
Application	PRU	PRU led test application	YES	×	√
Application	RS232	RS232 test application	YES	√	√
Application	RS485	RS485 test application	YES	√	√
Application	Camera	Camera test application	YES	√	√
Application	Audio	Audio test application	YES	√	×
Application	Qt	Qt Hello World demo application	YES	√	√
Tools	Cross Compiler	Linaro GCC 5.3	BIN	√	√
Tools	Win32DiskImager	TF/SD image write	EXE	√	√
Tools	Format Tools	TF/SD format tools	EXE	√	√

2. Deploy Development Environment

The following section covers the setup of hardware, deployment and verification of software development environment.

Software Preparation：

- One host PC with Ubuntu 12.04/14.04/16.04 64bit Desktop
- One host PC Windows7/Windows10

Cross Compiler：

- gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf.tar.xz
- ti_cgt_pru_2.1.3_linux_installer_x86.bin

Hardware Preparation:

MYD-C437X

Connect the RS232 Debug Interface J16(Debug UART in the following picture)to PC with a USB to DB9 RS232 converter cable and set the baudrate of serial port on host PC to 115200-8-n-1.If you need network debugging, please use the network cable to connect the development board J11 (Ethernet0 in the following picture).
An actual whole picture of MYD-C437X is shown below:

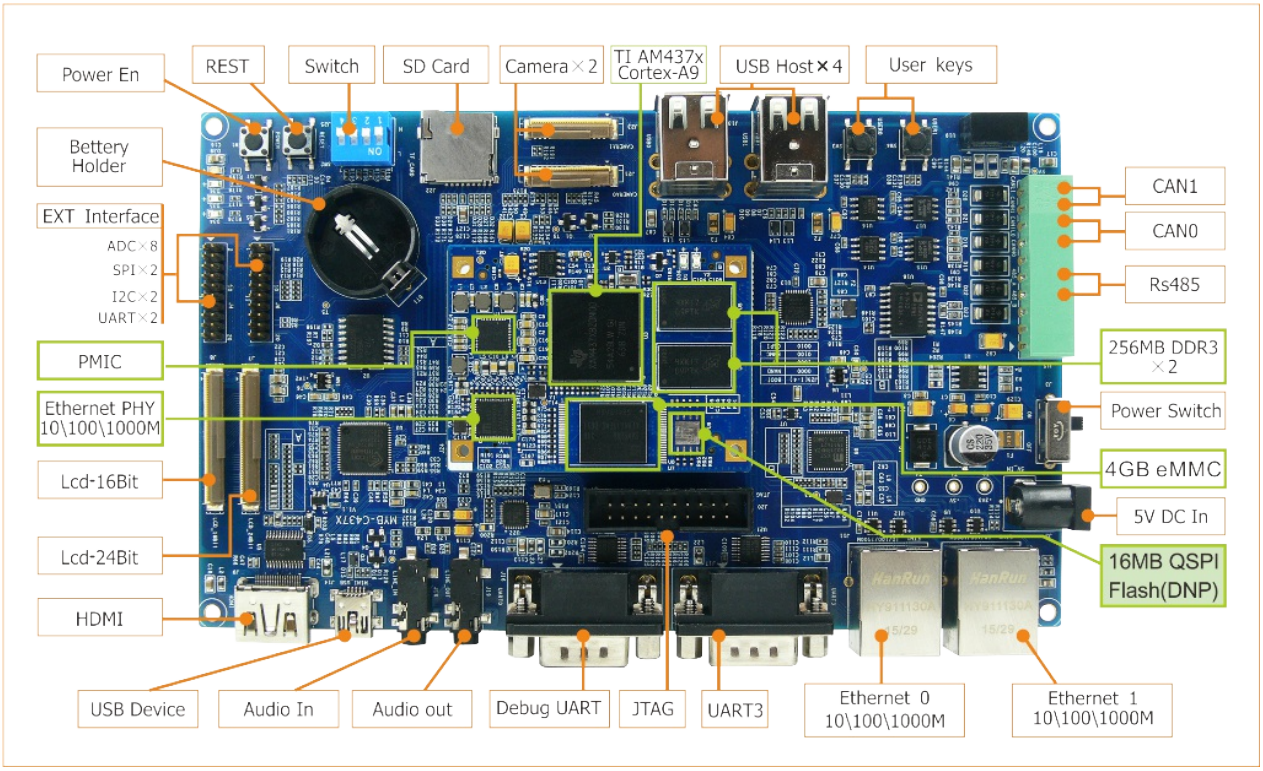

Figure 2-1 MYD-C437X Interface Definition

MYD-C437X-PRU

Connect the TTL Debug Interface J25(Debug UART in the following picture)to PC with a USB to TTL converter and set the baudrate of serial port on host PC to 115200-8-n-1. Ethernet interface J6(Giga Ethernet in the following picture) is corresponding to MAC0 of MYD-C437X-PRU; J26 and J27 are industrial ethernet interface shown as PRUETH0 and PRUETH1 in the following picture.
If you wat to debug with JTAG emulator, please connect the xds100v3 compatible emulator to J22(TI-JTAG interface).
An actual whole picture of MYD-C437X-PRU is shown below:

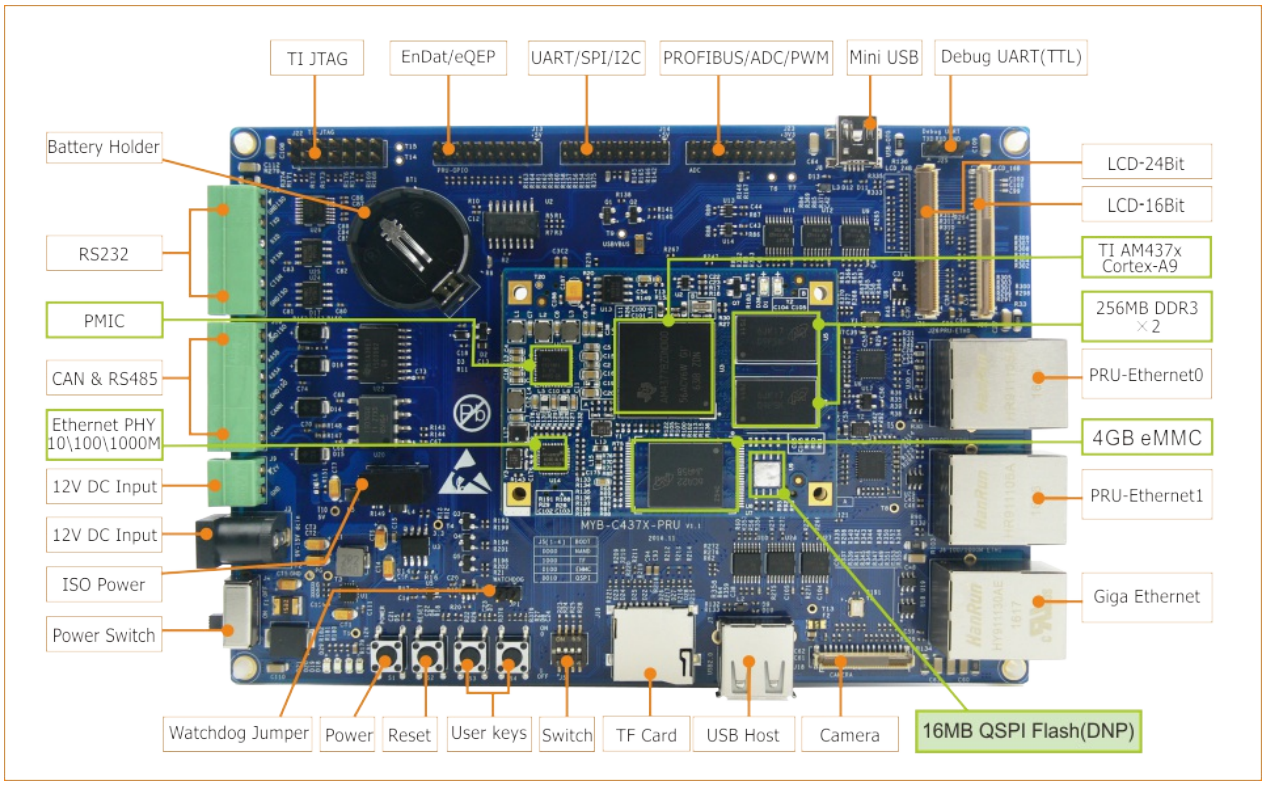

Figure 2-2 MYD-C437X-PRU Interface Definition

Create Work Directory:

Create a work directory and copy the resources from *04-Linux_Source* of our release package to the work directory on ubuntu host PC, `<WORKDIR>` here is defined by customers according to their development environment.

```
$ mkdir -p <WORKDIR>
$ cp -rf /cdrom/04-Linux_Source/* <WORKDIR>
$ ls  <WORKDIR>
Bootloader/  Examples/  Filesystem/  Kernel/  ToolChain/  Tools/
```

2.1 Install Tools

There are some essential tools and libraries needed to be installed during developing an embedded Linux system, such as `build-essential`, `zip`, `unzip` and so on.
For the sake of convenience we install all the tools and libraries before as shown below:

Ubuntu 12.04

```
$ sudo apt-get install build-essential git-core libncurses5-dev
$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev gettext
$ sudo apt-get install gperf libsdl-dev libesd0-dev libwxgtk2.6-dev
$ sudo apt-get install uboot-mkimage
$ sudo apt-get install g++ xz-utils
```

Ubuntu 14.04

```
$ sudo apt-get install build-essential git-core libncurses5-dev u-boot-tools
$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev gettext
$ sudo apt-get install gperf libsdl-dev libesd0-dev
$ sudo apt-get install g++ xz-utils
$ sudo apt-get install subversion
```

Ubuntu 16.04

```
$ sudo apt-get install build-essential git-core libncurses5-dev u-boot-tools
$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev gettext
$ sudo apt-get install gperf libsdl-dev libesd0-dev
$ sudo apt-get install g++ xz-utils
$ sudo apt-get install subversion
```

On 64bit Ubuntu OS, some 32bit runtime libraries should be installed as shown below:

```
$sudo apt-get install libc6-i386 lib32stdc++6 lib32z1
```

2.2 Setup GCC Toolchain

Before compiling U-boot or Kernel, we should set some environment variables on Ubuntu.
The path of the cross compile toolchain should be added to `PATH` environment variable, and `ARCH` environment variable should be set to `arm`, `CROSS_COMPILE` environment variable should be set to `arm-linux-gnueabihf-`.

```
$ cd <WORKDIR>/ToolChain
$ tar Jxvf  gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf.tar.xz
$ export PATH=$PATH:<WORKDIR>/ToolChain/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/bin
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-
```

All of the above processes are only effective in the current shell.
In order to make the environment variables effective to the current user, customers should set the profile configuration of the current user by adding or modifing
`~/.profile` in the home directory of the current user. It is shown as below:

```
vi ~/.profile
```

Add or modify `~./profile` at the end of the file:

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
export PATH=$PATH:<WORKDIR>/ToolChain/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/bin
```

Verify the Enviroment Variables：

```
$source ~/.profile
$echo $ARCH
arm
$echo $CROSS_COMPILE
arm-linux-gnueabihf-
```

Verify the Cross Compiler：

```
$ arm-linux-gnueabihf-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabihf-gcc
......
Thread model: posix
gcc version 5.3.1 20160113 (Linaro GCC 5.3-2016.02)
```

3. Build System

There are many open source tools for building an embedded Linux system, they are more convenient for embedded software engineers to build bootloader, kernel, filesystem all in one step. Currently, OpenWRT, Buildroot , Yocto are more commonly used.

Buildroot is a simple, efficient and easy-to-use tool to generate embedded Linux systems through cross-compilation,thanks to its kernel-like menuconfig, gconfig and xconfig configuration interfaces, building a basic system with Buildroot is easy, so it's very popular among embedded software engineers.

The following sections will explain U-boot, kernel, filesystem respectively, in the part of filesystem, we build a filesystem with QT5 included, customers can develop QT5 application based on this filesystem easily.

3.1 Bootloader

- Enter the directory of bootloader, uncompress the source code package as shown below：

```
$ cd <WORKDIR>/Bootloader
$ tar -zxvf myir-u-boot.tar.gz
$ cd myir-u-boot
```

- Compile U-Boot：

Different development boards correspond to different configuration files,the configuration of U-boot for MYD-AM437X series development board is located at *myir-u-boot/configs/*, the corresponding configuration file name and output files are shown below:

Board	Configuration File Name	Output
MYD-C437X	myd_c437x_evm_defconfig	MLO and u-boot.img
MYD-C437X-PRU	myd_c437x_idk_defconfig	MLO and u-boot.img

Compile U-boot for MYD-C437X development board：

```
$ make distclean
$ make myd_c437x_evm_defconfig
$ make
```

One of the second make in the configuration options see table,after build is completed , in the u-boot directory will generate MLO and u-boot.img files.the steps of compiling MYD-C437X-PRU is simliar to the one introudced of MYD-C437X development board.

After compiling U-boot, MLO and u-boot.img will be generated for mmc boot mode, They can be used for booting from TF Card(mmc0) and EMMC(mmc1).
Users can interrupt the running of U-boot by pressing `SPACE` key in the debug terminal on host PC to enter the console of U-boot. Please input `help` command in the U-boot console to get the usage of U-boot as shown below:

```
># help            -- Display help for U-boot
># echo            -- View a U-boot environment variable
```

3.2 Build Linux Kernel

- Enter the work directory and uncompress the Linux Kernel source code package：

```
$ cd <WORKDIR>/
$ tar -zxvf myir-kernel.tar.gz
$ cd myir-kernel
```

- Compile Kernel：

Different development boards correspond to different configuration files,The configuration of Kernel for MYD-AM437X series development board is located at *myir-kernel/arch/arm/configs/*, the corresponding configuration file name and output files are shown below:
Customers can compile Kernel as shown below:

Board	Kernel Configuration
MYD-C437X	myd_c437x_evm_defconfig
MYD-C437X-PRU	myd_c437x_idk_defconfig

The following to MYD-C437X development board, for example, that the kernel compiler process:

```
$ make mrproper
$ make myd_c437x_evm_defconfig
$ make zImage
$ make dtbs
```

One of the second make in the configuration options in the table above, the steps of compiling MYD-C437X-PRU is simliar to the one introudced of MYD-C437X development board.
After Compiling is completed, in the `arch/arm/boot` directory zImage file is generated, in the `arch/arm/boot/dts` directory device tree binary.dtb file is generated.

Different development boards have different device tree files, and the device tree files are located `myir-kernel/arch/arm/boot/dts` directory :

Board	Device Tree
MYD-C437X	myd_c437x_evm.dts、myd_c437x_evm_hdmi.dts
MYD-C437X-PRU	myd_c437x_idk.dts、myd_c437x_idk_lcd.dts

As the LCD and PRU Ethernet are hard multiplex on MYD-C437X-PRU development board, the two dtb files are designed to work in different mode. `myd_c437x_idk.dtb` is designed for industrial ethernet, and `myd_c437x_idk_lcd.dtb` is designed for LCD, please refer to the device tree source code of these two device tree binary files.

The two work mode of MYD-C437X-PRU are controlled by GPIO4-19 and GPIO4-21. When GPIO4-19 and GPIO4-21 output low level, PRU Ethernet and CAMERA0 are choosed,and LCD does not work, so the device node `&dss` for LCD should be set to `disabled` , the device node `&vpfe0` for camera0 and device node `pruss1_eth` for PRU Ethernets should be set to `okay` , as shown in `myd_c437x_idk.dts` file as shown below:

```
/*
 * File: myd_c437x_idk.dts
 */

......

    gpio4_pins: gpio4_pins_default {
        pinctrl-single,pins = <
        0x1fc ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AE23) cam1_data5.gpio4[19] */
        0x204 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AE24) cam1_data7.gpio4[21] */
        >;
    };
```

```
......

&gpio4 {
    pinctrl-names = "default";
    pinctrl-0 = <&gpio4_pins>;
    status = "okay";

    p19 {
        gpio-hog;
        gpios = <19 GPIO_ACTIVE_HIGH>;
        output-low;
        line-name = "SelPRUorLCDEN";
    };
    p21 {
        gpio-hog;
        gpios = <21 GPIO_ACTIVE_HIGH>;
        output-low;
        line-name = "SelPRUorLCDSEL";
    };
    /* SelPRUorLCDEN enable selects between PRU_ETH and LCD :
     * P19(OE)    | P21 (SEL)    | FUNCTION
     *----------|-----------|----------
     * LOW          |  LOW        |  PRU_ETH + CAMERA0
     * LOW          |     HIGHT      |     LCD
     * HIGH          |     ANY          |     NONE
     *
     * When changing this line make sure the newly
     * selected device node is enabled and the previously
     * selected device node is disabled.
     */
};

......

&dss {
    status = "disabled";
......
};

&vpfe0 {
    status = "okay";
......
};

&pruss1 {
    pruss1_mdio: mdio@54432400 {
        pinctrl-0 = <&pruss1_mdio_default>;
        pinctrl-names = "default";
        reset-gpios = <&gpio4 20 GPIO_ACTIVE_LOW>;
        status = "okay";

        pruss1_eth0_phy: ethernet-phy@0 {
            reg = <0>;
        };

        pruss1_eth1_phy: ethernet-phy@1 {
            reg = <1>;
        };
    };

    /* Dual mac ethernet application node on icss1 */
    pruss1_eth {
        compatible = "ti,am4372-prueth";
        pruss = <&pruss1>;
        sram = <&ocmcram_nocache>;
        status = "ok";

        pinctrl-0 = <&pruss1_eth_default>;
        pinctrl-names = "default";
```

```
        pruss1_emac0: ethernet-mii0 {
            phy-handle = <&pruss1_eth0_phy>;
            phy-mode = "mii";
            sysevent-rx = <20>;    /* PRU_ARM_EVENT0 */
            /* Filled in by bootloader */
            local-mac-address = [00 00 00 00 00 00];
        };

        pruss1_emac1: ethernet-mii1 {
            phy-handle = <&pruss1_eth1_phy>;
            phy-mode = "mii";
            sysevent-rx = <21>;    /* PRU_ARM_EVENT1 */
            /* Filled in by bootloader */
            local-mac-address = [00 00 00 00 00 00];
        };
    };
};
```

Otherwise, when GPIO4-19 outputs low level, GPIO4-21 outputs high level, the device node `&dss` for LCD should be set to `okay`, the device node `&vpfe0` for camera0 and device node `pruss1_eth` for PRU Ethernets should be set to `disabled`, please refer to `myd_c437x_idk.dts` file for detail.

Another example of device tree is relative to the SGX feature of AM437X. The three processors, AM4372, AM4376 and AM4377 have no SGX feature, so customers should set the device node `&sgx` to `disabled`.

```
&sgx {
        status = "disabled";
};
```

For AM4378 and AM4379, these two processors have SGX feature, the device node `&sgx` should be set to `okay` as shown below:

```
&sgx {
        status = "okay";
};
```

3.3 Build Filesystem

This section covers the buiding of filesystem with Buildroot.

> Note: After modifying source code of Kernel or U-boot, Buildroot can not update and build it automatically. Customers should commit it to the master branch of their local git repo manually.
>
> Note: If the source code of Kernel is updated, before building Buildroot again, customers should remove the package "myir-buildroot/dl/linux-master.tar.gz" and the "myir-buildroot/output/build/linux-master" and "myir-buildroot\output\build\linux-headers-master" directories manually. The same to rebuilding of U-boot.

3.3.1 Preparation before Building Buildroot

At the beginning of this document, we have setup the environment variables for Ubuntu, it is also effective for building Buildroot.

> Note: For Ubuntu 64bit OS, 32bit runtime libraries should be installed as shown below.

```
$sudo apt-get install libc6-i386 lib32stdc++6 lib32z1
```

Copy the Buildroot source package customized by MYIR from *04-Linux_Source/Filesystem/myir-buildroot.tar.gz* of our release package to work directory and uncompress it.
The content of myir-buildroot.tar.gz is shown below:

```
$ ls <WORKDIR>/Filesystem/myir-buildroot
arch   CHANGES           configs     dl    linux            output   support
board  Config.in         COPYING     docs  Makefile         package  system
boot   Config.in.legacy  DEVELOPERS  fs    Makefile.legacy  README   toolchain
```

For more details about the file structure of `Buildroot` , please refer to Buildroot manual https://buildroot.org/downloads/manual/manual.html.The part related to the MYIR-AM437X series development board is located in the `<WORKDIR>/Filesystem/myir-buildroot/board/myir/` directory.

3.3.2 Buildroot Configuration

The configuration files of MYD-AM437X series development board for Buildroot are all located at `<WORKDIR>/Filesystem/myir-buildroot/configs/` ,The configuration of MYD-AM437X series development board is shown in the following table:

Board	configuration file	Description
MYD-C437X	myd_c437x_evm_defconfig	Buildroot configuration without QT5 for MYD-C437X development board
MYD-C437X	myd_c437x_evm_qt5_defconfig	Buildroot configuration with QT5 for MYD-C437X development board
MYD-C437X-PRU	myd_c437x_idk_defconfig	Buildroot configuration without QT5 for MYD-C437X-PRU development board
MYD-C437X-PRU	myd_c437x_idk_qt5_defconfig	Buildroot configuration with QT5 for MYD-C437X-PRU development board

The following to MYD-C437X series development board, for example, that Buildroot configuration process:

```
$ make clean
$ make myd_c437x_evm_defconfig
$ make menuconfig
```

One of the second make in the configuration options see table, the steps of compiling MYD-C437X-PRU is simliar to the one introudced of MYD-C437X series development board.

- Configuration for Cross Compiler:

Buildroot can use internal cross compile toolchain generated by Buildroot itself, it can also use external cross compile toolchain. In this document, we choose the internal cross compile toochain, it will be generated and stored to `<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/bin/` after compiling.

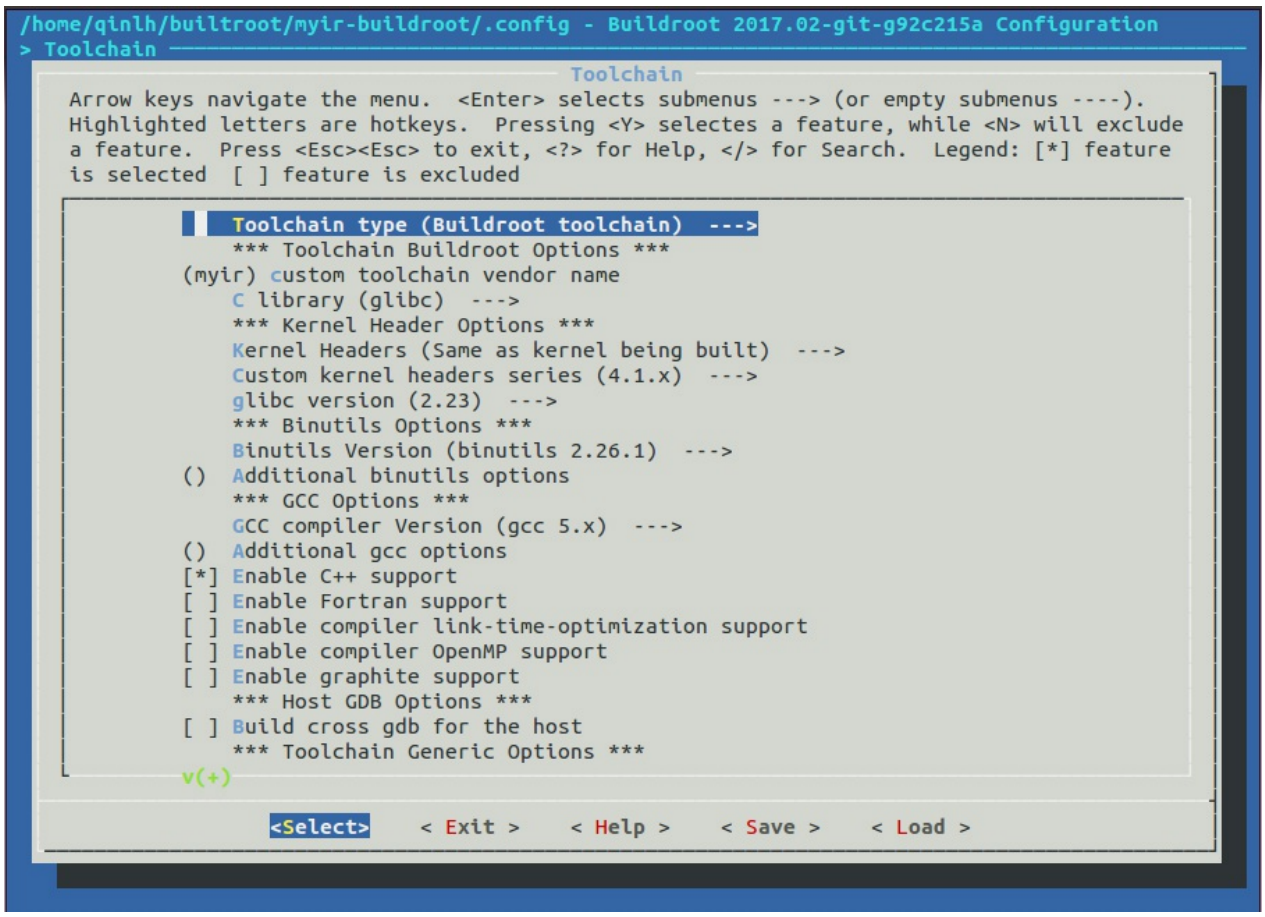

Figure 3-3-1 Configuration for Cross Compiler

- Configuration for System:

The configuration for system includes the name of the target system, the welcome message, the init subsystem (busybox/systemv/systemd) and device manage system,customers can also set the password for root user by configuration. For MYD-AM437X series development board, the password for root is set to `myirtech` as default, it is shown below.
If customers do not need to set password, they no need to config the password.

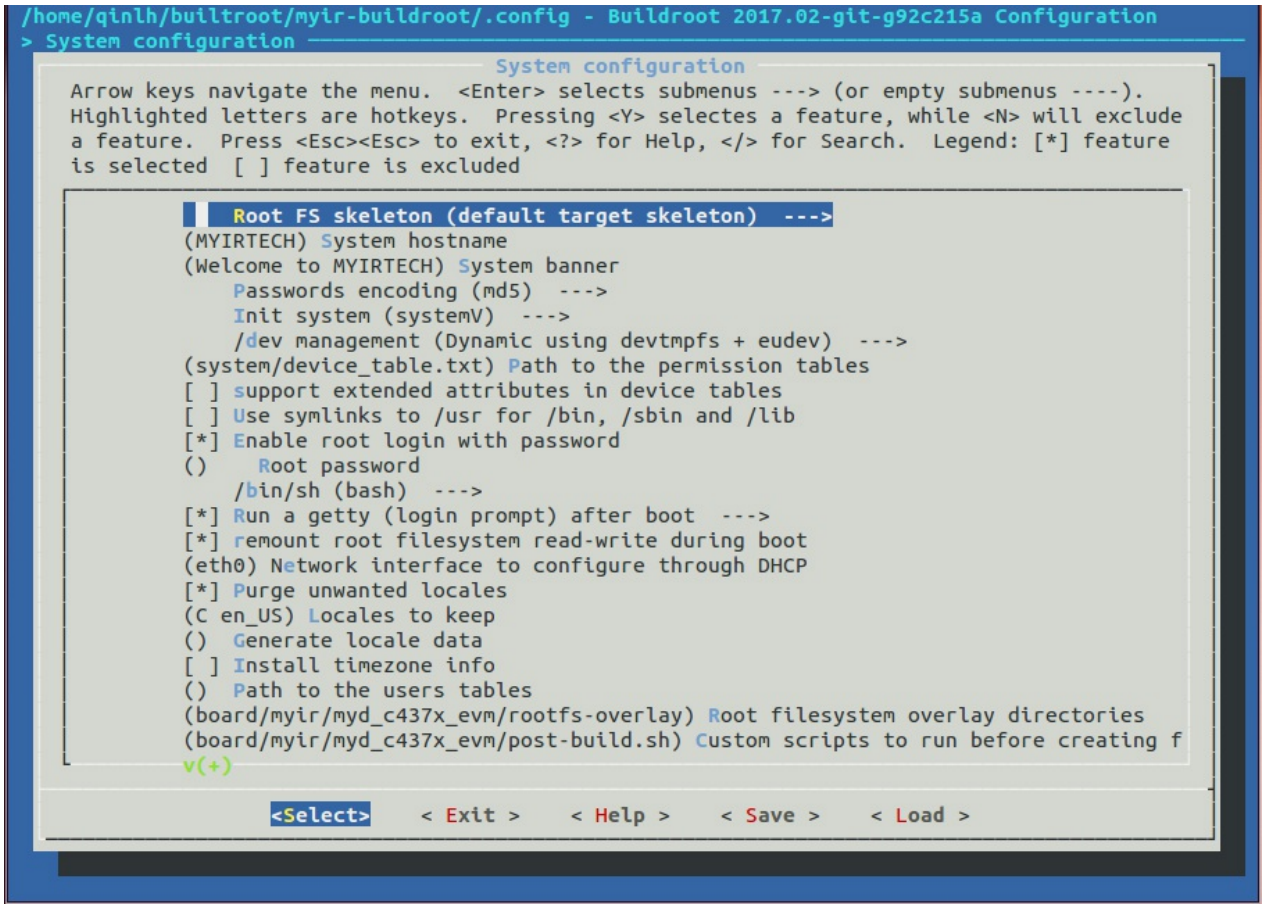

Figure 3-3-2 Configuration for System

- Configuration for Bootloader:

After the user gets the U-boot code, create the Git repository and replace BR2_TARGET_UBOOT_CUSTOM_REPO_URL configuration in the configuration file.Operation is as follows:
Create a U-boot code repository:

```
$ cd ~/
$ tar zxvf myir-u-boot.tar.gz
$ cd myir-u-boot
$ git init
$ git add . -f
$ git commit -m "Initial Version" -a
```

Modify the configuration file located at `<WORKDIR>/Filesystem/buildroot/configs/myd_c437x_evm_defconfig` ,modify the following two items:

```
BR2_TARGET_UBOOT_CUSTOM_REPO_URL="~/myir-u-boot/.git"
BR2_TARGET_UBOOT_CUSTOM_REPO_VERSION="master"
```

The configuration for Bootloader includes the URL of the source code of U-boot, the U-boot configuration file name, the output images of U-boot and so on. They are shown in Figure 3-3-3 below.
We fetch the source code of U-boot with git here, customers can use other protocols or even local directory. For other protocols, please refer to the Buildroot manual.

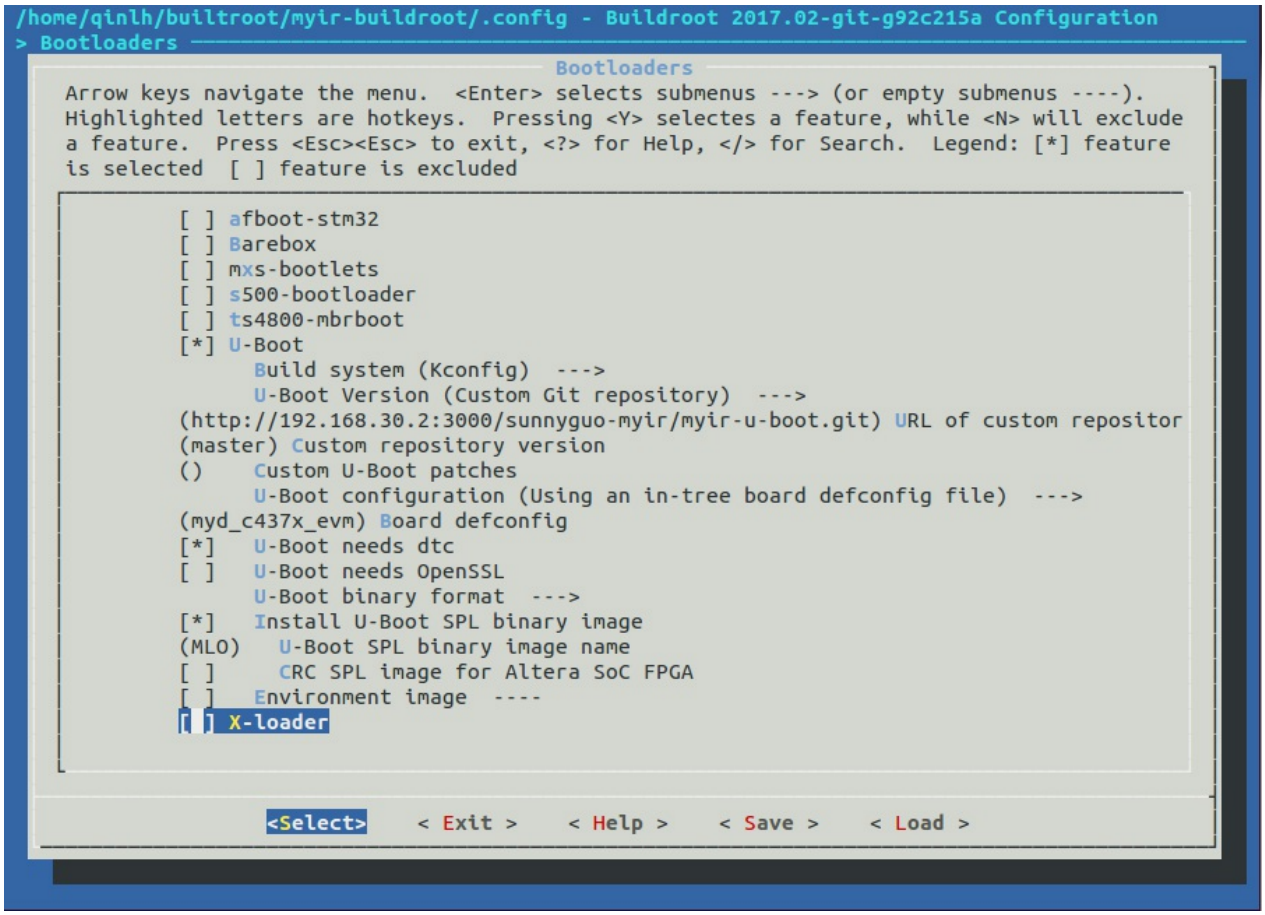

Figure 3-3-3 Configuration for Bootloader

- Configuration for Kernel:

After the user gets the Kernel code, create the Git repository and replace BR2_TARGET_UBOOT_CUSTOM_REPO_URL configuration in the configuration file.Operation is as follows:
Create a kernel code repository:

```
$ cd ~/
$ tar zxvf myir-kernel.tar.gz
$ cd myir-kernel
$ git init
$ git add . -f
$ git commit -m "Initial Version" -a
```

Modify the configuration file located at `<WORKDIR>/Filesystem/buildroot/configs/myd_c437x_evm_defconfig` ,modify the following two items:

```
BR2_TARGET_KERNEL_CUSTOM_REPO_URL="~/myir-kernel/.git"
BR2_TARGET_KERNEL_CUSTOM_REPO_VERSION="master"
```

The configuration for Kernel is similar with the configuration for Bootloader.

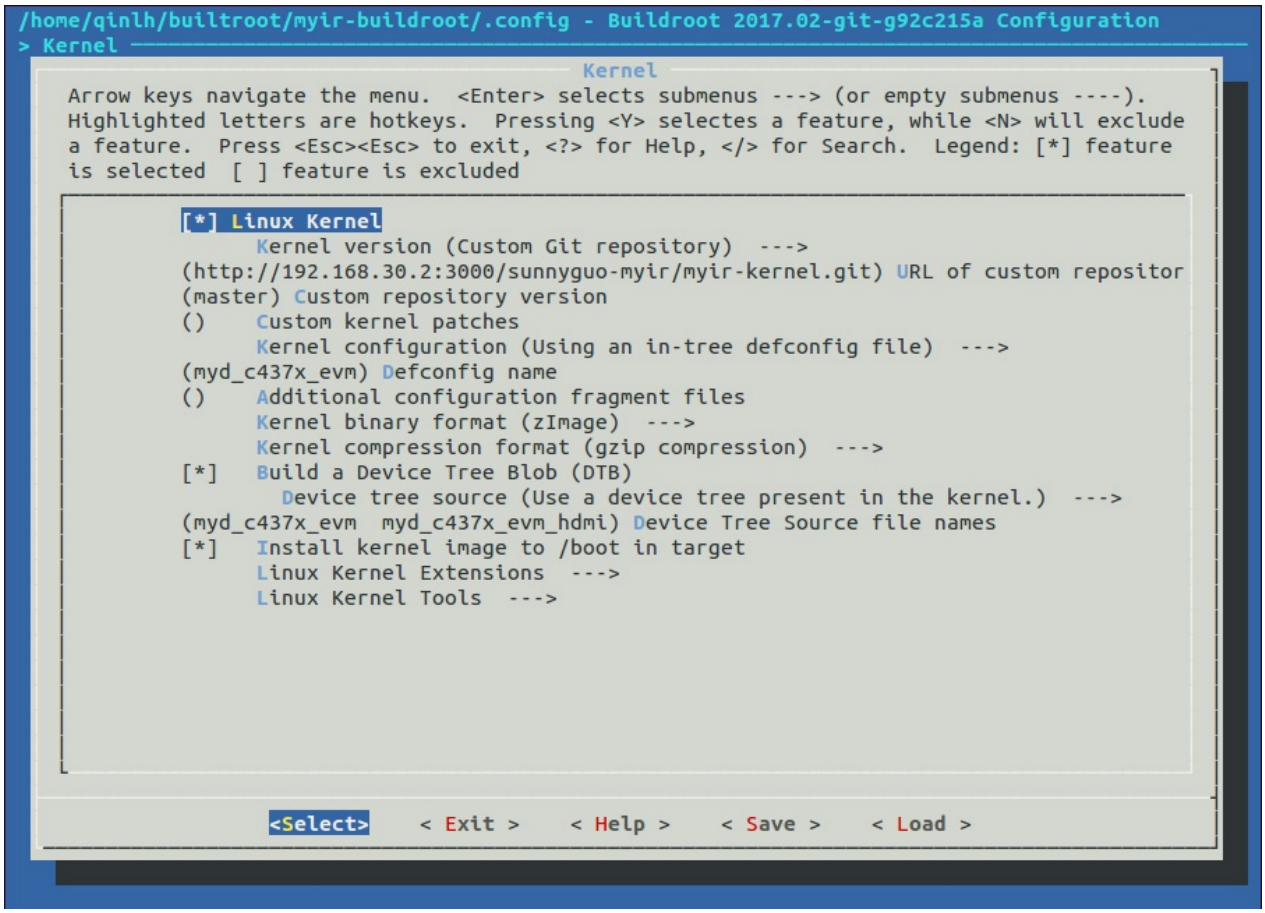

Figure 3-3-4 Configuration for Kernel

- Configuration for Filesystem:

The configuration for filesystem determines what filesystem images are generated in *myir-buildroot/output/images/* directory after compiling, If we choose created `ramdisk` in the configuration, we will get a ramdisk filesystem image. EXT2/4, UBIFS, and rootfs tar package can also be create if they are choosed in configuration.
By the way, the rootfs.tar.gz can be uncompressed and used as the nfsroot directory, it can also be made to other formats of filesystem images by host mtd-utils.
For example, we can create a UBIFS filesystem image without building Buildroot again after doing some modification for rootfs. Firstly, we create a file `ubinize.cfg` as shown below:

```
[ubifs]
mode=ubi
vol_id=0
vol_type=dynamic
vol_name=rootfs
vol_alignment=1
vol_flags=autoresize
image=rootfs.ubifs
```

Then, make a UBIFS image with UBIFS tools by the following processes:

```
$ export PATH=$PATH:<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/sbin
$ mkdir rootfs
$ tar zxvf rootfs.tar.gz -C ./rootfs
$ mkfs.ubifs -d rootfs -e 0x1f000 -c 2048 -m 0x800 -x lzo -F -o  rootfs.ubifs
$ ubinize -o rootfs.ubi -m 0x800 -p 0x20000 -s 512 -m 2048 -O 2048 ubinize.cfg
```

> Note: If `mkfs.ubifs` was installed already on Ubuntu OS, please rename it to another name. Users can check the path of `mkfs.ubifs` with command `which mkfs.ubifs` to make sure the path is located at `<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/sbin/` .

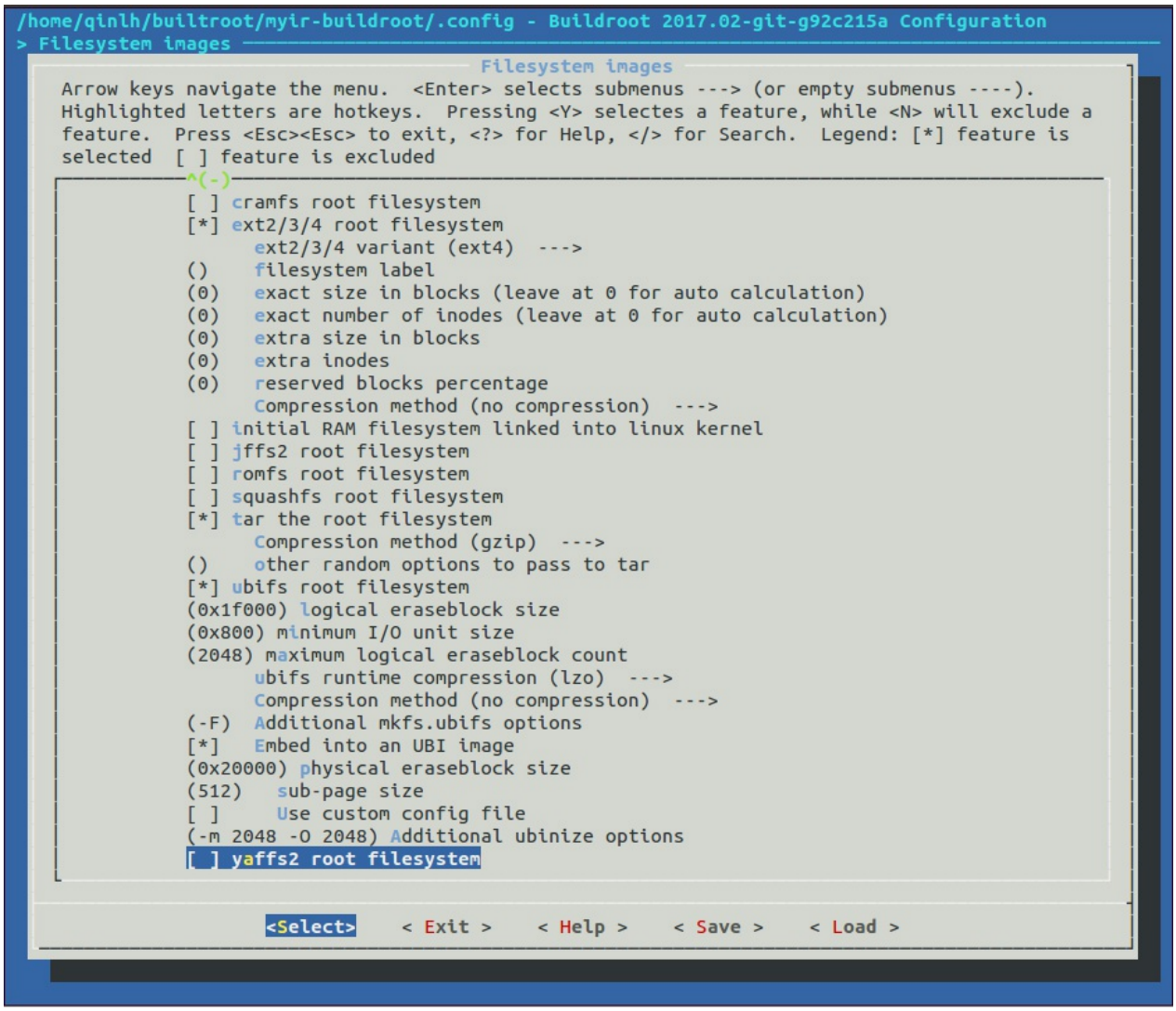

Figure 3-3-5 Configuration for Filesystem

- Configuration for Target Packages:

The configuration for target packages is easier, but it is changed more frequently. Customers can choose some hardware tools, such as I2C-tools, spi-tools, can-utils and so on, build them into the filesystem images for debugging. Some network utils, such as DHCP, TFTP, SSH and so on, can aslo be choosed and built into the filesystem images for production.
Most commonly used tools are included in the target packages of Buildroot. Customers can also write new target packages and integrate them to Buildroot, please refer to https://buildroot.org/downloads/manual/manual.html#adding-packages for details.

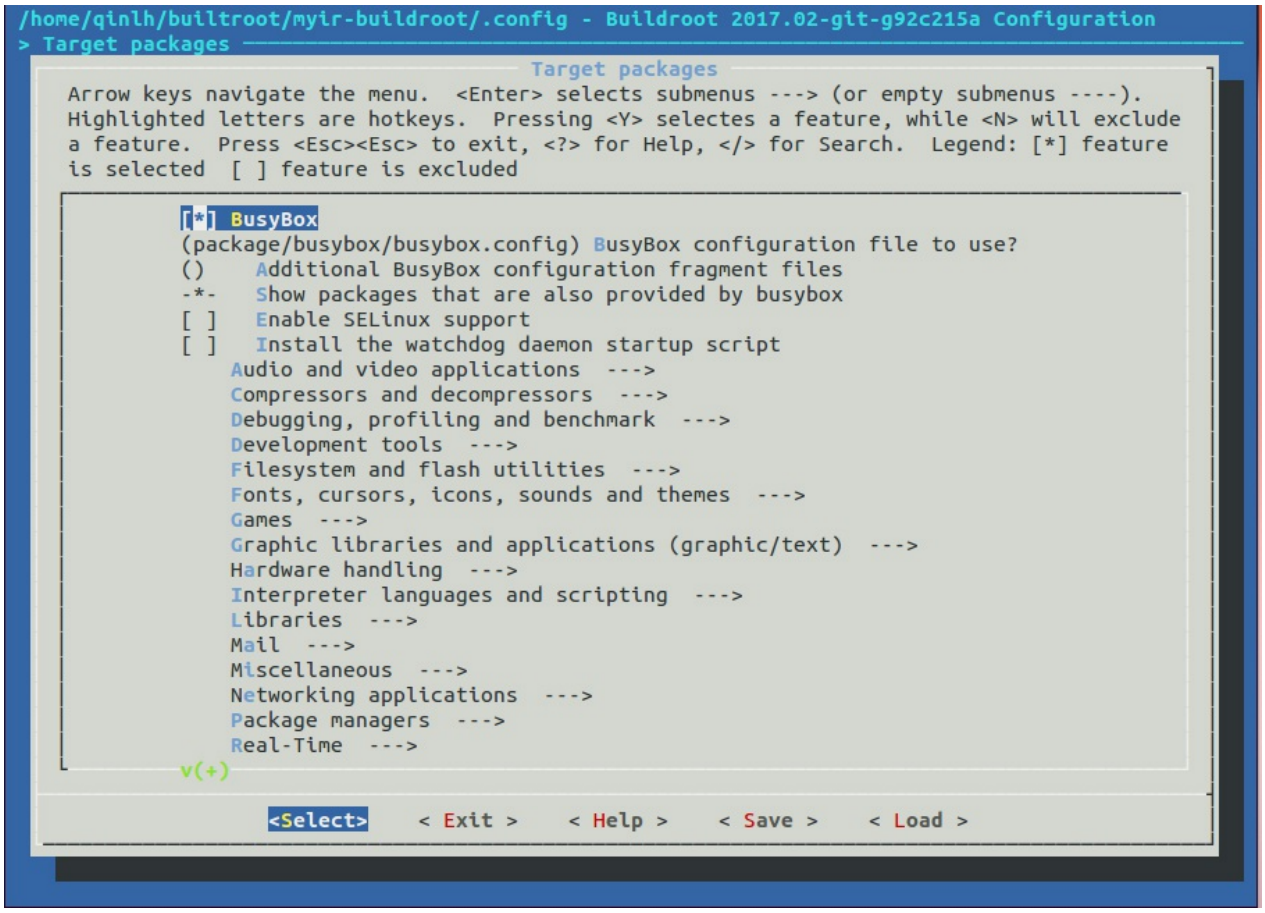

Figure 3-3-6 Configuration for Target Packages

3.3.3 Build Buildroot

Buildroot builds a process similar to the Linux kernel build, with just a simple command.

```
$ make clean
$ make myd_c437x_evm_defconfig
$ make
```

During compiling Buildroot, an output directory named as `output` will be created, and all the output images are all stored to the sub directory `images` of `output`.
The following files are images for MYD-C437X development board generated by Buildroot.

```
$ls -al output/images
boot.vfat              readme.txt         rootfs.ext4    sdcard.img        uEnv.txt
MLO                    rootfs.cpio        rootfs.tar     u-boot.img        zImage
myd_c437x_evm.dtb      rootfs.cpio.gz     rootfs.tar.gz  u-boot-spl.bin
myd_c437x_evm_hdmi.dtb  rootfs.cpio.uboot  rootfs.ubi     uEnv_mmc.txt
ramdisk.gz             rootfs.ext2        rootfs.ubifs   uEnv_ramdisk.txt
```

The bootloader, kernel and all kinds of filesystem images are generated all in one step, they will be introduced in the subsequent section.

3.3.4 Filesystem Built by Arago

Customers can also run an demo filesystem image created with Arago on a MYIR-AM437X series development board, it was created by TI, please refer to the WIKI page on TI's websit.
http://processors.wiki.ti.com/index.php/Processor_SDK_Building_The_SDK.

3.4 Build QT

QT5 is included in Buildroot as a target package, we have provided a config file with QT5 for MYD-C437X development board, so it is easy to build filesystem images with QT5 shown as below.

```
$ cd <WORKDIR>/Filesystem/myir-buildroot
$ make myd_c437x_evm_qt5_defconfig
$ make
```

The configuration options for the second make are shown in the table following . The configuration of the MYD-C437X-PRU is similar to that of the MYD-C437X.

Build Buildroot：

The following table shows the difference between the four config files.

Board	configuration file	Description
MYD-C437X	myd_c437x_evm_defconfig	Buildroot configuration without QT5 for MYD-C437X development board
MYD-C437X	myd_c437x_evm_qt5_defconfig	Buildroot configuration with QT5 for MYD-C437X development board
MYD-C437X-PRU	myd_c437x_idk_defconfig	Buildroot configuration without QT5 for MYD-C437X-PRU development board
MYD-C437X-PRU	myd_c437x_idk_qt5_defconfig	Buildroot configuration with QT5 for MYD-C437X-PRU development board

The MEasy HMI demo system is also generated from the buildroot configuration file with the QT5 runtime environment. For more information on MEasy HMI, refer to the MEasy HMI Development Guide.

In `myd_c437x_evm_qt5_defconfig` , the following items are different with `myd_c437x_evm_defconfig` have been chosen:

```
BR2_PACKAGE_QT5=y
BR2_PACKAGE_QT5BASE_LICENSE_APPROVED=y
BR2_PACKAGE_QT5BASE_EXAMPLES=y
BR2_PACKAGE_QT5BASE_WIDGETS=y
BR2_PACKAGE_QT5BASE_LINUXFB=y
BR2_PACKAGE_QT5BASE_EGLFS=y
BR2_PACKAGE_QT5BASE_GIF=y
BR2_PACKAGE_QT5BASE_JPEG=y
BR2_PACKAGE_QT5BASE_PNG=y
BR2_PACKAGE_QT5BASE_DBUS=y
BR2_PACKAGE_QT5BASE_TSLIB=y
BR2_PACKAGE_QT5QUICKCONTROLS=y
```

After compiling, the filesystem includes eglfs, linuxfb, minimal, offscreen platform plugins for running QT5 applications, the default platform plugin is eglfs.
If the processor is one of AM4372, AM4376 and AM4377 which has no SGX feature, the device node `&sgx` should be disabled in the device tree source file, thus the eglfs platform plugin does not work in the filesystems. Another platform plugin `linuxfb` should be choosed for running QT5 applications here and now as shown below:

```
$ ./helloqt5 --platform linuxfb:fb=/dev/fb0
```

After compiling with the config file `myd_c437x_evm_qt5_defconfig` , all the target images are generated at path `myir-buildroot/output/images` .
Beyond that, a cross compiler and a qmake tools for building QT5 applications are generated at path `myir-buildroot/output/host` .
These will be described in detail in the subsequent sections.

4. Linux Application Development

This section focuses on application development based on embedded Linux, the following examples provided by myirtech demonstrate how to take the control of some commonly used peripheral devices through Linux applications.

The source code of these examples is located at *04-Linux_Source\Examples* of our release package. Please follow the instructions provided in the readme file to set the environment variables, compile the source code and install the binary files into MYD-AM437X series development board.

```
$ cd <WORKDIR>/Examples/
```

Make sure the following environment variables are right.

```
$ export ARCH=arm
$ export CROSS_COMPILE=arm-linux-gnueabihf-
$ export PATH=$PATH:<WORKDIR>/ToolChain/gcc-linaro-5.3-2016.02-x86_64_arm-linux-gnueabihf/bin
```

After building Buildroot, a cross compile toolchain has been created at `myir-buildroot/output/host/usr/bin`, it can be used here by setting the environment variables instead of the above.

```
$ export CROSS_COMPILE=arm-linux-myir-gnueabihf-
$ export PATH=$PATH:<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/bin
```

Camera and Audio routines Makefile need to specify the dependency on the directory and the first directory of the file, the following examples of Camera routines:

```
$ cd <WORKDIR>/Examples/camera
$ cat Makefile
CC = $(CROSS_COMPILE)gcc
CFLAGS ?=-I /<WORKDIR>/Filesystem/myir-buildroot/output/host/usr/
                        arm-myir-linux-gnueabihf/sysroot/usr/include/libdrm/ \
                -I <WORKDIR>/Filesystem/myir-buildroot/output/host/usr/
                         arm-myir-linux-gnueabihf/sysroot/usr/include \
                -I <WORKDIR>/Filesystem/myir-buildroot/output/host/usr/
                arm-myir-linux-gnueabihf/sysroot/usr/include/omap
LDFLAGS ?=  -lpthread  -ljpeg -ldrm -ldrm_omap -L <WORKDIR>/Filesystem/
              myir-buildroot/output/host/usr/arm-myir-linux-gnueabihf/sysroot/usr/lib
TARGET = $(notdir $(CURDIR))_test
SRC =  $(shell ls *.c)
OBJS = $(patsubst %.c ,%.o ,$(SRC))
.PHONY: all
all: $(TARGET)
$(TARGET) : $(OBJS)
        $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^
%.o : %.c
        $(CC) $(CFLAGS) -c $< -o $@
clean:
        $(RM) *.o $(TARGET)
```

For building PRU test application, a PRU toolchain should be installed as shown below, it has been provided in our release package at path *03-Tools/ti_cgt_pru_2.1.3_linux_installer_x86.bin*.

```
$ cd <WORKDIR>/ToolChain/
$ ./ti_cgt_pru_2.1.3_linux_installer_x86.bin
$ export PRU_CGT=<WORKDIR>/ToolChain/ti-cgt-pru_2.1.3/
```

Customers could build examples all in one step,the following operation to MYD-C437X development board as an example:

```
$ cd <WORKDIR>/Examples/
$ make OPTION=MYD-C437X-EVM clean
$ make OPTION=MYD-C437X-EVM
$ make OPTION=MYD-C437X-EVM install
```

The installation path is specified by the PREFIX variable inside the Makefile defaults to `<WORKDIR>/ Examples/rootfs` , where the OPTION variable specifies the development board for the MYIR series. For details, see README.md in the `<WORKDIR>/Examples/` directory, the steps of compiling MYD-C437X-PRU is simliar to the one introudced of MYD-C437X development board.

or build the examples respectively:

```
$ cd <WORKDIR>/Examples/<APP_DIR>
$ make
```

If the binary files have no permission to run, please assign the running permission to them by chmod：

```
# chmod +x *
```

- Different applications require different device tree files, and the device tree file is modified as follows:

SD card boot : Edit in the SD card uEnv_ramdisk.txt in the `fdtfile` required for the application of the device tree file name, edit the file name to uEnv.txt. The content is changed as follows

```
# This uEnv.txt file can contain additional environment settings that you
# want to set in U-Boot at boot time.  This can be simple variables such
# as the serverip or custom variables.  The format of this file is:
#    variable=value
# NOTE: This file will be evaluated after the bootcmd is run and the
#       bootcmd must be set to load this file if it exists (this is the
#       default on all newer U-Boot images.  This also means that some
#       variables such as bootdelay cannot be changed by this file since
#       it is not evaluated until the bootcmd is run.
#optargs=video=HDMI-A-1:800x600

# Uncomment the following line to enable HDMI display and disable LCD display.
fdtfile=myd_c437x_evm.dtb
#fdtfile=myd_c437x_evm_hdmi.dtb
devtype=mmc
devnum=0
bootdir=/
bootpart=0:1
uenvcmd=if run loadimage;
        then run loadfdt; run loadramdisk; echo Booting from mmc${mmcdev} ...;
        run ramargs; print bootargs; bootz ${loadaddr} ${rdaddr} ${fdtaddr}; fi
```

EMMC boot : Boot the board from emmc ,into the system inside, hanging eMMC boot partition to / mnt directory below:

```
# mkdir /mnt/boot
# mount -t vfat /dev/mmcblk0p1 /mnt/boot/
# ls /mnt/boot/
MLO                     ramdisk.gz              uEnv.txt
myd_c437x_evm.dtb       u-boot.img              ws-calibrate.rules
myd_c437x_evm_hdmi.dtb  uEnv                    zImage
```

Use the `vi` command to edit the `fdtfile` of the uEnv.txt located in the */mnt /boot* directory, save and exit after editing is complete . The content is modified as follows:

```
# This uEnv.txt file can contain additional environment settings that you
# want to set in U-Boot at boot time.  This can be simple variables such
# as the serverip or custom variables.  The format of this file is:
#    variable=value
# NOTE: This file will be evaluated after the bootcmd is run and the
#       bootcmd must be set to load this file if it exists (this is the
#       default on all newer U-Boot images.  This also means that some
#       variables such as bootdelay cannot be changed by this file since
#       it is not evaluated until the bootcmd is run.
#optargs=video=HDMI-A-1:800x600

# Uncomment the following line to enable HDMI display and disable LCD display.
fdtfile=myd_c437x_evm.dtb
#fdtfile=myd_c437x_evm_hdmi.dtb
```

Save and exit after editing is complete, use the umount command to uninstall the boot partition. Restart the development board and use the specified device tree to configure the kernel.

```
#umount /mnt/boot
```

NFS boot : Directly copy the application of the required device tree file to the `tftp` directory below, and then enter the uboot console by setting the environment variable to select the corresponding device tree. The operation is as follows:

```
setenv serverip 192.168.30.2
setenv ipaddr 192.168.30.214
setenv rootpath /home/qinlh/export/rootfs
setenv fdtfile myd_c437x_evm.dtb            //Modify the device tree required for the application
saveenv
run netboot
```

4.1 GPIO Test

This example demonstrates how to use Linux API to control GPIO of MYD-AM437X series development board, please refer to the source code for detail.

Hardware Preparation：

- One MYD-AM437X series development board
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART

Software Preparation:

- Linux Kernel 4.1.18
- gpio_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- Copy the executable `gpio _test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `gpio_test` application as below:

```
# gpio_test -h
Usage: gpio_test [options]

Version 1.0
Options:
-n | -- number gpio    gpio number.
-g | -- get            get gpio level.
-s | -- set  level     set gpio level. 0: low; 1: high
-h | -- help            Print this message
```

- GPIO3_7 is used as the write-protect pin, write-protect is enabled when GPIO3_7 outputs high, disabled when GPIO3_7 outputs low.
  Test with gpio_test application as below:

```
# gpio_test -n 103 -s 1
==gpio3_7  direction is out
==gpio3_7  level is high
Set gpio3_7 level high success!
# gpio_test -n 103 -s 0
==gpio3_7  direction is out
==gpio3_7  level is low
Set gpio3_7 level low success!
```

Users can also control GPIO by executing `echo` and `cat` commands to write and read `/sys/class/gpio` in a shell script.
For example, there is a bash script named as `set_eeprom.sh` used for controlling GPIO3_7, the content of file is shown below:

```
#!/bin/bash

EEPROM_WP_GPIO_PIN=103

wait_gpio() {
        sleep 1
}

wp_init() {
        if [ ! -d "/sys/class/gpio/gpio$EEPROM_WP_GPIO_PIN" ]; then
                echo "$EEPROM_WP_GPIO_PIN" > /sys/class/gpio/export; wait_gpio
        fi

        echo "out" > /sys/class/gpio/gpio$EEPROM_WP_GPIO_PIN/direction; wait_gpio
}

up() {
        echo "0" > /sys/class/gpio/gpio$EEPROM_WP_GPIO_PIN/value; wait_gpio
}

down() {
        echo "1" > /sys/class/gpio/gpio$EEPROM_WP_GPIO_PIN/value; wait_gpio
}

if [ "$1" = "1" ]; then
        wp_init
        up
fi

if [ "$1" = "0" ]; then
        wp_init
        down
        echo "$EEPROM_WP_GPIO_PIN" > /sys/class/gpio/unexport; wait_gpio
fi
```

- Run `/usr/bin/set_eeprom.sh` to control the write-protect pin of EEPROM：

```
# chmod 777 /usr/bin/set_eeprom.sh
# set_eeprom.sh 1                           -- Enable write-protect
# set_eeprom.sh 0                           -- Disable write-protect
```

MYD-AM437X series of other development board GPIO test process is similar.

4.2 LCD Test

This example demonstrates the usage of Linux API for Linux framebuffer, users can use these API to paint points, lines and areas on LCD frame buffer.
At the end of this section, demonstrate drawing a picture on LCD framebuffer with `fbv` application built by Buildroot.

Hardware Preparation：

- One MYD-AM437X series development board
- MY-TFF070CV2 connects to LCD interface of MYD-AM437X series development board
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
LCD 16bit interface	J8 LCD_16Bit	J20 LCD_16B

Software Preparation:

- Linux Kernel 4.1.18
- framebuffer_test application
- fbv application built by Buildroot

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb
MYD-C437X-PRU	myd_c437x_idk_lcd.dtb

Test Steps:

- Copy the executable `framebuffer_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `framebuffer_test` application as below:

```
# chmod 777 /usr/bin/framebuffer_test
# framebuffer_test -h
Usage: framebuffer_test [options]

Version 1.0
Available options:
-d | --device name   framebuffer device name, default: /dev/fb0
-h | --help          Print this message

# framebuffer_test -d /dev/fb0
xres:800 >>> yres:480 >>> bpp:32>>>
```

During `framebuffer_test` running, serval colors of background are painted on LCD one by one, and then colorful points, lines, areas are painted.

- Copy a BMP file with 32BPP and resolution of 800*480 to `/media/1.bmp` of the development board, display the picture on LCD by fbv application:

```
# fbv
Usage: fbv [options] image1 image2 image3 ...

Available options:
 --help        | -h : Show this help
 --alpha       | -a : Use the alpha channel (if applicable)
 --dontclear   | -c : Do not clear the screen before and after displaying the image
 --donthide    | -u : Do not hide the cursor before and after displaying the image
 --noinfo      | -i : Supress image information
 --stretch     | -f : Strech (using a simple resizing routine) the image to fit onto
                                          screen if necessary
 --colorstretch| -k : Strech (using a 'color average' resizing routine) the image to
                                            fit onto screen if necessary
 --enlarge     | -e : Enlarge the image to fit the whole screen if necessary
 --ignore-aspect| -r : Ignore the image aspect while resizing
 --delay <d>   | -s <delay> : Slideshow, 'delay' is the slideshow delay in tenths of seconds.

Keys:
 r            : Redraw the image
 a, d, w, x   : Pan the image
 f            : Toggle resizing on/off
 k            : Toggle resizing quality
 e            : Toggle enlarging on/off
 i            : Toggle respecting the image aspect on/off
 n            : Rotate the image 90 degrees left
 m            : Rotate the image 90 degrees right
 p            : Disable all transformations
Copyright (C) 2000 - 2004 Mateusz Golicz, Tomasz Sterna.
Error: Required argument missing.

# fbv /meida/1.bmp
fbv - The Framebuffer Viewer
/media/1.bmp
800 x 480
```

After complete, the picture displays just right for the LCD.

MYD-AM437X series of other development board LCD test process is similar.

4.3 Touch Screen Test

This example demonstrates how to test touch screen by ts_calibrate application built with Buildroot.

Hardware preparation:

- One MYD-AM437X series development board
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1
- One MY-TFT070CV2 module connects to LCD interface of the MYD-AM437X series development board
- Or one MY-TFT070RV2 module connects to LCD interface of the MYD-AM437X series development board

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
LCD 16bit interface	J8 LCD_16Bit	J20 LCD_16B

Software Preparation:

- Linux Kernel 4.1.18
- TS_CALIBRATE application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb
MYD-C437X-PRU	myd_c437x_idk_lcd.dtb

Test Steps:

Due to the differences in the device tree, the `event` event number on the different MYD-AM437X series development board will be the same as the `event` event number corresponding to the capacitive touch and the resistance touch, as shown in the following table:

Touch screen	MYD-C437X	MYD-C437X-PRU
resistive	event1	event2
capacitive	event3	event3

- Power off ,Connect MY-TFT070CV2 module to LCD interface of the development board, power on the board and view the device node in */dev/input* directory

```
#  ls /dev/input/
by-path  event0   event1   event2   event3   mice     mouse0   mouse1

# cat /sys/class/input/event0/device/name
volume_keys@0

# cat /sys/class/input/event1/device/name
ti-tsc

# cat /sys/class/input/event2/device/name
tps65218_pwrbutton

# cat /sys/class/input/event3/device/name
ft5x06_ts
```

The result above shows the resistive touch screen is corresponding to */dev/input/event1*;
The capacitive touch screen is corresponding to */dev/input/event3*, so test capactive touch screen as below:

```
# export TSLIB_TSDEVICE=/dev/input/event3
# ts_calibrate
xres = 800, yres = 480
Took 4 samples...
Top left : X =   54 Y =   46
Took 4 samples...
Top right : X =  745 Y =   58
Took 4 samples...
Bot right : X =  745 Y =  421
Took 3 samples...
Bot left : X =   68 Y =  429
Took 3 samples...
Center : X =  394 Y =  245
-5.867981 1.023202 -0.019352
-2.867676 -0.003020 1.017846
Calibration constants: -384564 67056 -1268 -187936 -197 66705 65536
```

- Power off the development board, connect MY-TFT070RV2 module to LCD interface of the development board, power on the board and view the device node in */dev/input* directory

```
#  ls /dev/input/
by-path  event0   event1   event2   mice     mouse0   mouse1

# cat /sys/class/input/event0/device/name
volume_keys@0

# cat /sys/class/input/event1/device/name
ti-tsc

# cat /sys/class/input/event2/device/name
tps65218_pwrbutton
```

The result above shows the resistive touch screen is corresponding to */dev/input/event1*, so test resistive touch screen as below:

```
# export TSLIB_TSDEVICE=/dev/input/event2
# ts_calibrate
xres = 800, yres = 480
Took 3 samples...
Top left : X =   54 Y =   55
Took 3 samples...
Top right : X =  740 Y =   56
Took 3 samples...
Bot right : X =  737 Y =  419
Took 4 samples...
Bot left : X =   44 Y =  425
Took 4 samples...
Center : X =  395 Y =  243
-4.342529 1.015266 0.018063
-9.879883 0.003775 1.036696
Calibration constants: -284592 66536 1183 -647488 247 67940 65536
```

MYD-AM437X series of other development board Touch Screen test process is similar.

4.4 RTC Test

This example demonstrates how to use Linux API to read and write real time on RTC，please refer to the source code for detail.
Users can also test the RTC with `date` and `hwclock` command built with Buildroot.

Hardware Preparation:

- One MYD-AM437X series development board
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1.
- One CR2032 button cell

Interface	MYD-C437X	MYD-C437X-PRU
Debug serial	J16 UART0	J25 Debug UART

Software Preparation:

- Linux Kernel 4.1.18
- date, hwclock command
- rtc_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- Copy the executable `rtc_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `rtc_test` application as below:

```
# chmod 777 /usr/bin/rtc_test
# rtc_test -h
Usage: rtc_test [options]

Version 1.0
Options:
-d | --device name   rtc device name, default: /dev/rtc0
-w | --write  time     time string with format MMDDhhmm[CCYY][.ss]. such as: 111817582016.18
-h | --help          Print this message

# rtc_test -d /dev/rtc1 -w 111817582016.18

  date/time is updated to:  18-11-2016, 17:58:18.
```

- Power off the development board, wait for a while, power on again and read the rtc time by `rtc_test` as below:

```
# rtc_test -d /dev/rtc1

  Current RTC date/time is 18-11-2016, 17:59:12.
```

- Users can also use date and hwclock command to test RTC as below:

```
# date  081518002016.30                          -- Set system time to 2016/8/15 18:00:30
Mon Aug 15 18:00:30 UTC 2016
# date
Mon Aug 15 18:00:38 UTC 2016
# hwclock -w /dev/rtc1                          -- Write system time to rtc1
```

- Power off the development board, wait for a while, power on again and read the rtc time by hwclock as below:

```
# hwclock -r /dev/rtc1                          --
Mon Aug 15 18:11:08 2016  0.000000 seconds
```

MYD-AM437X series of other development board RTC test process is similar.

4.5 RS232 Test

This example demonstrates how to use Linux API to send and receive data from RS232, please refer to the source code for detail.

Hardware Preparation:

- One MYD-AM437X series development board
- Two data cables to connect RS232 interface of the two boards: GND<->GND，TXD<->RXD，RXD<->TXD, CTS<->RTS, RTS<->CTS
- Two USB to TTL converters, each connects MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1.

Interface	MYD-C437X	MYD-C437X-PRU
Debug serial	J16 UART0	J25 Debug UART
RS232 interface	J17 UART3	J12 RXD、TXD、RTS、CTS、GND

Software Preparation:

- Linux Kernel 4.1.18
- tty_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- Copy the executable `tty_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `tty_test` application as below:

```
# tty_test -h
Usage: tty_test [options]
Version 1.0
Options:
-d | --device name   tty device name, default: /dev/tty0
-m | --mode  mode   operate mode. 0: RS232, 1: RS485  default mode: 0
-f | --flow                    flow control
-b | --baudrate baudrate  set baudrate, default baudrate: 115200
-l | --loop              operate circularly
-w | --write  frame     frame string. such as: 0123456789
-h | --help          Print this message
```

- RS232 interface on the development board is a RS232 serial port with hardware flow control. It is corresponding to /dev/ttyO3 on a embedded Linux system. One board is used as sender, the other is used as receiver, they communicate with tty_test application as below:

```
# tty_test -d /dev/ttyO3 -b 9600 -m 0  -w 0123456789 -f -l
 SEND:0123456789
 SEND:0123456789
 SEND:0123456789
```

- Execute the following command at other board to receive data as below:

```
# tty_test -d /dev/ttyO3 -b 9600 -m 0 -f -l
 RECV:0123456789, total:10
 RECV:0123456789, total:10
 RECV:0123456789, total:10
```

- Exchange roles of the two boards, the result is the same.

MYD-AM437X series of other development board RS232 test process is similar.

4.6 RS485 Test

This example demonstrates how to use Linux APIs to send and receive data from RS485, please refer to the source code for detail.

Hardware Preparation:

- Two MYD-AM437X series development board
- Two data cables to connect RS485 interface of the two boards: 485A<->485A，485B<->485B，GND<->GND
- Two USB to TTL converters, each connects MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1.

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
RS485 interface	J15 485_A、485_B、GND	J10 485A、485B、GNDISO

Software Preparation:

- Linux Kernel 4.1.18
- tty_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- Copy the executable `tty_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `tty_test` application as below:

```
# tty_test -h
Usage: tty_test [options]
Version 1.0
Options:
-d | --device name   tty device name, default: /dev/tty0
-m | --mode  mode   operate mode. 0: RS232, 1: RS485  default mode: 0
-f | --flow                    flow control
-b | --baudrate baudrate  set baudrate, default baudrate: 115200
-l | --loop              operate circularly
-w | --write  frame     frame string. such as: 0123456789
-h | --help          Print this message
```

- RS485 interface on the development board is a RS485 serial port. It is corresponding to `/dev/ttyO5` on a embedded Linux system. One board is used as sender, the other is used as receiver, they communicate with tty_test application as below:

```
# tty_test -d /dev/ttyO5 -b 9600 -m 1  -w 0123456789 -l
 SEND:0123456789
 SEND:0123456789
 SEND:0123456789
```

- Execute the following command at other board to receive data as below:

```
# tty_test -d /dev/ttyO5 -b 9600 -m 1  -l
RECV:0, total:1
RECV:1, total:1
RECV:2, total:1
RECV:3, total:1
RECV:4, total:1
RECV:5, total:1
RECV:6, total:1
RECV:7, total:1
RECV:8, total:1
RECV:9, total:1
```

- Exchange roles of the two boards, the result is the same.

MYD-AM437X series of other development board RS485 test process is similar.

4.7 CAN Bus Test

This example demonstrates how to use Linux APIs to send and receive data from CAN bus, please refer to the source code for detail.

Hardware Preparation:

- Two MYD-AM437X series development board
- Two data cables to connect CAN Bus interface of the two boards: CANH<->CANH，CANL<->CANL，GND<->GND
- Two USB to TTL converters, each connects MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1.

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
CAN Bus interface	J15 CANH0、CANL1、GND、CANH1、CANL1	J10 CANH、CANL、GNDISO

Software Preparation:

- Linux Kernel 4.1.18
- can_test application
- ip link applicatoin
- can_utils tools built with Buildroot

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- Copy the executable `can_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `can_test` application as below:

```
# can_test --help
Usage: can_test [options]

Version 1.0
Options:
-d | --device name   can device name: can0
-b | --baudrate baudrate   set baudrate, default baudrate:50000
-l | --loop              operate circularly, default not operate circularly!
-w | --write  frame     frame string with format ID#MESSAGE. such as: 123#112233445566
-h | --help                Print this message
```

- Set the baudrate for CAN interface as below:

```
# ip link set can0 down
# ip link set can0 type can bitrate 50000 triple-sampling on
# ip link set can0 up
```

- The previous processes are no need to be executed manually. During running `can_test`, it will be set automatically. One board is used as sender, the other is used as receiver, they communicate with can_test application as below:

```
# chmod 777 /usr/bin/can_test
# can_test -d can0 -w 123#112233445566
[ 6862.997962] c_can_platform 481cc000.can can0: setting BTR=1c1d BRPE=0000
====== write  frame: ======
 frame_id = 0x123
 frame_len = 6
 frame_data = 0x11 0x22 0x33 0x44 0x55 0x66
============================
```

- Execute the following command at other board to receive data as below:

```
# chmod 777 /usr/bin/can_test
# can_test -d can0 -l
[ 6484.014811] c_can_platform 481cc000.can can0: setting BTR=1c1d BRPE=0000
  can0  0x123  [6]  0x11 0x22 0x33 0x44 0x55 0x66
```

> Note: -l option is used for operating circularly.
> Note: In case of the following error, please modify the value of "tx_queue_len" as below:

```
# can_test -d can0 -w 123#112233445566
  can raw socket write: No buffer space available

# echo 1000 > /sys/class/net/can0/tx_queue_len
```

- Exchange roles of the two boards, the result is the same.

MYD-AM437X series of other development board CAN test process is similar.

4.8 KEY Test

This example demonstrates how to read the keypad event information by Linux user APIs, please refer to the source code for detail.

Hardware Preparation:

- One MYD-AM437X series development board
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART

Software Preparation:

- Linux Kernel 4.1.18
- hexdump command
- keypad_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

MYD-AM437X series development board provides four keys POWER (SW1), RESET (SW2), USER0 (SW3), USER1 (SW4), where the RESET button will reset the system so it is not tested in this test.

Due to the differences in the device tree, the event number corresponding to the POWER key on the MYD-AM437X series development board will be different, as shown in the following table:

Key	MYD-C437X	MYD-C437X-PRU
POWER	event2	event1

- Copy the executable `keypad_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `keypad_test` application as below:

```
$ chmod 777 /usr/bin/keypad_test
$ keypad_test -h
Usage: keypad_test [options]

Version 1.0
Options:
-d | --device name   keypad device name, default: /dev/input/event0
-h | --help          Print this message
```

- View the device nodes of keypad, the following information shows `S3` and `S4` keypads are corresponding to `/dev/input/event0`, `S1` keypad is corresponding to `/dev/input/event1`.

```
#  ls /dev/input/
by-path  event0   event1   event2   event3   mice     mouse0   mouse1

# cat /sys/class/input/event0/device/name
volume_keys@0

# cat /sys/class/input/event1/device/name
tps65218_pwrbutton

# cat /sys/class/input/event2/device/name
ti-tsc

# cat /sys/class/input/event3/device/name
ft5x06_ts
```

- Test `S3` and `S4` keypads as below:

```
# keypad_test  -d /dev/input/event0
Event: Code = 115, Type = 1, Value=1               -- press S3
Event: Code = 0, Type = 0, Value=0
Event: Code = 115, Type = 1, Value=2
Event: Code = 0, Type = 0, Value=1
Event: Code = 115, Type = 1, Value=0               -- release S3
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=1               -- press S4
Event: Code = 0, Type = 0, Value=0
Event: Code = 114, Type = 1, Value=0               -- release S4
Event: Code = 0, Type = 0, Value=0
```

- Test `S1` keypad as below:

```
# kepad_test  -d /dev/input/event1
Event: Code = 116, Type = 1, Value=1               -- press S1
Event: Code = 0, Type = 0, Value=0
Event: Code = 116, Type = 1, Value=0               -- release S1
Event: Code = 0, Type = 0, Value=0
```

Each keypad has a event code as show above. the code should be consistent with the value in device tree source.
Users can use `hexdump` command to test keypads as below:

```
# hexdump /dev/input/event0
0000000 3912 57b2 cc9e 0007 0001 0073 0001 0000
0000010 3912 57b2 cc9e 0007 0000 0000 0000 0000
0000020 3912 57b2 056d 000b 0001 0073 0000 0000
0000030 3912 57b2 056d 000b 0000 0000 0000 0000

0000040 3915 57b2 1e47 0004 0001 0072 0001 0000
0000050 3915 57b2 1e47 0004 0000 0000 0000 0000
0000060 3915 57b2 6c14 0007 0001 0072 0000 0000
0000070 3915 57b2 6c14 0007 0000 0000 0000 0000
```

Test `S1` keypad with hexdump as below:

```
# hexdump /dev/input/event1
0000000 3a0a 57b2 93a2 0009 0001 0074 0001 0000
0000010 3a0a 57b2 93a2 0009 0000 0000 0000 0000
0000020 3a0a 57b2 348b 000d 0001 0074 0000 0000
0000030 3a0a 57b2 348b 000d 0000 0000 0000 0000
```

MYD-AM437X series of other development board KEY test process is similar.

4.9 LED Test

On an embbeded Linux system, the LEDs are commonly controlled by sysfs interface.
This example demonstrates how to control the LEDs by sysfs with `echo` command or `led_test` application.

Hardware Preparation:

- One MYD-AM437X series development board
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
LED_CPU0	core-board D36	core-board D36
LED_HeartBeat	baseboard D34	baseboard D20
LED_mmc1	baseboard D35	baseboard D19
LED_usr3	baseboard D36	baseboard D18

Software Preparation:

- Linux Kernel 4.1.18
- echo, led_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- View the device node of LED devices as below:

```
# ls /sys/class/leds/
myc:blue:cpu0       myd:blue:mmc1
myd:blue:heartbeat  myd:blue:usr3
```

- Control the LED by `echo` command as below:

```
#echo "0" > /sys/class/leds/myc:blue:cpu0/brightness
#echo "1" > /sys/class/leds/myc:blue:cpu0/brightness
#echo "0" > /sys/class/leds/myc:blue:cpu0/brightness

#echo "1" > /sys/class/leds/myd:blue:mmc1/brightness
#echo "0" > /sys/class/leds/myd:blue:mmc1/brightness

#echo "1" > /sys/class/leds/myd:blue:heartbeat/brightness
#echo "0" > /sys/class/leds/myd:blue:heartbeat/brightness

#echo "1" > /sys/class/leds/myd:blue:usr3/brightness
#echo "0" > /sys/class/leds/myd:blue:usr3/brightness
```

- Copy the executable `led_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `led_test` application as below:

```
# led_test -h
Usage: led_test [options]

Version 1.0
Options:
-d | --device name   led name myc:blue:cpu0
-l | --light brightness   led brightness. 0~255 0: off.
-h | --help                Print this message
# led_test -d myc:blue:cpu0 -l 0
Set led myc:blue:cpu0 off, brightness = 0
# led_test -d myc:blue:cpu0 -l 1
Set led myc:blue:cpu0 off, brightness = 1
```

- Observe the state of the LEDs

> Note: myc:blue:cpu0 is triggered by cpu0, so it can not be controlled directly.
> In order to control this LED, the trigger should be disabled by writing '0' to /sys/class/leds/myc:blue:cpu0/brightness, then it can be controlled normally as other LEDs.

MYD-AM437X series of other development board LED test process is similar.

4.10 EEPROM Test

This example demonstrates how to use Linux API to read and write the EEPROM of development board, please refer the source code for detail.

Hardware Preparation:

- One MYD-AM437X series development board
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1
- Make sure a EEPROM IC 24256E is soldered on the MYD-AM437X series development board

> The MYD-AM437X series development board default have the EEPROM IC.

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART

Software Preparation:

- Linux Kernel 4.1.18
- eeprom_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- Copy the executable `eeprom_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `eeprom_test` application as below:

```
# chmod 777 /usr/bin/eeprom_test
# eeprom_test -h
Usage: eeprom_test [options]

Version 1.0
Options:
-d | --device name   i2c device name: /dev/i2c-0
-a | --address addr     eeprom i2c address, default 0x50
-s | --start addr       start offset to read/write
-r | --read  count    read byte count
-w | --write frame      write frame string. such as: 0123456789
-h | --help               Print this message
```

- Before testing write on EEPROM, write-protect should be disabled by outputing low level on GPIO3_7. During running of eeprom_test, it set write-protect to be disabled automatically, users do not need to handle manually.

```
# echo 103 > /sys/class/gpio/export
# echo "out" > /sys/class/gpio/gpio103/direction
# echo 0 > /sys/class/gpio/gpio103/value
```

- Read and write EEPROM as below:

```
# eeprom_test -d /dev/i2c-0 -a 0x50 -w "hello world!"
WRITE:hello world!
WRITE SUCCESS!


# eeprom_test -d /dev/i2c-0 -a 0x50 -r 12
READ:hello world!
TOTAL 12 BYTES.
```

MYD-AM437X series of other development board EEPROM test process is similar.

4.11 USB Host Test

This example demonstrates how to use USB host to mount mass stroage device and verify the driver of USB host.

Hardware Preparation:

- One MYD-AM437X series development board
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1
- One USB disk

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
USB Host	J13、J12	J7

Software Preparation:

- Linux Kernel 4.1.18
- mount and umount commands

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- Plug the USB disk in the USB host interface of development board, use `mount` or `umount` command to load and unload USB disk. When users plug in the USB disk, Linux kernel dumps the message as below:

```
# [13752.969569] usb 1-1: new high-speed USB device number 2 using xhci-hcd
[13753.114361] usb 1-1: New USB device found, idVendor=0930, idProduct=6545
[13753.121504] usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[13753.129056] usb 1-1: Product: DT 101 G2
[13753.133580] usb 1-1: Manufacturer: Kingston
[13753.138021] usb 1-1: SerialNumber: 001D6095CA1EEC2146A90004
[13753.179033] usb-storage 1-1:1.0: USB Mass Storage device detected
[13753.189488] scsi host0: usb-storage 1-1:1.0
[13753.197187] usbcore: registered new interface driver usb-storage
[13754.266687] scsi 0:0:0:0: Direct-Access  Kingston DT 101 G2  PMAP PQ: 0 ANSI: 0 CCS
[13755.539278] sd 0:0:0:0: [sda] 15240576 512-byte logical blocks: (7.80 GB/7.27 GiB)
[13755.547768] sd 0:0:0:0: [sda] Write Protect is off
[13755.553880] sd 0:0:0:0: [sda] No Caching mode page found
[13755.559903] sd 0:0:0:0: [sda] Assuming drive cache: write through
[13755.591585]  sda: sda1
[13755.602727] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

- It show USB host works well and USB disk is detected, users can mount it to */mnt* directory of the embedded Linux system as below:

```
#
# mount /dev/sda1 /mnt
[ 1301.201855] FAT-fs (sdb1): Volume was not properly unmounted. Some data may be corrupt.
Please run fsck.

# ls /mnt
u-boot.img          MLO             helloworld
```

- Plug out the USB disk, Linux kernel dump the message as below:

```
#
# [14018.109698] usb 1-1: USB disconnect, device number 2
```

- Other USB host interface test methods refer to the above operation.

MYD-AM437X series of other development board USB Host test process is similar.

4.12 USB DEVICE Test

This example demonstrates how to use USB device interface and verify the driver of USB client.The development board works as a TF card reader, it is connected to the USB host of PC with a USB Type A to Mini B cable.

Hardware Preparation：

- One MYD-AM437X series development board
- One TF card
- One USB Type A to Mini B cable
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
USB DEVICE	J14	J8
SD card interface	J22	J19

Software Preparation:

- Linux Kernel 4.1.18
- lsmod, rmmod, modprobe command

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb、myd_c437x_idk_lcd.dtb

Test Steps:

- After the development is booted, USB Type A interface connect to PC USB ,USB Mini B interface connect to development board USB DEVICE interface , plugin a TF card to SD card interface. Load the mass storage gadget driver as below:

```
# modprobe g_mass_storage stall=0 file=/dev/mmcblk0p1 removable=1
[15151.225218] Mass Storage Function, version: 2009/09/11
[15151.231350] LUN: removable file: (no medium)
[15151.236837] LUN: removable file: /dev/mmcblk0p1
[15151.242148] Number of LUNs=1
[15151.245236] Number of LUNs=1
[15151.248511] g_mass_storage gadget: Mass Storage Gadget, version: 2009/09/11
[15151.256739] g_mass_storage gadget: userspace failed to provide iSerialNumber
[15151.264318] g_mass_storage gadget: g_mass_storage ready
[15151.270155] dwc3 48390000.usb: otg: gadget gadget registered
# [15152.172523] g_mass_storage gadget: high-speed config #1: Linux File-Backed Storage
```

- After g_mass_storage driver is loaded, a removable disk will be detected on PC. The content of this removable disk is just the same with the TF card in SD card interface.

> Note: Beyond that, users can load different gadget modules to achieve different functions. such as g_ether is used to make a RNDIS network interface,
> g_serial is used to make a serial port.

MYD-AM437X series of other development board USB Device test process is similar.

4.13 CAMERA Test

This example demonstrates in linux through the V4L2 video framework API interface to achieve camera display, please refer to the source code.

Hardware Preparation：

- One MYD-AM437X series development board
- Two MY-CAM011B camera modules
- One MY-TFT070-K display screen
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
camera interface	J21、J23	J18
LCD interface	J8	J20

> NOTE: For MYD-C437X-PRU development board camera and LCD can not used at the same time

Software Preparation：

- Linux Kernel 4.1.18
- camera_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb、myd_c437x_evm_hdmi.dtb
MYD-C437X-PRU	myd_c437x_idk.dtb

Test Steps：

MYD-C437X Test Steps：

- Development board shutdown, the two camera modules MY-CAM011B were connected to the camera interface, MY-TFT070-K display connect to the LCD interface, power on. Use the command to check whether the camera driver is loading successful and generate the devices：

```
# ls /dev/v*
/dev/vcs          /dev/vcsa         /dev/vga_arbiter  /dev/video1
/dev/vcs1         /dev/vcsa1        /dev/video0

/dev/v4l:
by-path
```

You can see the driver load is normal and successfully found two camera devices `video0, video1`

- Copy the executable `camera_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `camera_test` application as below：

```
#chmod +x /usr/bin/camera_test
# camera_test -h
Usage: camera_test [options]

Version 1.0
Options:
-h | --help               Print this message
```

You can see that this routine does not require any other parameters to run directly:

```
# camera_test
xres:800 >>> yres:480 >>> bpp:32>>>
CRTCs size: 800x480

Capture 0: Opened Channel

Capture 0: Capable of streaming

Capture 0: Init done successfully

Exported buffer fd = 7

Exported buffer fd = 9

Exported buffer fd = 11


Capture 1: Opened Channel

Capture 1: Capable of streaming

Capture 1: Init done successfully

Exported buffer fd = 14

Exported buffer fd = 16

Exported buffer fd = 18
```

After successful operation of the 7-inch LCD display can be seen on the two cameras screen.

MYD-C437X-PRU Test Steps：

- Development board shutdown, the camera module MY-CAM011B access to the camera interface, power on. Through the command to see whether the camera driver is loading successfully and generate the device:

```
# ls /dev/v
v4l/         vcs1         vcsa1        video0
vcs          vcsa         vga_arbiter
```

You can see the driver load is normal and successfully found a camera device video0.

- Insert the SD card to the SD card interface, mount the SD card to the system `/mnt/sdcard` directory:

```
# [  221.139812] mmc0: host does not support reading read-only switch, assuming write-enable

# mkdir /mnt/sdcard
# mount -t vfat /dev/mmcblk0p1 /mnt/sdcard
```

> Note: The device name of the SD card will change according to the system startup mode: When the SD card is started, the device name is /dev/mmcblk0p1. When the EMMC starts, the device name is /dev/mmcblk1p1

- Take a picture by command and stored in the SD card, the specific operation is as follows：

```
# v4l2grab -d /dev/video0 -o /mnt/sdcard/test.jpg -W 800 -H 600
Unable to set frame interval.

# ls /mnt/sdcard/
test.jpg
#umount /mnt/sdcard
```

- Unplug the SD card and insert it into the PC's card reader to view the pictures taken.

MYD-AM437X series of other development board CAMERA test process is similar.

4.14 Audio Test

This example demonstrates the use of the Linux ALSA audio framework and its alsa-utils tool set to achieve audio playback and recording functions.

Hardware Preparation：

- One MYD-AM437X series development board
- A pair of headphones
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
headphone interface	J19 LINE_OUT	no
microphone interface	J18 LINE_IN	no

Software Preparation：

- Linux Kernel 4.1.18
- alsa-utils tool set，audio_test application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm.dtb

Test Steps：

- The development board boot, the headphones into the development board headphone jack, the microphone head into the development board microphone interface
- Test audio playback, copy the prepared WAV format audio files to the development board `/media/test.wav` directory, use the alsa-utils tool set inside aplay to test audio playback：

```
# cd media/
# ls
test.wav
# aplay test.wav
Playing WAVE 'test.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

At this point through the headset can hear the music files being played, if you want to adjust the volume through the alsa-utils tool set inside the alsamixer to adjust the command execution and then there will be characters in the form of graphical interface, the use of four arrow keys can be adjusted：

```
+----------------------------- AlsaMixer v1.1.2 ------------------------------+
| Card: MYD-C437X-EVM                              F1:  Help               |
| Chip:                                            F2:  System information |
| View: F3:[Playback] F4: Capture  F5: All         F6:  Select sound card  |
| Item: Headphone [dB gain: -4.00, -4.00]          Esc: Exit               |
|                                                                          |
|     +--+                      +--+     +--+                              |
|     |  |                      |aa|     |  |                              |
|     |  |                      |aa|     |  |                              |
|     |  |                      |aa|     |  |                              |
|     |  |                      |aa|     |  |                              |
|     |  |                      |aa|     |  |                              |
|     |  |                      |aa|     |  |                              |
|     |aa|                      |aa|     |  |                              |
|     |aa|                      |aa|     |  |                              |
|     |aa|                      |aa|     |  |                              |
|     |aa|                      |aa|     |  |                              |
|     |aa|                      |aa|     |  |                              |
|     +--+     DAC      +--+     +--+     +--+    MIC_IN    +--+      |
|                       |OO|                      |MM|              |OO|      |
|                       +--+                      +--+              +--+      |
|    50<>50                    100<>100    0                                |
|  <Headphon>Headphon Headphon   PCM      Mic    Capture  Capture  Capture     |
+--------------------------------------------------------------------------+
```

Switch the item left and right with the arrow keys，The `[tem: Headphone [dB gain: -4.50, -4.50]` item is volume adjustment，Use the arrow keys to adjust the desired volume level up and down.

- Test the audio input, use the alsa-utils tool set inside the `arecord` to test：

```
# arecord -c 2 -r 44100 -f S16_LE record.wav
Recording WAVE 'record.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

End the recording, through aplay to play just the recording file：

```
# aplay record.wav
Playing WAVE 'record.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

Users can also use the audio_test application to test audio, the function of this application to synchronize the playback of audio recording.

- Copy the executable `audio_test` in the directory `<WORKDIR>/ Examples/rootfs/usr/bin/` to the development board `/usr/bin/` directory, run `audio_test` application as below：

```
# audio_test
rate set to 21999, expected 22000
Init capture successfully, rate: 21999, period_size: 128
rate set to 21998, expected 21999
Period size: 128 frames, buffer size: 256 bytes
^C38 frames
```

At this point through the microphone to speak, headphones can also hear the sound synchronization.

4.15 HDMI Test

This example demonstrates the interface through the linux libdmo library to access the kernel DRM display framework to achieve HDMI interface screen display.

Hardware Preparation：

- One MYD-AM437X series development board
- One HDMI cable,One monitor with HDMI interface
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
HDMI interface	J9 HDMI	no

Software Preparation：

- Linux Kernel 4.1.18
- modetest application

Development Board	Device Tree
MYD-C437X	myd_c437x_evm_hdmi.dtb

Test Steps：

- Plug one end of the HDMI cable into the HDMI interface on the development board and the other end to the HDMI interface on the monitor.
- Power on into the development board, run `modetest` application as below：

```
# modetest -s 25@27:1024x768
trying to open device 'i915'...failed
trying to open device 'amdgpu'...failed
trying to open device 'radeon'...failed
trying to open device 'nouveau'...failed
trying to open device 'vmwgfx'...failed
trying to open device 'omapdrm'...done
setting mode 1024x768-60Hz@XR24 on connectors 25, crtc 27
```

You can see the screen on the display.

MYD-AM437X series of other development board HDMI test process is similar.

4.16 PRU Test

This example demonstrates how to communicate between ARM and PRU with `rpmsg` and `remoteproc` feature of Linux kernel. The PRU test application writes a number from 0 to 7 to PRU, and the PRU set the three LEDs on MYD-C437X-PRU development board according to this number.

Hardware Preparation：

- One MYD-AM437X series development board
- One CAT5 cable ,One switches
- One USB to TTL converter used to connect MYD-AM437X series development board Debug serial port and PC, PC side baud rate setting 115200-8-n-1

Interface	MYD-C437X	MYD-C437X-PRU
debug serial	J16 UART0	J25 Debug UART
PRU-ETH0	no	J26 PRU-ETH0
PRU-ETH1	no	J27 PRU-ETH1

Software Preparation：

- Linux Kernel 4.1.18
- pru_led_test application

Development Board	Device Tree
MYD-C437X-PRU	myd_c437x_idk.dtb

Test Steps：

- Apply a patch to enable PRU-ICSS0 and disable PRU-ICSS1, because the three LEDs are controlled by PRU-ICSS0 instead of PRU-ICSS1

After appling the patch located at *04-Linux_sources/patches/0001-Enable-PRUSS0-instead-of-PRUSS1-on-the-MYD_C437x_PRU*, customers should rebuild
the Kernel and device tree source to generate `zImage` and device tree binary files again.

```
// set pinmux for AD21, AE22, AD22 to be controlled by PRUSS0
@@ -168,9 +148,10 @@
    leds_pins: leds_pins {
        pinctrl-single,pins = <
            0x244 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (F23) gpio5_11.gpio5[11] */
-            0x1f0 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AD21) cam1_data2.gpio4[16] */
-            0x1f4 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AE22) cam1_data3.gpio4[17] */
-            0x1f8 ( PIN_OUTPUT_PULLUP | MUX_MODE7 ) /* (AD22) cam1_data4.gpio4[18] */
+            0x1f0 ( PIN_OUTPUT_PULLUP | MUX_MODE4 ) /* (AD21) cam1_data2.pr0_pru1_gpo[10] */
+            0x1f4 ( PIN_OUTPUT_PULLUP | MUX_MODE4 ) /* (AE22) cam1_data3.pr0_pru1_gpo[11] */
+            0x1f8 ( PIN_OUTPUT_PULLUP | MUX_MODE4 ) /* (AD22) cam1_data4.pr0_pru1_gpo[12] */
+
```

- Compile the firmware running on PRU

The source code of PRU firmware is located at *04-Linux_Source\Examples\pru_led\PRU_RPMsg_LED0_1*, it should be compiled by ti_cgt_pru toolchain.
After compiling is complete, the firmware for PRU `PRU_RPMsg_LED0_1.out` is generated, please rename it to `am437x-pru0_1-fw` and copy to /lib/firmware/ of the embedded Linux
root file system. If ti_cgt_pru is not installed, please install it and compile `pru_led` test application as below:

```
$ cd <WORKDIR>/ToolChain/
$ ./ti_cgt_pru_2.1.3_linux_installer_x86.bin
$ export PRU_CGT=<WORKDIR>/ToolChain/ti-cgt-pru_2.1.3/
$ cd <WORKDIR>/04-Linux_Source\Examples\pru_led
$ make

**********************************************************
Building project: PRU_RPMsg_LED0_1

Finished building project: PRU_RPMsg_LED0_1
**********************************************************
```

- Restart the embedded Linux system rebuilt with PRU-ICSS0 enabled

A device node `/dev/rpmsg_pru31` generated after booting shows PRU working well, then users can move on to the following steps.
Users can use `rmmod` or `modprobe` to unload or reload PRU remoteproc driver and firmware for debugging as below:

```
# rmmod pru_rproc -f
[ 5702.650841] pru-rproc 54478000.pru1: pru_rproc_remove: removing rproc 54478000.pru1
[ 5702.660895] pruss-rproc 54440000.pruss: unconfigured system_events = 0x0800000000000000
host_intr = 0x00000002
[ 5702.672138]  remoteproc2: stopped remote processor 54478000.pru1
[ 5702.678938]  remoteproc2: releasing 54478000.pru1
[ 5702.684928] pru-rproc 54474000.pru0: pru_rproc_remove: removing rproc 54474000.pru0
[ 5702.694806] pruss-rproc 54440000.pruss: unconfigured system_events = 0x1000000000000000
host_intr = 0x00000001
[ 5702.706273]  remoteproc1: stopped remote processor 54474000.pru0
[ 5702.713233]  remoteproc1: releasing 54474000.pru0
# modprobe pru_rproc
[ 5707.305735]  remoteproc1: 54474000.pru0 is available
[ 5707.314208]  remoteproc1: Note: remoteproc is still under development and considered
experimental.
[ 5707.324384]  remoteproc1: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[ 5707.342644]  remoteproc1: powering up 54474000.pru0
[ 5707.347860]  remoteproc1: Booting fw image am437x-pru0_0-fw, size 84928
[ 5707.355459] pruss-rproc 54440000.pruss: configured system_events = 0x1000000000000000
intr_channels = 0x00000001 host_intr = 0x00000001
[ 5707.368458]  remoteproc1: remote processor 54474000.pru0 is now up
[ 5707.375567] virtio_rpmsg_bus virtio0: rpmsg host is online
[ 5707.381484]  remoteproc1: registered virtio0 (type 7)
[ 5707.386895] pru-rproc 54474000.pru0: PRU rproc node /ocp/pruss@54440000/pru0@54474000
probed successfully
[ 5707.397446] virtio_rpmsg_bus virtio0: creating channel rpmsg-pru addr 0x1e
[ 5707.405191]  remoteproc2: 54478000.pru1 is available
[ 5707.415107] rpmsg_pru rpmsg4: new rpmsg_pru device: /dev/rpmsg_pru30
[ 5707.421988]  remoteproc2: Note: remoteproc is still under development and considered
experimental.
[ 5707.432231]  remoteproc2: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[ 5707.448155]  remoteproc2: powering up 54478000.pru1
[ 5707.453860]  remoteproc2: Booting fw image am437x-pru0_1-fw, size 84360
[ 5707.461062] pruss-rproc 54440000.pruss: configured system_events = 0x0800000000000000
 intr_channels = 0x00000002 host_intr = 0x00000002
[ 5707.474342]  remoteproc2: remote processor 54478000.pru1 is now up
[ 5707.481129] virtio_rpmsg_bus virtio1: rpmsg host is online
[ 5707.487010]  remoteproc2: registered virtio1 (type 7)
[ 5707.492891] pru-rproc 54478000.pru1: PRU rproc node /ocp/pruss@54440000/pru1@54438000
probed successfully
[ 5707.503186] virtio_rpmsg_bus virtio1: creating channel rpmsg-pru addr 0x1f
# [ 5707.517016] rpmsg_pru rpmsg5: new rpmsg_pru device: /dev/rpmsg_pru31
```

- Use pru_led_test application to control LEDs as below:

```
# pru_led_test -h
Usage: pru_led_test [options]

Version 1.0
Options:
-d | --device name   pru rpmsg char device name /dev/rpmsg_pru31
-l | --loop          operate circularly, default not operate circularly!
-n | --number num    number send to pru! must be 0~7.
-h | --help                Print this message


# pru_led_test -d /dev/rpmsg_pru31 -n 7
Opened /dev/rpmsg_pru31, sending 1 messages

1 - Sent to PRU: 7
1 - Received from PRU:7

Received 1 messages, closing /dev/rpmsg_pru31
```

- Observe the state of the three LEDs on the motherboard of MYD-C437X-PRU development board.

They are turned on and off depending on the number along with `-n`. The variation regularity of the LEDs is relative to the binary value of the number.
For example, if customers set the number to 3(011b), then D18 will be turned off, D19 and D20 will be turned on.

5. Qt Application Development

In the previous sections, a build tool for QT5 application `qmake` has been created. It is used for generate Makefiles and other project files for QT5 application, after that, users can build QT5 application with cross compile toolchain. For development of larger QT5 application, a IDE tools named as `QtCreator` is commonly used.
There is a evaluation edition of QtCreator in our release package at the path *03-Tools/qt-creator-opensource-linux-x86_64-4.1.0.run*.

In the following sections, we will introduce the installation, configuration of QtCreator, and demonstrate how to create a simple QT5 application running on MYD-AM437X series development board.

5.1 Install QtCreator

On Ubuntu 64bit OS, customers can install QtCreator as shown below:

```
$ cd <WORKDIR>
$ cp /media/cdrom/03-Tools/Qt/qt-creator-opensource-linux-x86_64-4.1.0.run .
$ chmod a+x qt-creator-opensource-linux-x86_64-4.1.0.run
$ sudo ./qt-creator-opensource-linux-x86_64-4.1.0.run
```

QtCreator will be installed on Ubuntu OS step by step automatically.

5.2 Config QtCreator

- Configure Build&Run Environment:

Open QtCreator, choose `tools` -> `options`, then the `Build & Run` dialog popups. Please choose the `Compilers` tab to set compiler for QtCreator as shown below:

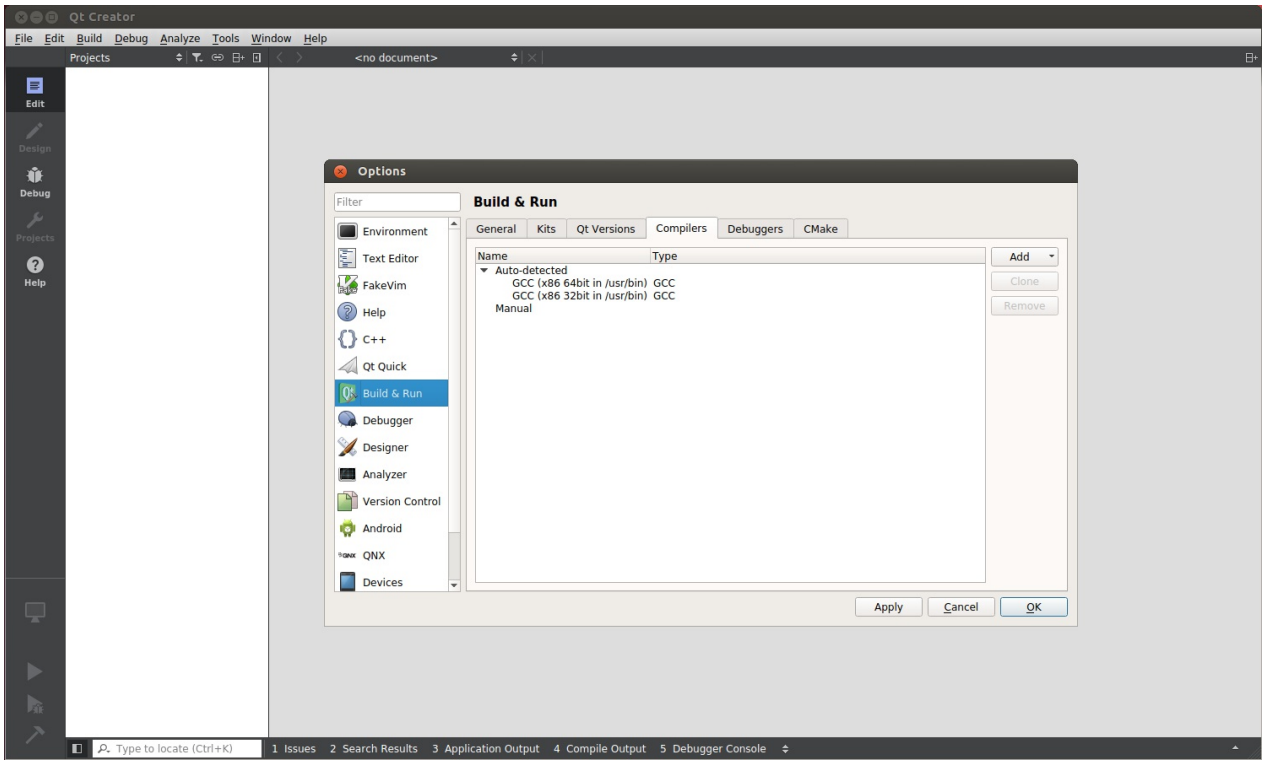

Figure 5-2-1 Settings of Compiler

Press `Add` button at the right side of this dialog, choose `Custom` in the dropdown list, and then set `Name`, `Compiler path`, `Make path` and `ABI` as shown below.
After complete, press the `Apply` button to save.

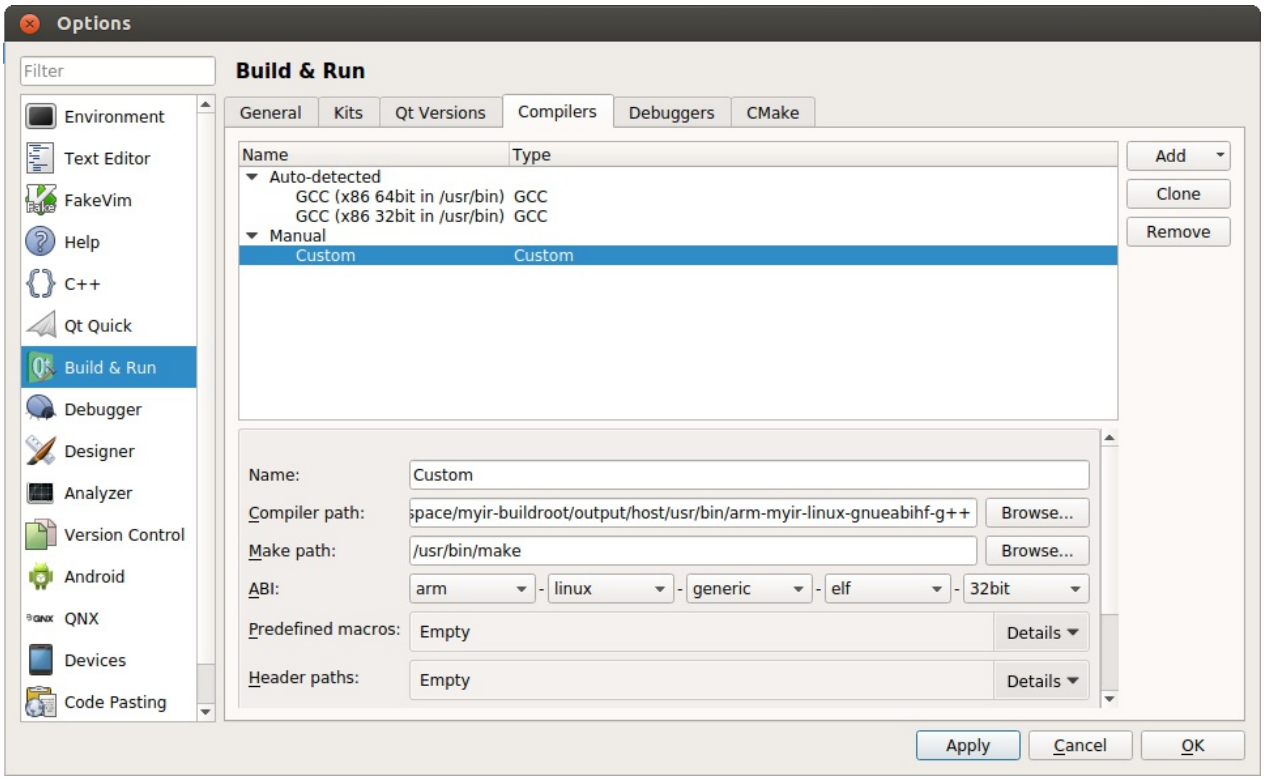

Figure 5-2-2 Add Compiler for QtCreator

At the same dialog, choose `Qt Version` Tab to add qmake, at the right side of this dialog, press `Add` button, then a new dialog popups,
please choose the qmake tools described at Chapter 3-4-1. After complete, press `Open` to set qmake, and then `Apply` to save.

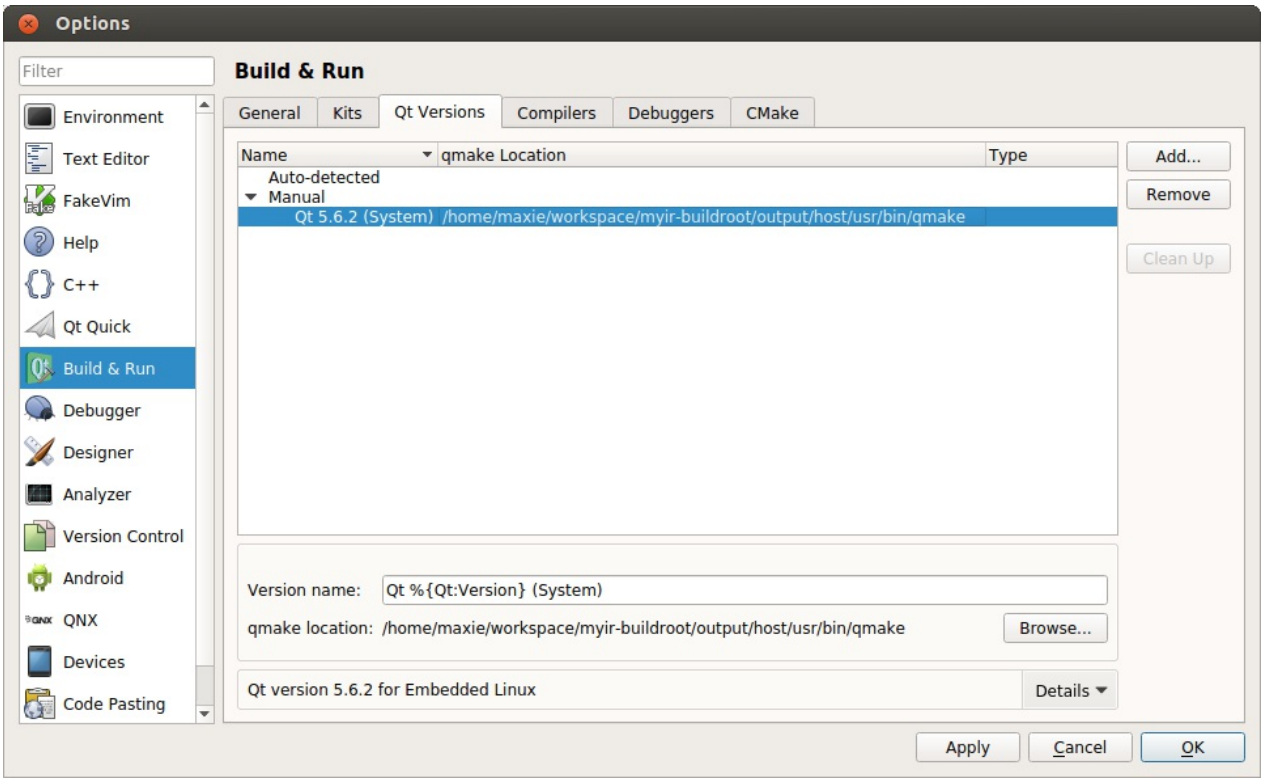

Figure 5-2-3 Choose qmake

In the `Build&Run` window, continue to choose `Kits` tab, and at the right side of this dialog, press `Add` button, then add settings of running environment for QT5 application. In the `Sysroot` editbox, write the path of cross compile toolchain, in the `Compiler` and `Qt Version` editboxs, write the settings being set before, set `Debugger` to `None` , set `CMake Tools` to default as shown below:

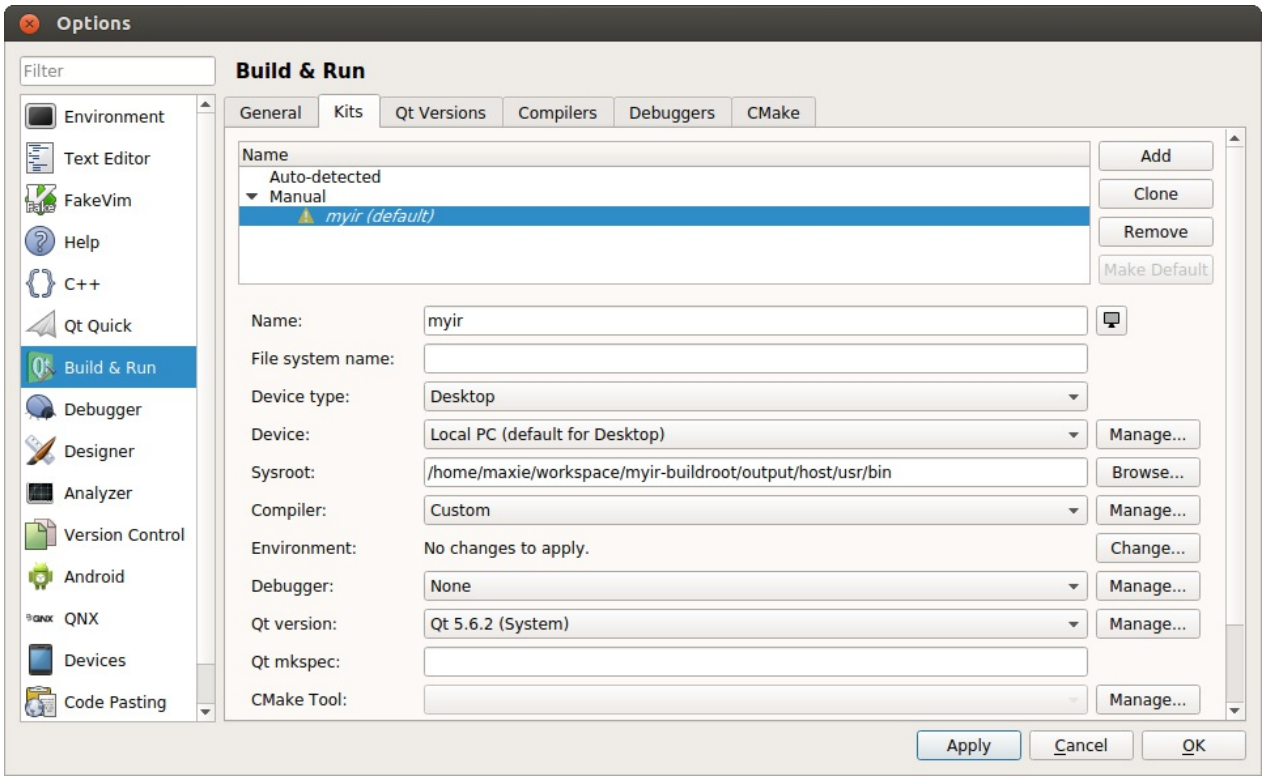

Figure 5-2-4 Add Kits for QtCreator

- Create Helloworld Project:

In the main menu of QtCreate, choose `File` -> `New File or Project` , and in the popup dialog, choose `Application` -> `Qt Widgets Application` as shown below:

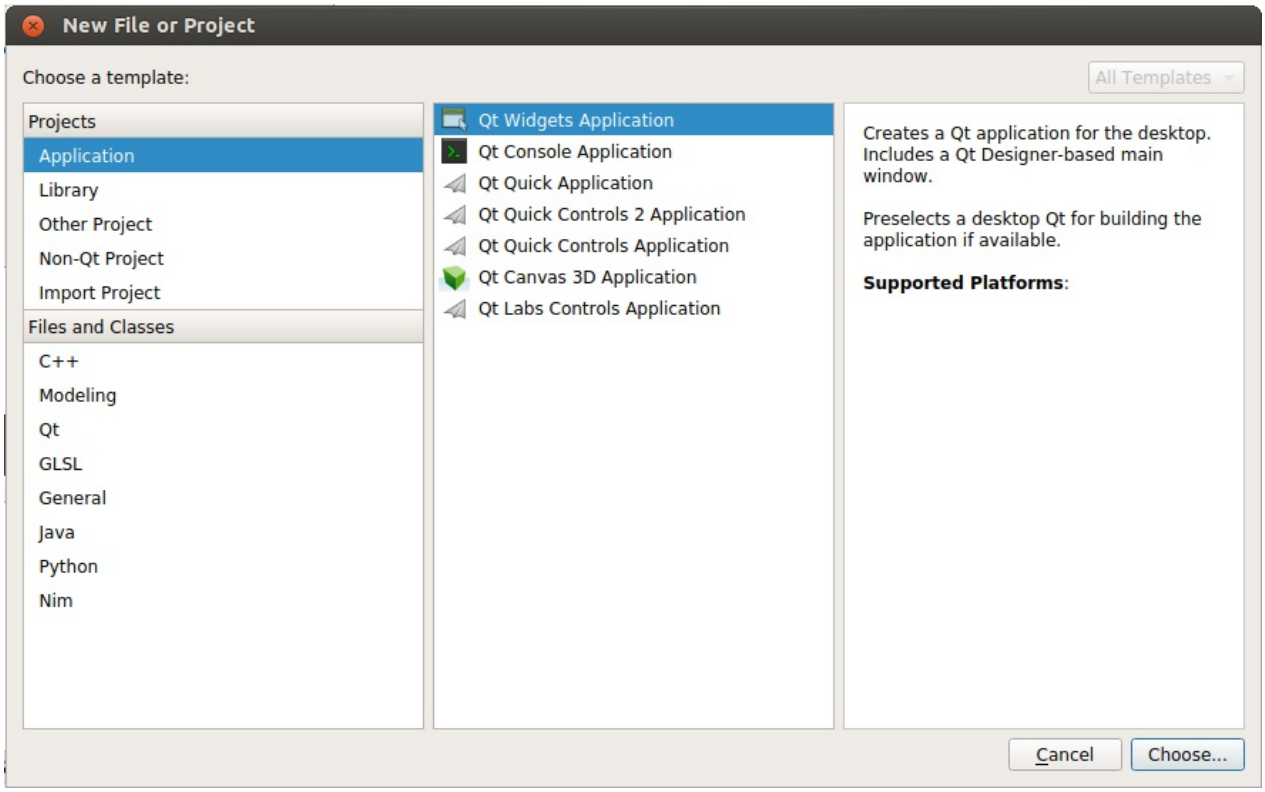

Figure 5-2-5 Create a QT Widgets Application

After pressing `Choose...` button, the `Qt Widgets Application` settings dialog popups, please set the name and path of the project as shown below in `Name` and `Create in` editboxs as shown below:

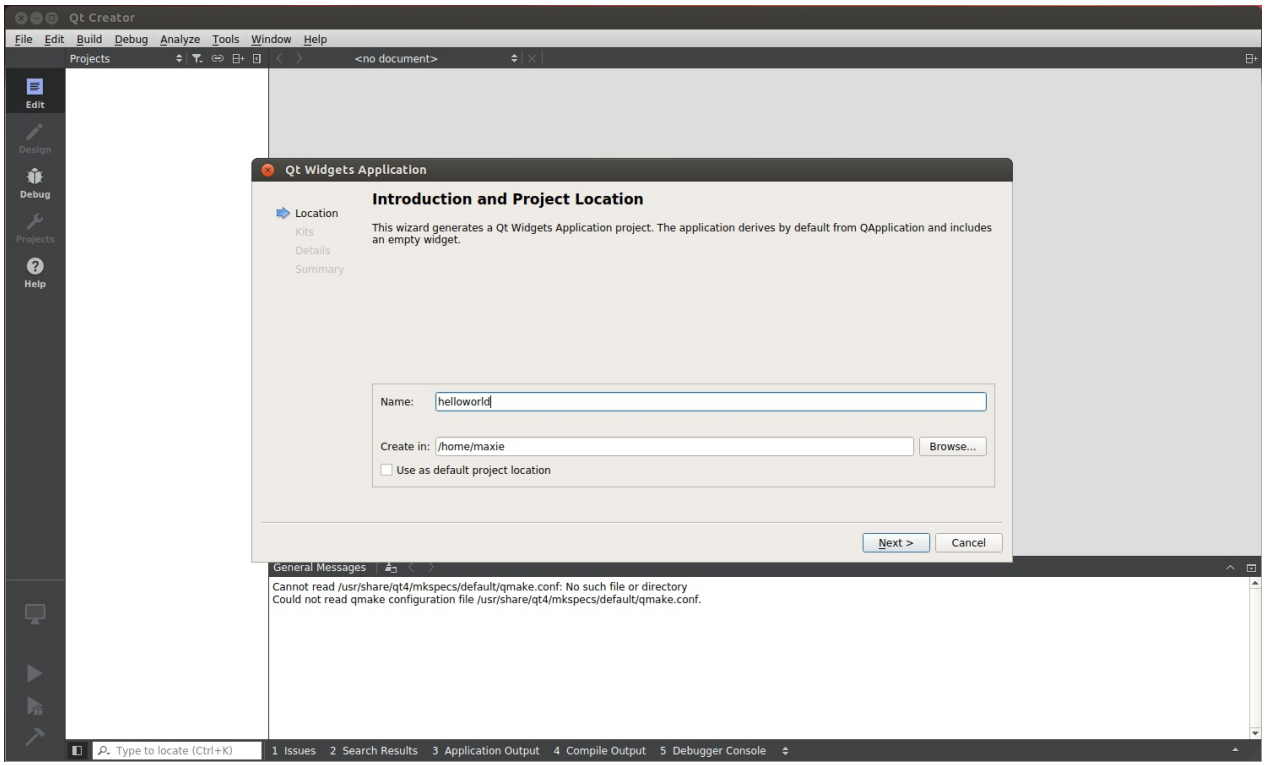

Figure 5-2-6 Set the Name and Path of the Project

Press `Next` button and choose the setting for `Kits` as below:

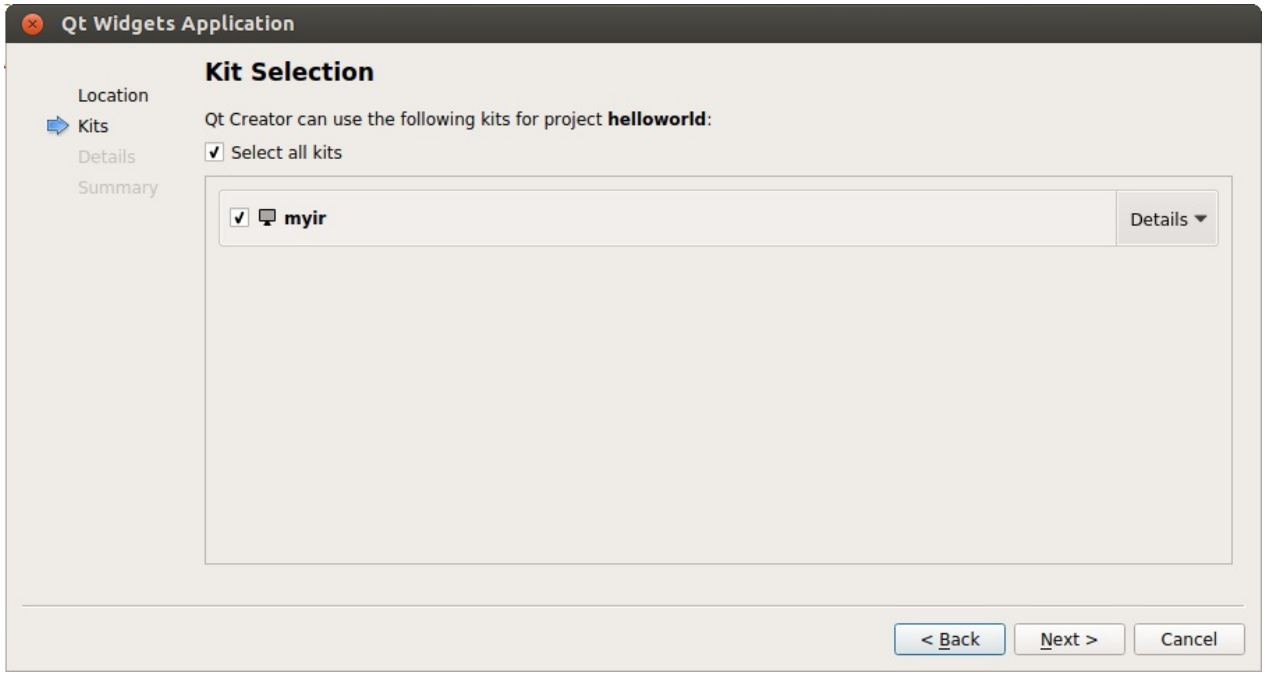

Figure 5-2-7 Set Kits for the Project

Choose the base class of the project as shown below:

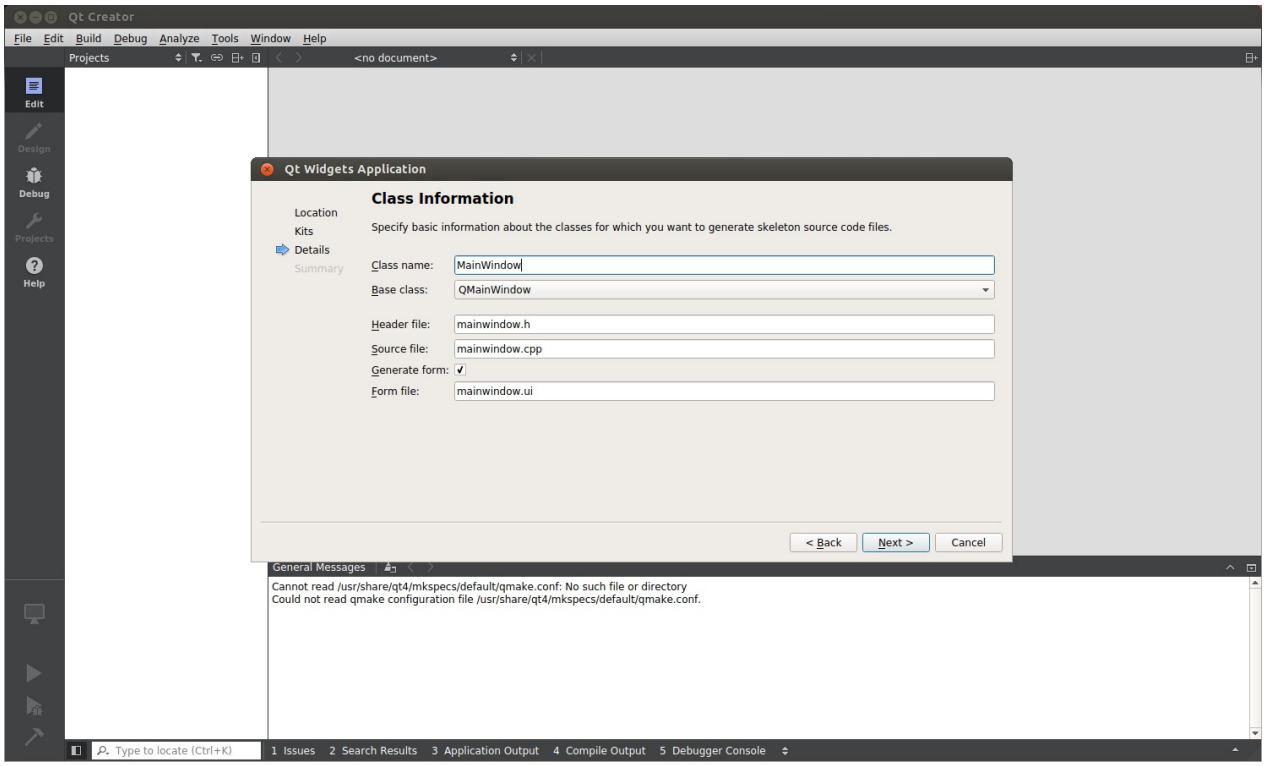

Figure 5-2-8 Choose Base Class

Press `Finish` button to create and save the project.

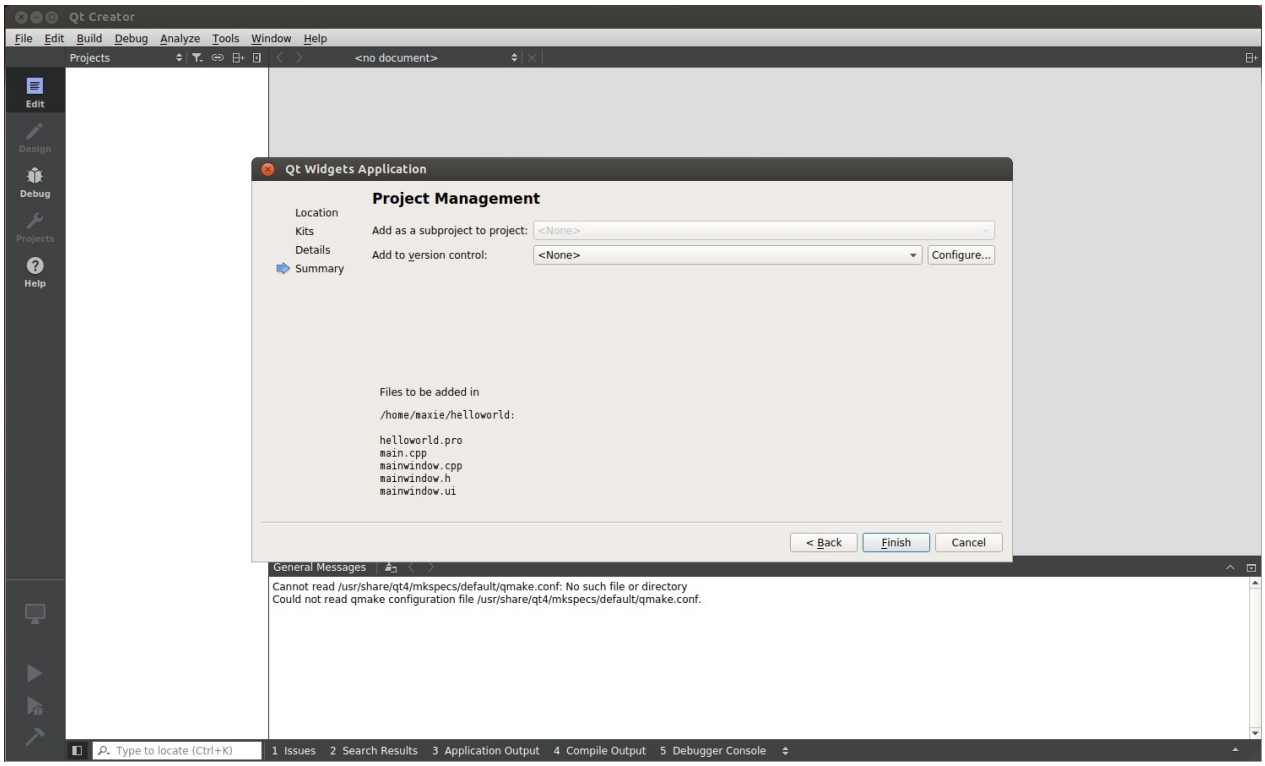

Figure 5-2-9 Finish Creating a New Project

5.3 Build QT Application

In the previous section, a QT5 project named as `helloworld` has been created.
It is shown in the project manage of QtCreator as below:

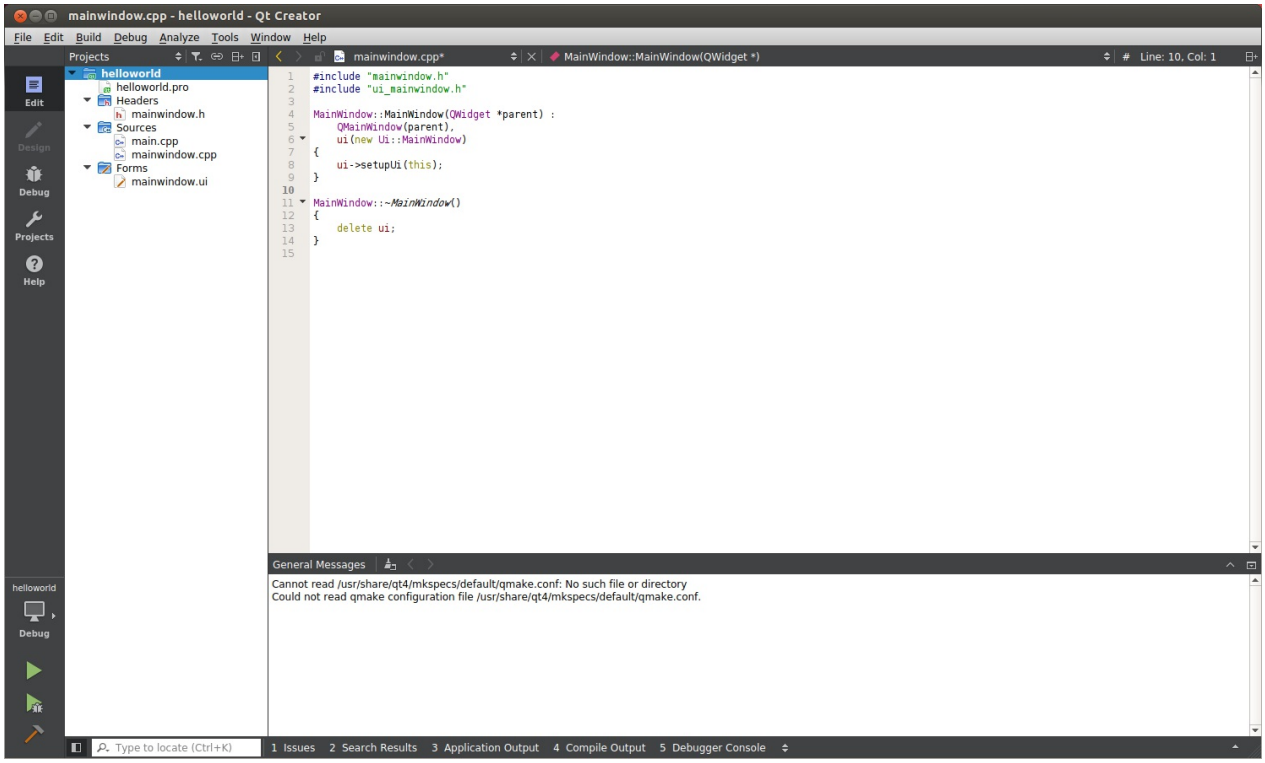

Figure 5-3-1 Project Manager of QtCreator

Double click the `mainwindow.ui` at the left side to open the `Design` view for designing a UI for helloworld project visually. Please drag a `Label` widget to the center of the mainwindow from the widgets list, double click the label and input `Hello, world!".

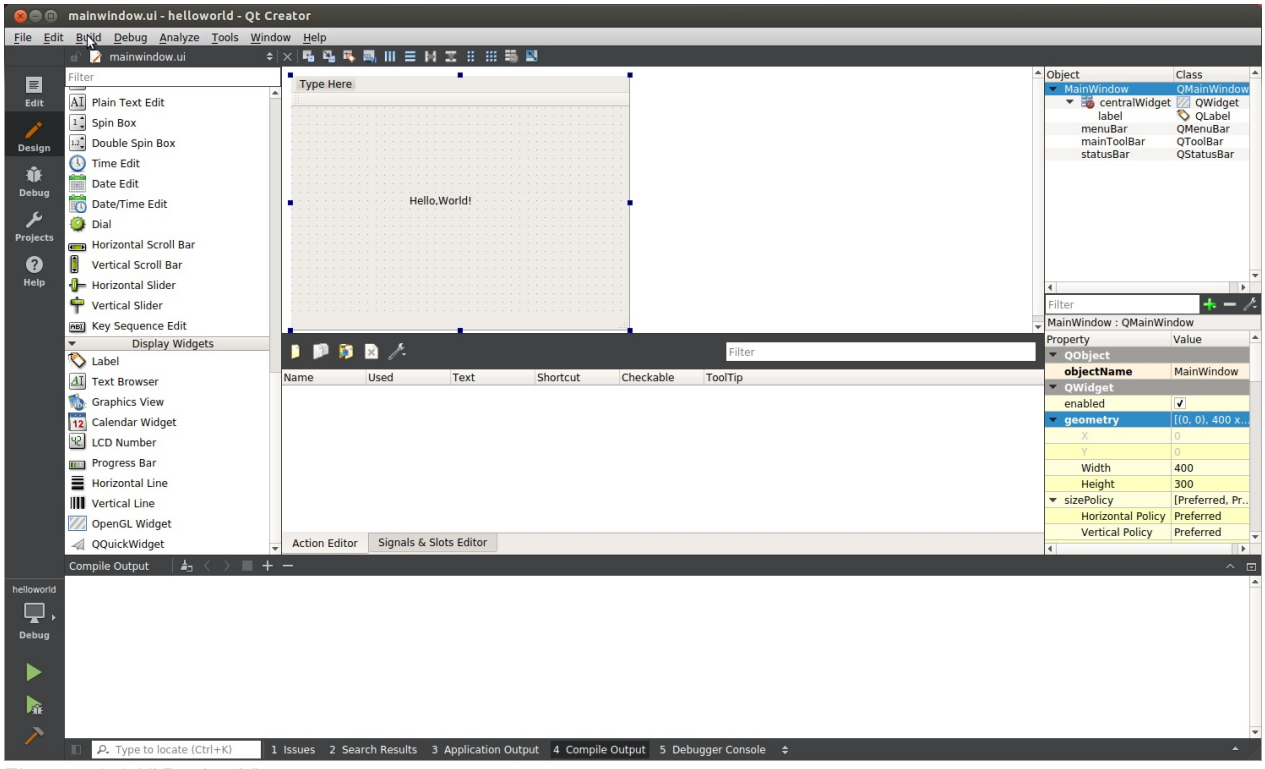

Figure 5-3-2 UI Design View

After complete, choose `Build` -> `Build Project` in the main menu of QtCreator to build the helloworld project.
Some log information outputs to the `Compile Output` sub window during compiling, in case of any erros and warning, please fix them and build again.

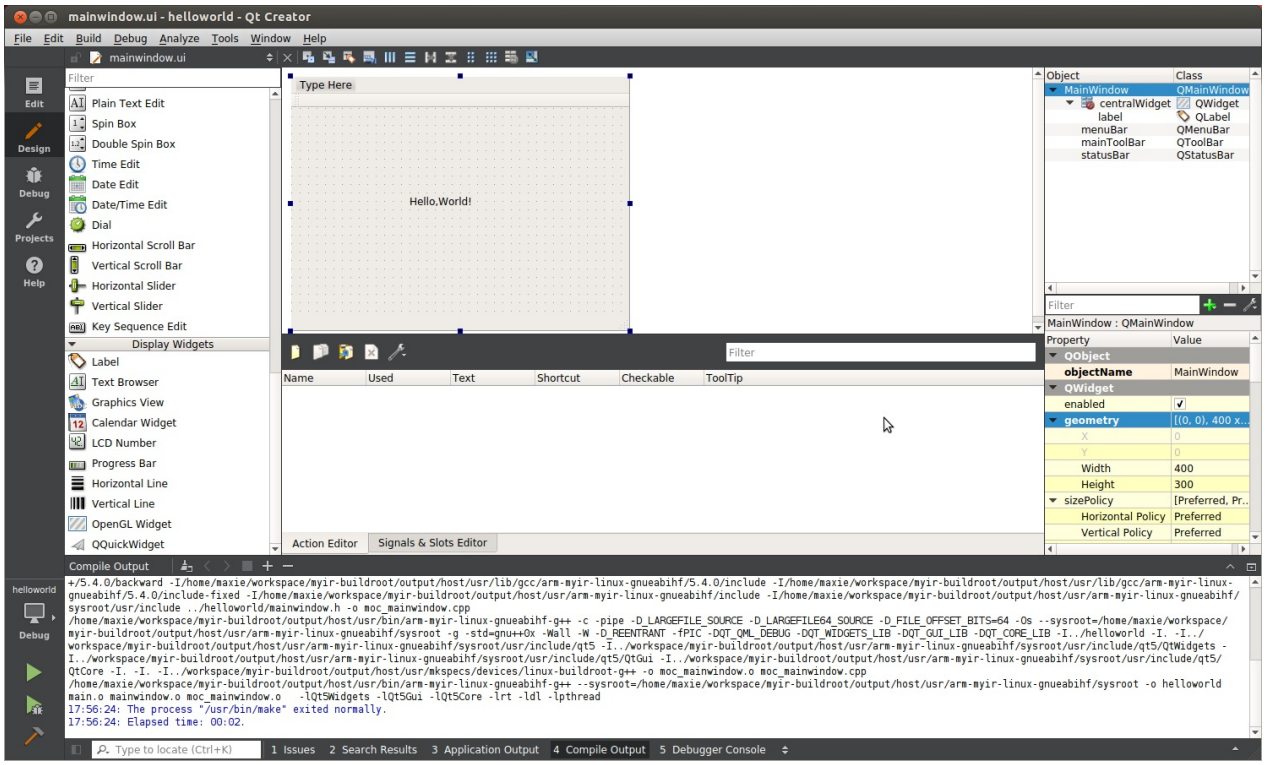

Figure 5-3-3 Build Project

After building, a binary format QT5 application is generated at *~/build-helloworld-myir_dev_kit-Debug/*, please use `file` command to check it and make sure it can work on ARM embedded Linux system as below:

```
file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (GNU/Linux), dynamically linked
 (uses shared libs), for GNU/Linux 4.1.0, not stripped
```

Finally, please copy the binary format application `helloworld` to `/usr/bin` directory of development board and execute as below:

```
# helloworld  --platform linuxfb:fb=/dev/fb0
```

A Window with a `Hello,World!` text label displays on the LCD screen as below:

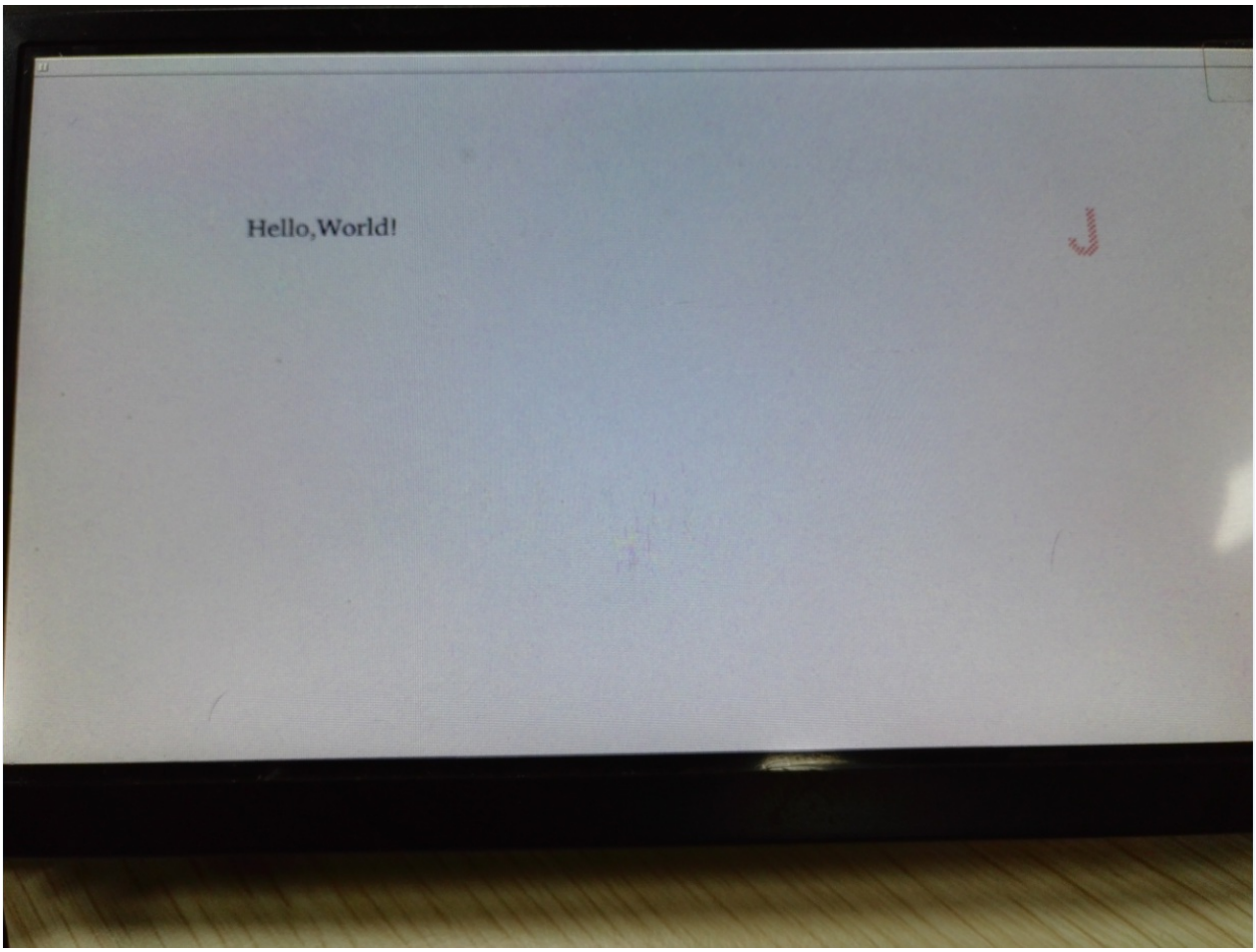

Figure 5-3-4 Execute QT5 Application

6. Update System

This chapter covers the updating of an embedded Linux system on MYD-AM437X series development board.
In our release package along with MYD-AM437X series development board , the prebuilt Linux system images located at *02-Images/linux* were provided for customers to evaluate the system quickly.

The MYD-AM437X series development board factory image file is located under the 02-Images directory of the CD-ROM, and the linux-images directory is a simplified version of the file system image.

The MEasy_HMI-images directory is a file system image that supports the MEasy_HMI demo system.
All the image files are shown in Table 6-1 as below:

Table 6-1 Prebuilt Images for MYIR-AM437X series development board

File Name	Description
MLO	First stage bootloader（SPL）
u-boot.img	Second stage bootloader
myd_c437x_evm.dtb	Device tree binary for LCD display mode of MYD-C437X
myd_c437x_evm_hdmi.dtb	Device tree binary for HDMI display mode of MYD-C437X
myd_c437x_idk.dtb	Device tree binary for PRU Ethernet work mode of MYD-C437X-PRU
myd_c437x_idk_lcd.dtb	Device tree binary for LCD work mode of MYD-C437X-PRU
zImage	Kernel image
uEnv.txt	Default environment variables for U-boot
uEnv_ramdisk.txt	Environment variables for boot ramdisk images on TF/SD card, it should be renamed to uEnv.txt and copied to TF/SD card
ramdisk.gz	ramdisk filesystem compressed by gzip
rootfs.ubi	UBIFS filesystem image
rootfs.ext4	EXT4 filesystem image
rootfs.tar.gz	Root directory compressed package
sdcard.img	TF/SD/EMMC disk image
matrix-rootfs.tar.gz	TI official file system directory compression package

There are four boot modes for MYD-AM437X series development board to run a embedded Linux system:

1. Boot from TF/SD card(EXT4 file system)
2. Boot from TF/SD card(Ramdisk file system)
3. Boot from EMMC（EXT4 file system)
4. Boot from Ethernet（NFS root for debug）

Boot from EMMC is the default for delivery. All the four boot modes will be illustrated in the following sections:

U-Boot Environment Variables

uEnv.txt is useful for U-boot to set default environment variables, During booting, U-boot loads a device tree binary file assigned by `fdtfile` environment variable.
For MYD-C437X series development board, the device tree binary file determines its work mode, so users can change the `fdtfile` environment variable in uEnv.txt to change the work mode of MYD-C437X development board. A sample uEnv.txt with `fdtfile` environment variable is shown as below:

```
fdtfile=myd_c437x_evm.dtb        # Used for LCD display mode
#fdtfile=myd_c437x_evm_hdmi.dtb   # User for HDMI dispaly mode
```

6.1 Boot from TF/SD Card(EXT4 file system)

After building Buildroot, a TF/SD card image file named as `sdcard.img` is generated at `<WORKDIR>/Filesystem/myir-buildroot/output/images` .

It consists of two partitions, one is FAT partition contains `MLO` , `u-boot.img` , `zImage` , `uEnv.txt` and device tree binary files for MYIR-AM437X series development board,the other partition is EXT4 partition, it will be used as the root partition of Linux.

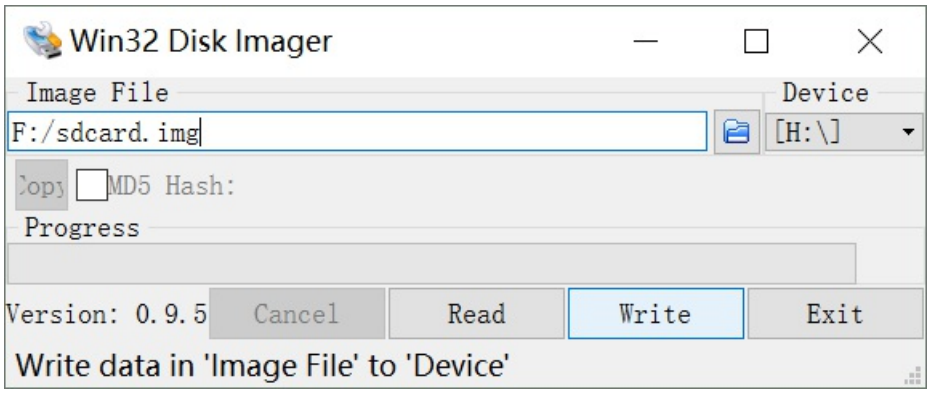

Figure 6-1-1 Write `sdcard.img` to TF/SD card with `win32diskimager`

- Put a TF card into the card reader, and connect the card reader to Windows host PC
- Write *02-Image\linux-images\sdcard.img* to the TF card with win32diskimager.exe as shown in Figure 6-1 above
- After writing, power off the development board, put the TF card to its TF slot, set the board to boot from TF/SD card by boot switch.
- Power on the development board, it will boot from TF card and mount the second partition of the TF card as root file system

> Note: Writing `sdcard.img` will format the TF card, please backup important files.

6.2 Boot from TF/SD Card(Ramdisk file system)

After building Buildroot, a compressed ramdisk file system image file named as `ramdisk.gz` is generated at `<WORKDIR>/Filesystem/myir-buildroot/output/images` .

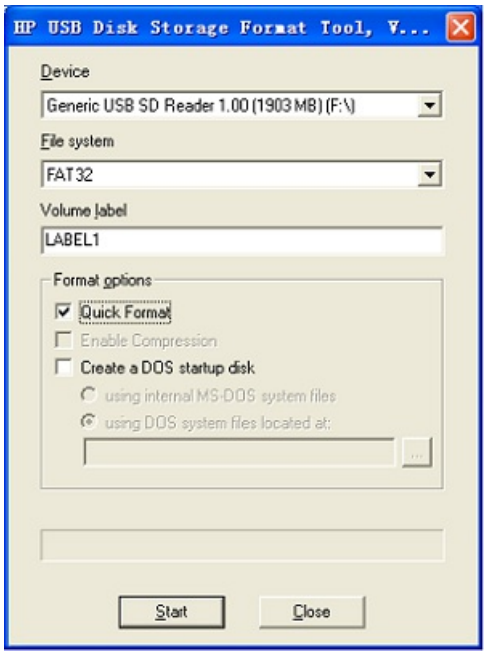

Figure 6-2-1 Formate TF/SD card

> Note: Please backup important files and use the format tool `HP USB Disk Storage Format Tool 2.0.6` in *03-Tools* of our release package to format

- Format the TF card as shown in Figure 6-2 above
- Put all the image files from *02-Images\linux-images* to the formated TF card, and rename the `uEnv_ramdisk.txt` to `uEnv.txt` , overwrite the old `uEnv.xt`
- power off the development board, put the TF card to its TF slot, set the board to boot from TF/SD card by boot switch.
- Power on the development board, it will boot from TF card and use ramdisk.gz in the TF card as root file system .

6.3 Boot from EMMC(EXT4 file system)

The `rootfs.tar.gz` file is also availabe for booting from EMMC, but it can not be written to emmc by `win32diskimager.exe` .
It can be written to EMMC after Linux booting from TF/SD card with ramdisk filesystem by some commands of Linux, please refer to `updatesys.sh` for details.

- Boot from TF/SD card with ramdisk file system
- After booting is complete, run `updatesys.sh` in Linux console to write `rootfs.tat.gz` on the TF card to EMMC as shown below.
- After writing emmc is complete, power off the development board, set it to EMMC boot mode by boot switch and take out the TF card
- Power on the development board, it will boot from EMMC and mount the second partition in the EMMC as root file system

The following to MYD-C437X development board as an example to illustrate the implementation of `updatesys.sh` process,the MYD-C437X-PRU development board is similar to the MYD-C437X.

```
#cd /
# ./updatesys.sh

All data on eMMC now will be destroyed! Continue? [y/n]
y           //User input, Verify that whether or not to erase all EMMC data
[  138.794247] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be
                corrupt. Please run fsck.
1024+0 records in
1024+0 records out
DISK SIZE - 3867148288 bytes
mkfs.fat 4.0 (2016-05-06)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
mkfs.fat 4.0 (2016-05-06)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
umount: /dev/mmcblk1p2: not mounted
mke2fs 1.43.3 (04-Sep-2016)
/dev/mmcblk1p2 contains a ext4 file system labelled 'rootfs'
        last mounted on /root/rootfs on Mon Jul  3 16:48:15 2017
Proceed anyway? (y,n) y //userinput,Verify that whether or not continue
Discarding device blocks: done
Creating filesystem with 261615 4k blocks and 65408 inodes
Filesystem UUID: ec25c446-3589-4f51-a6ff-d092053157e5
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

===> Update loader to emmc...
===> Updating kernel and devicetree to emmc...
===> Update uEnv to emmc...
===> Updating filesystem to emmc...


 Update system completed, The board can be booted from eMMC now
```

If you need to modify the size of the root partition can also modify the `updatesys.sh` script to achieve, which emmc_partition () function for the eMMC partition function.

```
emmc_partition()
{
    #
    # Format the eMMC, the partition table were be deleted
    #
    umount $EMMC_DRIVE"p1" > /dev/null 2>&1
    umount $EMMC_DRIVE"p2" > /dev/null 2>&1
    umount $EMMC_DRIVE"p3" > /dev/null 2>&1

    dd if=/dev/zero of=$EMMC_DRIVE bs=1024 count=1024
    if [ $? -ne 0 ]; then
        echo "===> Format emmc failed"
        exit 1
    fi

    SIZE=`fdisk -l $EMMC_DRIVE | grep Disk | awk '{print $5}'`

    echo DISK SIZE - $SIZE bytes

    CYLINDERS=475
    {
    echo ,395352,0x0C,*
    echo ,2092920,,-
    echo ,,,-
    } | sfdisk -u S $EMMC_DRIVE >/dev/null 2>&1

    if [ $? -ne 0 ]; then
        echo "===> eMMC partition failed"
        exit 1
    fi

    umount $EMMC_DRIVE"p1" > /dev/null 2>&1
    sleep 1
    mkfs.fat -F 32 -n "boot" "$EMMC_DRIVE"p1
    if [ $? -ne 0 ]; then
        echo "===> Creating boot partition failed"
        exit 1
    fi

    umount $EMMC_DRIVE"p3" > /dev/null 2>&1
    sleep 1
    mkfs.fat -F 32 -n "extented" "$EMMC_DRIVE"p3
    if [ $? -ne 0 ]; then
        echo "===> Create extended partition failed"
        exit 1
    fi

    umount $EMMC_DRIVE"p2" >> /dev/null
    sleep 1
    mkfs.ext4 -L "rootfs" "$EMMC_DRIVE"p2
    if [ $? -ne 0 ]; then
        echo "===> Creating rootfs partition failed"
        exit 1
    fi

    mkdir $EMMC_BOOT_MP
    mount $EMMC_DRIVE"p1" $EMMC_BOOT_MP
    mkdir $EMMC_ROOTFS_MP
    mount -t ext4 $EMMC_DRIVE"p2" $EMMC_ROOTFS_MP
}
```

Partition is mainly used to achieve the `sfdisk` command, format: start, size, ID, default partition information as shown below, the user can modify according to their own needs.

```
{
echo ,395352,0x0C,*   //Partition P1, boot partition
echo ,2092920,,-      //Partition P2, root partition
echo ,,,-             //Partition P3, extended partition
} | sfdisk -u S $EMMC_DRIVE >/dev/null 2>&1
```

After the partition is completed in the terminal using `fdisk -l` can see eMMC partition information:

```
Device         Boot   Start     End Sectors  Size Id Type
/dev/mmcblk0p1 *       2048  397399  395352  193M  c W95 FAT32 (LBA)
/dev/mmcblk0p2       399360 2492279 2092920 1022M 83 Linux
/dev/mmcblk0p3      2492416 7553023 5060608  2.4G 83 Linux
```

6.4 Boot from Ethernet(NFS root filesystem)

After building Buildroot, a compressed package named as `rootfs.tar.gz` is generated at `<WORKDIR>/Filesystem/myir-buildroot/output/images` .
This package can be used to work as NFS root for development board. In order to boot from ethernet, TFTP and NFS services should be
installed and configed as below:

- Install TFTP Service

```
$ sudo apt-get install tftp-hpa tftpd-hpa
```

- Config TFTP Service

Create a work directory for TFTP, open the configuration file for TFTP as shown below:

```
$ mkdir -p  <WORKDIR>/tftpboot
$ chmod 777 <WORKDIR>/tftpboot
$ sudo vi /etc/default/tftpd-hpa
```

Add or modify the parameters as shown below:

```
TFTP_DIRECTORY="<WORKDIR>/tftpboot"
TFTP_OPTIONS="-l -c -s"
```

Restart TFTP Service:

```
$ sudo service tftpd-hpa restart
```

Copy the `MLO` , `u-boot.img` , `zImage` , `ramdisk.gz` and device tree binary files to the work directory of TFTP service, then users can load these image files to the RAM of development board by TFTP in U-boot console, it is shown below:

```
>#  help tftpboot
tftpboot - boot image via network using TFTP protocol

Usage:
tftpboot [loadAddress] [[hostIPaddr:]bootfilename]
>#  tftpboot ${loadaddr} 192.168.1.111:zImage
```

- Install NFS Service

NFS(Network File System) is a file system can be mounted remotely through network. A directory on NFS server can be used as the root file system of an embedded Linux system. The installation and configuration of NFS service are described below:

```
$ sudo apt-get install nfs-kernel-server
```

- Config NFS Service

Edit the `/etc/exports` file of NFS server, and export a directory at the end of file:

```
$ sudo vi /etc/exports
```

Add or modify the directory to be exported , such as `/home/myir/rootfs` has been added as below:

```
/home/myir/rootfs *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
```

Restart NFS service:

```
$ cd /home/myir/rootfs
$ sudo tar zxvf  <WORKDIR>/images/rootfs.tar.gz
$ sudo service nfs-kernel-server restart
```

Verify NFS service on NFS server:

```
$ sudo mount -t nfs 127.0.0.1:/home/myir/rootfs /mnt
```

If NFS service works well, `/home/myir/rootfs` will be mounted at `/mnt` with NFS , and then the NFS server is available for development board.
For example, the IP address of the NFS server is set to `192.168.1.111` , then customers can set the IP of development board to the same sub network, such as
`192.168.1.112` in U-boot console, it is shown below:

```
>## setenv ipaddr 192.168.1.112
```

Verify the ethernet connection by `ping` command in U-boot console:

```
>## ping 192.168.1.111
```

Run `run netboot` command to load image files from tftpboot and boot with NFS root filesystem in U-boot console:

```
>## setenv serverip 192.168.1.111
>## setenv ipaddr 192.168.1.112
>## ping 192.168.1.111                   -- verify ethernet connection
>## setenv rootpath /home/myir/rootfs
>## run findfdt
>## echo $fdtfile                            -- display the dtb file name
>## run netboot
```

6.5 Matrix-rootfs Tutorial

Currently only EMMC boot and NFS startup are supported,only AM4378 and AM4379 cpu can be used.

EMMC boot：

Update the file-system in EMMC

- First, use start mode `Boot from TF/SD card (Ramdisk file system)` , confirm the TF card has matrix-rootfs.tar.gz compression package, start the development board.
- After the system is started, go to the linux console and modify the value of FILE_FILESYSTEM in updatasys.sh. The modification process is as follows：

```
# cd /
# vi updatesys.sh


# This script to update myd_c437x_evm or Rico Board system
# This script will respectively update the u-boot, device tree, zImage to
# QSPI.U_BOOT, QSPI.U-BOOT-DEVICETREE, QSPI.KERNEL, and update the filesystem
# to emmc
#
# Author: MYiR
# Email: support@myirtech.com
# Date: 2015.1.21
#       Initial Version
# Date: 2017.05.24
#       update to kernel4.1.18 and u-boot201605

#!/bin/sh

# The path sdcard mounted
SD_MOUNT_POINT="/media/mmcblk1p1"
# The rootfs partition would be mounted on current 'rootfs' directory
EMMC_BOOT_MP="boot"
EMMC_ROOTFS_MP="rootfs"

FILE_MLO="MLO"
if [ "$1" = "loader2qspi" ]; then
        FILE_UBOOT="u-boot.bin"
else
        FILE_UBOOT="u-boot.img"
fi
FILE_ZIMAGE="zImage"
FILE_DEVICETREE="myd_c437x_evm.dtb"
FILE_FILESYSTEM="matrix-rootfs.tar.gz"
FILE_RAMDISK="ramdisk.gz"
FILE_UBOOTENV="u-boot-env.bin"
FILE_UENV="uEnv"
FILE_DEFAULT_UENV="uEnv/uEnv.txt"
- updatesys.sh 1/286 0%
```

Change the value of FILE_FILESYSTEM to matrix-rootfs.tar.gz, save and exit, and then execute the updatasys.sh script to update the file system on the EMMC to matrix-rootfs.

- After the programming is complete, unplug the TF card, set the development board for EMMC boot mode, power can boot from the EMMC boot, and load the matrix-rootfs file system on EMMC.

NFS Boot：

The main process with reference to the previous program `4: Boot from Ethernet（NFS root for debug)` , where the matrix-rootfs.tar.gz extract to the NFS ROOT root file system directory.

```
$ cd /home/myir/rootfs
$ sudo tar zxvf  <WORKDIR>/images/matrix-rootfs.tar.gz
$ sudo service nfs-kernel-server restart
```

After the operation is completed, start the development board according to the NFS start mode, and enter the TI matrix-rootfs demo system.

Appendix A Warranty & Technical Support Services

MYIR Tech Limited is a global provider of ARM hardware and software tools, design solutions for embedded applications. We support our customers in a wide range of services to accelerate your time to market.

MYIR is an ARM Connected Community Member and work closely with ARM and many semiconductor vendors. We sell products ranging from board level products such as development boards, single board computers and CPU modules to help with your evaluation, prototype, and system integration or creating your own applications. Our products are used widely in industrial control, medical devices, consumer electronic, telecommunication systems, Human Machine Interface (HMI) and more other embedded
applications. MYIR has an experienced team and provides custom design services based on ARM processors to help customers make your idea a reality.

The contents below introduce to customers the warranty and technical support services provided by MYIR as well as the matters needing attention in using MYIR's products.

Service Guarantee

MYIR regards the product quality as the life of an enterprise. We strictly check and control the core board design, the procurement of components, production control, product testing, packaging, shipping and other aspects and strive to provide products with best quality to customers. We believe that only quality products and excellent services can ensure the long-term cooperation and mutual benefit.

Price

MYIR insists on providing customers with the most valuable products. We do not pursue excess profits which we think only for short-time cooperation. Instead, we hope to establish long-term cooperation and win-win business with customers. So we will offer reasonable prices in the hope of making the business greater with the customers together hand in hand.

Delivery Time

MYIR will always keep a certain stock for its regular products. If your order quantity is less than the amount of inventory, the delivery time would be within three days; if your order quantity is greater than the number of inventory, the delivery time would be always four to six weeks. If for any urgent delivery, we can negotiate with customer and try to supply the goods in advance.

Technical Support

MYIR has a professional technical support team. Customer can contact us by email (support@myirtech.com), we will try to reply you within 48 hours. For mass production and customized products, we will specify person to follow the case and ensure the smooth production.

After-sale Service

MYIR offers one year free technical support and after-sales maintenance service from the purchase date. The service covers:

Technical support service

- MYIR offers technical support for the hardware and software materials which have provided to customers;

- To help customers compile and run the source code we offer;
- To help customers solve problems occurred during operations if users follow the user manual documents;
- To judge whether the failure exists;
- To provide free software upgrading service.

However, the following situations are not included in the scope of our free technical support service:

- Hardware or software problems occurred during customers' own development;
- Problems occurred when customers compile or run the OS which is tailored by themselves;
- Problems occurred during customers' own applications development;
- Problems occurred during the modification of MYIR's software source code.

After-sales maintenance service

The products except LCD, which are not used properly, will take the twelve months free maintenance service since the purchase date. But following situations are not included in the scope of our free maintenance service:

- The warranty period is expired;
- The customer cannot provide proof-of-purchase or the product has no serial number;
- The customer has not followed the instruction of the manual which has caused the damage the product;
- Due to the natural disasters (unexpected matters), or natural attrition of the components, or unexpected matters leads the defects of appearance/function;
- Due to the power supply, bump, leaking of the roof, pets, moist, impurities into the boards, all those reasons which have caused the damage of the products or defects of appearance;
- Due to unauthorized weld or dismantle parts or repair the products which has caused the damage of the products or defects of appearance;
- Due to unauthorized installation of the software, system or incorrect configuration or computer virus which has caused the damage of products.

Warm tips:

1. MYIR does not supply maintenance service to LCD. We suggest the customer first check the LCD when receiving the goods. In case the LCD cannot run or no display, customer should contact MYIR within 7 business days from the moment get the goods.
2. Please do not use finger nails or hard sharp object to touch the surface of the LCD.
3. MYIR suggests user purchasing a piece of special wiper to wipe the LCD after long time use, please avoid clean the surface with fingers or hands to leave fingerprint.
4. Do not clean the surface of the screen with chemicals.
5. Please read through the product user manual before you using MYIR's products.
6. For any maintenance service, customers should communicate with MYIR to confirm the issue first. MYIR's support team will judge the failure to see if the goods need to be returned for repair service, we will issue you RMA number for return maintenance service after confirmation.

Maintenance period and charges

- MYIR will test the products within three days after receipt of the returned goods and inform customer the testing result. Then we will arrange shipment within one week for the repaired goods to the customer. For any special failure, we will negotiate with customers to confirm the maintenance period.
- For products within warranty period and caused by quality problem, MYIR offers free maintenance service; for products within warranty period but out of free maintenance service scope, MYIR provides maintenance service but shall charge some basic material cost; for products out of warranty period, MYIR provides maintenance service but shall charge some basic material cost and handling fee.

Shipping cost

During the warranty period, the shipping cost which delivered to MYIR should be responsible by user; MYIR will pay for the return shipping cost to users when the product is repaired. If the warranty period is expired, all the shipping cost will be responsible by users.

Products Life Cycle

MYIR will always select mainstream chips for our design, thus to ensure at least ten years continuous supply; if meeting some main chip stopping production, we will inform customers in time and assist customers with products updating and upgrading.

Value-added Services

1. MYIR provides services of driver development base on MYIR's products, like serial port, USB, Ethernet, LCD, etc.
2. MYIR provides the services of OS porting, BSP drivers' development, API software development, etc.
3. MYIR provides other products supporting services like power adapter, LCD panel, etc.
4. ODM/OEM services.

MYIR Tech Limited

Address: Room 04, 6th Floor, Building No.2, Fada Road, Yunli Smart Park, Bantian, Longgang District, Shenzhen, Guangdong, China 518129

Support Email: support@myirtech.com

Sales Email: sales@myirtech.com

Phone: +86-755-22984836

Fax: +86-755-25532724

Website: www.myirtech.com