EM-TF-EVK-AM5728

Linux Development Guide

Version: 1.0

2024-05-23

Disclaimer

- Shenzhen Emtop Co., Ltd. does not provide any type of guarantee for the program source code, software, documents, etc. provided with the product; Regardless of whether it is described or implied, including but not limited to the warranty of fitness for a particular purpose, the entire risk is born by the user.

Revision History：

Version	Date	Description
1.0	2024-05-23	First Release

Table of Contents

1. ENVIRONMENT SETUP 6

1.1 BUILD A DEVELOPMENT ENVIRONMENT 6

1.2 CONFIGURE COMPILATION ENVIRONMENT 6

1.3 OTHER TOOLS AND SERVICES 7

2. COMPILE SOURCE CODE 8

2.1 U-BOOT 8

2.1.1 GET U-BOOT SOURCE CODE 8

2.1.2 COMPILE U-BOOT 8

2.2 KERNEL 8

2.2.1 GET KERNEL SOURCE CODE 8

2.2.2 COMPILE KERNEL 9

2.3 EXTERNAL DRIVE 9

2.3.1 CONFIGURE ENVIRONMENT VARIABLES 9

2.3.2 COMPILE EXTERNAL DRIVER 11

2.3.3 INSTALL EXTERNAL DRIVER 11

3. CREATE TARGET IMAGE 14

3.1 MAKE IMAGE FILE 14

3.1.1 FORMAT THE PARTITIONS 14

3.1.2 COPY FIRMWARE 17

3.1.3 COPY ROOTFS 17

3.1.4 INSTALL KERNEL MODULES 18

3.1.5 INSTALL EXTERNAL DRIVER MODULE 18

3.2 BURN AND READ IMAGE 19

3.2.1 BURN IMAGE 19

3.2.2 READ IMAGE 19

4. TI SDK DEVELOPMENT 21

4.1 SDK INSTALLATION AND CONFIGURATION 21

4.2 TOP-LEVEL MAKEFILE USAGE 25

4.2.1 COMPILE THE ENTIRE SDK 25

4.2.2 COMPILE AND INSTALL A TARGET SEPARATELY 26

5. APPLICATION DEVELOPMENT 29

5.1 CROSS COMPILE AND RUN ORDINARY C PROGRAMS 29

5.1.1 WRITE C PROGRAM CODE 29

5.1.2 COMPILE ON HOST 29

5.1.3 COMPILE ON ARM BOARD 29

5.1.4 TRANSMIT TO ARM BOARD AND RUN 30

5.2 QT APPLICATION DEVELOPMENT 30

5.2.1 INSTALL QT CREATOR 30

5.2.2 CONFIGURE QT CREATOR 37

5.2.3 CREATE DEMO 41

5.2.4 RUNNING ON ARM BOARD 47

5.3 VIDEO CAPTURE DEMO 50

5.4 DUAL DISPLAY DEMO 51

6. ROOTFS BUILD BASED ON YOCTO 53

6.1 INSTALL REQUIRED TOOL SOFTWARE 53

6.2 CONFIGURE BASH 53

6.3 INSTALL COMPILER 54

6.4 OBTAIN OE-LAYERTOOL-SETUP.SH 54

6.5 BITBAKE BUILD 54

7. APPENDIX 56

1. ENVIRONMENT SETUP

1.1 BUILD A DEVELOPMENT ENVIRONMENT

Requirements for setting up the development environment:

- Hardware: At least 20GB disk space, 2GB RAM
- Software: Ubuntu 64 bit OS, 14.04 LTS, 16.04LTS or higher LTS version (Ubuntu Desktop or Ubuntu Server)

Note：

🕮 If you want to develop Qt, please install Ubuntu version with a GUI interface such as Ubuntu Desktop.

You can also use a virtual machine to run the Ubuntu 64 bit OS. After starting the Ubuntu system, run the following commands to install the software required for development.

- **$ sudo apt-get update**
- **$ apt-get install -y openssh-server git kpartx lzop lsb-core libncurses5**

1.2 CONFIGURE COMPILATION ENVIRONMENT

Copy gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf.tar.xz from the release folder to the $HOME directory in the Linux environment, and unzip it:

- **$ tar -Jxvf gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf.tar.xz**

Set environment variables:

- **$ export CROSS_COMPILE=$HOME/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-**
- **$ export ARCH=arm**

Note: Environment variables must be configured before compiling u-boot and kernel each time. For convenience, you can edit a script and then source the script:

- **$ cat $HOME/set_am57_env.sh**

```
#!/bin/bash

export CROSS_COMPILE=$HOME/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf
/bin/arm-linux-gnueabihf-
export ARCH=arm
```

- **$ source $HOME/set_am57_env.sh**

Note：

- The command strings starting with $ in this article should run under Ubuntu PC;
- $HOME appearing in the article is not mandatory and should be modified accordingly based on the user's actual directory;
- In this article, the command strings starting with "root@arm:~#" should run under the ARM board.

1.3 OTHER TOOLS AND SERVICES

Other tools and services may also be used during the development process, such as:

- ssh login ubuntu/serial port login: PuTTY software
- SAMBA service for transferring files between linux and windows
- NFS
- TFTP

The use of these conventional development tools isn't mentioned in the document. Please search on web for more information.

2. COMPILE SOURCE CODE

2.1 U-Boot

2.1.1 GET U-BOOT SOURCE CODE

Copy u-boot*.tar.gz to $HOME and unzip it:

- **$ cd $HOME**
- **$ tar -xzvf u-boot*.tar.gz**

2.1.2 COMPILE U-BOOT

- **$ cd $HOME/u-boot**
- **$ make distclean**
- **$ make som_am572x_defconfig**
- **$ make**

After compilation is completed, **u-boot.img** and **MLO** are generated in the **$HOME/ u-boot** directory.

2.2 KERNEL

2.2.1 GET KERNEL SOURCE CODE

Copy the Linux kernel source code package **linux*.tar.gz** to **$HOME** and unzip it:

- **$ tar -zxvf linux*.tar.gz**

2.2.2 COMPILE KERNEL

- **$ cd $HOME/linux**
- **$ make distclean**
- **$ make embest_ti_am57xx_defconfig**
- **$ make**

After compilation is completed, the target file **zImage** and **dtb** are generated:

- $HOME/linux/arch/arm/boot/zImage
- $HOME/linux/arch/arm/boot/dts/embest-SOM_AM572x_TM-mode0.dtb
- $HOME/linux/arch/arm/boot/dts/embest-SOM_AM572x_TM-mode0-LCD.dtb

embest-SOM_AM572x_TM-mode0.dtb sets HDMI as the main display, and **embest-SOM_AM572x_TM-mode0-LCD.dtb** sets LCD as the main display.

2.3 EXTERNAL DRIVE

Since the drivers for some of TI's peripheral modules are released separately, these drivers require additional compilation. These peripheral modules include 2D/3D image acceleration modules, hardware encryption/decryption modules, etc. Copy **extra.tar.gz** to $HOME and extract it.

- **$ cd $HOME**
- **$ tar -xzf extra.tar.gz**

2.3.1 CONFIGURE ENVIRONMENT VARIABLES

Edit the **Rules.make** file and modify the following variables to the corresponding values:

- **DESTDIR** The path of the root file system, such as the mounting path of the SD card that has been burned with the image under ubuntu, or it can be any other

blank directory.

- **CROSS_COMPILE** is the path where the cross-compilation tool chain is located.
- **LINUXKERNEL_INSTALL_DIR** is the Linux kernel source code path. Before compiling the external driver, make sure that the Linux kernel source code is correctly configured and compiled. Refer to **2.2 KERNEL**.

  - **$ cd extra**
  - **$ ls**

```
extra-drivers  Makefile  Rules.make
```

  - **$ cat Rules.make**

```
#platform
#platform
PLATFORM=am57xx-evm

#root of the target file system for installing applications
DESTDIR=$(HOME)/extra/fakeroot

#Cross compiler prefix
export
CROSS_COMPILE=$(HOME)/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin/ar
m-linux-gnueabihf-

#The directory that points to the SDK kernel source tree
LINUXKERNEL_INSTALL_DIR=$(HOME)/linux

CFLAGS= -march=armv7-a -marm -mfpu=neon -mfloat-abi=hard

#Strip modules when installing to conserve disk space
INSTALL_MOD_STRIP=1

export TOOLCHAIN_PREFIX=$(CROSS_COMPILE)
```

2.3.2 COMPILE EXTERNAL DRIVER

Before compiling, create a empty directory specified by the **DESTDIR** variable in **Rules.make**. Here we create the **fakeroot** directory:

- **$ cd extra**
- **$ mkdir fakeroot**
- **$ ls**

```
extra-drivers  fakeroot  Makefile  Rules.make
```

- **$ make clean**
- **$ make**

2.3.3 INSTALL EXTERNAL DRIVER

- **$ make install**

The compiled ko and related files will be installed under **fakeroot**. If you only need to update the ko file, you only need to copy the ko files to **lib/modules/4.9.28/extra** to the corresponding directory of the target board file system.

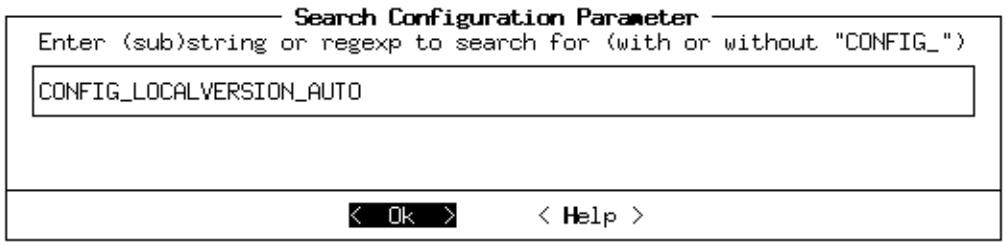
- **$ tree fakeroot**

```
fakeroot/
├── lib
│   ├── firmware
│   │   └── jailhouse.bin
│   └── modules
│       └── 4.9.28
│           ├── extra
│           │   ├── bc_example.ko
│           │   ├── cmemk.ko
│           │   ├── cryptodev.ko
│           │   ├── debugss_kmodule.ko
│           │   ├── driver
│           │   │   └── jailhouse.ko
```

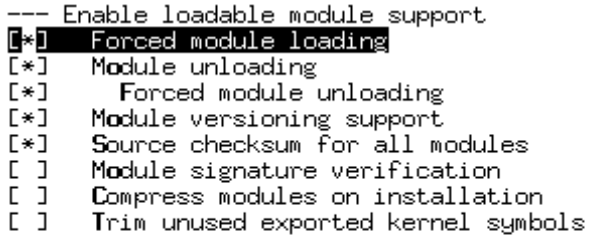
```
|   |   ├──galcore.ko
|   |   ├──gdbserverproxy.ko
|   |   ├──pvrsrvkm.ko
|   |   └──uio_module_drv.ko
|   ├──modules.alias
|   ├──modules.alias.bin
|   ├──modules.builtin.bin
|   ├──modules.dep
|   ├──modules.dep.bin
|   ├──modules.devname
|   ├──modules.softdep
|   ├──modules.symbols
|   └──modules.symbols.bin
└──usr
├──libexec
|   └── jailhouse
|   ├── jailhouse-cell-linux
|   ├── jailhouse-cell-stats
|   ├── jailhouse-config-create
|   ├── jailhouse-hardware-check
|   └──linux-loader.bin
├──sbin
|   └── jailhouse
└── share
├── bash-completion
|   └── completions
|   └── jailhouse
└── jailhouse
├── jailhouse-config-collect.tmpl
└── root-cell-config.c.tmpl

14 directories, 28 files
```

For development convenience, you can turn off the git version control option in the kernel. The kernel versions compiled in the future will always be linux-4.9.28, so there is no need to repeatedly compile extra-related driver modules.

Enter **make menuconfig** in the kernel root directory to open the configuration interf ace, type / in the menuconfig interface, and search CONFIG_LOCALVERSION_AUT O option, find the relevant option according to the prompts, and set it to [=n]:



During the development phase, CONFIG_MODULE_FORCE_LOAD can be enabled.



3. CREATE TARGET IMAGE

3.1 MAKE IMAGE FILE

There are prepared image files (such as **EM-TF-EVK-AM5728-TI-ShipmentImage-SDcard-V1.0.3r04.img**) in the release folder for users to use the **dd** command (Linux OS) and **Win32DiskImager.exe** (Windows OS) to write into the SD card or EMMC, please refer to the user manual for details. In this section we will introduce how to make the target files compiled in the previous sections into an image file that can be used for burning. The size of the image file we create should not exceed the capacity of the SD card/EMMC to be burned. Assuming the SD card capacity is 4GB, we can create a 3800MB blank disk file named **example.img** under Ubuntu:

- **$ cd $HOME**
- **$ sudo dd if=/dev/zero of=./example.img bs=1M count=3800**

```
3800+0 records in
3800+0 records out
3984588800 bytes (4.0 GB, 3.7 GiB) copied, 9.85035 s, 405 MB/s
```

3.1.1 FORMAT THE PARTITIONS

Format **example.img**, create two partitions: FAT32 (64MB, storing firmware) and EXT4 (storing rootfs).

- **$ cd $HOME**
- **$ sudo fdisk example.img**

```
Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
```

```
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0x928aa0d6.

Command (m for help): o
Created a new DOS disklabel with disk identifier 0x12076151.

Command (m for help): n
Partition type
p primary (0 primary, 0 extended, 4 free)
e extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-7782399, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-7782399, default 7782399): +64M

Created a new partition 1 of type 'Linux' and of size 64 MiB.

Command (m for help): t
Selected partition 1
Partition type (type L to list all types): c
Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.
Command (m for help): a
Selected partition 1 The bootable flag on partition 1 is enabled now.
Command (m for help): n
Partition type
p primary (1 primary, 0 extended, 3 free)
e extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2):
First sector (133120-7782399, default 133120):
Last sector, +sectors or +size{K,M,G,T,P} (133120-7782399, default 7782399):

Created a new partition 2 of type 'Linux' and of size 3.7 GiB.

Command (m for help): p
Disk example.img: 3.7 GiB, 3984588800 bytes, 7782400 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes I/O size (minimum/optimal): 512
```

```
bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x12076151

Device          Boot   Start     End         Sectors     Size   Id    Type
example.img1 *         2048      133119      131072      64M    c     W95 FAT32 (LBA)
example.img2           133120    7782399     7649280     3.7G   83    Linux

Command (m for help): w
The partition table has been altered.
Syncing disks.
```

Format the FAT32 partition and set the volume label to boot, format the ext4 partition and set the volume label to rootfs.

- **$ sudo losetup /dev/loop0 example.img**
- **$ sudo kpartx -av /dev/loop0**

```
add map loop0p1 (253:0): 0 131072 linear 7:0 2048
add map loop0p2 (253:1): 0 7649280 linear 7:0 133120
```

- **$ ls /dev/mapper/loop0p***

```
/dev/mapper/loop0p1  /dev/mapper/loop0p2
```

- **$ sudo mkfs.vfat -F 32 -n "boot" /dev/mapper/loop0p1**

```
mkfs.fat 3.0.28 (2015-05-16)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
unable to get drive geometry, using default 255/63
```

- **$ sudo mkfs.ext4 -L "rootfs" /dev/mapper/loop0p2**

```
mke2fs 1.42.13 (17-May-2015)
Discarding device blocks: done
Creating filesystem with 956160 4k blocks and 239040 inodes
Filesystem UUID: 0820d179-521d-4f91-816f-df13309eee87
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

3.1.2 COPY FIRMWARE

Create two temporary directories, respectively for mounting:

- **$ mkdir boot rootfs**
- **$ sudo mount /dev/mapper/loop0p1 boot/**
- **$ sudo mount /dev/mapper/loop0p2 rootfs/**

You can check whether the mount is successful with the **lsblk** command:

- **$ lsblk**

```
NAME        MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
loop0         7:0    0  3.7G  0 loop
├─loop0p2   253:1    0  3.7G  0 part  /home/david/rootfs
└─loop0p1   253:0    0   64M  0 part  /home/david/boot
```

- **$ sudo cp $HOME/u-boot/u-boot.img ./boot**
- **$ sudo cp $HOME/u-boot/MLO ./boot**
- **$ sudo cp $HOME/linux/arch/arm/boot/zImage ./boot**
- **$ sudo cp $HOME/linux/arch/arm/boot/dts/embest-SOM_AM572x_TM-mode0.dtb ./boot**
- **$ sudo cp $HOME/linux/arch/arm/boot/dts/embest-SOM_AM572x_TM-mode0-LCD.dtb ./boot**

Create uEnv.txt file in directory **boot** and specify the dtb file, for example, use HDMI as the main display

- **$ sudo touch ./boot/uEnv.txt**
- **$ sudo bash -c "echo fdtfile=embest-SOM_AM572x_TM-mode0.dtb > ./boot/uEnv.txt"**

3.1.3 COPY ROOTFS

Unzip **tisdk-rootfs-image-am57xx-evm.tar.xz** to **$HOME/rootfs-arago**, and copy all the

contents of **tisdk-rootfs-image-am57xx-evm.tar.xz** to the **rootfs** directory. Note that you need to use **cp -ap** option to ensure that the file's attributes remain unchanged.

- **$ mkdir rootfs-arago**
- **$ tar -Jxvf tisdk-rootfs-image-am57xx-evm.tar.xz -C rootfs-arago/**
- **$ sudo cp -ap rootfs-arago/* rootfs**

3.1.4 INSTALL KERNEL MODULES

Installing kernel modules requires root privileges, so you also need to set environment variables for the root account.

- **$ cd $HOME/linux**
- **$ sudo make modules_install INSTALL_MOD_PATH=$HOME/rootfs ARCH=arm CROSS_COMPILE=$HOME/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-**

3.1.5 INSTALL EXTERNAL DRIVER MODULE

Refer to **2.3.3 INSTALLING EXTERNAL DRIVER**, set the DESTDIR variable to **$HOME/rootfs**, and then install it.

If the user has not upgraded the kernel version, the **rootfs-arago.tar.gz** root file system we provide already contains the kernel module and external driver module, so there is no need to reinstall it.

After the copy operation completes, unmount the two partitions and synchronize the file system:

- **$ sudo umount boot rootfs**

- **$ sudo kpartx -d /dev/loop0**
- **$ sudo losetup -d /dev/loop0**
- **$ sync**

3.2 BURN AND READ IMAGE

3.2.1 BURN IMAGE

Please refer to the user manual.

3.2.2 READ IMAGE

During the development process, it is often necessary to read the image from the SD card for backup. You can use the following command to obtain the image file from the SD card.

- Under Linux OS

Connect the SD card into the card reader and connect it to the computer:

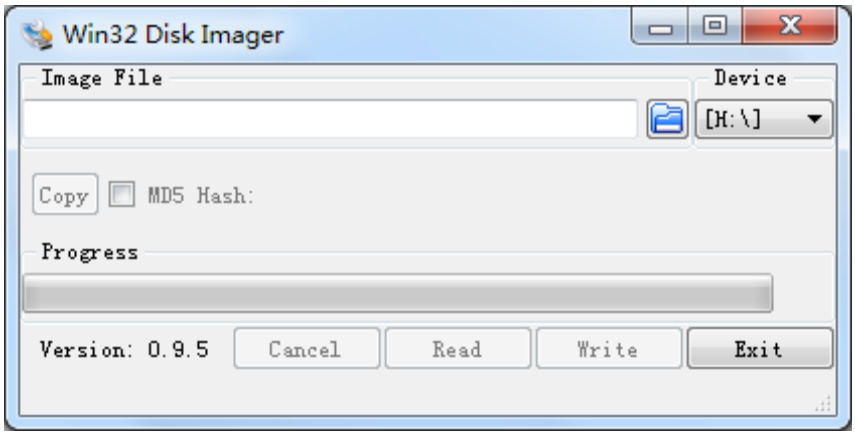
- **$ lsblk**

```
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdb 8:16 1 7.2G 0 disk
├─sdb2 8:18 1 3.8G 0 part
└─sdb1 8:17 1 64M 0 part
```

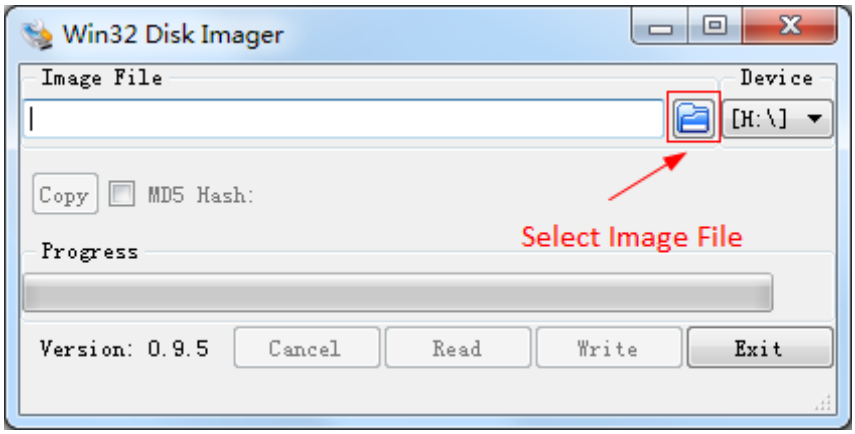
- **$ dd if=/dev/sdb of=./sdcard.img**

- Under Windows OS

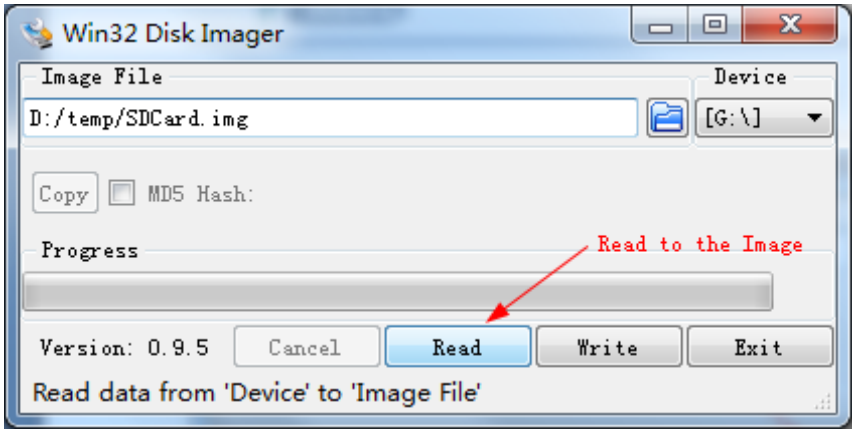
Connect the SD card to the computer and run Win32 Disk Imager



Select the storage path of the target image file, such as: D:/temp/SDCard.img



Click **Read** button to read the contents of the SD card into the image file:



After successful complete, you will get a complete SD image.

4. TI SDK DEVELOPMENT

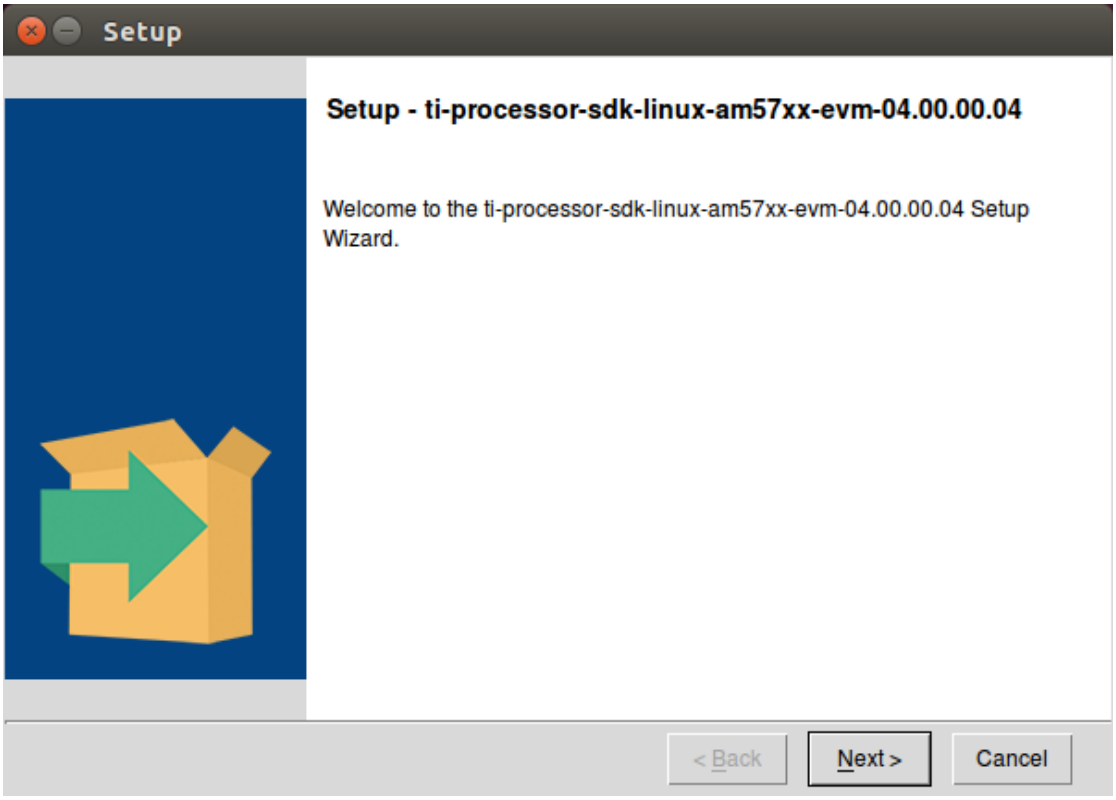
4.1 SDK INSTALLATION AND CONFIGURATION

The TI SDK installation package can be installed on the ssh command line or in the ubuntu desktop environment. Let's take the desktop environment installation as an example:

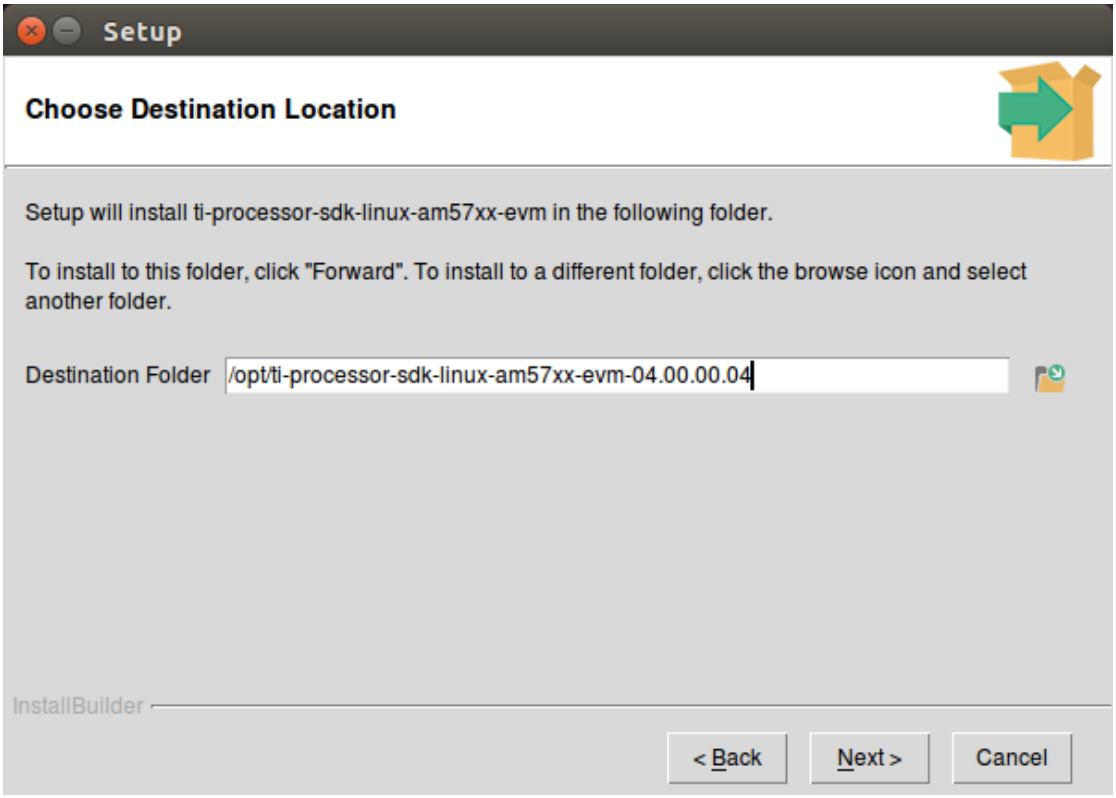
Press the key combination Ctrl+Alt+T on the Ubuntu desktop to open the console

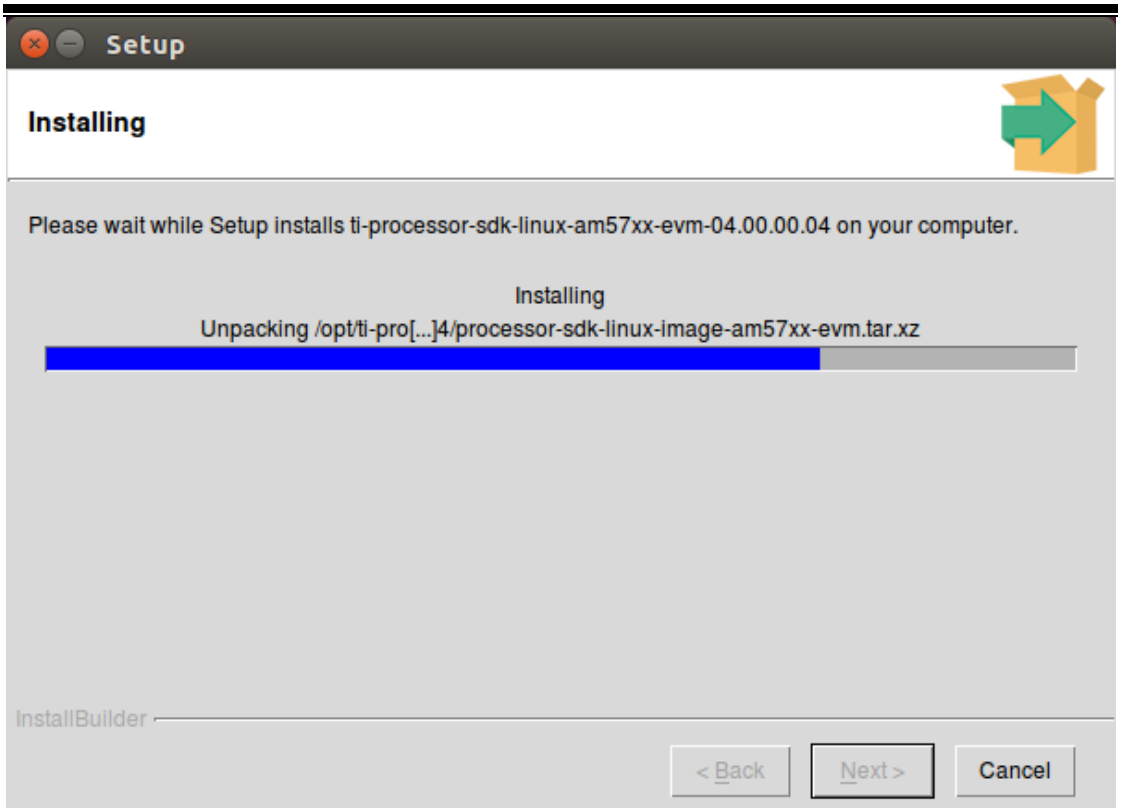
- **$ cd $HOME**
- **$ sudo chmod +x ti-processor-sdk-linux-am57xx-evm-04.00.00.04-Linux-x86-Install.bin**
- **$ sudo ./ti-processor-sdk-linux-am57xx-evm-04.00.00.04-Linux-x86-Install.bin**

Pop up a dialog:

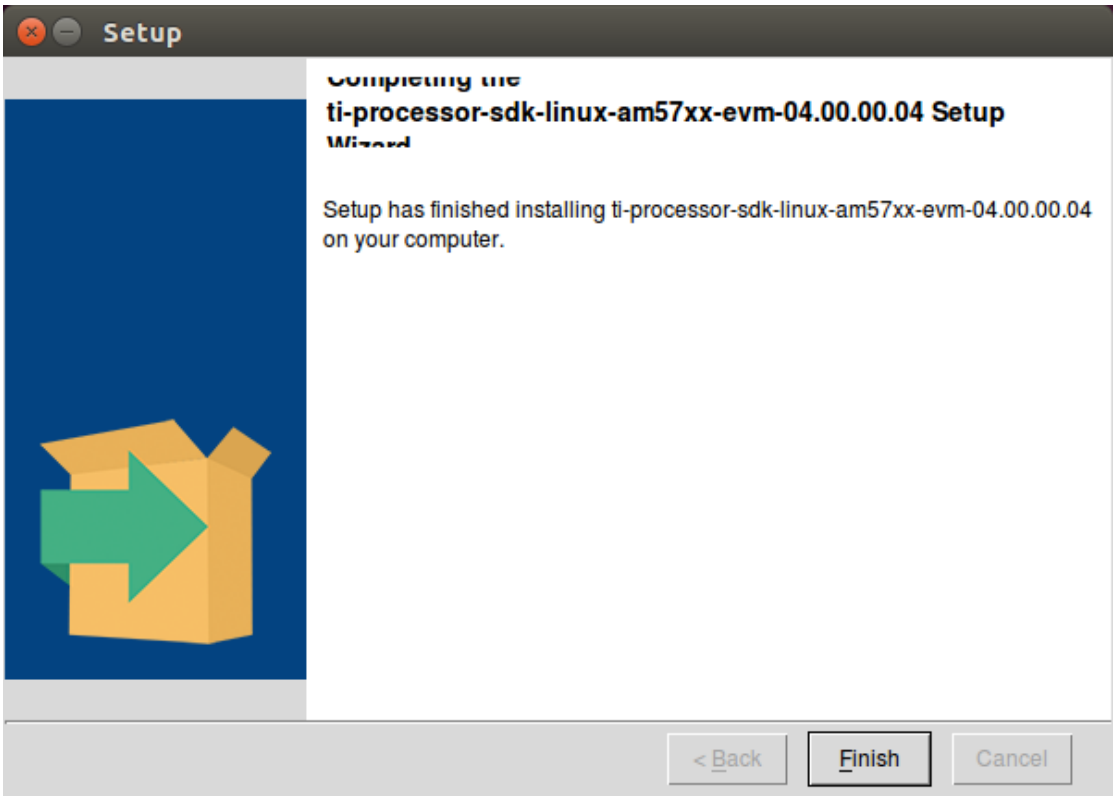


According to the guidance, click Next. When the installation path appears, you can use the default path for installation, or you can customize the installation path.





After about a few minutes, the installation is complete, click "Finish".



TI's SDK directory is organized as follows:

- **$ cd /opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/**
- **$ ls -l**

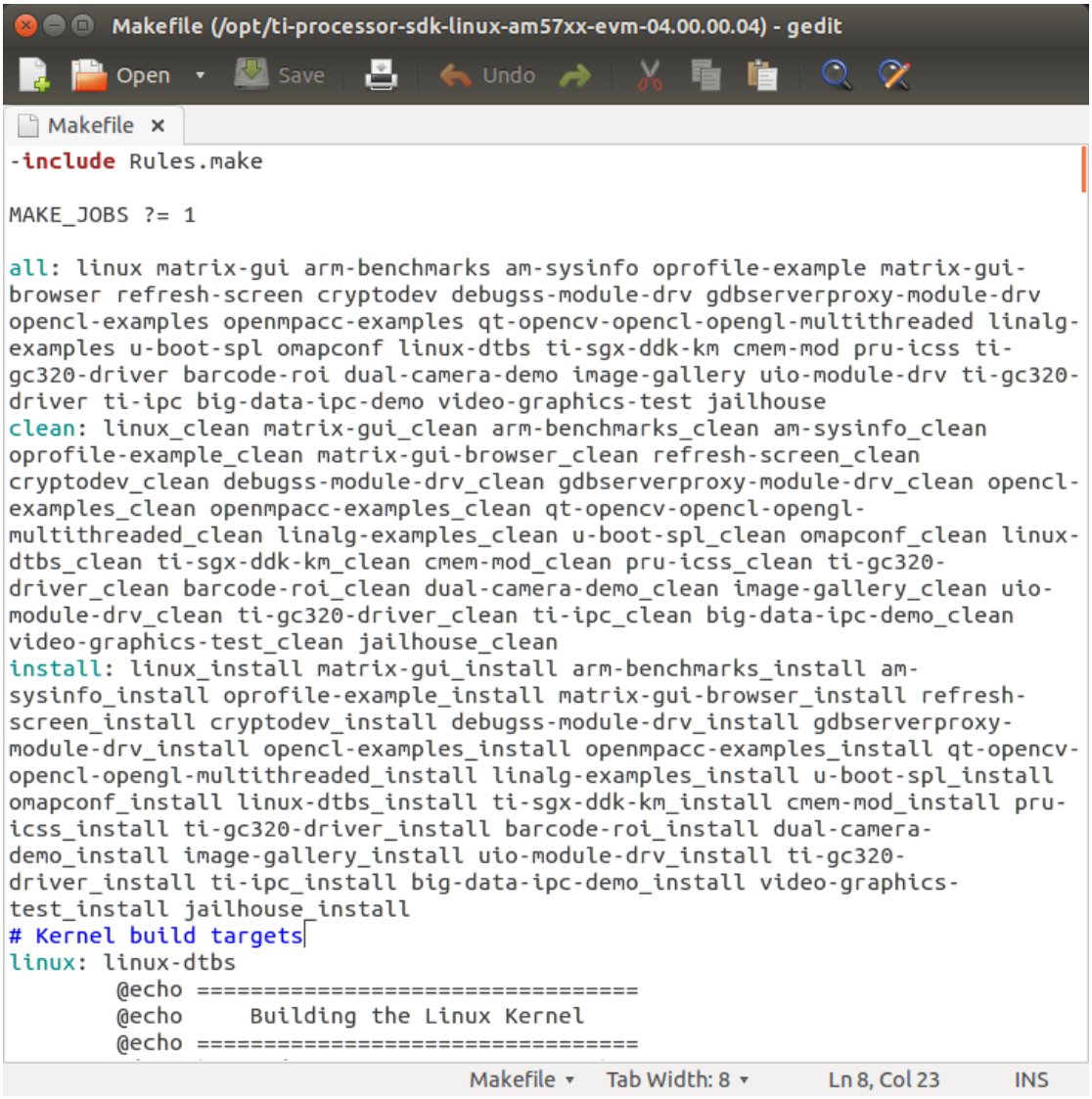
```
total 542220
drwxr-xr-x 2smbusersmbuser 4096 Aug 10 16:18 bin
drwxr-xr-x 6smbusersmbuser 4096 Jun 29 2017 board-support
drwxr-xr-x 3smbusersmbuser 4096 Jun 29 2017 docs
drwxr-xr-x 19 smbusersmbuser 4096 Jun 29 2017 example-applications
drwxr-xr-x 2smbusersmbuser 4096 Jun 29 2017 filesystem
drwxr-xr-x 3 root root 4096 Aug 10 16:18 linux-devkit
-rwxr-xr-x 1 smbusersmbuser 555147047 Jun 29 2017 linux-devkit.sh
-rwxr-xr-x 1 smbusersmbuser 44597 Jun 29 2017 Makefile
-rwxr-xr-x 1 smbusersmbuser 1324 Aug 10 16:18 Rules.make
-rwxr-xr-x 1 smbusersmbuser 4188 Jun 29 2017 setup.sh
```

The uses of each directory/file are as follows:

bin	Some tool scripts for making SD cards, setting up TFTP, etc.
board-support	1. u-boot source code; 2. Kernel source code; 3. External driver source code; 4. Pre-compiled firmware for TI's evaluation board
docs	SDK software list and license
example-applications	Demo program source code
filesystem	Pre-built root file system based on Arago
linux-devkit	The rootfs system required to compile the entire SDK, which includes the cross-compilation tool chain
linux-devkit.sh	The compressed package of the linux-devkit folder. This file is needed after moving the installed SDK path.
Makefile	The top-level Makefile can compile the entire SDK, including: 1. Linux Kernel 2. U-boot 3. External driver 4. Demo program
Rules.make	Some environment variables needed to compile the SDK, such as cross-editing tool chain and target root file system path
setup.sh	Setting up the development environment actually calls some tool scripts in the bin directory.

4.2 TOP-LEVEL MAKEFILE USAGE

The top-level Makefile in the SDK contains many targets. Through this Makefile, you can compile all, clean, and install them all, or you can specify a target individually for compilation, cleanup, and installation.



4.2.1 COMPILE THE ENTIRE SDK

The default target "all" of the top-level Makefile can compile the entire SDK, so only one

command is needed:

- **$ sudo make**

The compilation process starts from the first target dtbs and Linux. It takes some time. If you need to install the compiled product to the target file system after compilation is completed, you need to modify **DESTDIR** in **Rule.make** to the directory where the actual rootfs system is located.

- **$ cd /opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04**
- **$ sudo make install**
- **$ cd ../fakeroot**
- **$ ls -l**

```
total 12
drwxr-xr-x 2 root root 4096 Aug 10 17:39 boot
drwxr-xr-x 4 root root 4096 Aug 10 17:39 lib
drwxr-xr-x 4 root root 4096 Aug 10 17:39 usr
```

4.2.2 COMPILE AND INSTALL A TARGET SEPARATELY

Take compiling the dual-camera demo program as an example:

- **$ cd /opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04**
- **$ sudo make dual-camera-demo**

```
Makefile:715: warning: overriding commands for target `ti-gc320-driver'
Makefile:595: warning: ignoring old commands for target `ti-gc320-driver'
Makefile:723: warning: overriding commands for target `ti-gc320-driver_clean'
Makefile:603: warning: ignoring old commands for target `ti-gc320-driver_clean'
Makefile:731: warning: overriding commands for target `ti-gc320-driver_install'
Makefile:611: warning: ignoring old commands for target `ti-gc320-driver_install'
================================
Building Dual Camera Demo
================================
make[1]: Entering directory
`/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/example-applications/dual-camera-d
```

```
emo-1.0'
echo "manisha" am57xx-evm
manisha am57xx-evm
make[2]: Entering directory
`/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/example-applications/dual-camera-d
emo-1.0'
make[2]: Nothing to be done for `first'.
make[2]: Leaving directory
`/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/example-applications/dual-camera-d
emo-1.0'
make[1]: Leaving directory
`/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/example-applications/dual-camera-d
emo-1.0'
```

Install

- **$ sudo make dual-camera-demo_install**

```
Makefile:715: warning: overriding commands for target `ti-gc320-driver'
Makefile:595: warning: ignoring old commands for target `ti-gc320-driver'
Makefile:723: warning: overriding commands for target `ti-gc320-driver_clean'
Makefile:603: warning: ignoring old commands for target `ti-gc320-driver_clean'
Makefile:731: warning: overriding commands for target `ti-gc320-driver_install'
Makefile:611: warning: ignoring old commands for target `ti-gc320-driver_install'
==================================================
Installing Dual Camera Demo - Release version
==================================================
make[1]: Entering directory
`/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/example-applications/dual-camera-d
emo-1.0'
echo "manisha" am57xx-evm
manisha am57xx-evm
make[2]: Entering directory
`/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/example-applications/dual-camera-d
emo-1.0'
make[2]: Nothing to be done for `first'.
make[2]: Leaving directory
`/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/example-applications/dual-camera-d
emo-1.0'
dual_camera release version installed.
```

```
make[1]: Leaving directory
`/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/example-applications/dual-camera-demo-1.0'
```

5. APPLICATION DEVELOPMENT

5.1 CROSS COMPILE AND RUN ORDINARY C PROGRAMS

5.1.1 WRITE C PROGRAM CODE

Let's get a 4G communication program **application.tar.gz** as example, unzip it to $HOME directory.

5.1.2 COMPILE ON HOST

- **$ cd $HOME/application/4g_test**
- **$ $HOME/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc 4G_test.c -o 4G_test**
- **$ ls**

```
4G_test 4G_test.c readme.md
```

- **$ file 4G_test**

```
4G_test: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=69a5b54a2de0c56b075f871fff6710797250a72c, not stripped
```

5.1.3 COMPILE ON ARM BOARD

The compilation toolchain is already installed in the rootfs system of our release, so the C source code files can be copied to compile the application directly on the board using gcc

- root@arm:~# **gcc 4G_test.c -o 4G_test**

- root@arm:~# **ls**

```
4G_test  4G_test.c  readme.md
```

5.1.4 TRANSMIT TO ARM BOARD AND RUN

- Connect the ARM board and computer to the local network and use the scp command to transfer files.
  - root@arm:~# **scp <UbuntuUser>@<UbuntuIPAddr>:/home/david/application_test_programs/4g_test/4g_test ./**
- Copy using storage media such as USB flash drive.

.....

5.2 QT APPLICATION DEVELOPMENT

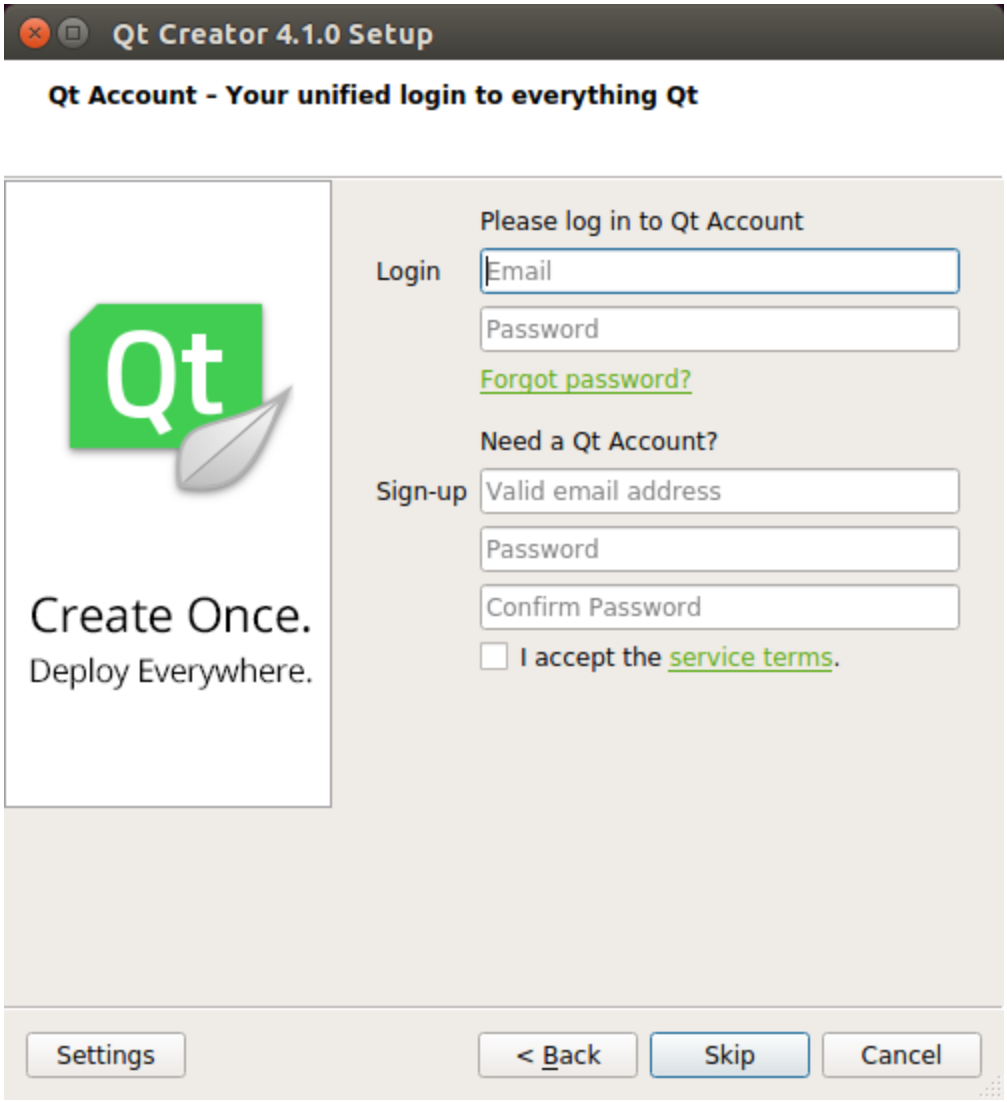
5.2.1 INSTALL QT CREATOR

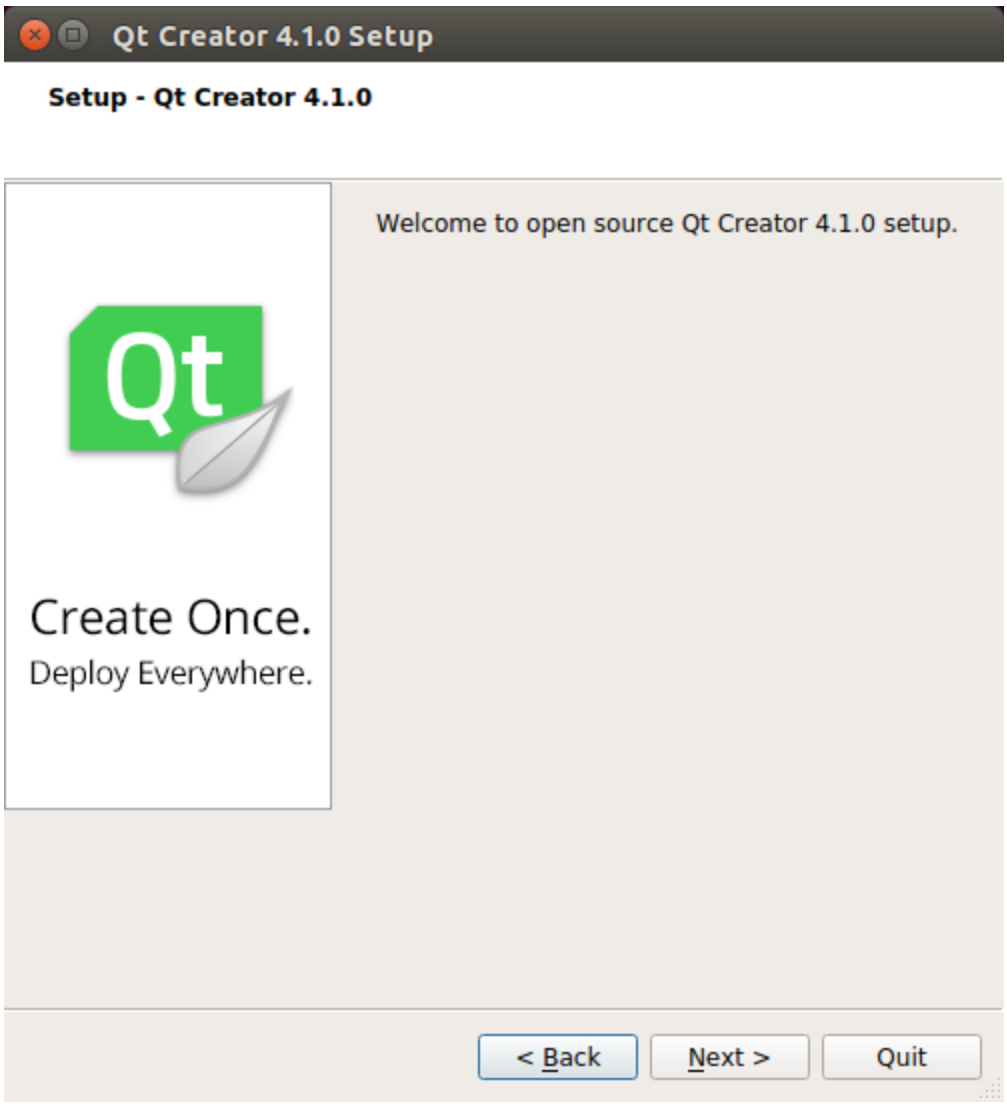
Qt Creator is a graphical designer, so this section of the operation is performed in the Ubuntu desktop environment. Copy **qt-creator-opensource-linux-x86_64-4.1.0.run** to **$HOME** and add executable permissions.

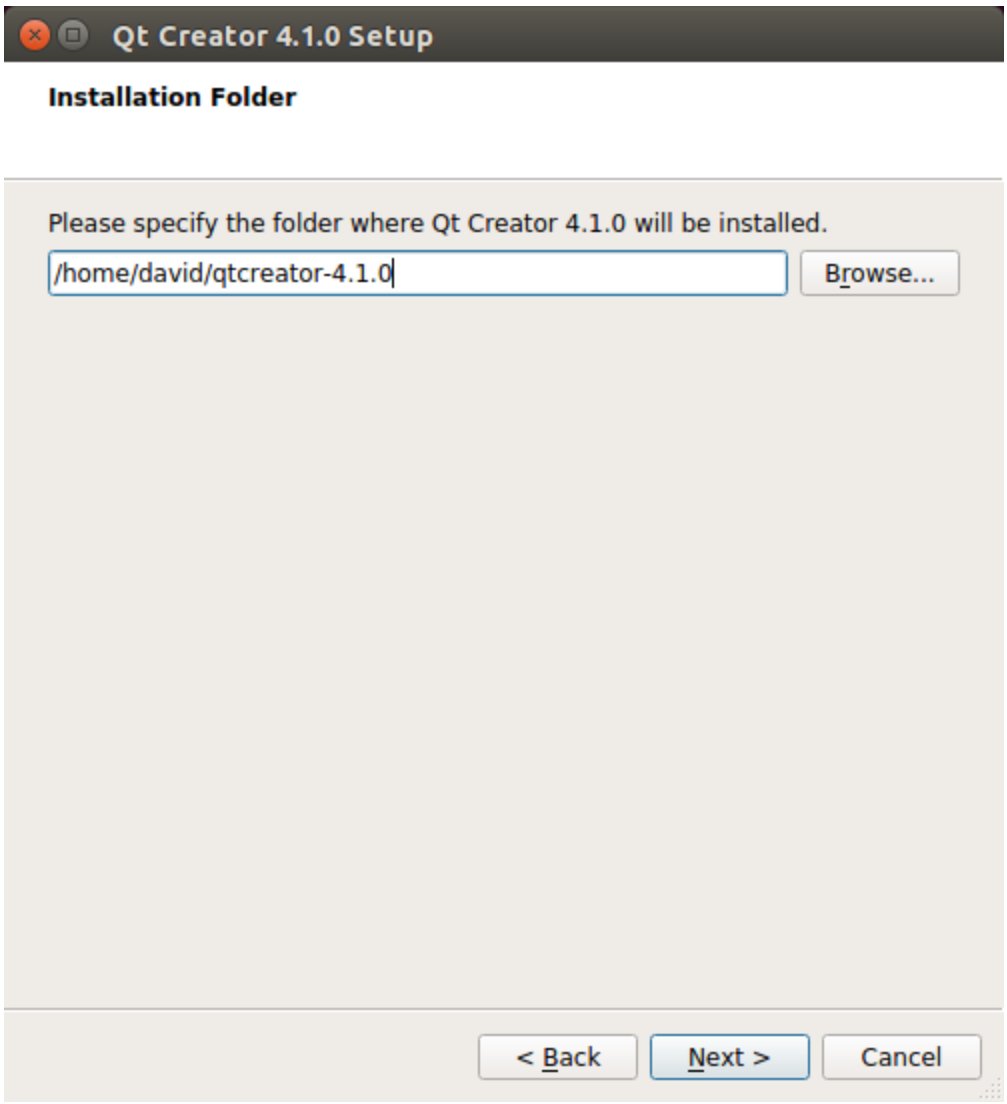
- **$ sudo chmod +x qt-creator-opensource-linux-x86_64-4.1.0.run**

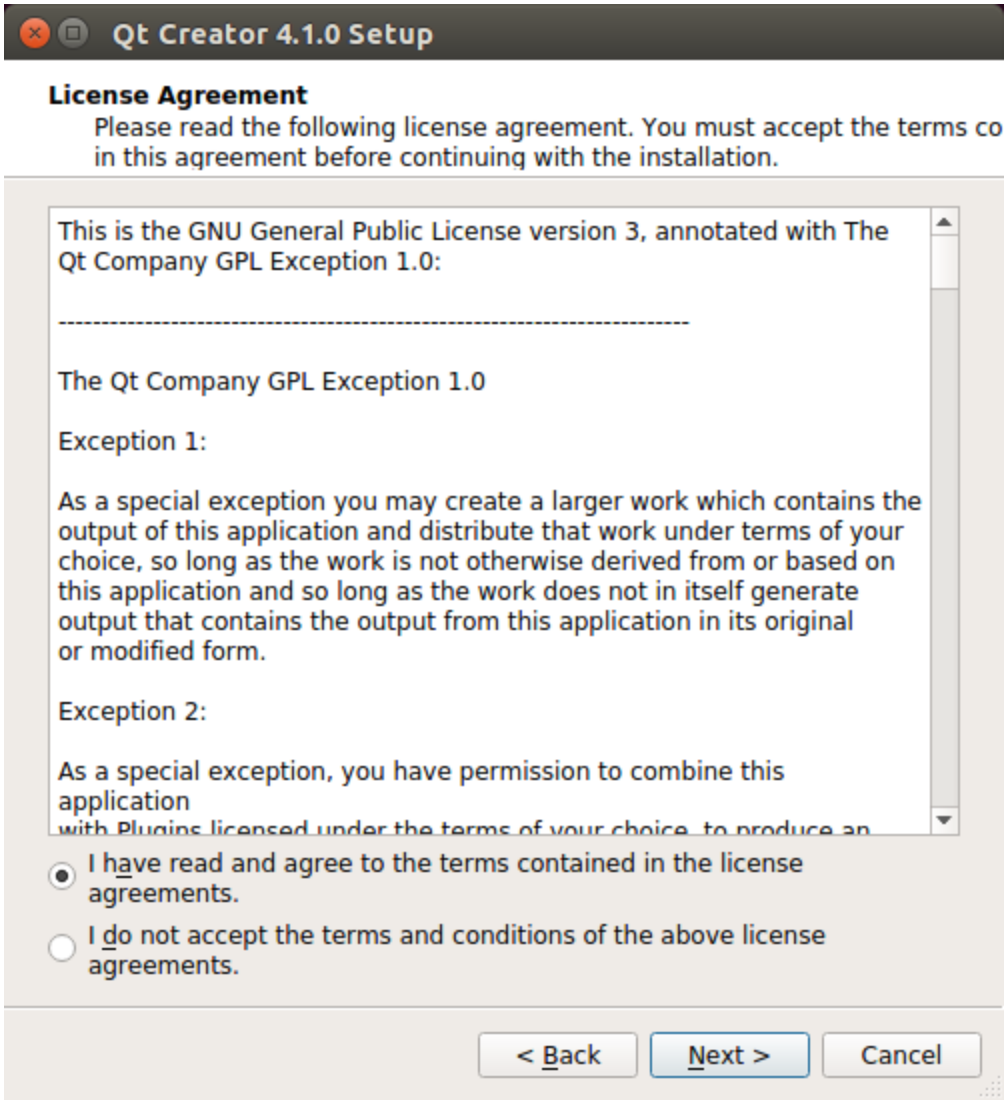
Start to install:

- **$ ./qt-creator-opensource-linux-x86_64-4.1.0.run**

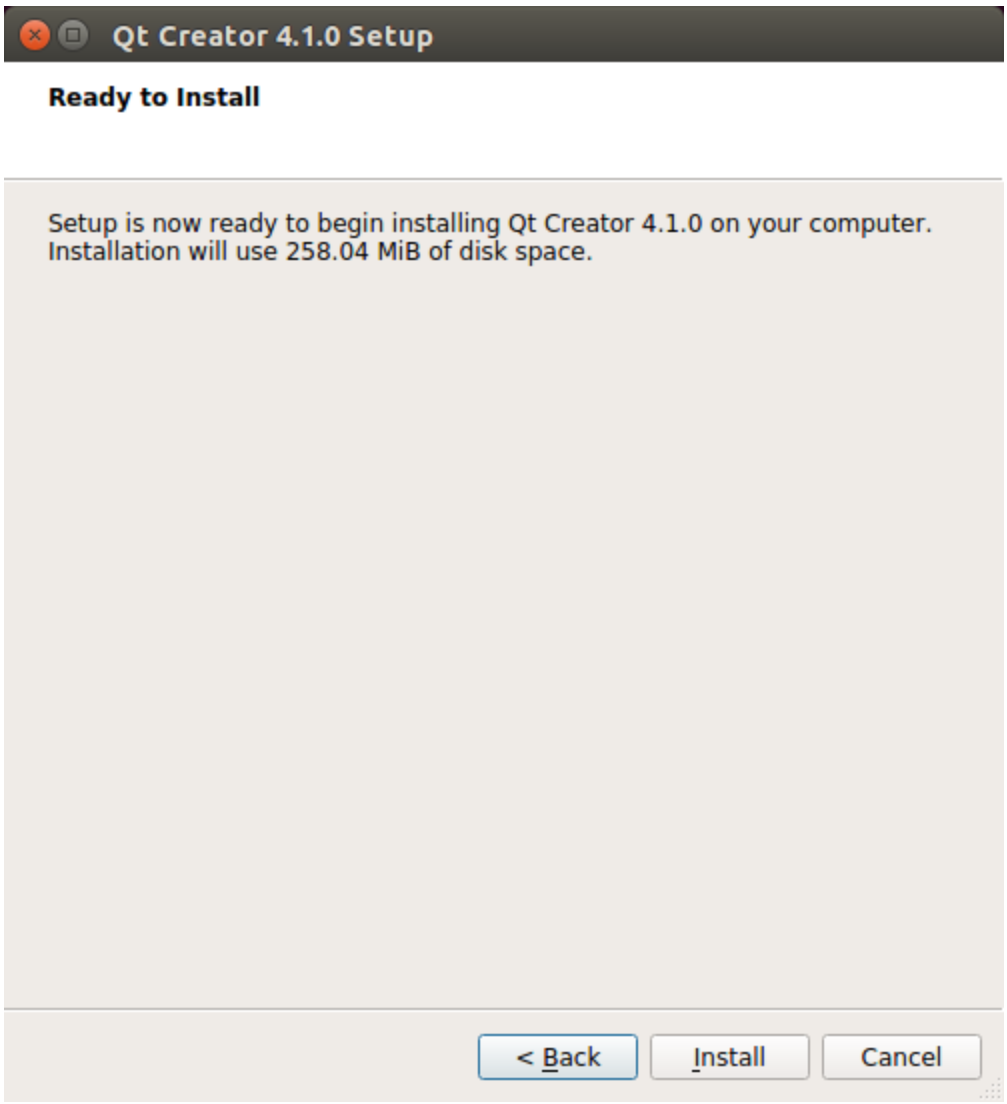

Qt Creator 4.1.0 Setup
Qt Account - Your unified login to everything Qt
Create Once.
Deploy Everywhere.
Please log in to Qt Account
Login
Email
Password
Forgot password?
Need a Qt Account?
Sign-up
Valid email address
Password
Confirm Password
I accept the service terms.
Settings
< Back
Skip
Cancel


Qt Creator 4.1.0 Setup
Setup - Qt Creator 4.1.0
Welcome to open source Qt Creator 4.1.0 setup.
Create Once.
Deploy Everywhere.
< Back
Next >
Quit


Qt Creator 4.1.0 Setup
Installation Folder
Please specify the folder where Qt Creator 4.1.0 will be installed.
/home/david/qtcreator-4.1.0
Browse...
< Back
Next >
Cancel

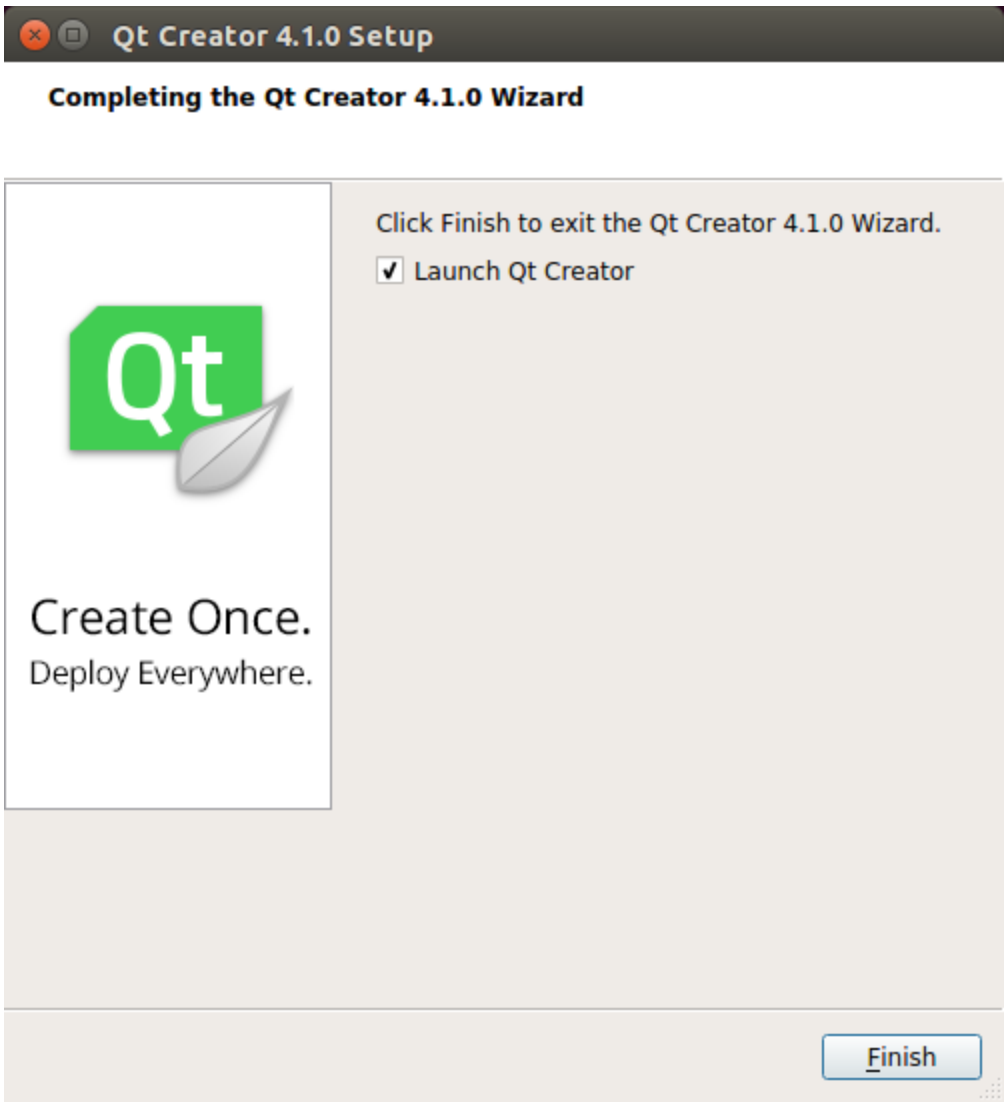

Qt Creator 4.1.0 Setup
License Agreement
Please read the following license agreement. You must accept the terms co
in this agreement before continuing with the installation.
This is the GNU General Public License version 3, annotated with The
Qt Company GPL Exception 1.0:

The Qt Company GPL Exception 1.0
Exception 1:
As a special exception you may create a larger work which contains the
output of this application and distribute that work under terms of your
choice, so long as the work is not otherwise derived from or based on
this application and so long as the work does not in itself generate
output that contains the output from this application in its original
or modified form.
Exception 2:
As a special exception, you have permission to combine this
application
I have read and agree to the terms contained in the license
agreements.
I do not accept the terms and conditions of the above license
agreements.
< Back
Next >
Cancel



According to the guidance, install Qt creator to **$HOME/qtcreator-4.1.0**.

After the installation is completed, the following picture appears:



After the last step, please do not start Qt Creator right now. You need to start it from the TI SDK because there are some necessary environment variables that need to be set.

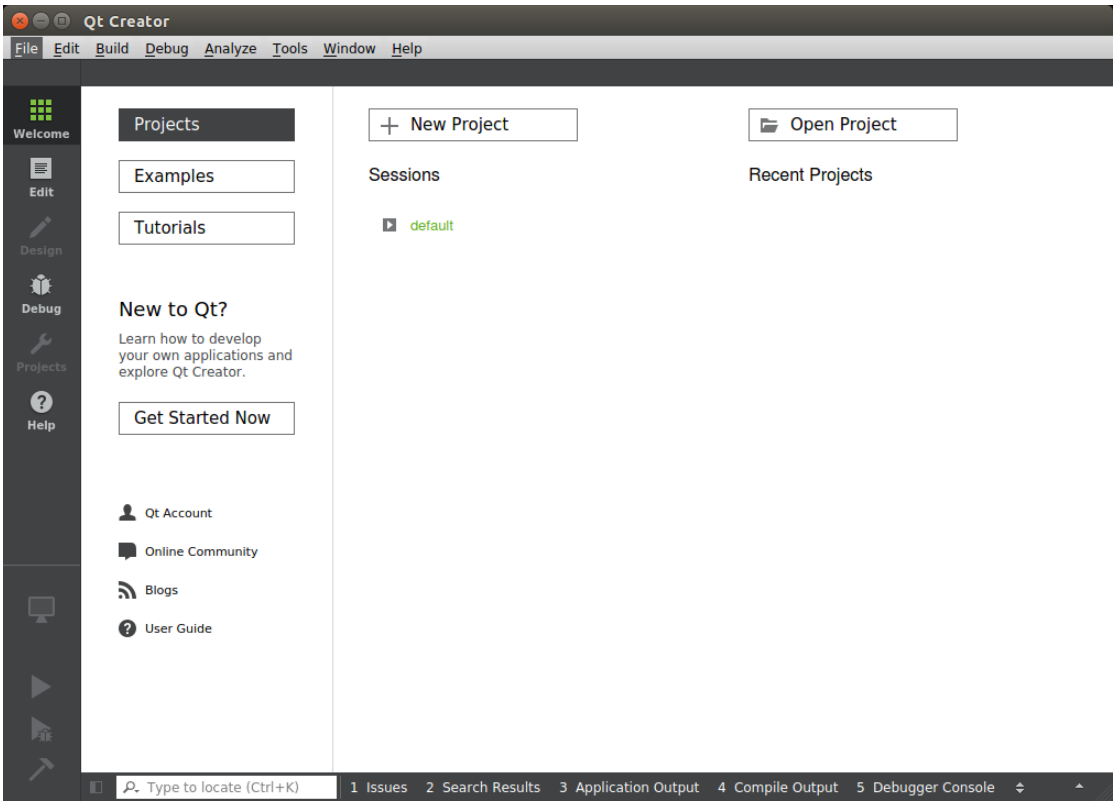
- **$ source /opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/linux-devkit/environment-setup**

Then switch to the Qt Creator installation directory./qtcreator starts

- **[linux-devkit]:~> cd ~/qtcreator-4.1.0/bin/**
- **[linux-devkit]:~/qtcreator-4.1.0/bin> ./qtcreator**

5.2.2 CONFIGURE QT CREATOR

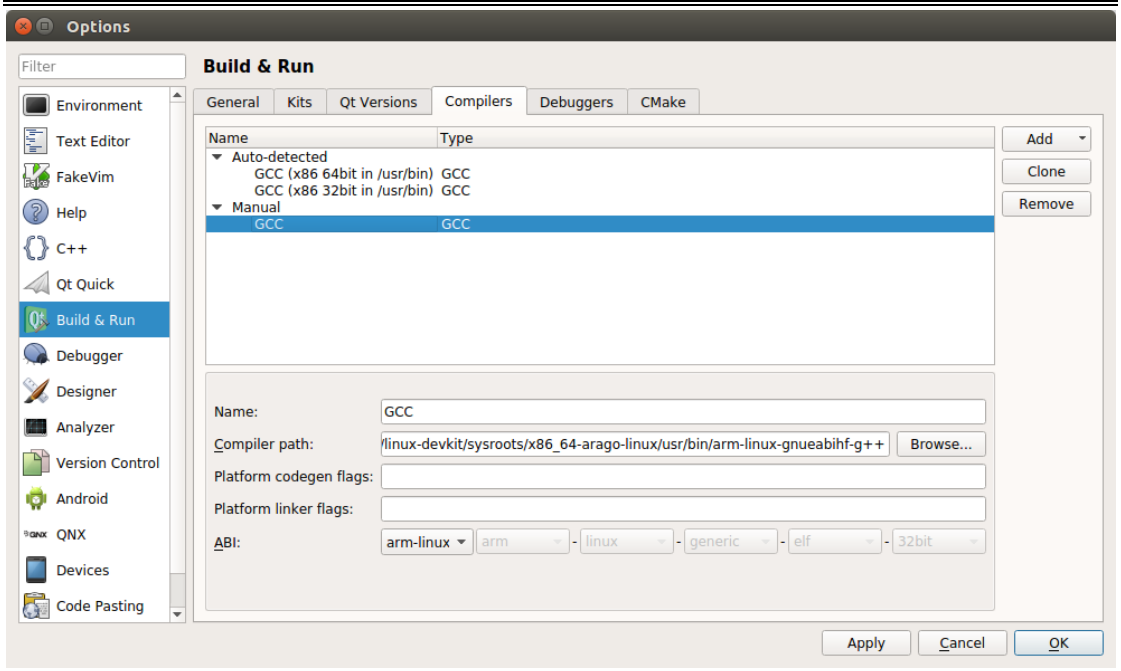
Qt Creator starts as shown below:



Before using Qt Creator to develop a program that runs on AM5728, you need to configure the cross-compilation tool, QT version, Debuggers version, and Kits.

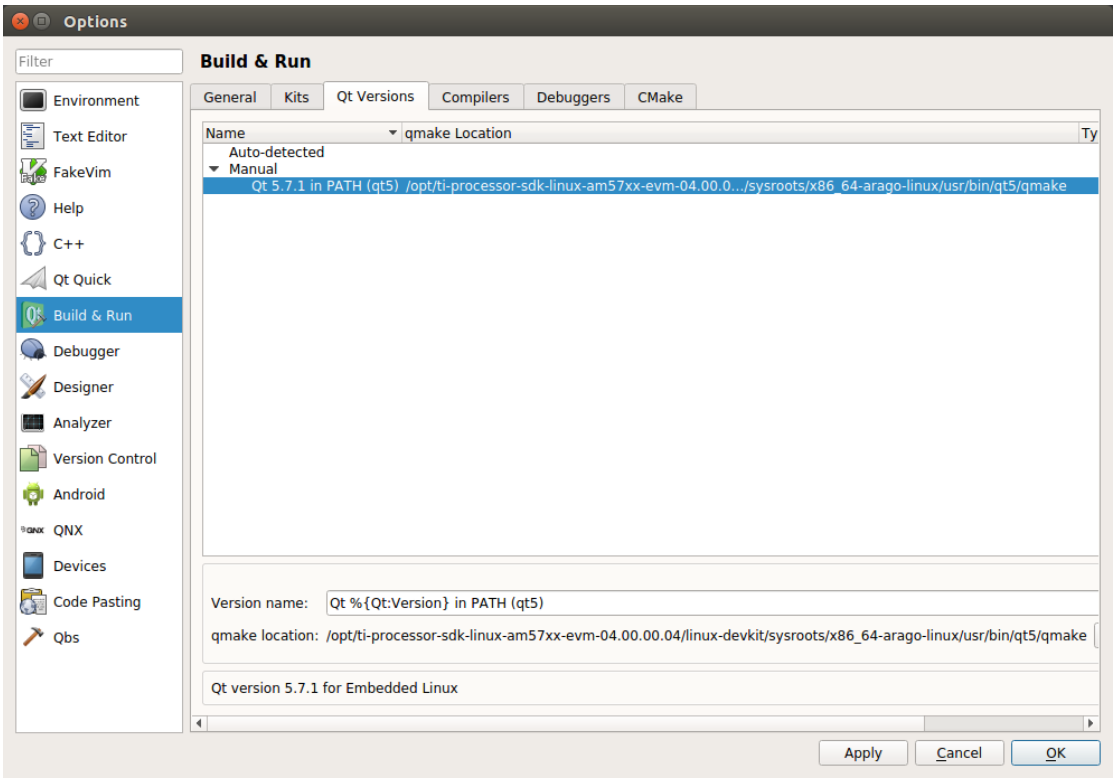
- Configure the cross-compilation tool chain

Click the menu bar Tool->Options->Build&Run->Compilers->Add->GCC, click Browse to select the Compiler path as **/opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/linux-devkit/sysroots/x86_64-arago-linux/usr/bin/arm-linux-gnueabihf-g++** (gcc path of TI SDK), then click Open and click Apply.



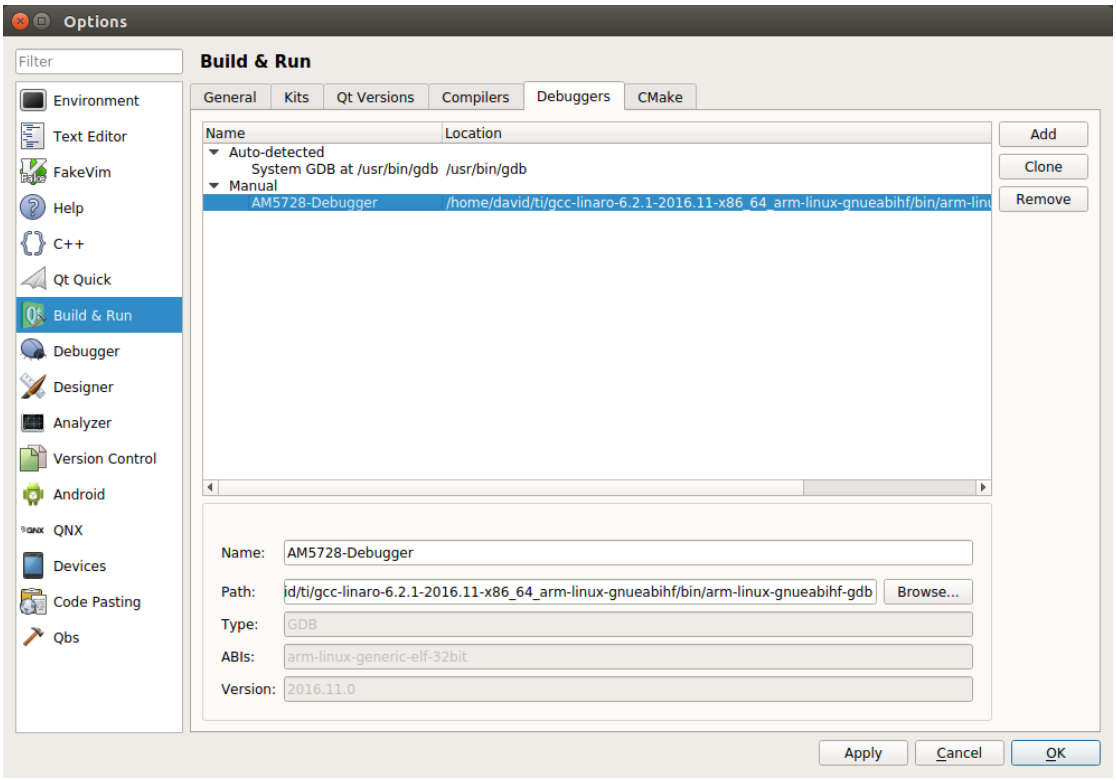
- Configure Qt version

Click "Qt Versions->Add" and select the qmake file path of TI SDK: /opt/ti-processor-sdk-linux-am57xx-evm-04.00.00.04/linux-devkit/sysroots/x86_64-arago-linux/usr/bin/qt5/qmake, then click Apply button.



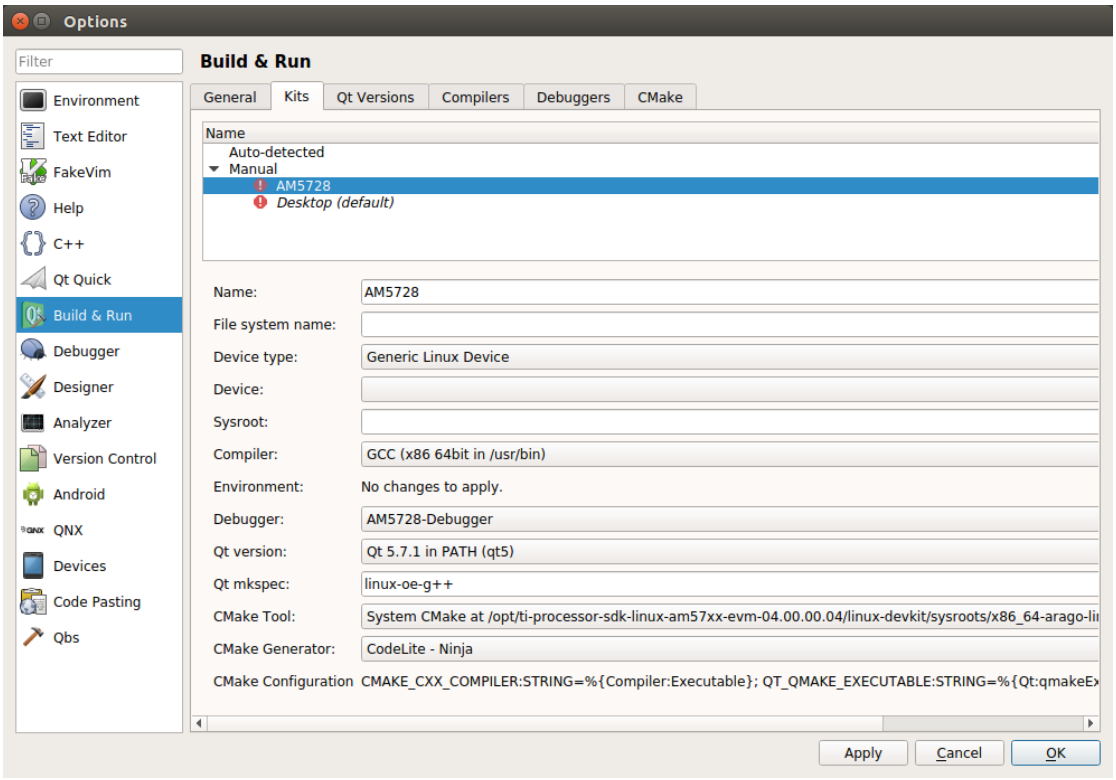
- Configure Debuggers version

Click the Debuggers option, click Add, click Browse, and select the GDB compiler in the cross-compiler installation directory, such as /home/david/ti/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gdb. Change the Name option, enter AM5728-Debugger, and click Apply to complete the setting.



- Configure Kits

Click "Tool->Options->Build & Run->Kits->Add" in the menu bar, change Name to AM5728, Device type to Generic Linux Device, enter linux-oe-g++ in the Qt mkspec option, and click Apply after configuration, click OK

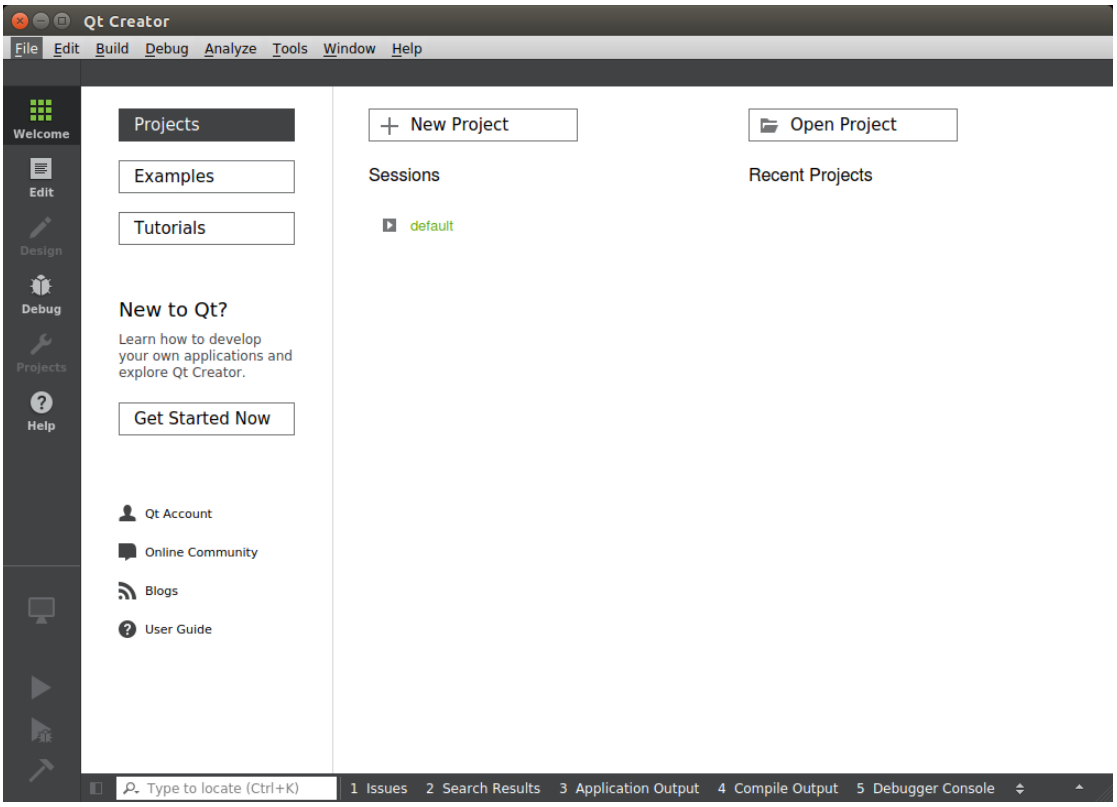


5.2.3 CREATE DEMO

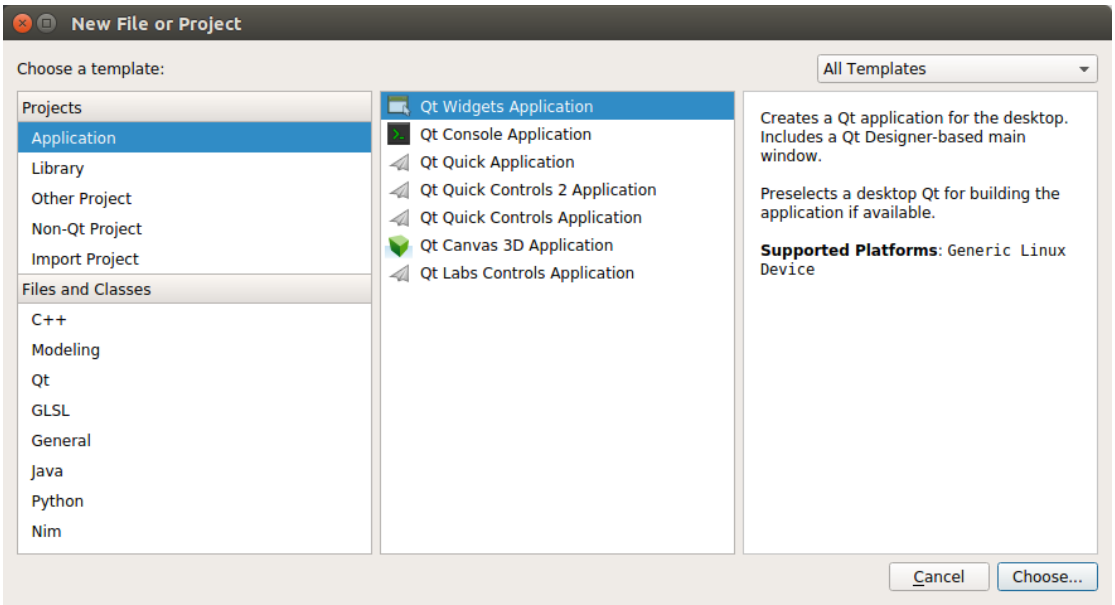
In this chapter, let's create a simple widget program hello-world to introduce the development process of Qt interface.

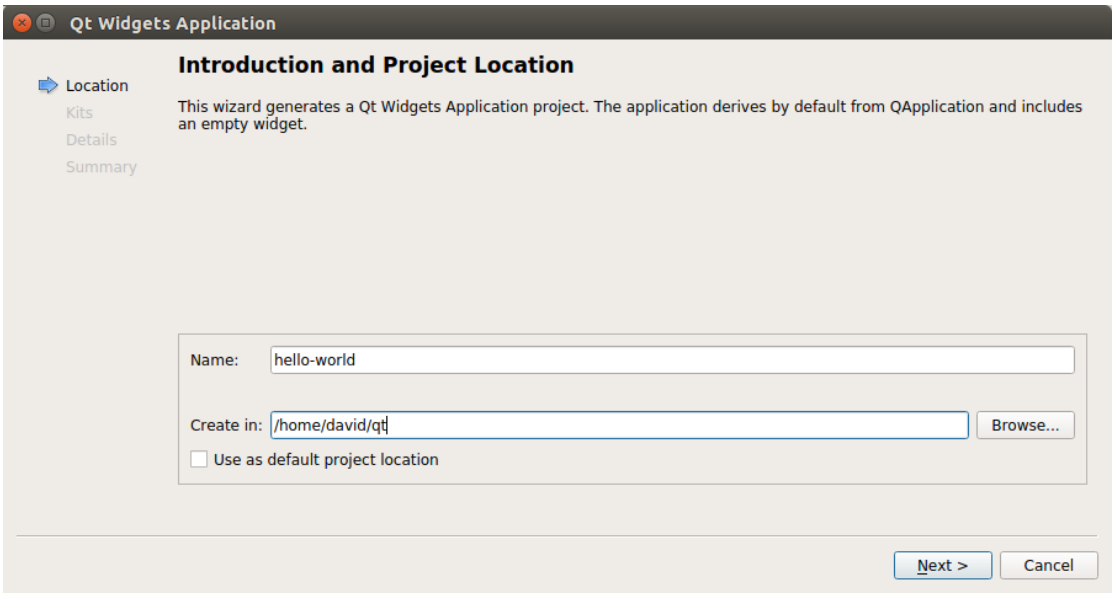
- Create a new project

Similarly, start Qt Creator from the SDK and store the source code in **/home/david/ti**.

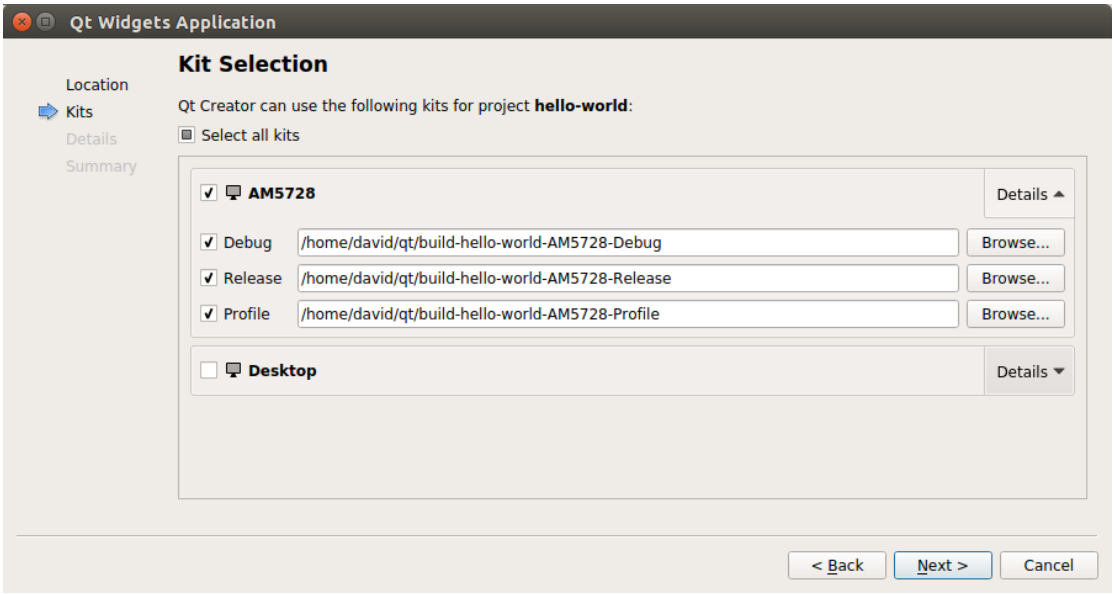


Click the File -> New File or Project menu in the top left corner

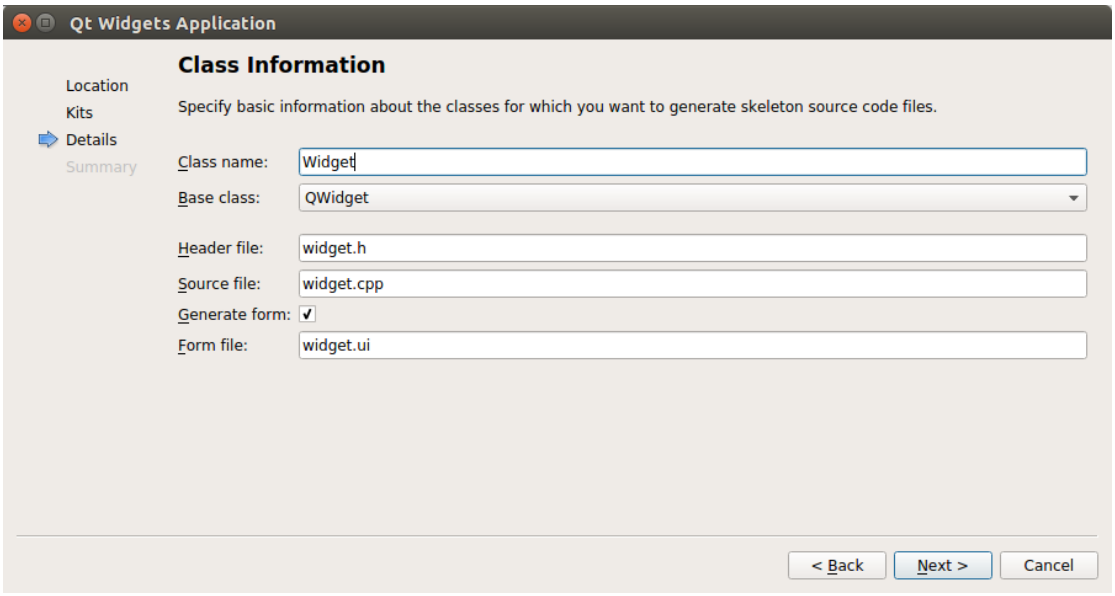




Select Kit: AM5728

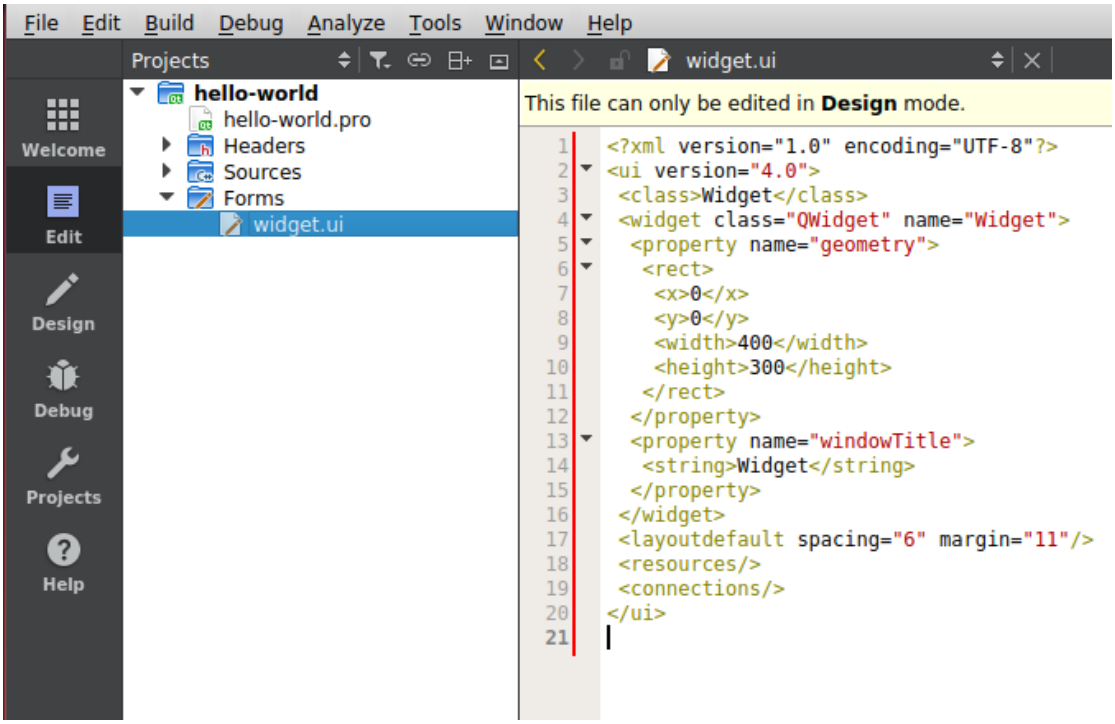


Select Base Class: Qwidget

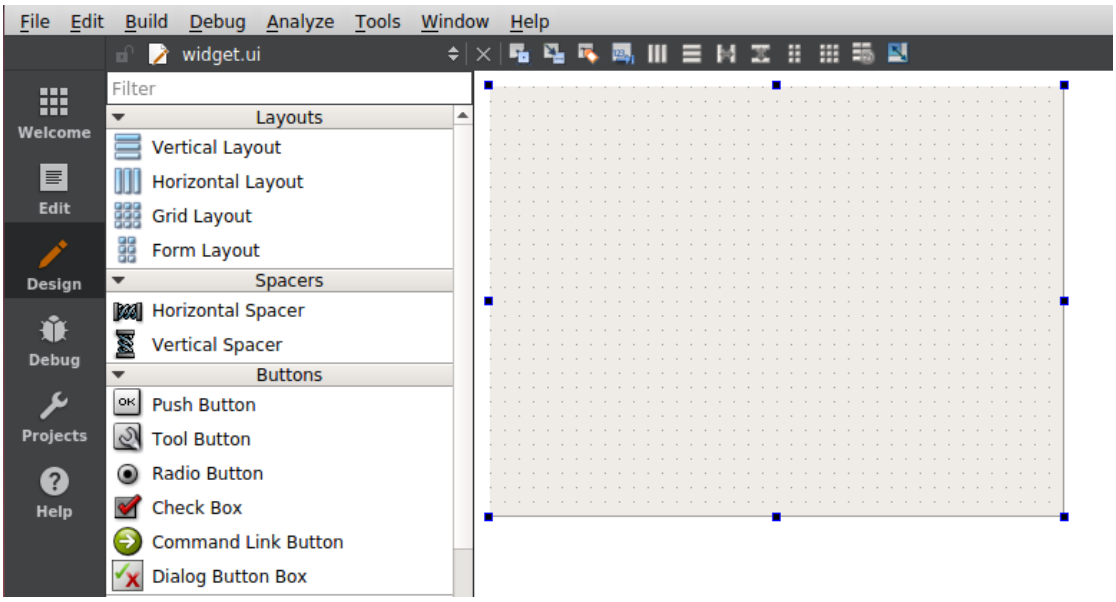


Click Finish to enter the new project

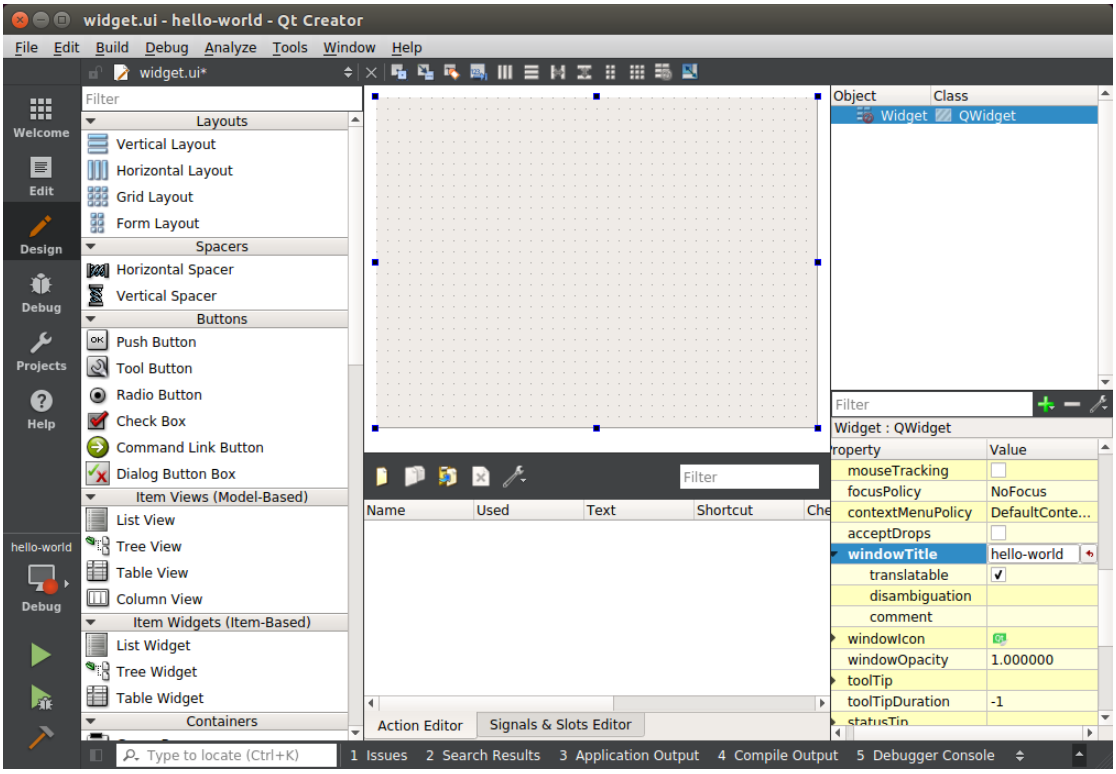
- Design UI interface



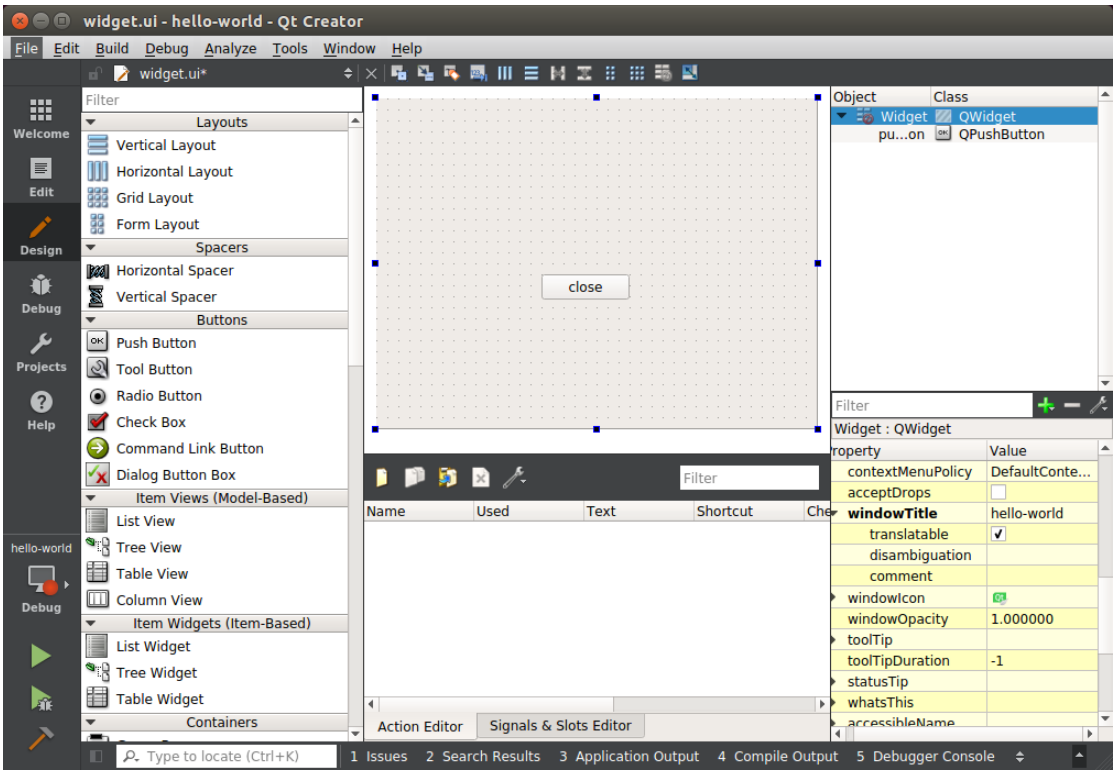
Double-click widget.ui to enter the UI designer



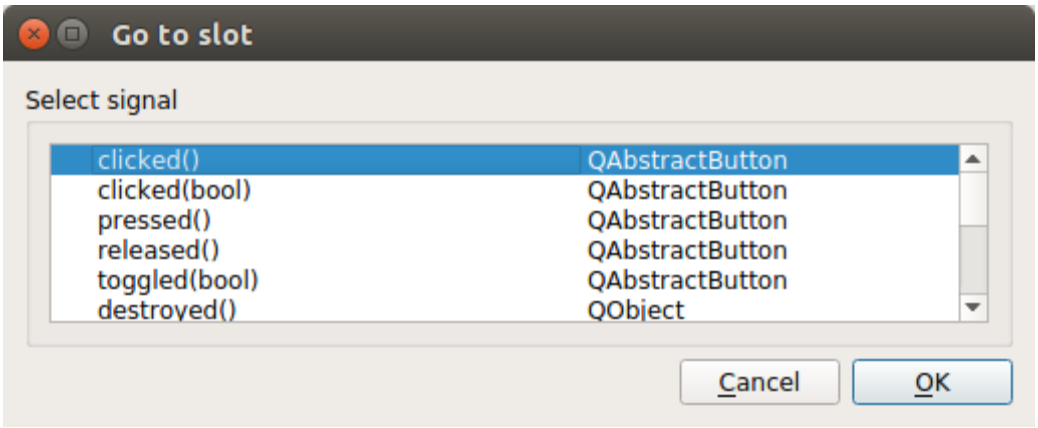
In the Property window in the lower right corner, modify the windowTitle field and enter hello-world.



Drag a Push Button to the UI in the Buttons window on the left, double-click and change the name to "close".



Select the close button, right-click and select Go to slot to enter code editing.



- Edit code

Edit the response code of the on_pushButton_clicked function. Enter close() here. The function is to close the window.

- Compile

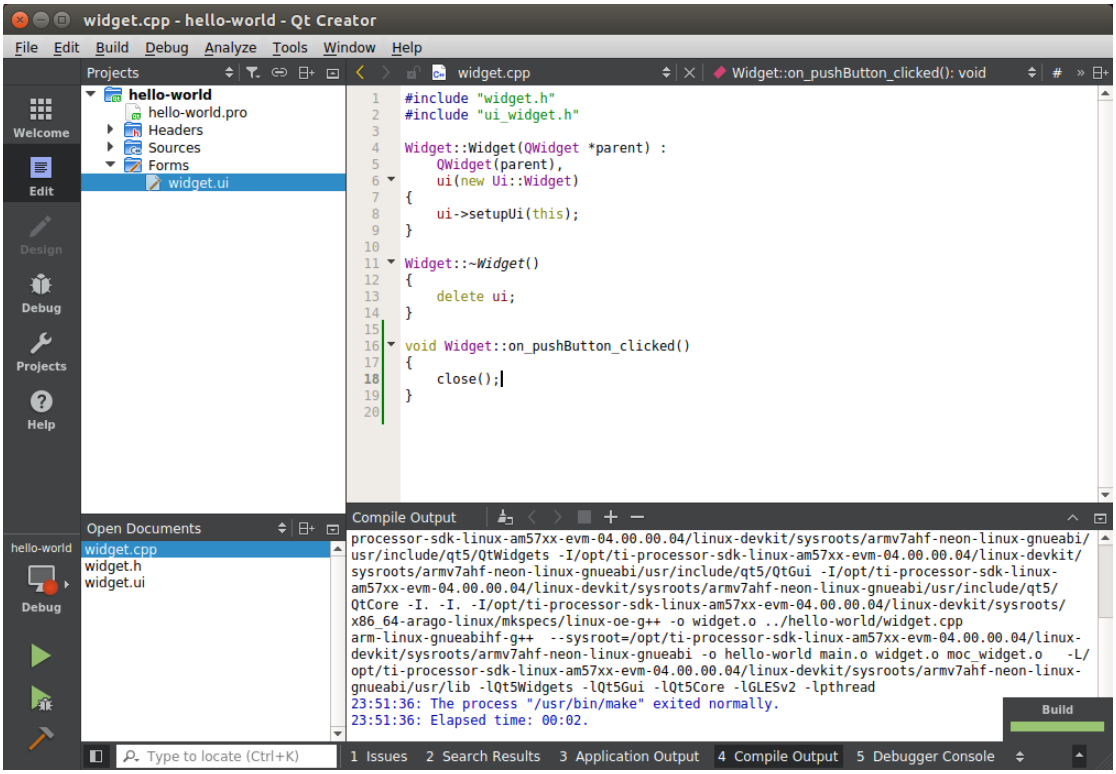
Click the hammer-shaped shortcut button in the lower left corner to compile the pro

ject and produce the executable file in directory **$HOME/qt/build-hello-world-AM5728-Debug**.

Check the ELF with file command:

- **$ file hello-world**

```
hello-world: ELF 32-bit LSB executable, ARM, EABI5 version 1 (GNU/Linux), dynamically linked (uses shared libs), for GNU/Linux 2.6.32, BuildID[sha1]=cd9018247cd88be33eb2f59fb56fe7af7fee37ea, not stripped
```

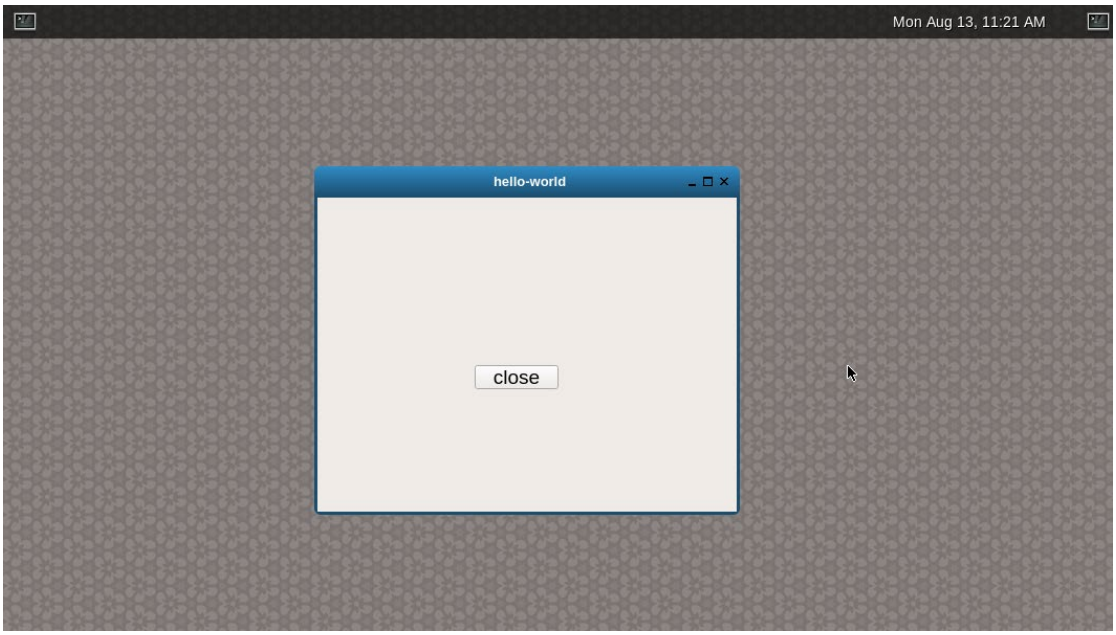


5.2.4 RUNNING ON ARM BOARD

- **Run under weston desktop**

Copy the generated executable file hello-world to the AM5728 board and run ./hello-world.

The display will show the following window:



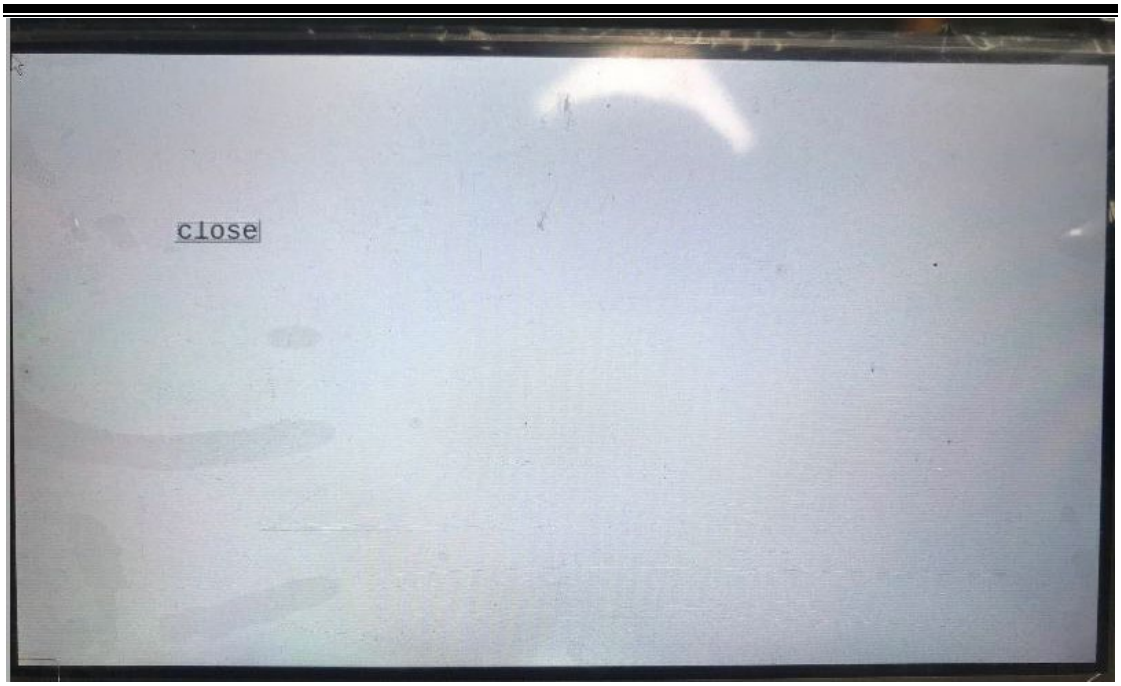
Click the close button with the mouse (or click on the touch screen) to close the window.

- **Run without weston desktop**

Qt programs can also be run without weston desktop. Close weston first.

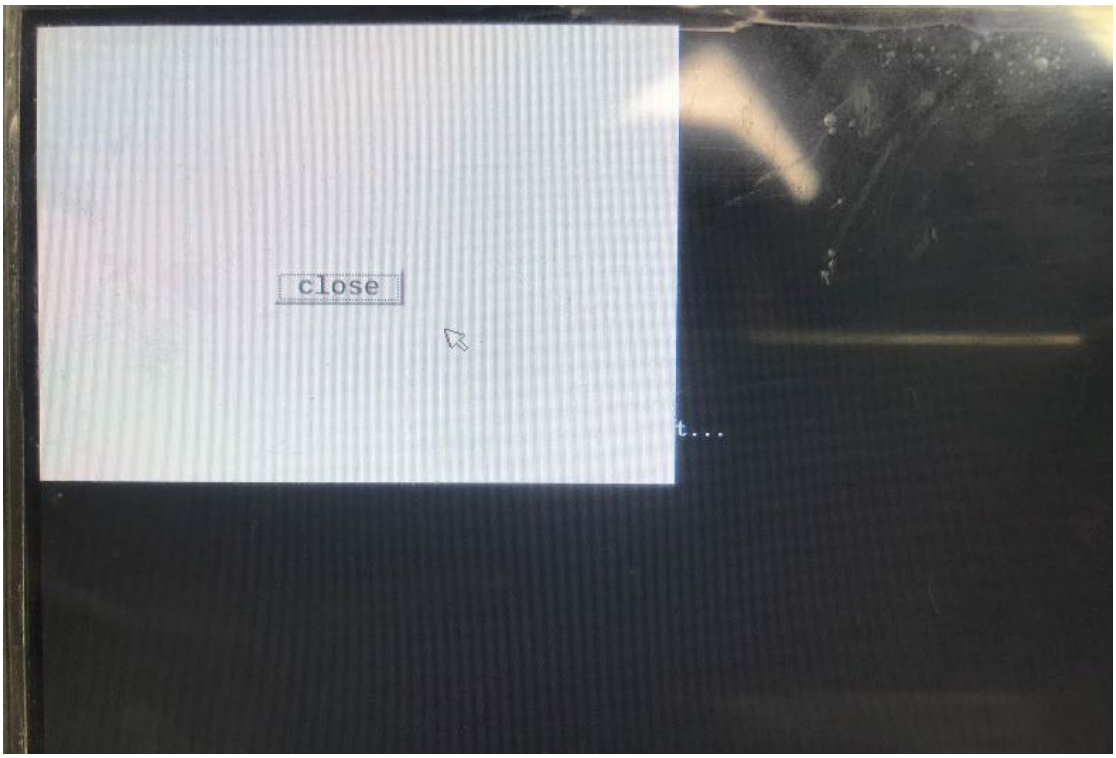
- root@arm:~# **/etc/init.d/weston stop**

- root@arm:~# **./hello-world -platform eglfs**



Or

- root@arm:~# **./hello-world -platform linuxfb**



5.3 VIDEO CAPTURE DEMO

The video collection function includes preview, photo taking, and video recording. Run the Camera application in the UI interface, the icon is shown below.

You can test video preview and capturing.

- Capture: photo shutter, used to take a picture.
- Switch: Swap the main display window and secondary display window on the screen.
- PIP: Close the secondary display window.
- Exit: Terminate the application.

The photos taken can be browsed in Gallery, the icon is

The photo manager interface is as below:

The Delete, Quit, Previous, and Next buttons are to delete the current picture, exit the application, previous picture, and next picture respectively.

Video recording is the video recording function, which is implemented by gstreamer.

Command line reference:

- root@arm:~# **gst-launch-1.0 -e v4l2src device=/dev/video2 num-buffers=1000 io-mode=5 ! 'video/x-raw, \ format=(string)NV12, width=(int)1920, height=(int)1080, framerate=(fraction)30/1' ! ducatimpeg4enc bitrate=12000 ! \ queue !mpeg4videoparse !qtmux !filesink location=out.mp4**

To play the file recorded above, use the following command (assuming the full path of the generated mp4 file is: **/media/out.mp4**):

- root@arm:~# **gst-launch-1.0 playbin uri=file:///media/out.mp4 video-sink=kmssink**

5.4 DUAL DISPLAY DEMO

It's a demo program of dual displays that can demonstrate OpenGL, dual cameras, and

dual displays working at the same time. This application constructs a rotating cube through OpenGL, obtains dual camera data through the V4L2 video acquisition interface, and then displays it on the LCD and HDMI display screens respectively. It is used to demonstrate the powerful video processing and display capabilities of the platform. The Weston should be terminated before testing.

- root@arm:~# **/etc/init.d/weston stop**

The specific test steps:

- Dual screens display a rotating cube and a camera respectively
  - root@arm:~# **kmscube-camera -a**
- Dual screens display rotating cube and #0 camera respectively
  - root@arm:~# **kmscube-camera -a -i 0**
- Dual screens showing rotating cube and #1 camera respectively
  - root@arm:~# **kmscube-camera -a -i 1**
- Single screen display rotating cube
  - root@arm:~# **kmscube-camera -c 32**
  - root@arm:~# **kmscube-camera -c 36**
- Dual screen, one screen displays the rotating cube, and other alternately displays two cameras
  - root@arm:~# **kmscube-camera -a -i 2**

6. ROOTFS BUILD BASED ON YOCTO

Yocto can build a complete embedded system image. This section focuses on building the rootfs system. For more information about Yocto, please visit the official website: https://www.yoctoproject.org/docs/2.5/mega-manual/mega-manual.html

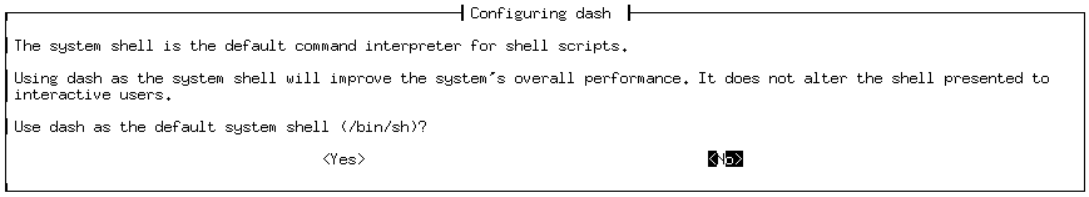
Setting up the Yocot build environment requires a PC with suitable hardware performance, sufficient memory and hardware space, and sufficient network bandwidth. It is recommended that at least 200G free space and 8GB memory be used.

6.1 INSTALL REQUIRED TOOL SOFTWARE

- **$ sudo apt-get install git build-essential python diffstattexinfo gawk chrpath dos2unix wget unzip socatdoxygen libc6:i386 libncurses5:i386 libstdc++6:i386 libz1:i386**

6.2 CONFIGURE BASH

- **$ sudo dpkg-reconfigure dash**



Select "No".

6.3 INSTALL COMPILER

If the cross-compilation tool chain has been installed before, please skip the step.

- **$ wget https://releases.linaro.org/components/toolchain/binaries/6.2-2016.11/arm-linux-gnueabihf/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf.tar.xz**
- **$ tar -Jxvf gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf.tar.xz -C $HOME**

6.4 OBTAIN oe-layertool-setup.sh

- **$ git clone git://arago-project.org/git/projects/oe-layersetup.git tisdk**
- **$ cd tisdk**
- **$ ./oe-layertool-setup.sh -f configs/processor-sdk/processor-sdk-04.00.00.04-config.txt**

6.5 BITBAKE BUILD

- **$ cd ..**
- **$ cd build**

- **$. conf/setenv**
- **$ export PATH=$HOME/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/bin:$PATH**
- **$ MACHINE=am57xx-evm bitbake arago-core-tisdk-image**

The initial build takes a long time, ranging from a few hours to dozens of hours, and the specific time varies depending on PC performance and network bandwidth.

After the build is complete, the target file system is **tisdk/build/arago-tmp-external-linaro-toolchain/deploy/images/am57xx-evm/tisdk-rootfs-image-am57xx-evm.tar.xz**.

7. APPENDIX