

DES8168-EVM 实验

0 实验准备.....	5
1 基础部分.....	7
1.1 CCS 软件的基本使用	7
1.1.1 如何导入已有的工程.....	7
1.1.2 如何编译工程.....	8
1.1.3 如何使用仿真器连接目标板.....	10
1.1.4 如何下载程序至目标板.....	12
1.1.5 如何下载程序至给定的片上内存地址.....	12
1.1.6 如何加载 gel 文件.....	13
1.1.5 如何另存内存中的数据.....	13
1.2 模块测试（CCS 部分）	14
1.2.1 测试 SPI ROM.....	14
1.2.2 测试 NANDFLASH	15
1.2.3 测试 DDR	15
1.2.4 测试 MII、GMII	16
1.2.5 测试 UART	17
1.2.6 测试 VPSS Capture	18
1.2.7 测试 VPSS Display.....	21
2 Linux 系统的搭建及简单应用	24
2.1 系统安装.....	24
2.1.1 虚拟机的安装.....	24
2.1.2 ubuntu 的安装、配置	24
2.1.3 软件开发包安装.....	27
2.1.4 NFS 服务器安装	29
2.1.5 TFTP 服务器安装	30
2.1.6 QT embedded 安装.....	31
2.2 制作镜像文件.....	32
2.2.1 制作 uboot 镜像文件	32
2.2.2 制作内核镜像文件.....	33
2.2.3 制作 UBI 文件系统.....	34
2.3 DVRRDK 镜像烧写.....	34
2.3.1 uboot 启动测试	34
2.3.2 烧写 uboot 到 NandFlash	35
2.3.3 烧写内核及文件系统镜像到 NandFlash	36
2.3.4 NAND 启动 DVRRDK 系统	37
2.4 EZSDK 镜像烧写.....	37
2.4.1 SD 卡制作	38
2.4.2 制作 SD 卡启动脚本	39
2.4.3 SD 卡启动 EZSDK 系统.....	40
2.5 搭建 NFS 网络文件系统	40
2.5.1 搭建 EZSDK 环境网络文件系统.....	41

2.5.2 搭建 DVRRDK 环境网络文件系统.....	42
2.6 模块测试（Linux 部分）	43
2.6.1 测试 HDMI 接口	43
2.6.2 测试 USB 接口（2 个）	44
2.6.3 测试 SATA 硬盘接口	44
2.6.4 测试 MMC 模块.....	45
2.6.5 测试音频接口.....	46
2.6.6 测试 5158 其他 3 个输入视频.....	46
3 基于 EZSDK 的应用开发.....	48
3.1 Matrix-gui	48
3.1.1 运行 Matrix.....	48
3.1.2 编译示例.....	49
3.2 OMTB	49
3.2.1 运行 OMTB	49
3.2.2 编译示例.....	50
3.3 OpenMax.....	50
3.3.1 运行 OpenMax.....	51
3.3.2 编译示例.....	51
3.4 Qt 示例.....	52
3.4.1 运行 Qt 示例.....	52
3.4.2 编译示例.....	52
3.5 SysLink 示例	53
3.5.1 运行 SysLink 示例	53
3.5.2 编译示例.....	54
3.6 Codec Engine 示例	55
3.6.1 运行 Codec Engine 示例	55
3.6.2 编译示例.....	56
3.7 GStreamer	58
3.7.1 运行 GStreamer	58
3.7.2 编译示例.....	58
3.8 Graphics SDK	59
3.8.1 运行 Graphics SDK Demo	59
3.8.2 编译示例.....	60
4 基于 MCFW 的应用开发	62
4.1 修改 DEMO 示例.....	62
4.1.1 标清直通示例.....	62
4.1.2 标清编解码示例.....	66
4.1.3 标清编码示例.....	66
4.1.4 高清解码示例.....	67
4.2 示例编译.....	68
4.3 示例运行.....	68
4.3.1 标清编解码示例.....	68
4.3.2 标清编码示例.....	69
4.3.3 高清解码示例.....	70

4.3.4 标清直通示例.....	70
4.4 视频直通示例加入 sobel 算法	71
4.4.1 源码修改.....	71
4.4.2 源码编译.....	75
4.4.3 运行算法 demo.....	75
4.5 M3_VPSS 测试源码编译	75
4.5.2 编译 M3_VPSS 测试源码	76

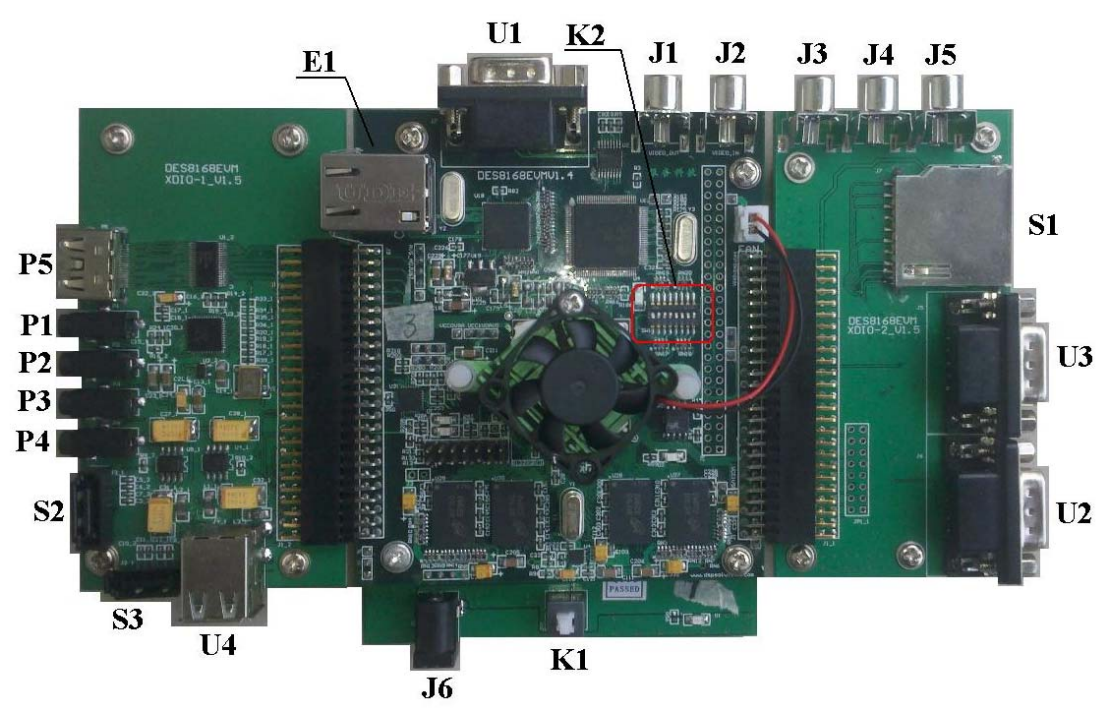
电子科大信工实验室

0 实验准备

实验所需的软件、工具等都位于《experiment》文件目录下，各个文件夹中内容如下表所示：

目录	文件表述
DVRRDK4.0	DVRRDK 软件开发包安装文件，安装在 ubuntu10.04 系统下。
EZSDK5.05	EZSDK 软件开发包安装文件，安装在 ubuntu10.04 系统下。
Flashburn	CCS 烧写 NandFlash 使用的可执行文件、uboot、内核及文件系统镜像文件。
GNU	EZSDK 和 DVRRDK 环境下用于编译内核和 uboot 使用的交叉编译工具。
NFS	搭建网络文件系统使用到的 uboot 和两种环境使用到的内核镜像文件。
RawPlayer	原始视频播放器安装文件，安装在 Windows 系统下。
Sobel	在视频直通程序中添加 Sobel 算法修改的源文件参考文件和算法库文件。
ubuntu	虚拟机中安装 ubuntu10.04 使用的光盘镜像文件。
Vm_Tools	虚拟机中 Linux 系统和当前 Windows 系统之间的文件直接拷贝工具。
VMware_Workstation_wmb	虚拟机安装文件及注册密钥文件，安装在 Windows 系统下。
evm816x	CCS 软件中使用到的 GEL 文件和测试工程文件(包括头文件和库文件)。
SevureCRT	串口终端安装文件压缩包。
Capture_Display_test_pacage	TVP5158 捕获测试可执行性文件和源码、标清显示测试可执行性文件和源码
Video	H264 原码视频文件和 ini 配置文件，用于 DVRRDK 下解码示例演示。
ubi_fs_tool	编译生成 UBI 文件系统使用到的可执行程序，用于 ubuntu_32bit
Prebuilt	定制的 EZSDK 和 DVRRDK 环境下的 uboot、内核和 ubi 文件系统等
Vm	已安装完成的 ubuntu 系统镜像，VMware 软件直接打开使用

EVM 板接口示意图:



EVM 板接口名称:

接口	名称	接口	名称
J1	标清视频输出口 1	S2	SATA 接口
J2	标清视频输入口 1	S3	SATA 接口
J3	标清视频输入口 4	K1	电源开关
J4	标清视频输入口 3	K2	拨码开关
J5	标清视频输入口 2	P1	Line IN
J6	5V 直流电源输入口	P2	麦克风输入
U1	串口 2	P3	耳机输出
U2	串口 1	P4	Line Out
U3	串口 0	P5	HDMI 输出接口
U4	USB 接口 0、1	E1	以太网接口
S1	SD 卡插槽		

1 基础部分

1.1 CCS 软件的基本使用

CCS 软件操作演示是在 CCSv5.2 版本下进行的，其余版本与此类似。CCS 测试工程源码位于《evm816x》文件夹目录下。

1.1.1 如何导入已有的工程

在 CCS 软件中导入现有工程，具体步骤如下：

1. 启动 CCSv5.2 软件，等待初始化完成；
2. 将《experiment》文件夹中的《evm816x》文件夹拷贝到 CCS 的 Workspace 的目录下。该文件夹包含：各个工程的源文件、头文件、库文件和初始化使用的 gel 文件。现使用 D:\DSPresearch 为当前 ccs 软件的 Workspace 目录；
3. 选择工具栏[Project]→[Import Existing CCS Eclipse Project]，如图 1-1-1 所示：

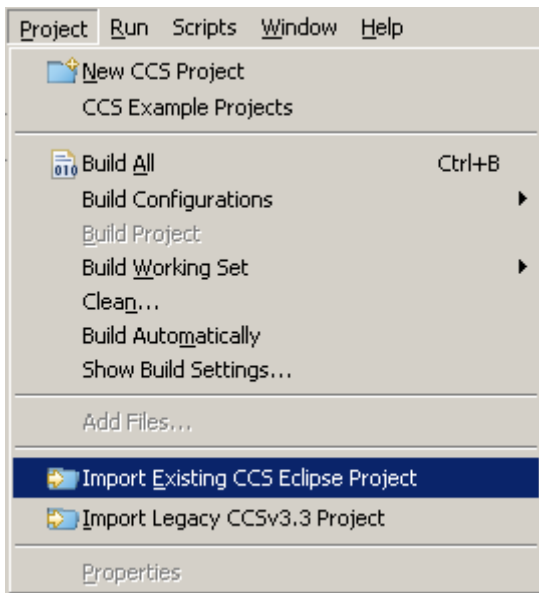


图 1-1-1

4. 选择[Import Existing CCS Eclipse Project]后，点击[Browse]按钮浏览文件夹，选择源码目录：D:\DSPresearch\evm816x\tests，点击[Select ALL]选择所有工程，如图 1-1-2 所示：

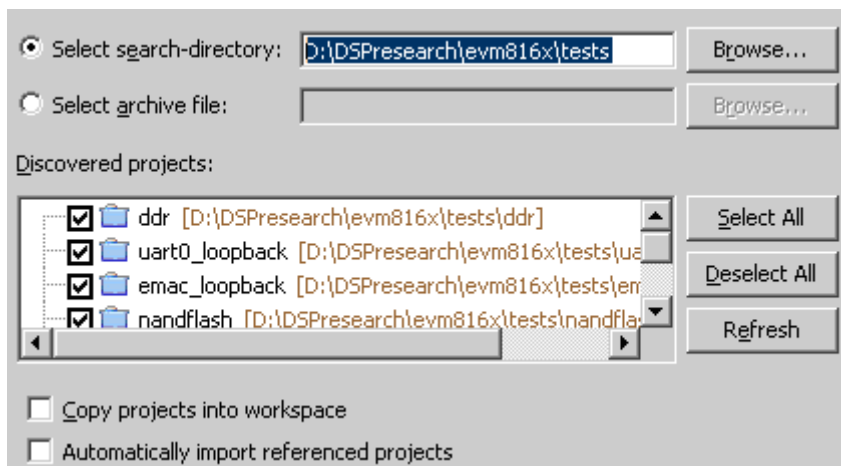


图 1-1-2

5. 最后按下[Finish]，等待软件导入工程完成。至此工程成功导入到 CCS 软件中，可在[Project Explorer]窗口中看到导入的工程目录，如图 1-1-3 所示：

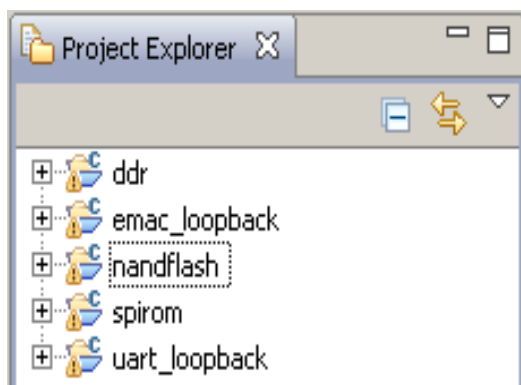


图 1-1-3

1.1.2 如何编译工程

导入工程后，源码需要编译、链接后生成可执行.out 文件才能被 DSP 执行，编译步骤如下：

1. 右键点击待编译的工程选择[Build Project]选项，如图 1-1-4 所示：

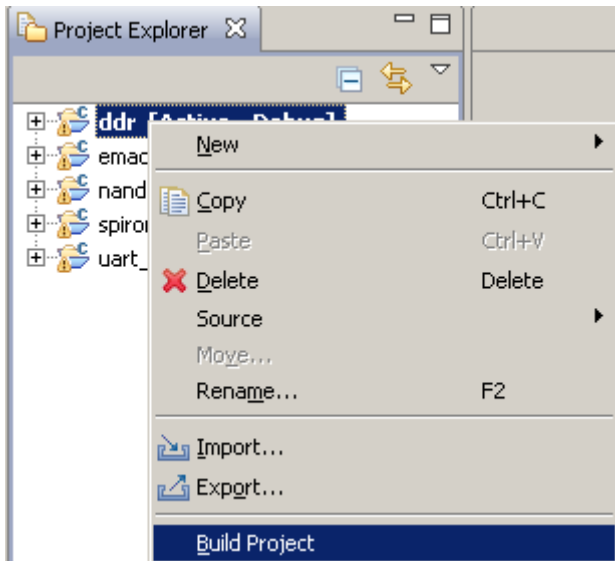


图 1-1-4

2. 编译完成后可在 Console 和 Problem 窗口看见反馈信息，如图 1-1-5 所示：

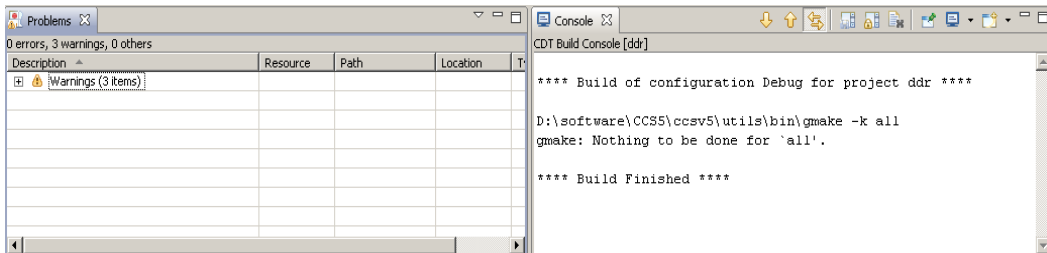


图 1-1-5

注：对应工程的 Properties 中的 Compile version 为 TI v5.0.1。

3. 如需编译所有工程，也可以使用菜单栏中[Project]→[Build All]选项，编译所有工程，如图 1-1-6 所示：

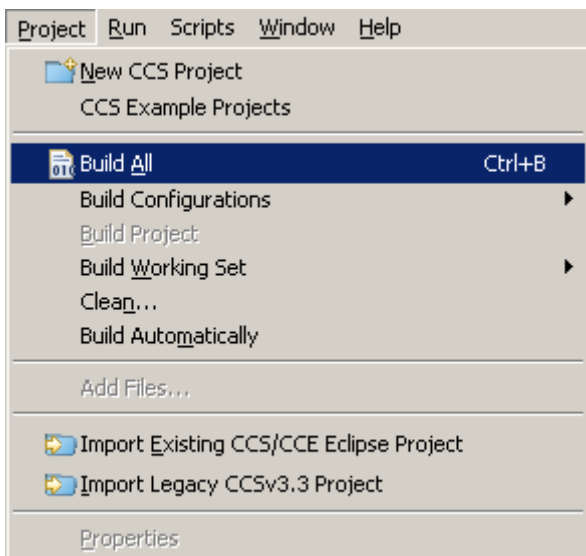


图 1-1-6

1.1.3 如何使用仿真器连接目标板

工程编译生成.out 可执行文件后，就可以通过仿真器载入到芯片片上运行，在此之前需要通过仿真器与芯片连接。现使用仿真器 XDS560 为例，其余仿真器使用于此类似，连接目标板步骤如下：

1. 打开 CCS5.2 软件并等待启动完成后，选择工具栏[View]→[Target Configurations]，弹出 Target Configurations 窗口，如图 1-1-7 所示：

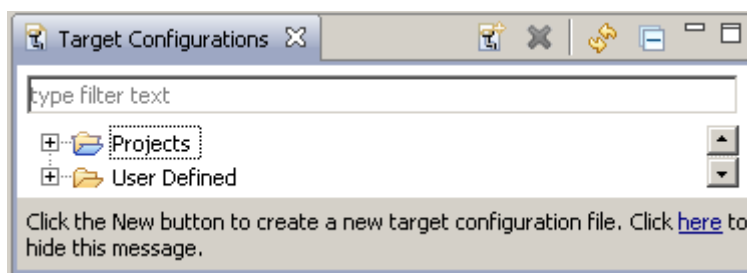


图 1-1-7

2. 在 Target Configurations 窗口中点击右键，选择[New Target Configuration]选项，创建新的配置文件，并命名为 xds560，如图 1-1-8 所示：

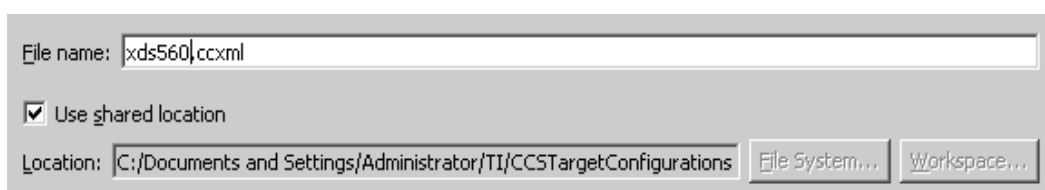


图 1-1-8

注：命名可根据自己需求修改

3. 设置配置文件名称后，点击[Finish]进入参数设置界面。[Connection]选项选择 TI XDS560 Emulator，[Board or Device]选项选择 Ti816x，并保存设置。配置完成后，如图 1-1-9 所示：

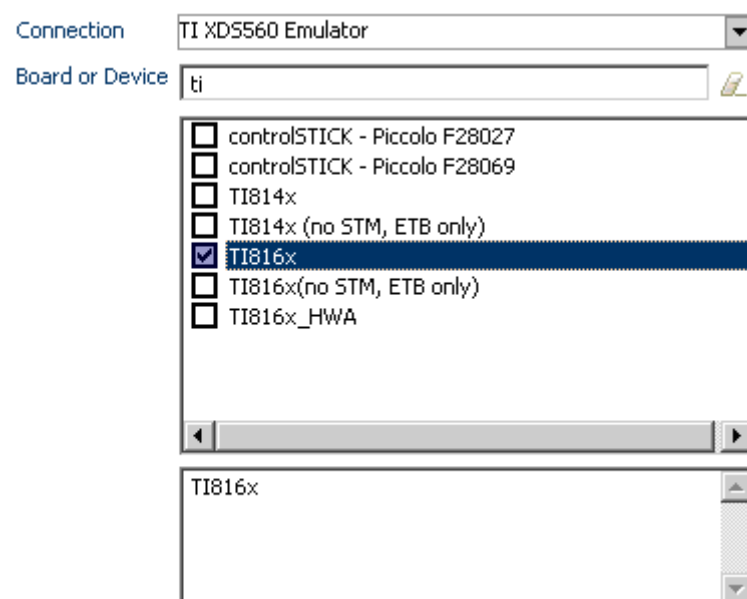


图 1-1-9

注：[Connection]选项中的类型根据实际使用的仿真器型号选择。

4. 连接目标板的串口和仿真器到 PC 机，并打开 PC 机 SecureCRT.exe 串口通讯软件，设置波特率为 115200。
5. 开启 DES8168-EVM 板电源，观察 SecureCRT 软件中是否有打印信息，若有出现读秒提示时，立即按下回车键停止目标板继续启动。随后，在新建的配置文件 xds560.ccml 上右键选择[Launch Selected Configuration]，按照配置启动仿真器，如图 1-1-10 所示：

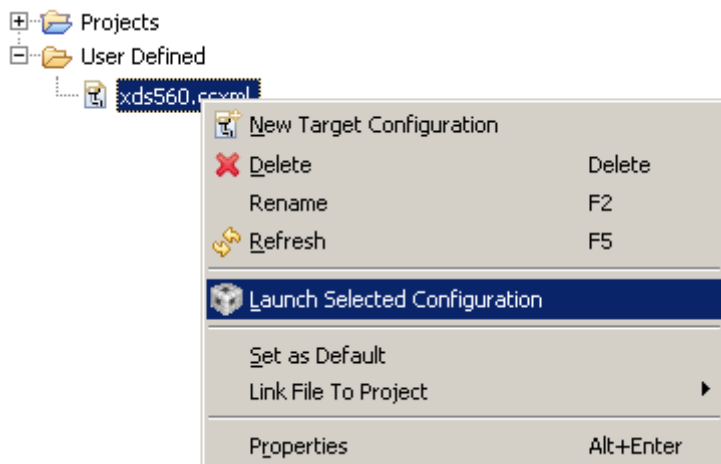


图 1-1-10

6. 等待仿真器配置完成后，我们就可以看到目标板的各个核。在需要连接的核上右键选择[Connect Target]连接到对应的核。例如：连接 Cortex-A8 核后如图 1-1-11 所示：

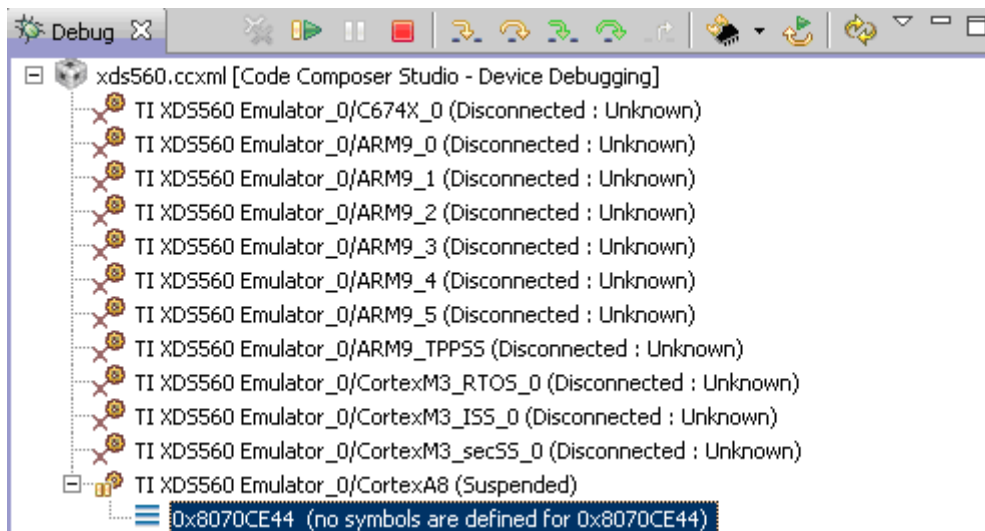


图 1-1-11

注：其中 TI XDS560 Emulator_0/C674X_0 为 DSP 核；
TI XDS560 Emulator_0/CortexA8 为 ARM 核；
TI XDS560 Emulator_0/CortexM3_ISS_0 为 M3_VPSS 核；
TI XDS560 Emulator_0/CortexM3_RTOS_0 为 M3_VIDEO 核；

1.1.4 如何下载程序至目标板

仿真器连接到目标板指定的核之后，我们就可以下载程序到指定核的片上运行。以 Cortex-A8 核上运行程序为例，步骤如下：

1. 开启 CCS 软件后，使用仿真器连接到目标板的 Cortex-A8 核，左键选择列表中已连接的 Cortex-A8 核；
2. 点击工具栏中[Run]→[Load]→[Load Program]，然后在新窗口中选择.out 可执行文件。如图 1-1-12 所示：

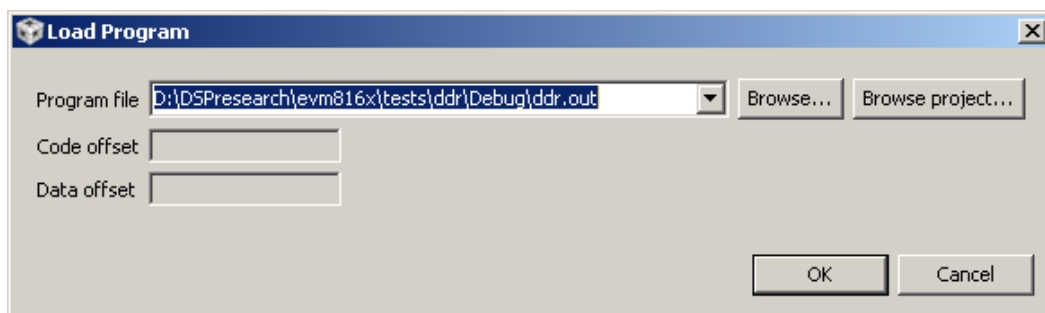


图 1-1-12

3. 点击[OK]后，等待下载完成，按下 F8 键就可以运行载入的程序。更多操作按钮在 Debug 窗口标题处，如图 1-1-13 所示：



图 1-1-13

1.1.5 如何下载程序至给定的片上内存地址

调试时，我们可能需要把代码下载到指定的内存空间运行，例如 uboot。后面的 Linux 应用会使用到，步骤如下：

1. 开启 CCS 软件后，使用仿真器连接到目标板的 Cortex-A8 核，左键选择列表中已连接的 Cortex-A8 核；
2. 选择工具栏[Tools]→[Load Memory]，然后选择指定的文件路径；
3. 选择文件路径后，点击[Next]，设置载入起始地址。其中 [Type Size]选择 32bit，然后点击[Finish]等待载入完成，如图 1-1-14 所示：

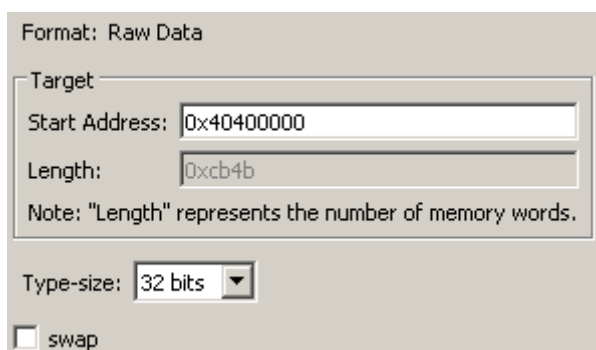


图 1-1-14

1.1.6 如何加载 gel 文件

GEL 文件的加载可以实现对基于 DSP 系统各个核以及外设的初始化，配置 CCS 集成环境，并且可以访问 DSP 系统资源以及仿真器的数据资源。GEL 文件加载步骤如下：

1. 开启 CCS 软件后，启动反序器，不要连接待连接的核；
2. 在待载入 GEL 文件的核上点击右键选择[Open Gel Files View]，打开 Gel 文件窗口，如图 1-1-15 所示：

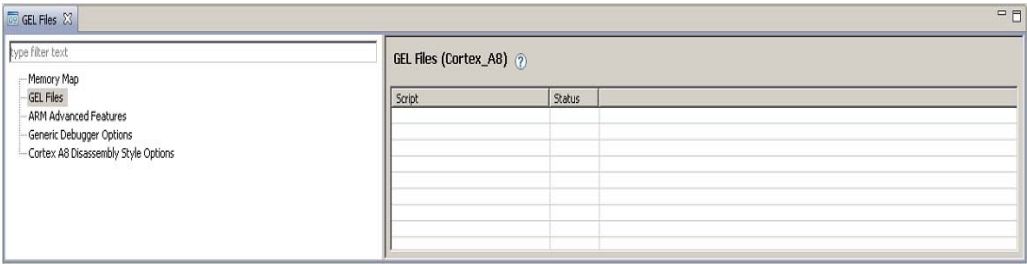


图 1-1-15

3. 在打开的 GEL Files 窗口中右键选择[Load GEL]，然后选 gel 文件路径，载入后如图 1-1-16 所示：

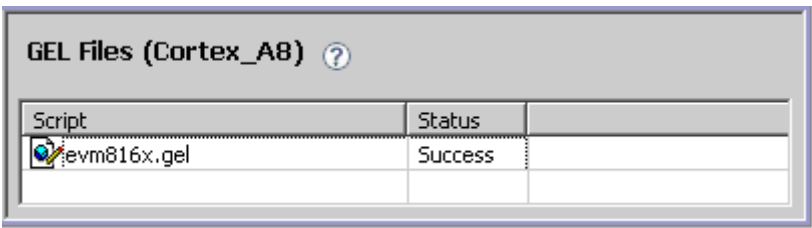


图 1-1-16

4. 此后，连接已载入 gel 文件的核，连接时就会根据载入的 gel 文件配置连接的核。

1.1.5 如何另存内存中的数据

在调试图像文件时，经常会将其从 DDR 中另存出来使用专用的软件工具查看，例如通过 Cortex-A8 核将 DDR 中的数据保存至文件，步骤如下：

1. 开启 CCS 软件后，使用仿真器连接到目标板的 Cortex-A8 核，左键选择列表中已连接的 Cortex-A8 核；
2. 选择工具栏[Tools] → [Save Memory]。然后设置将要保存为的文件的的路径和名称；
3. 设置完路径和名称后点击[Next]，设置数据格式、起始地址、长度、字长，如图 1-1-17 所示。最后点击[Finish]等待数据保存完毕。

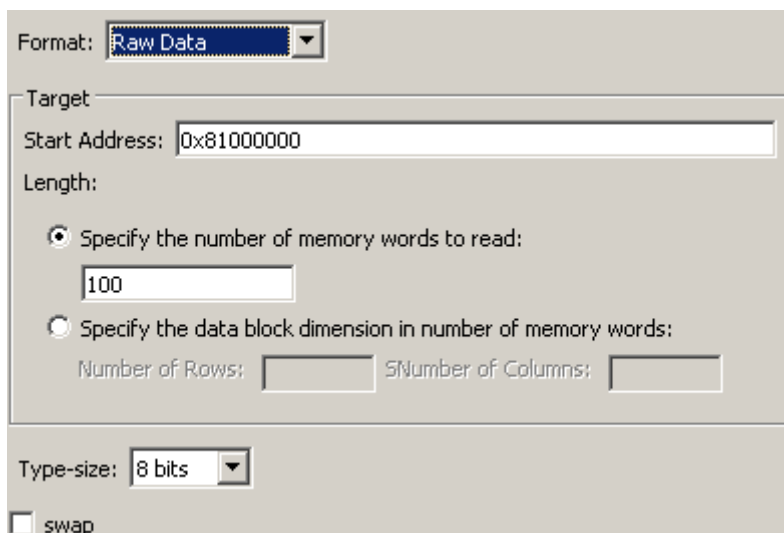


图 1-1-17

1.2 模块测试（CCS 部分）

在测试前，需按照章节 [1.1.1](#) 导入《evm816x》文件夹下测试源码工程。其中 1.2.2~1.2.5 中各个待测试模块代码是在 Cortex-A8 核上运行。1.2.6 和 1.2.7 中各个待测试模块代码在 M3_ISS 核运行，其测试源码的编译是在 Linux 下进行的，在 4.5 章节讲述。

另外，还需要把《experiment》文件夹中《Capture_Display_test_pacage》文件夹拷贝到 Workspace D:\DSPresearch 目录下，并安装《experiment》文件夹中的 RawPlayer 用于播放捕获的图像。

1.2.1 测试 SPI ROM

SPI ROM 测试关键源码位于 spirom_test.c 中 Int16 spirom_test()。此代码对 spirom 的前 4 页作了读写测试。Spirom 的硬件参数位于 spirom.h 中，包括 SPIROM_PAGESIZE、SPIROM_SECTORSIZE。在不同的 spirom 型号中要根据实际的 spirom 硬件参数修改。

另外，需要注意 SPI ROM 测试后会损坏以前保存在 ROM 上的数据。SPI ROM 测试步骤如下：

1. 编译 spirom 工程，生成可执行文件；
2. 打开 DES8168-EVM 板电源开关，若在串口终端中看到 uboot 启动时的读秒提示信息立即键入回车键停止其继续启动。如没有烧写 uboot 则需要在 Cortex-A8 核上加载 gel 文件，按照章节 [1.1.6](#) 加载 D:\DSPresearch\Capture_Display_test_pacage\gel 目录下的 evm816x.gel。完成上述操作后，打开 CCS 软件使用仿真器连接到 Cortex-A8 核。
3. 按照 [1.1.4](#) 章节步骤载入 spirom 测试程序，可执行文件位于以下目录下：D:\DSPresearch\evm816x\tests\mcspi_spirom
4. 测试通过后，控制台打印信息如图 1-2-1 所示：

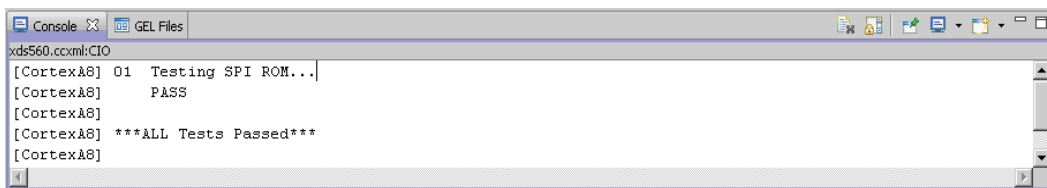


图 1-2-1

1.2.2 测试 NANDFLASH

NandFlash 测试关键源码位于 `nandflash.c` 中 `Int16 nandflash_test ()`。此代码测试 NandFlash 的前 8 个块，以写入数据再读出做对比测试 Flash 的读写。

NandFlash 的硬件参数位于 `nandflash.h` 中，包括 `NANDFLASH_BLOCKS`、`NANDFLASH_PAGESPERBLOCK`、`NANDFLASH_PAGESIZE`、`NANDFLASH_SPARESIZE`。在不同的 `nandflash` 型号中要根据实际的硬件参数设定。

另外，需要注意 NandFlash 测试会损坏烧写在 NandFlash 上的数据。
NandFlash 测试步骤如下：

1. 编译 `nandflash` 工程，生成可执行文件；
2. 打开 DES8168-EVM 板电源开关，若在串口终端中看到 `uboot` 启动时的读秒提示信息立即键入回车键停止其继续启动。如没有烧写 `uboot` 则需要在 Cortex-A8 核上加载 `gel` 文件，按照章节 [1.1.6](#) 加载 `D:\DSPresearch\Capture_Display_test_pacage\gel` 目录下的 `evm816x.gel`。完成上述操作后，打开 CCS 软件使用仿真器连接到 Cortex-A8 核。
3. 按照 [1.1.4](#) 章节载入 NandFlash 测试程序，可执行文件位于以下目录：
`D:\DSPresearch\evm816x\tests\nandflash\Debug`
4. 测试通过后，控制台打印信息如图 1-2-2 所示：

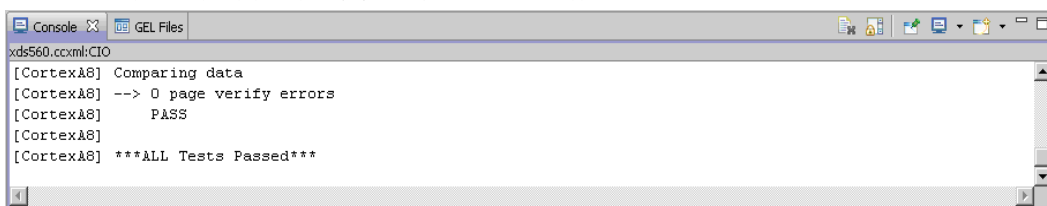


图 1-2-2

1.2.3 测试 DDR

DDR 测试关键源码位于 `ddr_test.c` 中 `Int16 ddr_test ()`。此代码测试 `ddr` 的部分区域，分别写入特定数据和按位取反地址数据，再读出做对比测试 DDR 的读写功能。

DDR 测试步骤如下：

1. 编译 `ddr` 工程，生成可执行文件；
2. 打开 DES8168-EVM 板电源开关，若在串口终端中看到 `uboot` 启动时的读秒提示信息立即键入回车键停止其继续启动。如没有烧写 `uboot` 则需要在 Cortex-A8 核上加载 `gel` 文件，按照章节 [1.1.6](#) 加载 `D:\DSPresearch\`

Capture_Display_test_pacage\gel 目录下的 evm816x.gel。完成上述操作后，打开 CCS 软件使用仿真器连接到 Cortex-A8 核。

- 按照 1.1.4 章节下载 ddr 测试程序，可执行文件位于以下目录：

D:\DSPresearch\evm816x\tests\ddr\Debug

- 测试通过后，控制台打印信息如图 1-2-3 所示：

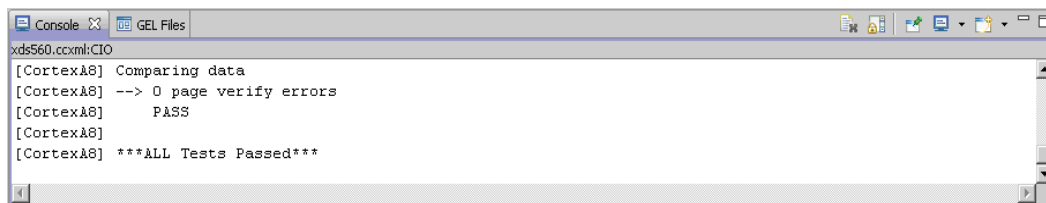


图 1-2-3

1.2.4 测试 MII、GMII

以太网测试关键源码位于 emac_gmii_test.c 中 Int16 emac_gmii_test () 和 emac_mii_test.c 中 Int16 emac_mii_test ()。此代码分别对 1000M 和 100M 网卡进行了回环测试，因此在测试时需在目标板网络接口插入回环网络插头。以太网测试步骤如下：

- 编译 emac_loopback 工程，生成可执行文件；
- 打开 DES8168-EVM 板电源开关，若在串口终端中看到 uboot 启动时的读秒提示信息立即键入回车键停止其继续启动。如没有烧写 uboot 则需要在 Cortex-A8 核上加载 gel 文件，按照章节 1.1.6 加载 D:\DSPresearch\Capture_Display_test_pacage\gel 目录下的 evm816x.gel。完成上述操作后，使用仿真器连接到 Cortex-A8 核。
- 按照 1.1.4 章节载入以太网测试程序，可执行文件位于以下目录：
D:\DSPresearch\evm816x\tests\emac_loopback\Debug\
- 测试通过后，控制台打印信息如图 1-2-4 所示：

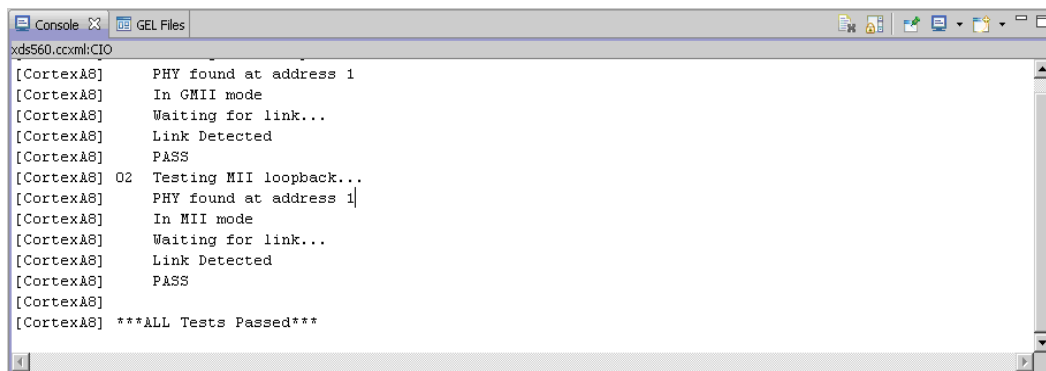


图 1-2-4

注：回环网络插头制作方法如下：

网络接头定义如图 1-2-5 所示，百兆网卡网线，连接 1,3 脚和 2,6 脚；千兆网卡网线，需在前面基础上再连接 4,7 脚和 5,8 脚。

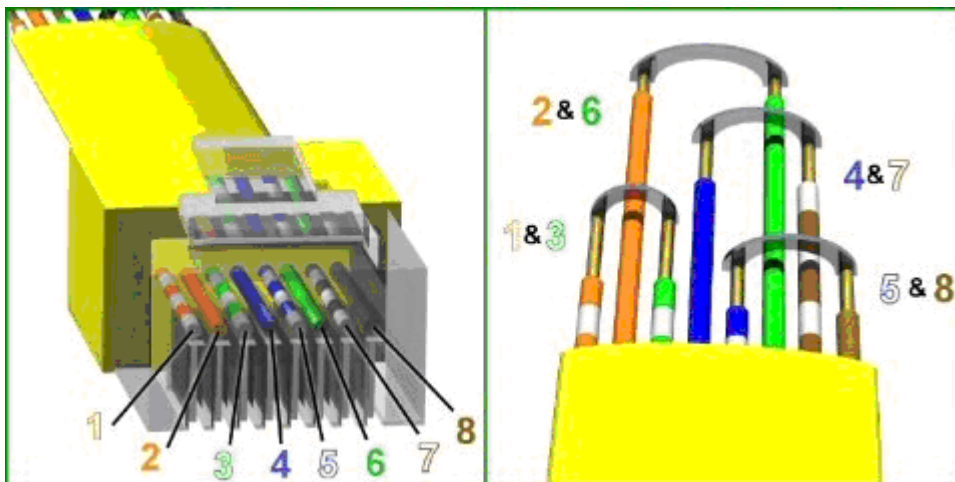


图 1-2-5

1.2.5 测试 UART

串口测试关键源码位于 `uart_loopback_test.c` 中 `Int16 uart_loopback_test ()`。此代码使用串口收发 256 字节数据做回环测试，因此在测试各个串口时需在对应的串口上插入回环插头。DES8168-EVM 板上共有 3 个串口，分别单独测试。

串口测试步骤如下：

1. 将 `uart_loopback_test ()` 函数中 `EVM816X_UART_open` 打开的串口修改为对应的串口。如图 1-2-6 中“UART_0”，对应串口 0，可将其修改为“UART_1”和“UART_2”分别对应于串口 1 和串口 2；

```
62     if ( ( uart1 = EVM816X_UART_open( UART_0, /*115200*/ 13 ) ) == 0 )
63         return -1;
```

图 1-2-6

2. 编译 `uart_loopback` 工程，生成可执行文件；
3. 打开 DES8168-EVM 板电源开关，若在串口终端中看到 `uboot` 启动时的读秒提示信息立即键入回车键停止其继续启动。如没有烧写 `uboot` 则需要在 Cortex-A8 核上加载 `gel` 文件，按照章节 [1.1.6](#) 加载 `D:\DSPresearch\Capture_Display_test_package\gel` 目录下的 `evm816x.gel`。完成上述操作后，使用仿真器连接到 Cortex-A8 核。
4. 按照 [1.1.4](#) 章节载入串口测试程序，可执行文件位于以下目录：
`D:\DSPresearch\evm816x\tests\emac_loopback\Debug\`
5. 测试通过后，控制台打印信息如图 1-2-7 所示：

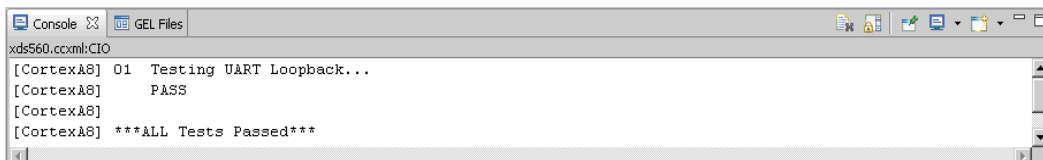


图 1-2-7

1.2.6 测试 VPSS Capture

TVP5158 视频捕获测试源码及编译在 4.5 章节讲述，现将其编译生成的可执行文件直接使用。测试前 DES8168-EVM 板的标清视频输入 1 连接到视频输入，标清视频输出口连接到标清显示器。测试步骤如下：

1. CCS 软件启动后，打开 DES8168-EVM 板电源开关，若在串口终端中看到 uboot 启动时的读秒提示信息立即键入回车键停止其继续启动。若没有打印提示信息提示则直接启动仿真器。
2. 选中 Cortex-A8 核并载入 gel 文件 TI816x_evm_A8_ddr3.gel，位于 D:\DSPresearch\Capture_Display_test_package\gel 目录下。载入 gel 文件后如图 1-2-8 所示：

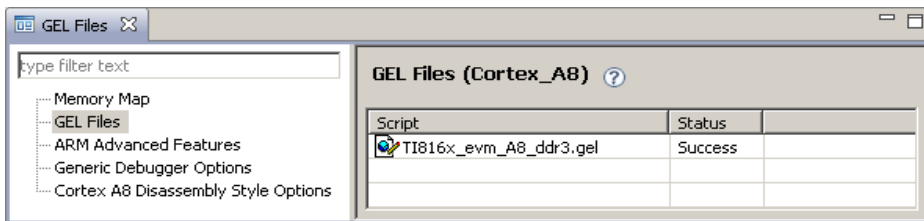


图 1-2-8

3. 选中 CortexM3_ISS 核并载入 gel 文件 TI816x_evm_ducati.gel，位于 D:\DSPresearch\Capture_Display_test_package\gel 目录下。载入 gel 文件后如图 1-2-9 所示：

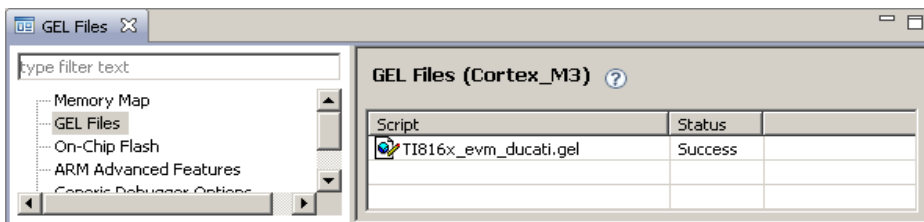


图 1-2-9

4. 连接 Cortex-A8 核，等待连接完成后选择工具栏 [Scripts] → [TI816x HDVPSS Init] → [HDVPSSInit],运行该脚本，如图 1-2-10 所示：



图 1-2-10

5. 连接 CortexM3_ISS 核，等待连接完成后选择工具栏 [Scripts] → [UniCacheEnableDisable] → [Ducati_Cache_Enable],运行该脚本，如图 1-2-11 所示：



图 1-2-11

6. 载入测试代码 hdpss_examples_captureVip_m3vpss_debug.xem3，位于：D:\DSPresearch\Capture_Display_test_package\fireware 目录下；

7. 选择 debug 窗口中 CortexM3_ISS 核选项下选中 [main() at VcaptureVip_main.c],在函数观察窗口中将提示定位源码路径, 如图 1-2-12 所示。按下[Locate File...]定位源码文件, 源文件位于: D:\DSPresearch\Capture_Display_test_package\src\capture\captureVip\src 目录下;

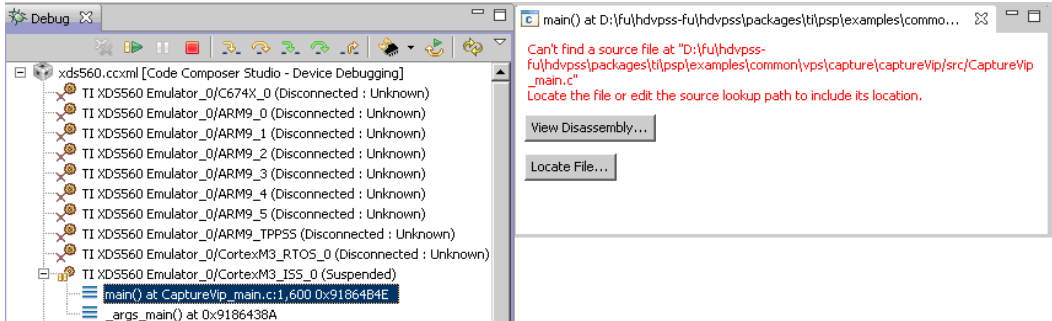


图 1-2-12

8. 定位到源文件后, 函数观察窗口会显示源码。在 CaptureApp_callback 函数中[gCaptureApp_ctrl.totalFieldCount += frameList.numFrames]处如图 1-2-13 红色方框所示处加入断点。鼠标放在该行, 右键选择[Breakpoint(Code Composer Studio)] → [Breakpoint]即可在该位置加入断点。

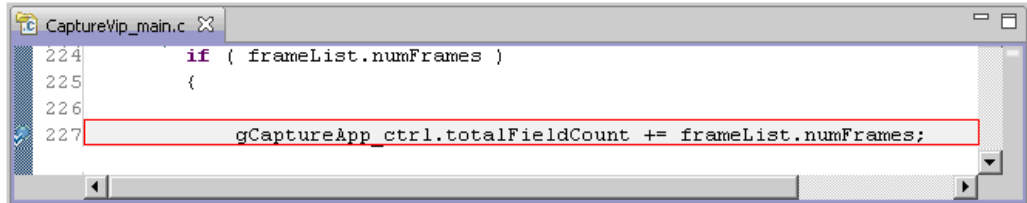


图 1-2-13

9. 添加断点后即可在[Breakpoint]列表窗口看到该断点, 如图 1-2-14 所示:

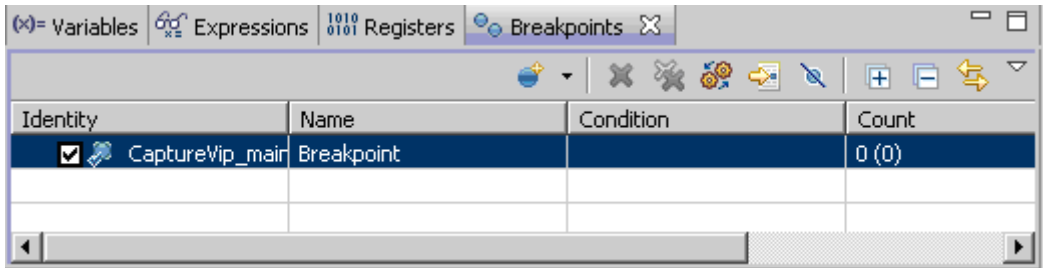


图 1-2-14

10. 双击函数观察窗口中 framelist 变量, 将其选中。然后右键选择[Add Watch Expression...], 将 framelist 添加[Expressions]变量观察窗口, 如图 1-2-15 所示:

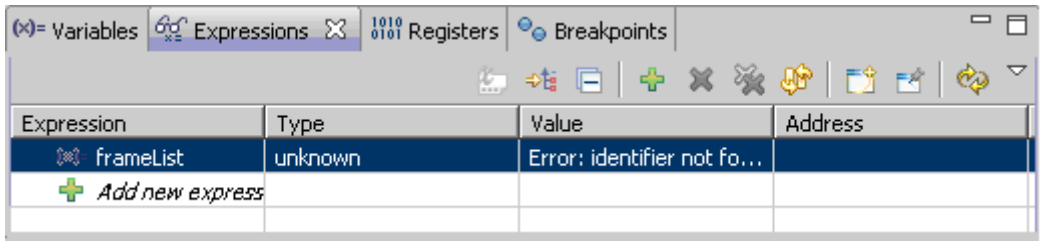
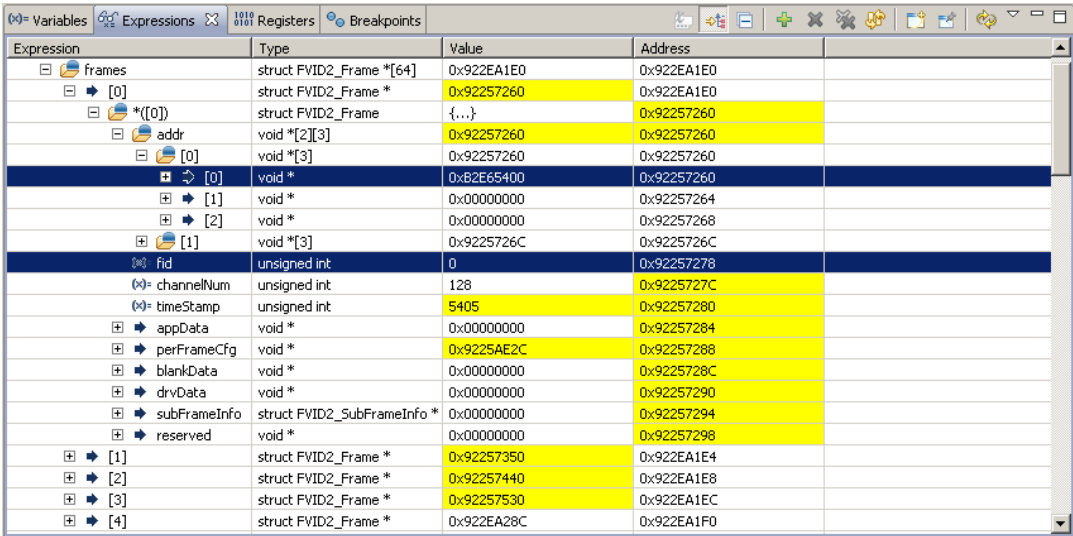


图 1-2-15

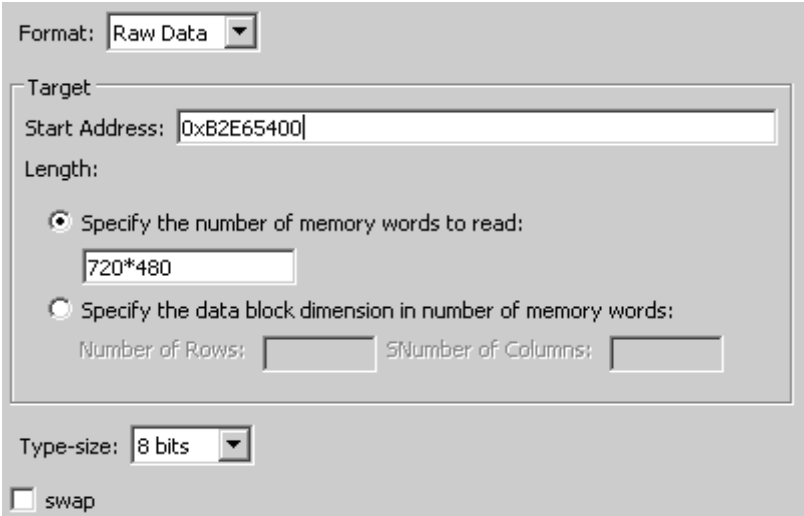
11. 按下 F8 运行代码直到停在我们设置的断点处，并多次运行，观察变量窗口 framelist。当 frame[0]的 fid=0 时，如图 1-2-16 所示。查看 addr[0][0]的值为 0xB2E65400，这个值就是视频帧保存的地址，每次可能不同，将其复制下来。



Expression	Type	Value	Address
frames	struct FVID2_Frame *[64]	0x922EA1E0	0x922EA1E0
frames[0]	struct FVID2_Frame *	0x92257260	0x922EA1E0
frames[0] -> {...}	struct FVID2_Frame	{...}	0x92257260
frames[0] -> addr	void *[2][3]	0x92257260	0x92257260
frames[0] -> addr[0]	void *[3]	0x92257260	0x92257260
frames[0] -> addr[0][0]	void *	0xB2E65400	0x92257260
frames[0] -> addr[0][1]	void *	0x00000000	0x92257264
frames[0] -> addr[0][2]	void *	0x00000000	0x92257268
frames[0] -> addr[1]	void *[3]	0x9225726C	0x9225726C
frames[0] -> fid	unsigned int	0	0x92257278
frames[0] -> channelNum	unsigned int	128	0x9225727C
frames[0] -> timeStamp	unsigned int	5405	0x92257280
frames[0] -> appData	void *	0x00000000	0x92257284
frames[0] -> perFrameCfg	void *	0x9225AE2C	0x92257288
frames[0] -> blankData	void *	0x00000000	0x9225728C
frames[0] -> drvData	void *	0x00000000	0x92257290
frames[0] -> subFrameInfo	struct FVID2_SubFrameInfo *	0x00000000	0x92257294
frames[0] -> reserved	void *	0x00000000	0x92257298
frames[1]	struct FVID2_Frame *	0x92257350	0x922EA1E4
frames[2]	struct FVID2_Frame *	0x92257440	0x922EA1E8
frames[3]	struct FVID2_Frame *	0x92257530	0x922EA1EC
frames[4]	struct FVID2_Frame *	0x922EA28C	0x922EA1F0

图 1-2-16

12. 保存数据，选择工具栏[Tools] → [Save Memory]。保存文件路径为 D:\DSPresearch\video.bin，然后[Format]选择 Raw Data, [Start Address]为刚才复制的 addr[0][0]的值，[Length]为 720*480，[type-size]选择 8-bit，如图 1-2-17 所示：



Format: Raw Data

Target

Start Address: 0xB2E65400

Length:

☒ Specify the number of memory words to read:

720*480

☐ Specify the data block dimension in number of memory words:

Number of Rows: Number of Columns:

Type-size: 8 bits

☐ swap

图 1-2-17

13. 保存完毕后将 video.bin 重命名为 video.yuv，并使用 RawPlayer 打开该文件。启动 RawPlayer 后选择[文件] → [打开]选项，提示设置界面，设置尺寸为 720*240，帧率为 30 帧/秒，如图 1-2-18 所示。在对话框中找到 video.yuv 路径，点击[打开]即可预览捕获的一帧图像。



图 1-2-18

1.2.7 测试 VPSS Display

TVP5158 标清显示测试源码及编译在 4.5 章节讲述，现将其编译生成的可执行文件直接使用。测试前 EVM 板的标清视频输出口需要连接到标清显示器。测试步骤如下：

1. CCS 软件启动后，打开 DES8168-EVM 板电源开关，若在串口终端中看到 uboot 启动时的读秒提示信息立即键入回车键停止其继续启动。若没有打印信息提示则直接启动仿真器。
2. 选中 Cortex-A8 核并载入 gel 文件 TI816x_evm_A8_ddr3.gel，位于 D:\DSPresearch\Capture_Display_test_package\gel 目录下。载入 gel 文件后如图 1-2-19 所示：

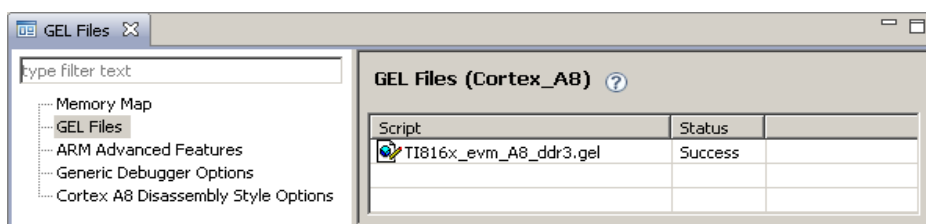


图 1-2-19

1. 选中 CortexM3_ISS 核并载入 gel 文件 TI816x_evm_ducati.gel，位于 D:\DSPresearch\Capture_Display_test_package\gel 目录下。载入 gel 文件后如图 1-2-20 所示：

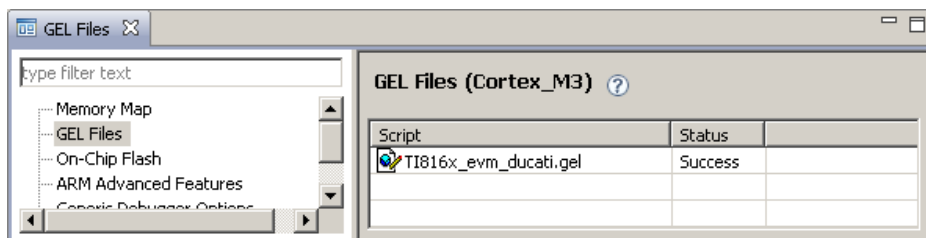


图 1-2-20

2. 连接 Cortex-A8 核，等待连接完成后选择工具栏[Scripts] → [TI816x HDVPSS Init] → [HDVPSSInit]，运行该脚本，如图 1-2-21 所示：

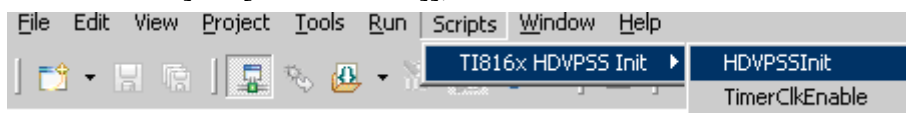


图 1-2-21

3. 连接 CortexM3_ISS 核, 选择工具栏[Scripts] → [UniCacheEnableDisable] → [Ducati_Cache_Enable],运行该脚本如图 1-2-21 所示:

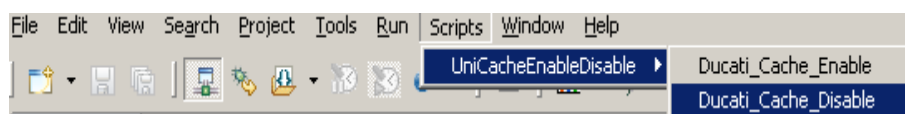


图 1-2-22

4. 载入测试代码
hdivpss_examples_sdDisplay_m3vpss_whole_program_debug.xem3,位于:
D:\DSPresearch\Capture_Display_test_package\fireware 目录下;
5. 按下 F8 运行代码, 观察控制台输出, 提示选择显示模式, 如图 1-2-23 所示。
然后输入 1(NTSC 模式)和回车;

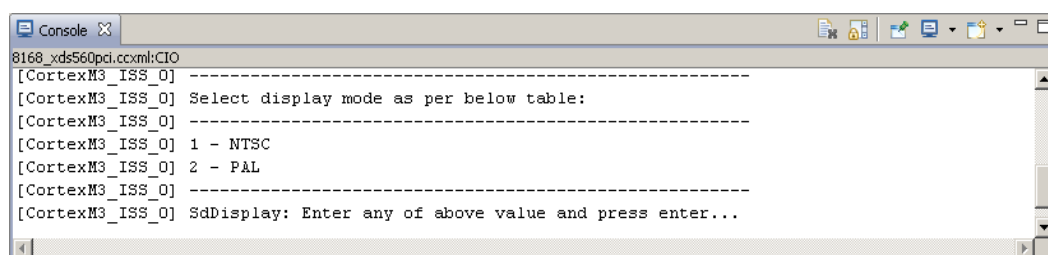


图 1-2-23

6. 接着提示选择显示输入数据格式, 如图 1-2-24 所示。选择 2 (YUV422I 格式), 按下回车确认;

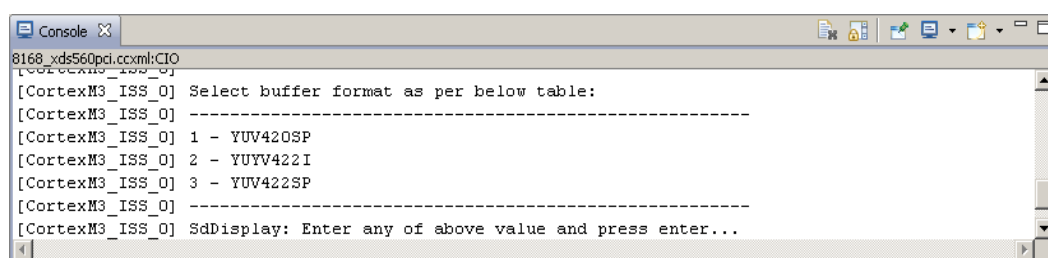


图 1-2-24

7. 接着提示选择显示输出视频格式, 如图 1-2-25 所示。选择 2 (COMPOSITE 格式), 按下回

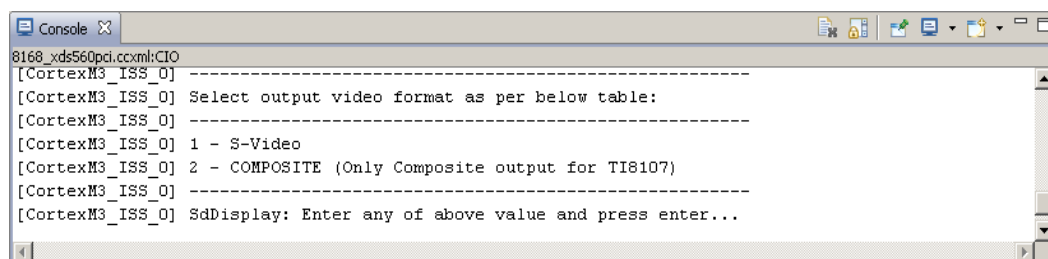


图 1-2-25

8. 随后提示载入数据, 如图 1-2-26 所示。暂停 M3 运行, 选择工具栏[Tools] → [Load Memory]下载数据文件到地址 0xa0800000。测试视频文件路径为:
D:\DSPresearch\Capture_Display_test_package\test_video\PSP10_720x480.yuyv422。

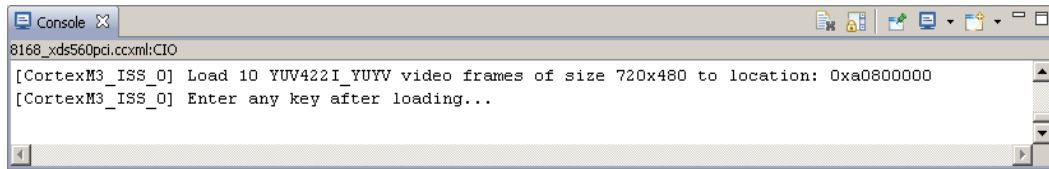


图 1-2-26

9. 载入数据后，按下 **F8** 运行，并在控制台中键入任意数字并按下回车。此后在目标板板标清视频输出口将会有一段测试视频输出，可在标清显示器上观察到。

2 Linux 系统的搭建及简单应用

2.1 系统安装

2.1.1 虚拟机的安装

DES8168 的开发主要在 Linux-ubuntu10.4 版本下进行。ubuntu 安装在 Windows 系统虚拟机环境下。虚拟机的安装直接运行《VMware_Workstation_wmb》目录下 VMware-workstation-full-8.0.2-591240.exe 可执行程序。在安装完成后使用《VMware-workstation 注册机_keygen》目录下《有效注册密钥.txt》注册虚拟机。

ubuntu 系统可直接使用《Vm》目录下的已经安装完成的 Linux-ubuntu10.4 系统文件。使用虚拟机启动 Linux 系统：

- 1) 打开虚拟机软件 vmware。
- 2) 选择工具栏[File] → [Open]，打开《Vm》目录下的虚拟机 Ubuntu.vmx。
- 3) 鼠标左键点击 vmware 软件[Commands]窗口中的[Power on this virtual machine]启动 ubuntu 系统。

另外，如需重新安装 Linux-ubuntu10.4 系统，则需按照 2.1.2~2.1.6 章节安装、配置各个组件。

2.1.2 ubuntu 的安装、配置

1. ubuntu 安装：

- 1) 虚拟机安装完成后，打开虚拟机[home]目录下 New Virtual Machin 图标(或者工具栏[File] → [New] → [Virtual Machin])安装 ubuntu。使用到的 ISO 文件位于《ubuntu》目录下；

注：

建议分配内存 512M，分配硬盘 100G；

安装时需要主机连接到外网，以便系统自动安装各个组件；

其中的[user name]设置栏设置为：user（此步骤是为了和文档后续 Linux 使用的路径相匹配，也可按自己需求设置）

- 2) 选择配置完成后等待安装完成，至此 ubuntu 系统已成功安装。

2. 配置 VMware 虚拟机：

- 1) 进入 ubuntu 系统，首先打开虚拟机软件 vmware，然后选择工具栏[File] → [Open]打开安装的虚拟机，后缀为.vmx。
- 2) 配置虚拟机网络适配器方式为：桥接方式(Bridged)，如图 2-1-1 所示：

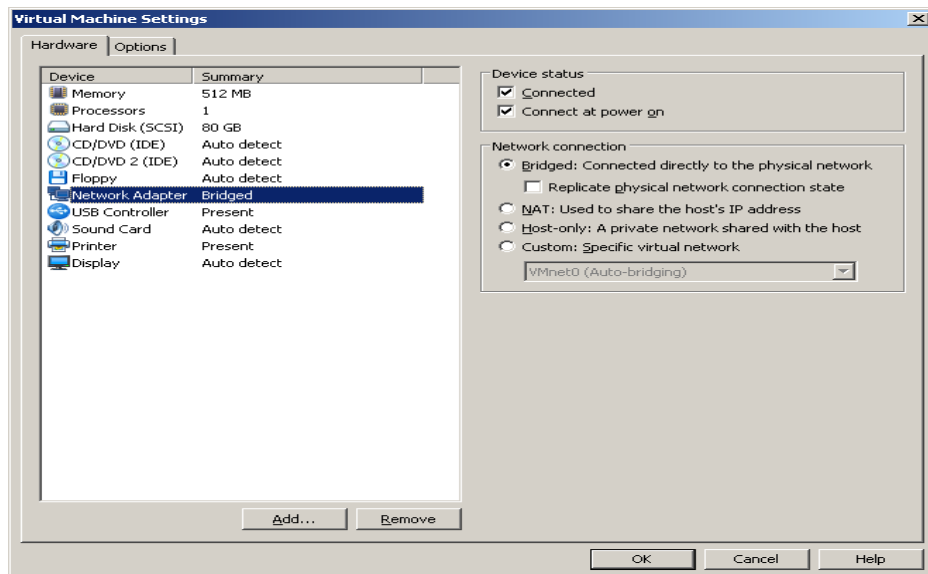


图 2-1-1

3. 配置 ubuntu 系统:

- 1) 进入 ubuntu 操作系统后设置虚拟机网络 IP，在 ubuntu 菜单栏中选择 [System] → [Preference] → [Network Connections] 进入设置界面，如图 2-1-2 所示：



图 2-1-2

- 2) 选中 Auto eth0，并按下 Edit 按钮对其进行修改。切换到 [IPv4 Settings] 选项，[Method] 选择 Manul，然后添加 IP 地址和子网掩码等。其中 IP 地址应与 windows 下主机在一个段，例如：设置主机的 IP 地址为 192.168.1.43，虚拟机中 IP 地址为：192.168.1.41。设置结果如图 2-1-3 所示：

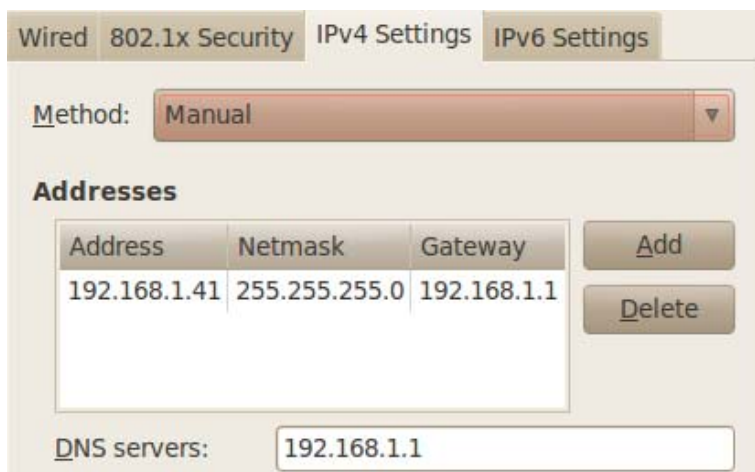


图 2-1-3

- 3) 在 ubuntu 菜单栏中选择[Applications] → [Accessories] → [Terminal]即可进入命令终端，如图 2-1-4 所示：

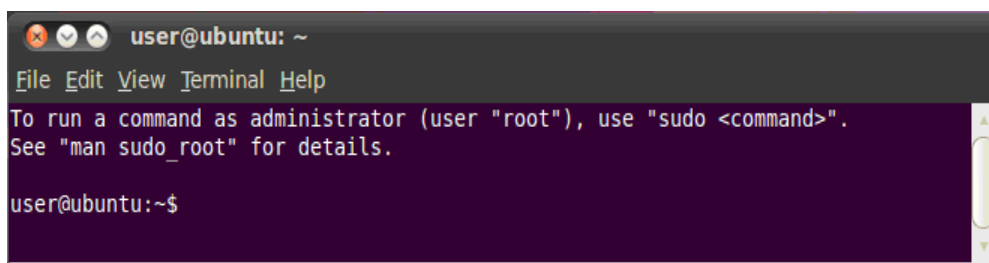


图 2-1-4

- 4) 在终端中执行如下命令进入连接管理配置文件编辑界面：

```
ubuntu$ sudo gedit /etc/NetworkManager/nm-system-settings.conf
```

注：sudo 是允许系统管理员让普通用户执行一些或者全部的 root 命令的一个工具，使用后提示输入安装 ubuntu 时设定的密码。密码输入时没有字符提示。

编辑界面，如图 2-1-5 所示

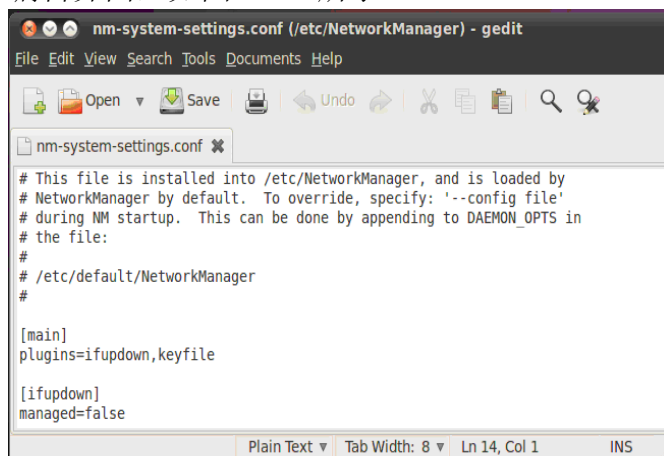


图 2-1-5

- 5) 修改配置文件中[ifupdown]managed 赋值为 true，并按下[save]保存文件。然后在终端中重启连接管理器，执行：

```
ubuntu$ sudo /etc/init.d/network-manager restart
```

或者执行：

```
ubuntu$ sudo /etc/init.d/service network-manager restart
```

4. 设置 root 帐户密码、获取 root 权限

安装 ubuntu 后，默认的情况下 root 的密码没有设置，如果使用 su 命令是无法切换到 root 权限下的。因此，需要设置 root 密码，在终端中执行：

```
ubuntu$ sudo passwd root
```

然后，按照提示首先输入安装 ubuntu 时设定的密码，再输入两次 root 帐户密码。执行成功后如图 2-1-6 所示：

```
user@ubuntu:~$ sudo passwd root
[sudo] password for user:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

图 2-1-6

此后，可在终端中执行 su 命令，然后输入刚才配置的 root 帐户密码获取 root 权限。在下面的 linux 系统下的操作都应先执行 su 命令获取 root 权限，避免权限问题带来的错误。执行 su 命令后如图 2-1-7 所示：

```
user@ubuntu:~$ su
Password:
root@ubuntu:/home/user#
```

图 2-1-7

5. 安装 Vm Tools:

为实现 ubuntu 和 Windows 的文件的复制与粘贴，需安装 Vm Tools 工具。拷贝《Vm_Tools》目录下 windows.iso 到 vmware 虚拟机安装根目录(并非 ubuntu 系统的安装目录)下即可。

2.1.3 软件开发包安装

在 DM8168 开发环境中有两套软件开发包：EZSDK 和 DVRRDK。在安装之前，选择 ubuntu 左上角菜单栏中 [Places] → [Home Folder] 进入 [HOME] 目录下，新建一个文件夹，如 workspace。把《experiment》目录下的 DVRRDK4.0、EZSDK5.05 和 GNU 文件夹下的安装文件拷贝到 ubuntu 下新建的 workspace 目录下。

注意每个 ubuntu 系统的 [HOME] 目录名称是在安装时设置的 [user name] 选项设置的，例如我们现在使用的叫作 user。在使用时要区分开文档和自己路径的不同，根据实际情况修改。

EZSDK 软件开发包需要另外安装 CodeSourcery 工具包，用于编译链接各个模块。此外在 DVRRDK 中同样使用这个编译工具包编译 uboot 及内核。

1. 交叉编译工具 CodeSourcery 安装步骤：

- 1) 打开命令终端，使用命令切换工作目录到 workspace 目录下，执行：

```
ubuntu# cd /home/user/workspace/
```

- 2) 改变交叉编译工具安装包使用权限为所有人可使用，执行：

```
ubuntu# chmod 777 arm-2009q1-203-arm-none-linux-gnueabi.bin
```

- 3) 运行安装程序，执行：

```
ubuntu# ./arm-2009q1-203-arm-none-linux-gnueabi.bin
```

注：安装使用默认配置即可，在完成后可取消 View “Getting Started” guide 的勾选选项；安装完成后在[HOME]目录下可看到 GCC 安装文件夹 CodeSourcery；

- 4) 编辑 Shell 的启动配置文档，设置 CodeSourcery 编译器路径，执行：

```
ubuntu# gedit /root/.bashrc
```

- 5) 在文档最后添加编译工具绝对路径，需注意由于[HOME]目录设置的名字可能不同，需对应实际而定，例如按照当前路径应添加：

```
export PATH=/home/user/CodeSourcery/Sourcery_G++_Lite/bin:$PATH
```

如图 2-1-6 所示：

```
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile)
# sources /etc/bash.bashrc).
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#   . /etc/bash_completion
#fi

export PATH=/home/user/CodeSourcery/Sourcery_G++_Lite/bin:$PATH
```

图 2-1-6

- 6) 设置完成后保存配置文档，需要重新登录到 Linux 才生效。

2. DVRRDK_04.00.00.03 安装步骤如下：

- 1) 打开命令终端，切换工作目录到 workspace 目录下：

```
ubuntu# cd /home/user/workspace/
```

- 2) 运行 DVRRDK4.0 安装文件，执行：

```
ubuntu# ./DVRRDK_04.00.00.03--Linux-x86-Install.bin
```

- 3) 此后将在该目录下生成 dvrrdk 文件夹，切换到 dvrrdk 目录下，执行：

```
ubuntu# cd dvrrdk
```

- 4) 解压得到 DVRRDK 软件开发包，执行：

```
ubuntu# tar --lzma -xvpf DVRRDK_04.00.00.03.tar.lzma
```

- 5) 剪切解压后得到的文件夹 DVRRDK_04.00.00.03 到[HOME]目录下；

- 6) 切换到文件系统 target 目录下，执行：

```
ubuntu# cd /home/user/DVRRDK_04.00.00.03/target/
```

- 7) 解压文件系统压缩包，执行：

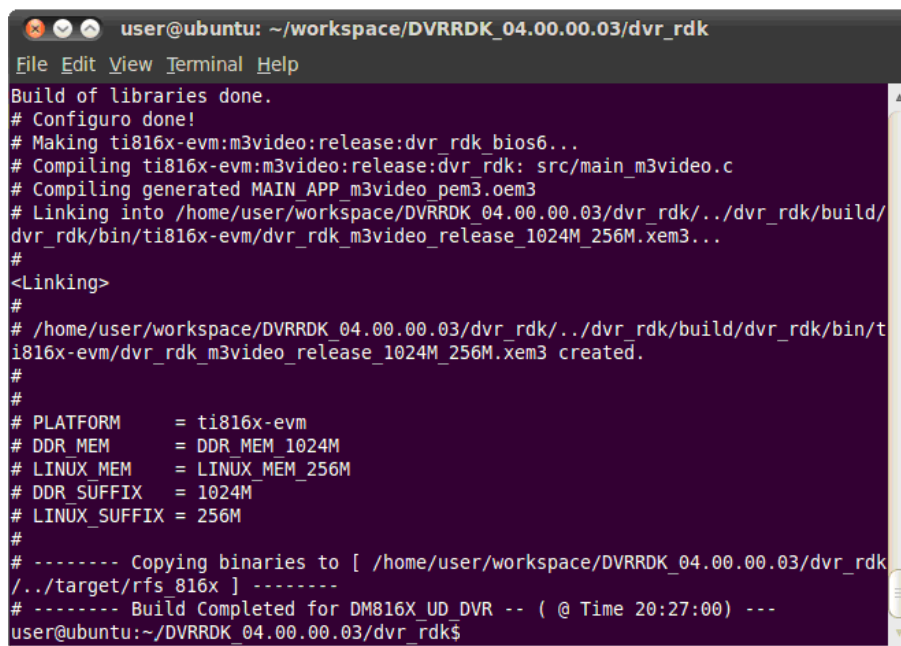
```
ubuntu# tar --lzma -xvpf nfs_DM816x_UD_DVR.tar.lzma
```

- 8) 切换到 dvr_rdk 目录并编译整个包，依次执行：

```
ubuntu# cd /home/user/DVRRDK_04.00.00.03/dvr_rdk/
```

```
ubuntu# make -s sys
```

编译完成后如图 2-1-6 所示：



```
user@ubuntu: ~/workspace/DVRRDK_04.00.00.03/dvr_rdk
File Edit View Terminal Help
Build of libraries done.
# Configur done!
# Making ti816x-evm:m3video:release:dvr_rdk_bios6...
# Compiling ti816x-evm:m3video:release:dvr_rdk: src/main_m3video.c
# Compiling generated MAIN_APP_m3video_pem3.oem3
# Linking into /home/user/workspace/DVRRDK_04.00.00.03/dvr_rdk/./dvr_rdk/build/
dvr_rdk/bin/ti816x-evm/dvr_rdk_m3video_release_1024M_256M.xem3...
#
<Linking>
#
# /home/user/workspace/DVRRDK_04.00.00.03/dvr_rdk/./dvr_rdk/build/dvr_rdk/bin/t
i816x-evm/dvr_rdk_m3video_release_1024M_256M.xem3 created.
#
#
# PLATFORM      = ti816x-evm
# DDR_MEM       = DDR_MEM_1024M
# LINUX_MEM     = LINUX_MEM_256M
# DDR_SUFFIX    = 1024M
# LINUX_SUFFIX  = 256M
#
# ----- Copying binaries to [ /home/user/workspace/DVRRDK_04.00.00.03/dvr_rdk
/./target/rfs_816x ] -----
# ----- Build Completed for DM816X_UD_DVR -- ( @ Time 20:27:00) ---
user@ubuntu:~/DVRRDK_04.00.00.03/dvr_rdk$
```

图 2-1-6

注：编译完成需要 20 分钟左右，请耐心等待；

3. EZSDK_5_05_01_04 安装步骤如下：

- 1) 打开命令终端切换工作目录到 workspace 目录下，执行：

```
ubuntu# cd /home/user/workspace/
```

- 2) 运行 EZSDK 安装程序，执行：

```
ubuntu# ./ezsdk_dm816x-evm_5_05_01_04_setuplinux
```

注：均使用默认配置即可。

- 3) 安装完成后，可在[HOME]目录看到 ti-ezsdk_dm816x-evm_5_05_01_04 文件夹。切换到 filesystem 目录下，执行：

```
ubuntu# cd /home/user/ti-ezsdk_dm816x-evm_5_05_01_04/filesystem
```

- 4) 创建 NFS 文件系统文件目录，执行：

```
ubuntu# mkdir ezsdk-dm816x-evm-rootfs
```

- 5) 切换到 ezsdk-dm816x-evm-rootfs 目录下，执行：

```
ubuntu# cd ezsdk-dm816x-evm-rootfs
```

- 6) 解压文件系统压缩包到当前文件系统目录下，执行：

```
ubuntu# tar -zxvf ../ezsdk-dm816x-evm-rootfs.tar.gz
```

- 7) 切换到 EZSDK 根目录下，并编译整个包，执行：

```
ubuntu# cd /home/user/ti-ezsdk_dm816x-evm_5_05_01_04/
```

```
ubuntu# make -s all
```

注：编译完成需要时间较长，请耐心等待；

2.1.4 NFS 服务器安装

Linux 系统的文件系统可以通过网络挂载，需要在服务端运行网络文件系统 (NFS-NetWork FileSystem) 服务器。NFS 服务器的安装使用在线安装方式，在安装前保证网络通畅，安装步骤如下：

1. 打开命令终端，安装 NFS 服务器，执行：

```
ubuntu# apt-get install nfs-kernel-server
```

2. 配置 NFS 文件夹路径，进入配置编辑界面，执行：

```
ubuntu# gedit /etc/exports
```

3. 然后在最后添加 NFS 文件系统文件夹路径及参数，如下：

```
#DVERRDK:
```

```
/home/user/DVERRDK_04.00.00.03/target/rfs_816x
```

```
*(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
```

```
#EZSDK:
```

```
/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/filesystem/ezsdk-dm816x-evm
```

```
-rootfs *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
```

注：文件路径与后面的参数之间只有一个空格。

添加完成后，如图 2-1-7 所示。

```
# Example for NFSv4:
# /srv/nfs4      gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)
#
/home/user/DVERRDK_04.00.00.03/target/rfs_816x/ *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/filesystem/ezsdk-dm816x-evm-rootfs *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
```

图 2-1-7

4. 保存、退出后，重启 NFS Server，执行：

```
ubuntu# /etc/init.d/nfs-kernel-server start
```

5. 系统启动后，查看 NFS Server 的各个文件系统路径，执行：

```
ubuntu# showmount -e localhost
```

如图 2-1-8 所示：。

```
Export list for localhost:
/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/filesystem/ezsdk-dm816x-evm-rootfs *
/home/user/DVERRDK_04.00.00.03/target/rfs_816x *
root@ubuntu:/home/user#
```

图 2-1-8

2.1.5 TFTP 服务器安装

TFTP 服务器可实现 Linux 开发环境与目标板通过网络方式传输文件，操作方便、快捷。TFTP 服务器的安装配置步骤，如下：

1. 打开命令终端，安装 TFTP 服务器组件，在终端中执行：

```
ubuntu# apt-get install tftpd-hpa tftp-hpa
```

2. 安装完成后，进入 TFTP 配置编辑界面，执行：

```
ubuntu# gedit /etc/default/tftpd-hpa
```

3. 修改[TFTP_DIRECTORY]后的值为 TFTP 服务器文件夹的路径，[TFTP_OPTIONS]的值为“-l -c -s”，如图 2-1-9 所示：

```
# /etc/default/tftpd-hpa
```

```
TFTP_USERNAME="tftp"
```

```
TFTP_DIRECTORY="/home/user/DVERRDK_04.00.00.03/tftphome/"
```

```
TFTP_ADDRESS="0.0.0.0:69"
```

```
TFTP_OPTIONS="-l -c -s"
```

图 2-1-9

4. 保存、退出后，重启 TFTP 服务器，执行：

- ```
ubuntu# service tftpd-hpa restart
```
5. 查看 TFTP 服务器状态，执行：
- ```
ubuntu# service tftpd-hpa status
```
- 如图 2-1-10 所示：
- ```
user@ubuntu:~$ service tftpd-hpa status
tftpd-hpa start/running, process 916
```

图 2-1-10

## 2.1.6 QT embedded 安装

在应用开发 Qt 时，使用到的组件、库等需要事先编译和安装，具体步骤如下：

1. 拷贝《experiment》目录下《QT》文件夹中 qt-everywhere-opensource-src-4.6.4.tar.gz 到虚拟机中，例如[HOME]目录下；
2. 切换工作目录到该文件目录下，执行：

```
ubuntu# cd /home/user/
```
3. 解压源码包，执行：

```
ubuntu# tar xzf qt-everywhere-opensource-src-4.6.4.tar.gz
```
4. 解压《QT》文件夹中的 dvrapp.rar，并将<dvrapp>/dvrgui/linux-TIarmv7-g++ 文件夹拷贝到虚拟机中<qt-everywhere-opensource-src-4.6.4>/mkspecs/qws/ 目录下；
5. 拷贝<dvrapp>/dvrgui/linux-TIarmv7-g++/qmake.conf 到虚拟机如下目录下：  
<qt-everywhere-opensource-src-4.6.4>/mkspecs/qws/
6. 切换到 qt-everywhere-opensource-src-4.6.4 目录下，执行：

```
ubuntu# cd /home/user/qt-everywhere-opensource-src-4.6.4
```
7. 配置 QT 安装包，检查当前的环境是否满足要安装软件的依赖关系，并生成 makefile 文件，执行：

```
ubuntu# ./configure -embedded arm -platform qws/linux-x86-g++ -xplatform qws/linux-TIarmv7-g++ -depths 16,24,32 -little-endian -no-mmx -no-3dnow -no-sse -no-sse2 -no-glib -no-cups -no-largefile -no-accessibility -no-openssl -no-gtkstyle -qt-freetype -qt-mouse-pc -qt-mouse-linux -plugin-mouse-linux -plugin-mouse-pc -fast -qt-gfx-transformed -opensource
```
8. 配置完成后，编译、安装源工具码包，依次执行：

```
ubuntu# make
ubuntu# make install
```

注：编译时间较长，请耐心等待。
9. 添加 qmake 工具及编译支持库等环境变量，进入系统配置文件编辑界面，执行：

```
ubuntu# gedit /root/.bashrc
```

在配置文件最后添加如下配置信息：

```
export PATH=/home/user/qt-everywhere-opensource-src-4.6.4/bin:ubuntu#PATH
export QTDIR=/home/user/qt-everywhere-opensource-src-4.6.4
export QTLIB=/home/user/qt-everywhere-opensource-src-4.6.4/lib
export QTINC=/home/user/qt-everywhere-opensource-src-4.6.4/include
```

- 注：其中的路径需要按照实际安装位置进行修改。
10. 添加后保存、退出，重新登录后才生效。

## 2.2 制作镜像文件

### 2.2.1 制作 uboot 镜像文件

uboot 启动可以从 SD 卡、NandFlash、NorFlash、SPI 和 EMAC 启动，分为一级启动和两级启动方式（在 SD 卡中使用两级启动方式）。因此需要根据实际需求进行不同的配置，生成所需的 uboot 镜像文件。

EZSDK 和 DVRRDK 两种开发环境都包含 uboot 源码包，使用相同的编译方式。编译可以使用完全编译和增量编译两种方式，增量编译方式省略下面步骤 2 和 3。增量编译仅使用于源文件稍作修改之后再重新编译，且在修改之前已经进行过一次完全编译。

uboot 完全编译步骤如下：

1. 进入 uboot 源码目录下，执行：

EZSDK：

```
ubuntu# cd
/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/board-support/u-boot-2010.06-
psp04.04.00.01/
```

DVRRDK：

```
ubuntu# cd
/home/user/DVRRDK_04.00.00.03/ti_tools/linux_lsp/uboot/u-boot-dvr-rdk/
```

2. 清除生成的.o 文件、配置文件和镜像文件，执行：

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
distclean
```

3. 选择配置参数并写入文件，面命令行中的 xxx 在 EZSDK 环境下为：evm，DVRRDK 环境下为 dvr：

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_xxx_config_nand
```

用于烧写到 NandFlash，生成文件名为：u-boot.noxip.bin

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_xxx_config_nor
```

用于烧写到 NorFlash，生成文件名为：u-boot.bin

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_xxx_config_spi
```

用于烧写到 SPIROM，生成文件名为：u-boot.noxip.bin.spi

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_xxx_config_quick_mmc_min
```

用于 SD 卡中第一级启动 uboot，生成文件名为：u-boot.min.sd



```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_XXX_config_sd
```

用于 SD 卡中第二级启动 uboot，生成文件名为：u-boot.bin

4. 编译 uboot 在当前目录下生成对应的镜像文件，执行命令：

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
u-boot.ti
```

## 2.2.2 制作内核镜像文件

内核编译有两种默认配置：ti8168\_dvr\_defconfig 和 ti8168\_evm\_defconfig，在 EZSDK 环境下使用 ti8168\_evm\_defconfig 配置方式，在 DVRRDK 环境下 ti8168\_evm\_defconfig 配置方式。

EZSDK 和 DVRRDK 两种开发环境都包含内核源码包，使用相同的编译方式。编译可以使用完全编译和增量编译两种方式，增量编译方式省略下面步骤 2 和 3。增量编译仅使用于源文件的修改和内核配置选项修改之后的重新编译，且在修改之前已经进行过一次完全编译。内核完全编译步骤如下：

1. 进入内核目录，执行：

EZSDK：

```
ubuntu# cd
/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/board-support/linux-2.6.37-psp
04.04.00.01/
```

DVRRDK：

```
ubuntu# cd ~/DVRRDK_04.00.00.03/ti_tools/linux_lsp/kernel/linux-dvr-rdk/
```

2. 清除生成的.o 文件、配置文件和镜像文件，执行：

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
distclean
```

3. 选择配置参数并写入文件，执行：

EZSDK：

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_evm_defconfig
```

DVRRDK：

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_dvr_defconfig
```

4. 进入内核图形配置界面，使能或者关闭驱动、协议等选项，执行：

```
ubuntu# make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
menuconfig
```

5. 编译内核，执行：

```
ubuntu# make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
uImage
```

6. 编译完成后在内核目录下的 arch/arm/boot 目录下生成的二进制文件 uImage，即是我们需要的内核镜像文件。

## 2.2.3 制作 UBI 文件系统

UBI 文件系统是将 DVRRDK 环境下的文件系统压缩后得到, 主要用于烧写到 NandFlash 中。第一次制作时需稍作配置, 此后可以直接执行第 3、4 步即可。制作步骤如下:

1. 拷贝《ubi\_fs\_tool》目录下的文件到<DVRRDK\_04.00.00.03>/ti\_tools/mtd\_utils/sbin 目录下替换以前的文件。
2. 安装 liblzo2 库的驱动, 执行:  
`ubuntu# apt-get install liblzo2-dev`
3. 切换工作目录带 dvr\_rdk 下, 执行:  
`ubuntu# cd /home/user/DVRRDK_04.00.00.03/dvr_rdk`
4. 生成 UBI 文件系统镜像文件, 执行:  
`ubuntu# make ubifs_128`

等待制作完成后, 生成的 ubi\_128\_DM816X\_UD\_DVR.img 即是我们需要的文件系统镜像, 用于烧写到 NandFlash 中。该文件会被自动拷贝到 <DVRRDK\_04.00.00.03>/tftphone 目录下。

## 2.3 DVRRDK 镜像烧写

DVRRDK 开发环境下的各个镜像制作完成后, 可烧写到 NandFlash 中。烧写完成后, 配置环境变量, 并且设置 DES8168-EVM 板的拨码开关为 NandFlash 启动方式。此后, 目标板即可从 NandFlash 启动 DVRRDK 系统。

### 2.3.1 uboot 启动测试

uboot 可以使用 CCS 软件载入到 DM8168 片上运行, 以此对其进行测试。uboot 镜像文件制作步骤参照 [2.2.1](#) 章节。uboot 加载、测试步骤如下:

1. 将 Linux 中编译生成的 u-boot.noxip.bin 拷贝到 Windows 目录下(注意不要包含中文路径);
2. 连接目标板电源、串口、仿真器, 并在 PC 机上打开串口终端, 设置波特率为 115200;
3. 打开目标板电源, 若串口终端中看到 uboot 的打印信息, 在其读秒时立即则按下回车键停止其继续启动;
4. 启动 CCS 软件, 使用仿真器连接到 Cortex-A8 核, 选择工具栏[Tools]→[Load Memory]选项选定 u-boot.noxip.bin 载入到地址 0x40400000。其中 Type-size 设置应设置为 32bit;
5. 选择工具栏[View]→[disassembly]选项, 打开汇编窗口。在该窗口中键入 0x40400008 并按下回车跳到该地址, 如图 2-3-1 所示:

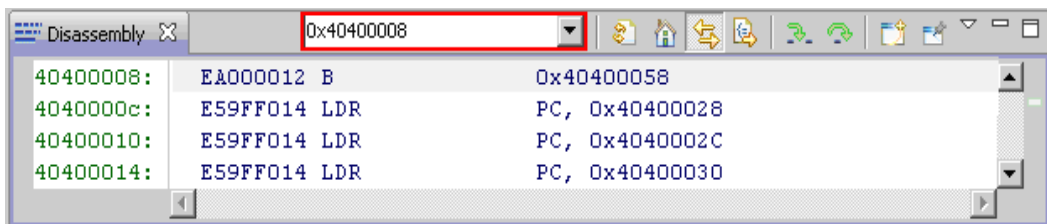


图 2-3-1

6. 使用鼠标左键点击该行，将光标移到此处。然后右键选择[Move To Line]，使程序指针移到这一行；
7. 按下 F8 运行程序，观察串口终端中是否有打印信息。正常启动后可看到 uboot 打印信息中的眼图，如图 2-3-2 所示：

```
I2C: ready
DRAM: 1 GiB
NAND: HW ECC BCH8 Selected
256 MiB
MMC: OMAP SD/MMC: 0
Net: Detected MACID:4:e4:51:60:6a:84
Ethernet PHY: GENERIC @ 0x01
Davinci EMAC
Hit any key to stop autoboot: 0
TI8168_EVM#
```

图 2-3-2

## 2.3.2 烧写 uboot 到 NandFlash

uboot 测试无误后就可以烧写到 NandFlash 中，调整拨码开关设置启动方式后，目标板就可以从 NandFlash 读入 uboot 自行启动。其中，uboot 镜像文件的制作参照 2.2.1 章节，uboot 测试参照 2.3.1 章节。

uboot 二进制文件可使用《Flashburn》目录下的预编译文件。该目录下还存放了烧写 uboot 使用到的可执行二进制文件。烧写 uboot 到 NandFlash 步骤如下：

1. 将待烧写的 u-boot.noxip.bin 拷贝到 Windows 任意目录下（注意不能有中文路径）；
2. 连接目标板电源、串口、仿真器，并在 PC 机上打开串口终端，设置波特率为 115200；
3. 打开目标板电源，若串口终端中看到 uboot 的打印信息，在其读秒时立即按下回车键停止其继续启动；
4. 启动 CCS 软件，使用仿真器连接到 Cortex-A8 核，选择工具栏[Run]→[Load]→[Load Program]选项，载入《Flashburn》文件夹中 nand-flash-writer.out 文件（注意不能有中文路径）；
5. 按下 F8 运行，即可在 Console 控制台窗口中看到提示输入，如图 2-3-3 所示：

```

[CortexA8] Choose your operation
[CortexA8] Enter 1 ---> To Flash an Image
[CortexA8] Enter 2 ---> To ERASE the whole NAND
[CortexA8] Enter 3 ---> To DDR Simple test
[CortexA8] Enter 4 ---> To EXIT

```

图 2-3-3

6. 首先选择 2，并按下回车键，擦除整个 NAND；
7. 等待擦除完成后，选择工具栏[Run]→[Restart]，从新回到程序入口。再次按下 F8 运行，选择 1 烧写镜像，并按提示输入路径 uboot 文件路径，例如当前路径为：D:\DSPresearch\Flashburn\u-boot.noxip.bin；
8. 等待烧写完成，在 Consol 控制台中打印信息如图 2-3-4 所示：

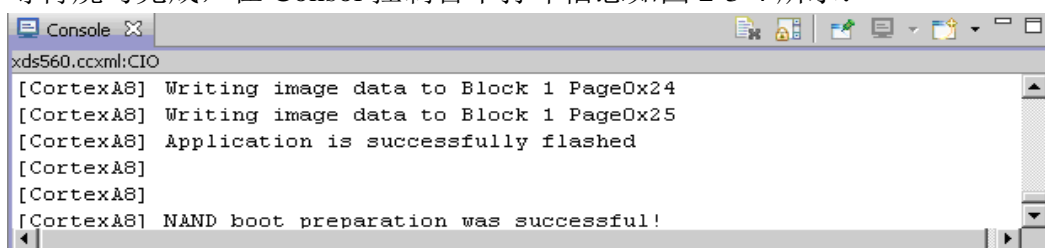


图 2-3-4

9. 设置拨码开关 1~8 号依次为 01000001（0 表示：OFF/1 表示：ON），重新启动 EVM 板，如果烧写成功就可以看见串口终端中 uboot 打印的信息。

### 2.3.3 烧写内核及文件系统镜像到 NandFlash

NandFlash 用于存放 DVRRDK 开发环境下的，烧写方式使用 TFTP 方式。使用 TFTP 方式烧写内核和文件系统必须有 uboot 的支持，即需要首先烧写 uboot 到 NandFlash，并从 NAND 启动 uboot。其中，uboot 的烧写参照 [2.3.2](#) 章节。烧写使用到的内核和文件系统镜像文件的制作分别参照章节 [2.2.2](#) 和章节 [2.2.3](#)。

烧写各镜像到 NandFlash 步骤如下：

1. 拷贝 Linux 中编译生成的内核镜像和文件系统镜像到 [2.1.3](#) 章节安装的 TFTP 服务器目录下。可使用《Prebuilt》文件夹中的《DVR》文件夹下的内核和文件系统镜像。

**注：**现在假定内核镜像文件名为：uImage\_DM816X\_UD\_DVR，

文件系统镜像名为：ubi\_128\_DM816X\_UD\_DVR.img；

2. 连接 EVM 板的串口和网口至 PC 机，打开 PC 机串口终端，设置波特率为 115200；
3. 启动 EVM 板，若看到串口终端中 uboot 打印的读秒信息时立即按下回车键；
4. 设置服务器 IP：192.168.1.41，本机 IP：192.168.1.21。注意服务器 IP 要与 [2.1.1](#) 章节设置 ubuntu 的 IP 地址对应。在串口终端中分别执行：

```
target# setenv serverip 192.168.1.41
```

```
target# setenv ipaddr '192.168.1.21'
```

5. 烧写内核镜像到 NAND 中，需注意目前烧写的内核镜像名称为：

uImage\_DM816X\_UD\_DVR。执行：

```
target# mw.b 0x81000000 0xFF 0x300000;tftp 0x81000000
```

```
uImage_DM816X_UD_DVR; nand erase 0x00580000 0x440000; nand
write.i 0x81000000 0x00580000 0x300000
```

6. 烧写 ubi 文件系统镜像到 NAND 中，需注意目前烧写的文件系统镜像名称为：ubi\_128\_DM816X\_UD\_DVR.img。执行：

```
target# mw.b 0x81000000 0xFF 0x4C00000;tftp 0x81000000
ubi_128_DM816X_UD_DVR.img;nand erase 0x009c0000 0xC820000;nand
write 0x81000000 0x009c0000 0x4C00000
```

文件系统烧写时间较长，请耐心等待烧写完成。

## 2.3.4 NAND 启动 DVRRDK 系统

烧写完 uboot、内核、文件系统后，还需要设置启动时使用到的环境变量，在串口终端中执行：

```
target# setenv bootargs 'console=ttyO2,115200n8 noinitrd
ip=192.168.1.21:192.168.1.43:192.168.1.43:255.255.255.0::eth0:off
mem=256M rootwait=1 rw ubi.mtd=4,2048 rootfstype=ubifs root=ubi0:rootfs
init=/init vram=20M notifyk.vpssm3_sva=0xBEE00000 stdin=serial
ddr_mem=1024M'
```

```
target# setenv bootcmd 'nand read 0x81000000 0x00580000 0x300000; bootm
0x81000000'
```

保存环境变量，执行：

```
target# saveenv
```

设置 DES818-EVM 板拨码开关 1~8 号依次为 01000001 (0 表示：OFF/1 表示：ON)。目标板上电，启动完成后输入：root，即可登陆到系统，如图 2-3-5 所示：



```
Arago Project http://arago-project.org dm816x ttyO2

Arago 2012.10 dm816x ttyO2

dm816x login: root
```

图 2-3-5

## 2.4 EZSDK 镜像烧写

EZSDK 开发环境下的各个镜像制作完成后，可以存放到 SD 卡中。拷贝完成后，添加环境变量配置脚本文件，并且设置 DES8168-EVM 板的拨码开关为 SD 卡启动方式。此后，目标板即可从 SD 卡启动 EZSDK 系统。

## 2.4.1 SD 卡制作

目标板从 SD 卡启动 Linux 系统，需要使用两级 uboot 来启动内核和文件系统。因此，我们需要重新编译 uboot 源码包得到第一级启动使用到的 uboot 文件。对于 SD 卡需要重新进行分区，在对应分区放入 uboot、uImage 和文件系统等文件。这些过程都在 Linux 下进行，步骤如下：

1. 切换工作目录到 EZSDK 的 uboot 源码目录下，在 Linux 终端下执行：

```
ubuntu# cd
/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/board-support/u-boot-2010.06-
psp04.04.00.01
```

2. 编译 uboot 源码包，得到第一级启动使用到的 uboot 文件。

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
distclean
```

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_evm_config_quick_mmc_min
```

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
u-boot.ti
```

3. 编译后，重命名当前路径下生成的文件 u-boot.min.sd 为 MLO；

4. 编译第二级 uboot，执行：

```
ubuntu# make CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
ti8168_evm_config_sd
```

5. 重命名当前目录下的 u-boot.noexec.bin 为：u-boot.bin。

6. 参照章节 [2.2.2](#)，在 EZSDK 环境下编译得到 uImage 内核镜像文件。若已有可用内核，则跳过此步骤；

7. 文件系统使用 EZSDK 环境提供的文件系统，位于虚拟机：

```
<ti-ezsdk_dm816x-evm_5_05_01_04>/filesystem/ezsdk-dm816x-evm-rootfs.tar
.gz
```

8. 在 ubuntu 系统下的 workspace 目录新建一个文件夹，命名为 ti816x。拷贝上述文件到此文件夹下，包括 MLO、u-boot.bin、uImage、ezsdk-dm816x-evm-rootfs-tar.gz 和 mksd-ti816x.sh。

注：mksd-ti816x.sh 位于

```
<ti-ezsdk_dm816x-evm_5_05_01_04>/board-support/host-tools 目录下；
```

9. 使用读卡器将 SD 卡连接到 PC 机上，鼠标点击 ubuntu 系统桌面右下角 [Realtek USB2.0-CRW] 图标查看 SD 卡是否连接到 Linux 下，如图 2-4-1 所示，目前已连接到 Linux 下。若已经连接到 Windows 主机，点击第一个选项将 SD 卡从主机重新连接到 Linux；

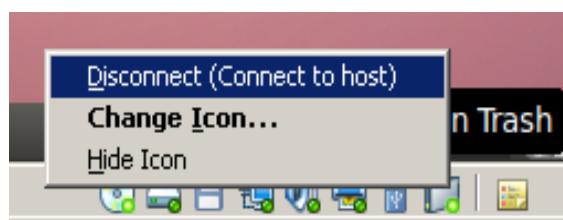


图 2-4-1

10. 查看设备点，一般为/dev/sdb，执行：

```
ubuntu# fdisk -l
```

如图 2-4-2 所示：

```
Disk /dev/sdb: 7948 MB, 7948206080 bytes
255 heads, 63 sectors/track, 966 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

 Device Boot Start End Blocks Id System
/dev/sdb1 * 1 9 72261 c W95 FAT32 (LBA)
/dev/sdb2 10 966 7687102+ 83 Linux
```

图 2-4-2

11. 查看当前设备挂载情况，执行：

```
ubuntu# df
```

如图 2-4-3 所示。

```
user@ubuntu:~$ df
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/sda1 81099528 23983004 52996872 32% /
none 250364 316 250048 1% /dev
none 254608 264 254344 1% /dev/shm
none 254608 104 254504 1% /var/run
none 254608 0 254608 0% /var/lock
none 254608 0 254608 0% /lib/init/rw
/dev/sdb1 71133 2800 68333 4% /media/boot
```

图 2-4-3

注：其中最后一项可见 SD 卡已经自动挂载到 Linux 系统了，需将其卸载，如果没有此项则跳过。在终端中执行：

```
ubuntu# umount /dev/sdb1
```

12. 准备工作完成后，分区 SD 卡并拷贝文件，执行：

```
ubuntu# cd /home/user/workspace/ti816x/
```

```
ubuntu# ./mksd-ti816x.sh /dev/sdb MLO u-boot.bin uImage
```

```
ezsdk-dm816x-evm-rootfs.tar.gz
```

注：拷贝时间较长，请耐心等待。等待文件拷贝完成后，SD 卡分区已经制作完成，包括 root 和 rootfs 两个分区，并放入了启动时所需的各个文件。其中在 root 分区中存放 MLO,u-boot.bin 和 uImage,而 rootfs 中为 EZSDK 文件系统。

### 2.4.2 制作 SD 卡启动脚本

目标板从 SD 卡启动过程中需要配置环境变量，SD 卡中的环境变量是以脚本文件的形式完成的，制作脚本文件步骤如下：

1. 在<ti-ezsdm816x-evm\_5\_05\_01\_04>/board-support/u-boot-2010.06-psp04.04.00.01/tools 目录下创建一个空白文件，命名为 boot.txt。

2. 打开 boot.txt 将如下环境变量复制到该文件中并保存；
 

```
setenv bootargs 'notifyk.vpssm3_sva=0xBF900000 vram=50M
ti816xfb.vram=0:16M,1:16M,2:6M console=ttyO2,115200n8
root=/dev/mmcbk0p2 mem=128M rootwait'
setenv bootcmd 'mmc rescan 0; fatload mmc 0 0x81000000 uImage; bootm
0x81000000'
boot
```
3. 切换工作目录到该目录下，在终端中执行：
 

```
ubuntu# cd
/home/user/ti-ezsdm816x-evm_5_05_01_04/board-support/u-boot-2010.06-
psp04.04.00.01/tools
```
4. 生成脚本文件 boot.scr，执行：
 

```
ubuntu# ./mkimage -A arm -O linux -T script -C none -n TI_script -d boot.txt
boot.scr
```
5. 使用读卡器将 boot.scr 拷贝到 SD 卡的第一个 boot 分区。

### 2.4.3 SD 卡启动 EZSDK 系统

SD 卡中写入 uboot、内核、文件系统和环境变量脚本后，还需对 EVM 板的启动方式进行配置。设置码开关 1~8 号依次为 01110001 (0 表示：OFF/1 表示：ON)。设置完成后，插入 SD 卡，启动目标板板，等待启动完成后输入：root，登陆到系统。启动完成后终端中可见如下打印信息，如图 2-4-4 所示：

```
Starting system message bus: dbus.
Starting telnet daemon.
Starting syslogd/klogd: done
Starting tthtpd.
Starting PVR
Starting Matrix GUI application.
```



```
Arago Project http://arago-project.org dm816x-evm ttyO2
Arago 2011.09 dm816x-evm ttyO2
dm816x-evm login:root
root@dm816x-evm:~#
```

图 2-4-4

## 2.5 搭建 NFS 网络文件系统

调试程序时，文件系统可以通过网络挂载而并不需要烧写到 NAND 或者 SD 卡中，使得在修改文件系统中文件时更加容易操作。搭建文件系统前首先拷贝 windows 下《NFS》文件夹到 D:\DSPresearch 目录下备用，该文件夹下存放 DES8168-EVM 板启动时使用的 uboot 和 EZSDK、DVRRDK 两种环境下的内核镜像文件。

网络文件系统的搭建需要先安装 NFS 服务器，并且配置文件系统文件夹的



路径，若未完成上述步骤则需按照 [2.1.3](#) 章节“NFS 服务器安装”安装、配置 NFS 服务器。

## 2.5.1 搭建 EZSDK 环境网络文件系统

### 1. 启动 NFS 服务器：

打开 Vm 虚拟机进入到 ubuntu 系统，通常状况下 NFS 服务器会自行启动。查看 nfs 开启状态，在 Linux 终端执行：

```
ubuntu# service nfs-kernel-server status
```

执行后，如图 2-5-1 所示。

```
user@ubuntu:~$ service nfs-kernel-server status
nfsd running
```

图 2-5-1

如未启动，通过命令启动 NFS 服务器，执行：

```
ubuntu# nfs-kernel-server start
```

查看 NFS 服务器是否配置了 EZSDK 文件系统路径，在 Linux 终端中执

```
ubuntu# showmount -e localhost
```

执行后，如图 2-5-2 所示，可见配置的 EZSDK 文件系统路径；

```
user@ubuntu:~$ showmount -e localhost
Export list for localhost:
/home/user/ti-ezsdk_dm816x-evm 5 05 01 04/filesystem/ezsdk-dm816x-evm-rootfs *
/home/user/DVRRDK_04.00.00.03/target/rfs_816x/ *
```

图 2-5-2

### 2. 启动目标板：

- 1) 连接硬件，包括：网线，串口 2，电源；
- 2) 启动串口终端 SecureCRT.exe，设置波特率为 115200；
- 3) 开启 EVM 板电源，若在串口终端中看到 uboot 打印信息，在其提示读秒时按下回车键停止其继续启动。若未看到打印信息，则需要按照 [2.3.1](#) 章节通过 CCS 载入 u-boot 并运行。等到读秒提示时按下回车键停止其继续启动。

### 3. 载入内核、挂载文件系统：

- 1) 若内核已经烧写内核到 NAND，只需要从 NandFlash 读入 DDR，在串口终端执行：

```
target# nand read 0x81000000 0x00580000 0x400000
```

- 2) 若已经烧写内核到 SD 卡，在串口终端执行：

```
target# mmc rescan 0
```

```
target# fatload mmc 0 0x81000000 uImage
```

- 3) 若未烧写内核镜像，则需要使用 CCS 软件通过仿真器将 uImage 载到 DSP 片内地址 0x81000000，uImage 文件为 D:\DSPresearch\NFS 文件夹下 uImage\_ez；

### 4. 在串口终端中设置环境变量，注意确保 NFS 服务器配置的文件系统路径和当前系统配置一致，执行：

```
target# setenv bootargs 'console=ttyO2,115200n8 rootwait root=/dev/nfs rw
```

```
rootftype=jffs2 mem=364M@0x80000000 vram=50M
nfsroot=192.168.1.41:/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/filesyste
m/ezsdk-dm816x-evm-rootfs
ip=192.168.1.21:192.168.1.43:192.168.1.43:255.255.255.0::eth0:off
notifyk.vpssm3_sva=0xBF900000 earlyprintk'
```

注：虚拟机中服务器的 ip 地址为 192.168.1.41，windows 主机的 ip 地址为 192.168.1.43，目标板的 ip 地址为 192.168.1.21。

5. 启动内核，执行：

```
target# bootm 0x81000000
```

6. 等待启动完毕后，在串口终端中输入:root，登录到系统。

## 2.5.2 搭建 DVRRDK 环境网络文件系统

1. 启动 NFS 服务器：

打开 Vm 虚拟机进入到 ubuntu 系统，通常状况下 NFS 服务器会自行启动。

查看 nfs 开启状态，在 Linux 终端执行：

```
ubuntu# service nfs-kernel-server status
```

执行后，如图 2-5-4 所示。

```
user@ubuntu:~$ service nfs-kernel-server status
nfsd running
```

图 2-5-4

如未启动，通过命令启动 NFS 服务器，执行：

```
ubuntu# nfs-kernel-server start
```

查看 NFS 服务器是否配置了 DVRRDK 文件系统路径，在 Linux 终端中执行：

```
ubuntu# showmount -e localhost,
```

执行后，如图 2-5-2 所示，可见配置的 DVRRDK 文件系统路径：

```
user@ubuntu:~$ showmount -e localhost
Export list for localhost:
/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/filesystem/ezsdk-dm816x-evm-rootfs *
/home/user/DVRRDK_04.00.00.03/target/rfs_816x/ *
```

图 2-5-5

2. 启动目标板：

- 1) 连接硬件，包括：网线，串口 2，电源；
- 2) 启动串口终端 SecureCRT.exe，设置波特率为 115200；
- 3) 开启 EVM 板电源，若在串口终端中看到 uboot 打印信息，在其提示读秒时按下回车键停止其继续启动。若未看到打印信息，则需要按照 [2.3.1](#) 章节通过 CCS 载入 u-boot 并运行。等到读秒提示时按下回车键停止其继续启动。

3. 载入内核、挂载文件系统：

- 1) 若内核已经烧写内核到 NAND，只需要从 NandFlash 读入 DDR，在串口终端执行：

```
target# nand read 0x81000000 0x00580000 0x400000
```

- 2) 若已经烧写内核到 SD 卡，在串口终端执行：

```
target# mmc rescan 0
```

```
target# fatload mmc 0 0x81000000 uImage
```

- 3) 若未烧写内核镜像，则使用 CCS 软件通过仿真器将 uImage 载到 DSP 片内地址 0x81000000，uImage 文件为 D:\DSPresearch\NFS 文件夹下 uImage\_dvr;

- 4) 在串口终端中设置环境变量，注意确保 NFS 服务器配置的文件系统路径和当前系统配置一致，执行：

```
target# setenv bootargs 'mem=256M console=ttyO2,115200n8
```

```
root=/dev/nfs rw rootwait=1
```

```
nfsroot=192.168.1.41:/home/usr/DVRRDK_04.00.00.03/target/rfs_816x
```

```
ip=192.168.1.21:192.168.1.43:192.168.1.43:255.255.255.0::eth0:off
```

```
vram=20M notifyk.vpssm3_sva=0xBEE00000 ddr_mem=1024M'
```

4. 启动内核，执行：

```
target# bootm 0x81000000
```

5. 等待启动完毕后，在串口终端中输入:root，登录到系统。

## 2.6 模块测试（Linux 部分）

### 2.6.1 测试 HDMI 接口

1. 连接 DES-8168EVM 板的 HDMI 接口到高清显示器；
2. 按照 [2.5.1](#) 章节搭建 EZSDK 网络文件系统或者从 SD 卡启动 EZSDK-Linux 系统；
3. 等待系统启动后，即可在显示器上看到 LinuxGUI 界面,如图 2-6-1 所示：



图 2-6-1

### 2.6.2 测试 USB 接口（2 个）

1. 连接 DES-8168EVM 板的 HDMI 接口到高清显示器；
2. 按照 [2.5.1](#) 章节搭建 EZSDK 网络文件系统或者从 SD 卡启动 EZSDK-Linux 系统；
3. 待系统启动后，插入鼠标即可移动显示屏幕中的光标；
4. 插入 U 盘，在终端中查看磁盘分区情况，执行：

target# df

执行后即可看到所有磁盘分区及挂载情况，如图 2-6-2 所示。其中的/dev/sda4 即为 usb 存储设备，/media/sda4 为挂载目录。设备名称和挂载目录为自动分配，每次可能有所不同。

```
root@dm816x-evm:~# df
Filesystem 1k-blocks Used Available Use% Mounted on
192.168.1.41:/home/user/ti-ezsdk_dm816x-evm_5_05_01_04/filesystem/ezsdk-dm816x-evm-rootfs
81099584 73904512 3075456 96% /
devtmpfs 1024 52 972 5% /dev
tmpfs 40 0 40 0% /mnt/.splash
none 1024 52 972 5% /dev
/dev/sda4 15221888 16 15221872 0% /media/sda4
tmpfs 16384 128 16256 1% /var/volatile
tmpfs 156472 0 156472 0% /dev/shm
tmpfs 16384 0 16384 0% /media/ram
```

图 2-6-2

5. 切换到 U 盘挂载目录下，执行：

target# cd /media/sda4

6. 创建名为 test 的文件夹，执行：

target# mkdir test

7. 列出目录内容，执行：

target# ls

执行后即可看到当前目录下的目录和文件列表，如图 2-6-3 所示。其中的 test 即为新建的文件目录。

```
root@dm816x-evm:~# cd /media/sda4/
root@dm816x-evm:/media/sda4# mkdir test
root@dm816x-evm:/media/sda4# ls
test
root@dm816x-evm:/media/sda4#
```

图 2-6-3

8. 删除创建目录，执行：

target# rmdir test

### 2.6.3 测试 SATA 硬盘接口

1. DES-8168EVM 板的 SATA 接口插入 SATA 盘，并供电；
2. 按照 [2.5.1](#) 或者章节搭建 EZSDK 网络文件系统或者从 SD 卡启动 EZSDK-Linux 系统；
3. 在终端中查看磁盘分区情况，执行：

target# df

执行后即可看到所有磁盘分区及挂载情况，如图 2-6-4 所示。其中的/dev/sda4 即为硬盘存储设备，/media/sda4 为挂载目录。设备名称和挂载目录为自动分配，

每次可能有所不同。

```
root@dm816x-evm:~# df
Filesystem 1k-blocks Used Available Use% Mounted on
192.168.1.41:/home/user/ti-ezsdm816x-evm_5_05_01_04/filesystem/ezsdk-dm816x-evm-rootfs
81099584 73904512 3075456 96% /
devtmpfs 1024 52 972 5% /dev
tmpfs 40 0 40 0% /mnt/.splash
none 1024 52 972 5% /dev
/dev/sda4 15221888 16 15221872 0% /media/sda4
tmpfs 16384 128 16256 1% /var/volatile
tmpfs 156472 0 156472 0% /dev/shm
tmpfs 16384 0 16384 0% /media/ram
```

图 2-6-4

9. 切换到 SATA 盘挂载目录下，执行：

```
target# cd /media/sda4
```

10. 创建名为 test 的文件夹，执行：

```
target# mkdir test
```

11. 列出目录内容，执行：

```
target# ls
```

执行后即可看到当前目录下的目录和文件列表，如图 2-6-5 所示。其中的 test 即为新建的文件目录。

```
root@dm816x-evm:~# cd /media/sda4/
root@dm816x-evm:/media/sda4# mkdir test
root@dm816x-evm:/media/sda4# ls
test
root@dm816x-evm:/media/sda4#
```

图 2-6-5

12. 删除创建目录，执行：

```
target# rmdir test
```

## 2.6.4 测试 MMC 模块

1. DES-8168EVM 板的 SD 接口插入 SD 卡；
2. 按照 [2.5.1](#) 或者章节搭建 EZSDK 网络文件系统或者从 SD 卡启动 EZSDK-Linux 系统；
3. 在终端中查看磁盘分区情况，执行：

```
target# df
```

执行后即可看到所有磁盘分区及挂载情况，如图 2-6-6 所示。其中的 /dev/mmcblk0p1 即为 MMC 存储设备，/media/mmcblk0p1 为挂载目录。设备名称和挂载目录为自动分配，每次可能有所不同。

```
root@dm816x-evm:/# df
Filesystem 1k-blocks Used Available Use% Mounted on
192.168.1.41:/home/user/ti-ezsdm816x-evm_5_05_01_04/filesystem/ezsdk-dm816x-evm-rootfs
81099584 73904512 3075456 96% /
devtmpfs 1024 52 972 5% /dev
tmpfs 40 0 40 0% /mnt/.splash
none 1024 52 972 5% /dev
tmpfs 16384 128 16256 1% /var/volatile
tmpfs 156472 0 156472 0% /dev/shm
tmpfs 16384 0 16384 0% /media/ram
/dev/mmcblk0p1 7744216 4 7744212 0% /media/mmcblk0p1
```

图 2-6-6

4. 切换到 MMC 挂载目录下，执行：

```
target# cd /media/mmcblk0p1
```

5. 创建名为 test 的文件夹，执行：

```
target# mkdir test
```

6. 列出目录内容，执行：

```
target# ls
```

执行后即可看到当前目录下的目录和文件列表，如图 2-6-7 所示。其中的 test 即为新建的文件目录。

```
root@dm816x-evm:/# cd /media/mmcblk0p1
root@dm816x-evm:/media/mmcblk0p1# mkdir test
root@dm816x-evm:/media/mmcblk0p1# ls
test
root@dm816x-evm:/media/mmcblk0p1# █
```

图 2-6-7

7. 删除创建目录，执行：

```
target# rmdir test
```

## 2.6.5 测试音频接口

1. DES-8168EVM 板的 P3 口插入耳机，P1 口插入电脑机箱音频输出；
2. 按照 [2.5.1](#) 或者章节搭建 EZSDK 网络文件系统或者从 SD 卡启动 EZSDK-Linux 系统；
3. 配置混音器的捕获和播放音量，执行：

```
target# amixer cset name='PCM Playback Volume' 100%,100%
```

```
target# amixer cset name='PGA Capture Volume' 10%,10%
```

4. 执行捕获和播放命令，100s 后自动停止；

```
target# arecord -f dat -d 100 -v | aplay -f dat -d 100 -v
```

5. 在 Windows 中播放音频文件，此后在耳机中可听到 PC 机播放的声音。

## 2.6.6 测试 5158 其他 3 个输入视频

1. 连接 DES-8168EVM 板的 HDMI 接口到高清显示器，以及子卡 4 路视频输入(也可以使用 1 路视频输入，测试时分别连接)；
2. 按照 [2.5.2](#) 章节搭建 DVRRDK 网络文件系；
3. 切换到/opt/dvr\_rdk/ti816x 目录下，执行：

```
target# cd /opt/dvr_rdk/ti816x
```

4. 依次执行 init.sh、load.sh、run.sh 三个脚本文件，进入示例选择界面如图 2-6-8 所示：

```
target# ./init.sh; ./load.sh; ./run.sh
```

```
root@dm816x:/opt/dvr_rdk/ti816x# sh run.sh

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

1: VCAP + VENC + VDEC + VDIS - Progressive SD Encode + Decode
2: VCAP + VENC + VDIS - SD Encode ONLY
3: VCAP + VENC + VDEC + VDIS - Progressive HD Encode + Decode
4: VDEC + VDIS - SD/HD Decode ONLY
5: VCAP + VDIS - NO Encode or Decode + 4Channel
a: 960H DVR usecase

e: Exit
```

图 2-6-8

5. 输入 5 选择直通测试，等待执行完成后即可在高清显示器上看到 4 路输入视频画面。

## 3 基于 EZSDK 的应用开发

### 3.1 Matrix-gui

Matrix 是在 EZSDK 环境下提供的应用和示例的一个启动程序，目前有两种形式的 Matrix：一是图形用户界面（GUI）形式，一是文字用户界面（TUI）形式。默认情况下当 EVM 板启动后启动图形界面形式的 Matrix，使用鼠标作为输入。当启动 EZSDK 文件系统后，Matrix-gui 应用程序将会自动运行，使用鼠标进行操作。

Matrix GUI 基于 HTML 和 CSS（层叠式样表）设计，非常容易定制化。在 EZSDK 文件系统目录/usr/share/matrix/html 下可以看到许多 HTML 文件，这些文件包含了所有生成菜单窗口和子窗口的全部信息。所有启动应用程序的信息也包含在上述 HTML 文件中。因此，对于定制自己的 Matrix GUI 和添加新的应用操作相对比较容易。

#### 3.1.1 运行 Matrix

EZSDK 系统启动后在默认配置下会自行启动 matrix 示例。在启动 EVM 板时设置拨码开关为 0111 0001，进入 SD 卡启动方式，另外还需要连接 HDMI 接口至高清显示器。等待目标板启动完成后，在串口终端中执行下面命令可停止/启动 Matrix 示例的运行：

```
target# /etc/init.d/matrix-gui-e stop
```

```
target# /etc/init.d/matrix-gui-e start
```

Matrix 示例运行后可在高清显示器上看到各个应用列表，如图 3-1-1 所示：



图 3-1-1



这些示例中包括：2D、3D 图形界面演示，多媒体编解码演示等。接入鼠标，即可在高清显示器中选择运行各个演示示例。

### 3.1.2 编译示例

Matrix 示例工程基于 QT/WebKit/C++设计，源码工程位于以下路径：  
<ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/example-applications/ matrix-gui-e-1.3。编译步骤如下：

1. 切换到 EZSDK 根目录下，执行：

```
ubuntu# cd /home/user/ti-ezsdk_dm816x-evm_5_05_01_04/
```

2. 编译整个工程，执行：

```
ubuntu# make matrix
```

编译完成后，会在 matrix-gui-e-1.3 目录下生成 Matrix 的可执行文件：

```
matrix_gui
```

3. 如需清除编译 matrix 生成的各个文件，执行：

```
ubuntu# make matrix_clean
```

## 3.2 OMTB

OMTB 全称 OpenMAX Test Bench，用于创建和连接多媒体软件模块实例，可以实现对他们作为同一个示例统一配置和运行。例如：

OMTB 包提供示例：

- 捕获视频→编码→存储
- 解码→显示

创建一个新的示例：

- 简单示例：捕获→显示，捕获→裁剪→显示
- 综合示例：捕获→编码→解码→显示

用户使用脚本语言形式创建应用示例和改变配置或链接。在脚本中使用的命令行接口编写文件。

### 3.2.1 运行 OMTB

EZSDK 系统启动后在默认配置下会自行启动 Matrix 示例，而 Matrix 和 OMTB 都会使用到图形显示。因此，在运行 OMTB 示例之前需先关闭图形应用程序。

在启动 EVM 板时设置拨码开关设置为 0111 0001，进入 SD 卡启动方式，并连接高清显示器。等待目标板启动后，运行 OMTB 示例过程如下：

1. 关闭 Graphics Planes，在终端中执行：

```
target# echo 0 > /sys/devices/platform/vpss/graphics0/enabled
```

```
target# echo 0 > /sys/devices/platform/vpss/graphics1/enabled
```

2. 进入 OMTB 示例目录，执行：

```
target# cd /usr/share/ti/ti-omb
```

3. 列出目录下的脚本文件，如图 3-2-1 所示：

```
target# ls
root@dm816x-evm:/usr/share/ti/ti-omtb# ls
capture_dei_encode.oms
capture_dei_encode_30fps.oms
decode_scale_display.oms
decode_scale_display_30fps.oms
dual_display_encode_decode.oms
dual_display_encode_decode_nopause.oms
fread_aacdec_fwrite.oms
fread_vdec_fwrite.oms
fread_venc_fwrite.oms
omtb_dm81xxbm_a8host.xv5T
```

图 3-2-1

4. 运行各个 OMTB 示例，执行：

```
target# ./omtb_dm81xxbm_a8host.xv5T <script-name>.oms
```

注：其中的<script-name>需修改成当前目录下可用的脚本文件名，例如：  
运行解码示例：

```
target# ./omtb_dm81xxbm_a8host.xv5T decode_scale_display.oms
```

5. 执行上述命令后，等待组件初始化完毕即可在高清显示器中看到高清解码输出视频。

注：由于目前在硬件上不支持高清视频的捕获，当前 EVM 板可以运行的 demo 只有图 3-2-1 中的 decode\_scale\_display.oms

### 3.2.2 编译示例

OMTB 示例工程源码工程位于以下路径：

<ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/example-applications/omtb\_01\_00\_01\_07。

编译步骤如下：

1. 切换到 EZSDK 根目录下，执行：

```
t ubuntu# cd /home/user/ti-ezsdk_dm816x-evm_5_05_01_04/
```

2. 编译整个工程，执行：

```
ubuntu# make omtb
```

编译完成后，会在 omtb\_01\_00\_01\_07 目录下生成名为 bin 文件夹，存放可执行程序：omtb\_dm81xxbm\_a8host.xv5T。

3. 如需清除编译 omtb 生成的各个文件，执行：

```
ubuntu# make omtb_clean
```

## 3.3 OpenMax

OpenMax 提供了广泛的流媒体编解码和应用。包含加速多媒体组件的开发，并且在多个操作系统、硅片平台上集成和编程，OpenMax 具有很好的移植性。

OpenMax 的实现简化了多处理器架构，使得用户可以在 A8 处理器上创建自己的应用。其他的多媒体应用（如：OMTB、Gstreamer）就可以利用 OpenMax 组件，把他们当作插件应用到现有的框架中。

OpenMax 有多个功能模块组成，每一个功能模块都被赋予一个特定的多媒体处理功能，例如：视频捕获和视频编码，都是以一个组件的形式呈现。组件

被视为面向对象的类，基础的功能继承于 OMX\_BASE 类，而标准的 API 被扩展开来以满足组件的要求。TI 提供的 OpenMax 组件有：OMX\_VFCC(捕获)、DEI(反交叠)、NF(滤波)、VENC(编码)、VDEC(解码)、SC(裁剪)、VFDC(显示)。

### 3.3.1 运行 OpenMax

EZSDK 系统启动后在默认配置下会自行启动 Matrix 示例，而 Matrix 和 OMTB 都会使用到图形显示。因此，在运行 OMTB 示例之前需先关闭 Matrix 示例程序。

在启动 EVM 板时设置拨码开关设置为 0111 0001，进入 SD 卡启动方式，并连接高清显示器。等待目标板启动后，运行 OpenMax 示例过程如下：

1. 关闭 Matrix 示例程序，执行：

```
target# /etc/init.d/matrix-gui-e stop
target# /etc/init.d/pvr-init stop
```

2. 进入 OpenMax 示例目录，执行：

```
target# cd /usr/share/ti/ti-omx
```

3. 列出 OpenMax 各个示例，如图 3-2-1 所示：

```
target# ls
adec_snt_a8host_debug.xv5T
audio_encode_a8host_debug.xv5T
c6xtest_a8host_debug.xv5T
capture_encode_a8host_debug.xv5T
decode_a8host_debug.xv5T
decode_display_a8host_debug.xv5T
decode_mosaicdisplay_a8host_debug.xv5T
display_a8host_debug.xv5T
encode_a8host_debug.xv5T
```

图 3-2-1

4. 例：执行彩色条图显示示例，执行：

```
target# ./display_a8host_debug.xv5T -d 0
```

### 3.3.2 编译示例

OpenMax 示例工程源码工程位于以下路径中：

<ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/component-sources/omx\_05\_02\_00\_46/examples/ti/omx/demos/。编译步骤如下：

1. 切换到 EZSDK 根目录下，执行：

```
ubuntu# cd /home/user/ti-ezsdk_dm816x-evm_5_05_01_04/
```

2. 编译整个工程，执行：

```
ubuntu# make omx
```

编译完成后，会在 omx\_05\_02\_00\_46 目录下生成名为 bin 文件夹，在该文件下有各自独立的文件夹存放可执行程序。

3. 如需清除编译 omx 生成的各个文件，执行：

```
ubuntu# make omx_clean
```

## 3.4 Qt 示例

Qt/Embedded 是一个图形用户界面工具包,用于给 Linux 帧缓冲设备渲染图形画面,已经集成在 EZSDK 软件开发包。此外,Qt 工具包也可以给 X11 图形用户界面渲染图形画面。

### 3.4.1 运行 Qt 示例

EZSDK 系统启动后在默认配置下会自行启动 Matrix 示例,而 Matrix 和 OMTB 都会使用到图形显示。因此,在运行 Qt 示例之前需先关闭 Matrix 示例程序。

在启动 EVM 板时设置拨码开关设置为 0111 0001,进入 SD 卡启动方式,并连接高清显示器。等待目标板启动后,运行 Qt 示例过程如下:

1. 关闭 Matrix,在终端中执行:

```
target# /etc/init.d/matrix-gui-e stop
```

2. 切换到 Qt 示例目录,在终端中执行:

```
target# cd /usr/bin/qtopia/examples
```

3. 查看当前可执行 Qt 示例,如图 3-4-1 所示:

```
root@dm816x-evm: /usr/bin/qtopia/examples# ls
README effects linguist qws tutorials
animation examples.pro mainwindows richtext uitools
dbus gestures multimedia script webkit
declarative graphicsview network sql widgets
designer help opengl statemachine xml
desktop ipc painting threads xmlpatterns
dialogs itemviews phonon tools
draganddrop layouts qtestlib touch
```

图 3-2-1

4. 在该目录下有多个示例,需切换到各自目录下使用./运行,注意 Qt 程序都需要跟参数。例如运行日历示例,在终端中执行:

```
target# cd richtext/calendar
```

```
target# ./calendar -qws -geometry 320x200+50+20
```

5. 运行后按下 Ctrl+C 结束当前示例。

### 3.4.2 编译示例

Qt 示例源码位于如下路径:

<ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/filesystem/ezsdk-dm816x-evm-rootfs/usr/bin/qtopia/.。依次执行如下命令对他们进行编译,在各自目录下生成对应的可执行文件:

```
ubuntu# cd <应用程序目录>
ubuntu# qmake -project
ubuntu# qmake
ubuntu# make
```

## 3.5 SysLink 示例

SysLink: 类似于 DSP/BIOS 的更名, 是 DSP Link 的高版本, SYSLink 是实现 ARM 与 DSP 核间通信的软件, Codec Engine 用 SYSLink 实现 ARM 和 DSP 之间底层的通信和信息传递。

### 3.5.1 运行 SysLink 示例

在运行示例时需注意, SysLink 示例与默认的 EZSDK 配置相比使用不同的内存映射, 在启动系统时应配置不同的 Linux 内存。即把 bootargs 中 MEN 的值修改为 169M。

在 EZSDK 中文件系统中有多个 Codec Engine 应用示例, 在启动 EVM 板时需将拨码开关设置为 0111 0001 选择从 SD 卡启动 EZSDK 系统。等待目标板启动后, 在目标板上依次执行如下命令:

1. 关闭 Graphics Planes 和运行的固件, 在终端中执行:

```
target# /etc/init.d/pvr-init stop
target# /etc/init.d/matrix-gui-e stop
target# /etc/init.d/load-hd-firmware.sh stop
```

2. 切换到 SysLink 示例目录下, 在终端中执行:

```
target# cd /usr/share/ti/syslink-examples/TI816X
```

3. 载入 SysLink 模块, 在终端中执行:

```
target# modprobe syslink
```

执行结果, 如图 3-5-1 所示:

```
root@dm816x-evm:/usr/share/ti/syslink-examples/TI816X# modprobe syslink
SysLink version : 2.20.00.14
SysLink module created on Date:Oct 23 2013 Time:13:05:58
```

图 3-5-1

4. 在该目录下有多个示例分别切换到各自文件夹下运行 run.sh 脚本文件, 如运行 helloworld 示例:

```
target# cd helloworld
target# ./run.sh
```

执行结果, 如图 3-5-2 所示:

```

root@dm816x-evm:/usr/share/ti/syslink-examples/TI816X# cd helloworld/
root@dm816x-evm:/usr/share/ti/syslink-examples/TI816X/helloworld# ./run.sh
+ ./slaveloader startup DSP server_dsp.xe674
Attached to slave procId 0.
Loading procId 0.
Loaded file server_dsp.xe674 on slave procId 0.
Started slave procId 0.
+ ./app_host DSP
--> App_exec:
App_exec: event received from procId=0
<-- App_exec: 0
+ ./slaveloader shutdown DSP
Stopped slave procId 0.
Unloaded slave procId 0.
Detached from slave procId 0.

```

图 3-5-2

## 3.5.2 编译示例

SysLink 示例位于以下路径中：

<ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/component-sources/syslink\_2\_20\_00\_14 目录下。在编译示例之前需要修改配置变量。Syslink 根目录下 products.mak 文件修改如下：

- 1) 修改芯片类型：  
DEVICE := TI816X
  - 2) 修改处理器操作系统：  
GPPOS =Linux
  - 3) 修改软件开发包名称  
SDK = NONE
  - 4) 修改 EZSDK 文件系统路径：  
EXEC\_DIR  
= <ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/filesystem/ezsdk-dm816x-evm-rotfs  
DEPOT = <ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/component-sources
  - 5) 修改组件根目录下个组件路径：  
##### For TI816X device #####  
else ifeq ("ubuntu#(DEVICE)","TI816X")  
LINUXKERNEL = <ezsdk>/board-support/linux-2.6.37-psp04.04.00.01  
CGT\_ARM\_INSTALL\_DIR =  
/home/user/CodeSourcery/Sourcery\_G++\_Lite  
CGT\_ARM\_PREFIX =  
ubuntu#(CGT\_ARM\_INSTALL\_DIR)/bin/arm-none-linux-gnueabi-  
IPC\_INSTALL\_DIR = <ezsdk>/component-sources/ipc\_1\_24\_03\_32  
BIOS\_INSTALL\_DIR = <ezsdk>/component-sources/bios\_6\_33\_05\_46  
XDC\_INSTALL\_DIR =  
<ezsdk>/component-sources/xdctools\_3\_23\_03\_53  
CGT\_C674\_ELF\_INSTALL\_DIR = <ezsdk>dsp-devkit/cgt6x\_7\_3\_1
  - 6) 在 CGT\_C674\_ELF\_INSTALL\_DIR 下面添加加 M3 编译器路径：  
CGT\_M3\_ELF\_INSTALL\_DIR =  
<DVRDK\_04.00.00.03>/ti\_tools/cgt\_m3/cgt470\_4\_9\_5
- 注：填写的路径应与实际环境一致。

syslink 依赖于当前环境下的 Linux 内核因此在编译 syslink demo 之前需要先编译内核组件。在 EZSDK 根目录下执行如下命令：

```
ubuntu# make linux
```

上述编译完成后，即可编译 syslink demo 生成可执行文件。生成的可执行文件位于< syslink\_2\_20\_00\_14>/examples 目录下，编译步骤如下：

```
ubuntu# cd < syslink_2_20_00_14 >
```

```
ubuntu# make -s examples
```

示例编译完成后需将编译生成的 demo 可执行文件和 driver 拷贝到文件系统中。安装路径在 products.mak 文件中 EXEC\_DIR 变量配置。安装执行：

```
ubuntu# make install
```

## 3.6 Codec Engine 示例

Codec Engine 可以让应用程序通过调用一系列相同 API 函数，轻松实例化和运行 xDM 标准的算法。对于应用程序开发人员来说 Codec Engine 就是一系列 API 函数，并且这个 API 对于以下的所有情况来说没有区别：算法运行在通用处理器(GPP)上，例如 ARM；还是运行在远端处理器（如 DSP）上，这些 API 都相同。

支持的处理器可以是 ARM+DSP 的异构处理器，也可以是单核 DSP 处理器，还可以是单核 ARM 处理器。所有处理器使用的 API 相同，所有支持的系统所用的 API 也相同，系统包括 Linux, WinCE, BIOS 等。

### 3.6.1 运行 Codec Engine 示例

Codec Engine 示例支持本地和远端应用配置，易于移植到多个平台。基于 Linux 的系统要求载入一个或多个内核驱动，使得这些示例能够正常运行。Codec Engine 示例需要载入 CMEM 驱动，并且如果支持远端处理器还需要载入 syslink 驱动。在示例目录下软件开发包已提供了方便载入这些驱动的一个脚本文件：loadmodules.sh。

在运行示例时需注意，Codec Engine 示例与默认的 EZSDK 配置相比使用不同的内存映射，需在启动时配置不同的 Linux 内存。在启动 Linux 时把 bootargs 中 MEN 的值修改为 169M。

在 EZSDK 文件系统中有多多个 Codec Engine 应用示例，在启动 EVM 板时需将拨码开关设置为 0111 0001 选择从 SD 卡启动 EZSDK 系统。等待目标板启动后，在目标板上依次执行如下命令：

1. 关闭 Graphics Planes 和运行的固件，在终端中执行如下命令：

```
target# /etc/init.d/pvr-init stop
```

```
target# /etc/init.d/matrix-gui-e stop
```

```
target# /etc/init.d/load-hd-firmware.sh stop
```

2. 切换到 Codec Engine 示例目录下，在终端中执行：

```
target# cd /usr/share/ti/ti-codec-engine-examples
```

3. 执行脚本文件载入 sysLink 和 cmen 模块，在终端中执行如下命令：

```
target# ./loadmodules.sh
```

执行结果如图 3-6-1 所示:

```
root@dm816x-evm:/usr/share/ti/ti-codec-engine-examples# ./loadmodules.sh
SysLink version : 2.20.00.14
SysLink module created on Date:Oct 23 2013 Time:13:05:58
Trace disabled
Trace entry/leave prints enabled
Trace SetFailureReason enabled
CMEWK module: built on Oct 10 2012 at 03:20:54
Reference Linux version 2.6.37
File /swcoe/sdk/cm/netra/arago-tmp/work/dm816x-evm-none-linux-gnueabi/ti-linuxutils-1_3_22_00_02-r4d/linuxutils_3_22_00_02/package
s/ti/sdo/linuxutils/cmcm/src/module/cmcm.c
allocated heap buffer 0xd1000000 of size 0x4ac000
cmcm initialized
```

图 3-6-1

#### 4. 运行 Codec Engine 示例命令格式:

**app.xv5T [-p proc] [-e engine] [-s serverSuffix] [-m mapFile] input-file output-file**

**-p proc:** 如果使用了远端 server, proc 就应该修改为需要载入 server 的处理  
器名称, 如: "DSP"或者"VIDEO-M3"。如果 proc 没有定义, 默认为"DSP"。

**-e engine:** 指定将要打开的 engine 的名字, 在运行本地应用程序时使用此  
选项。例如: 本地运行 speech1\_copy, 执行: ./app\_local.xv5T -e  
"speech1\_copy"

**-s serverSuffix:** 如果使用到了远端 server, serverSuffix 就应该修改为 server  
的名字的扩展添加的名称, 以此指示编译生成的目标类型。例如: "xe674"  
代表 C674 Elf 目标类型。默认的 serverSuffix 是"x64P"。

**-m mapFile:** 如果用于远端处理器载入 server 的 MMU 被使能, mapFile 就  
用于指定包含远端处理器内存映射的文件名称。Codec Engine 将使用  
mapFile 映射 MMU 的入口。示例映射文件位于:

CE\_INSTALL\_DIR/examples/ti/sdo/ce/examples/buildutils

例如: 运行 universal\_copy client 应用示例, 以 DSP 作为 server 端, 执行:

```
target# cd /usr/share/ti/ti-codec-engine-examples/universal_copy/
```

```
target# ./app_remote.xv5T -p DSP -s xe674
```

以 VIDEO-M3 或者 VPSS-M3 作为远端 server, 执行如下:

```
target# ./app_remote.xv5T -p VIDEO-M3 -s xem3
```

```
target# ./app_remote.xv5T -p VPSS-M3 -s xem3
```

**注:** 目前 SD 卡中文件系统没有 M3 端的可执行程序, 需要重新编译生成,  
参考 3.6.2 章节。

验证应用程序是否运行成功, 可以使用 ls 命令查看当前示例目录下是否生  
成 out.dat 文件, 并对比 in.dat 是否一致。运行成功后执行 ls, 如图 3-6-2 所示:

```
root@dm816x-evm:/usr/share/ti/ti-codec-engine-examples/universal_copy# ls
all_DSP.xe674 app_remote.xv5T in.dat out.dat
```

图 3-6-2

### 3.6.2 编译示例

在 Codec engine/example 目录下包括了在开发 CE 当中可能用到的各种情形  
示例, 在编译示例之前需要修改配置参数。

1) <ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/component-sources/codec\_engine\_3\_22  
\_01\_06/examples 目录 xdcpaths.mak 文件

1) 修改芯片类型:

DEVICE := TI816X

2) 修改 Codec engine 安装路径:

CE\_INSTALL\_DIR:=



- /home/user/ti-ezsdk\_dm816x-evm\_5\_05\_01\_04/component-sources/codec\_engine\_3\_22\_01\_06
- 2) <ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/component-sources/codec\_engine\_3\_22\_01\_06 目录 products.mak 文件
- 1) 修改 Graphics SD、syslink、ipc 等组件根目录路径：  
DEPOT =  
/home/user/ti-ezsdk\_dm816x-evm\_5\_05\_01\_04/component-sources
  - 2) 修改组件根目录下个组件路径：  
IPC\_INSTALL\_DIR = ubuntu#(DEPOT)/ipc\_1\_24\_03\_32  
FC\_INSTALL\_DIR =  
ubuntu#(DEPOT)/framework\_components\_3\_22\_01\_07  
LINK\_INSTALL\_DIR =  
ubuntu#(DEPOT)/syslink\_2\_20\_00\_14  
OSAL\_INSTALL\_DIR = ubuntu#(DEPOT)/osal\_1\_22\_01\_09  
XDAIS\_INSTALL\_DIR = ubuntu#(DEPOT)/xdais\_7\_22\_00\_03  
CMEM\_INSTALL\_DIR =  
ubuntu#(DEPOT)/linuxutils\_3\_22\_00\_02  
EDMA3\_LLD\_INSTALL\_DIR =  
ubuntu#(DEPOT)/edma3lld\_02\_11\_05\_02  
XDC\_INSTALL\_DIR =  
ubuntu#(DEPOT)/xdctools\_3\_23\_03\_53  
BIOS\_INSTALL\_DIR = ubuntu#(DEPOT)/bios\_6\_33\_05\_46
  - 3) 修改 DSP C674x 编译器路径：  
ti.targets.elf.C674 ?=  
<DVRSDK\_04.00.00.03>/ti\_tools/cgt\_dsp/cgt6x\_7\_3\_5/
  - 4) 添加 M3 编译器路径：  
ti.targets.arm.elf.M3 ?=  
<DVRSDK\_04.00.00.03>/ti\_tools/cgt\_m3/cgt470\_4\_9\_5
  - 5) 添加 ARM 编译器路径：  
CGTOOLS\_V5T ?= <CodeSourcery/Sourcery\_G++\_Lite>/  
CC\_V5T ?= bin/arm-none-linux-gnueabi-gcc
- 注：填写的路径应与当前环境一致。

Codec engine 依赖于当前环境下的内核和 syslink，因此在编译 Codec engine demo 之前需要先编译内核组件和 syslink。在 EZSDK 根目录下执行：

```
ubuntu# cd /home/user/ti-ezsdk_dm816x-evm_5_05_01_04
ubuntu# make linux
```

在 syslink\_2\_20\_00\_14 根目录下执行：

```
ubuntu# make syslink
```

注：执行上述命令之前需要按照 [3.5.2](#) 章节修改 syslink 配置文件。

上述编译完成后，即可编译 Codec engine Demo 生成可执行文件。生成的可执行文件位于各目录下的 bin 文件夹中，编译执行：

```
ubuntu# cd <codec_engine_3_22_01_06>/example
ubuntu# gmake -s clean
```

```
ubuntu# gmake
```

注：在执行 gmake 命令时，提示找不到 gmake 命令，在/usr/bin/目录下创建 gmake 连接文件，执行：

```
ubuntu# ln -s <ti-ezsdk>/component-sources/xdctools_3_23_03_53/gmake
/usr/bin/gmake
```

在运行 Codec engine 示例是需先将

<ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/component-sources/codec\_engine\_3\_22\_01\_06/examples/ti/sdo/ce/examples/apps 目录下各个示例生成的可执行文件和  
<ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/component-sources/codec\_engine\_3\_22\_01\_06/examples/ti/sdo/ce/examples/servers/all\_codecs 目录下生成的 server 可执行文件放在同一目录下，然后在运行。

## 3.7 GStreamer

GStreamer 是一个开源多媒体框架，我们可以建立 pipeline 来处理多媒体的内容。gst-openmax 插件使用 OpenMax 对多媒体进行加速处理。

### 3.7.1 运行 GStreamer

在启动 EVM 板时需将拨码开关设置为 0111 0001 选择从 SD 卡启动 EZSDK 系统，并连接高清显示器。等待目标板启动后，我们首先关闭 graphics planes。运行 GStreamer 示例过程如下：

1. 关闭 Graphics Planes，在终端中执行如下命令：

```
target# echo 0 > /sys/devices/platform/vpss/graphics0/enabled
target# echo 0 > /sys/devices/platform/vpss/graphics1/enabled
target# echo 0 > /sys/devices/platform/vpss/graphics2/enabled
```

2. 执行以下命令解码一个 H.264 源码数据流，并且通过 HDMI 显示：

```
target# gst-launch -v filesrc
location=/usr/share/ti/data/videos/dm816x_1080p_demo.264 \! 'video/x-h264' !
h264parse access-unit=true ! omx_h264dec ! omx_scaler \! omx_ctrl
display-mode=OMX_DC_MODE_1080P_60 ! omx_videosink sync=false
```

### 3.7.2 编译示例

EZSDK Linux 软件开发包包含了 GStreamer 开发头文件、库和封装配置，位于

<ti-ezsdk\_dm816x-evm\_5\_05\_01\_04>/component-sources/gst-openmax\_GST\_DM81XX\_00\_06\_00\_00 目录下。编译 GStreamer 执行：

```
ubuntu# cd /home/user/ti-ezsdk_dm816x-evm_5_05_01_04
ubuntu# make -s gstomx
```

执行如下命令安装 GStreamer 到文件系统/usr/bin 目录下：

```
target# make gstomx_install
```

## 3.8 Graphics SDK

Ti8168 配备了 3D 核，拥有硬件加速 3D 操作的能力，这类专用硬件基于 SGX 系列芯片设计。Graphics 核只能用于加速图形操作，不能用于视频解码。硬件加速的 API 有：

- OpenGL ES1.1
- OpenGL ES2.0

编程和使用 Graphics Engine，需要使用到上述 API 之一。这些 API 可以用于创建独立的应用，但是都通常作为后端加速各种高级应用框架，如 Android, Qt, Xorg。TI Linux Graphics SDK 包含了 graphics drivers 、OpenGL ES1.1,2.0，当前 EZSDK 软件开发包中 Graphics SDK 只包含软件开发包不含有 Demo，Demo 和 SDK 可在 TI 网站中下载得到：

[http://software-dl.ti.com/dsps/dsps\\_public\\_sw/sdo\\_sb/targetcontent/gfxsdk/latest/index\\_FDS.html](http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/gfxsdk/latest/index_FDS.html)

### 3.8.1 运行 Graphics SDK Demo

在 SD 卡中文件系统中有多组 Graphics 3D 图形示例，在启动 EVM 板时需将拨码开关设置为 0111 0001 选择从 SD 卡启动 EZSDK 系统，并连接高清显示器。等待目标板启动后，在目标板上依次执行如下命令：

1. 关闭 Matrix GUI 固件，在终端中执行：
2. 切换到示例目录下查看可用的示例，在终端中执行：

```
target# /etc/init.d/matrix-gui-e stop
```

```
target# cd /usr/bin/SGX/demos/Raw
```

```
target# ls
```

当前可用示例如图 3-8-1 所示：

```
root@dm816x-evm:/usr/bin/SGX/demos/Raw# ls
OGLES2ChameleonMan OGLESEvilSkull OGLESPolyBump
OGLES2Coverflow OGLESFilmTV OGLESShadowTechniques
OGLES2FilmTV OGLESFiveSpheres OGLESSkybox
OGLES2PhantomMask OGLESFur OGLESTrilinear
OGLES2Shaders OGLESLighting OGLESUserClipPlanes
OGLES2Skybox2 OGLESMouse OGLESVase
OGLES2Water OGLESOptimizeMesh
OGLESCoverflow OGLESParticles
```

图 3-8-1

3. 在该目录下有多个示例，使用 ./ 执行。以运行人脸旋转示例为例，在终端中执行：
4. 执行后可在高清显示器中看到 3D 图形界面。在终端中按下 q 结束当前 demo 的运行。

```
target# ./OGLESPolyBump
```

## 3.8.2 编译示例

EZSDK Linux 软件开发包中只有 Graphics 的软件开发包， Demo 示例需要另行安装，具体步骤如下：

1. 拷贝《experiment》文件夹中《GraphicsDemo》中的安装文件拷贝到虚拟机中，文件名为 Graphics\_SDK\_setuplinux\_4\_10\_00\_01\_minimal\_demos.bin。
2. 在虚拟机命令终端中安装文件，执行：

```
ubuntu# ./Graphics_SDK_setuplinux_4_10_00_01_minimal_demos.bin
```

注：其中的安装路径应该为 ti-ezsdk\_dm816x-evm\_5\_05\_01\_04 安装目录下的/component-sources/graphics-sdk\_4.04.00.02

安装组件选择如图 3-8-2 所示：

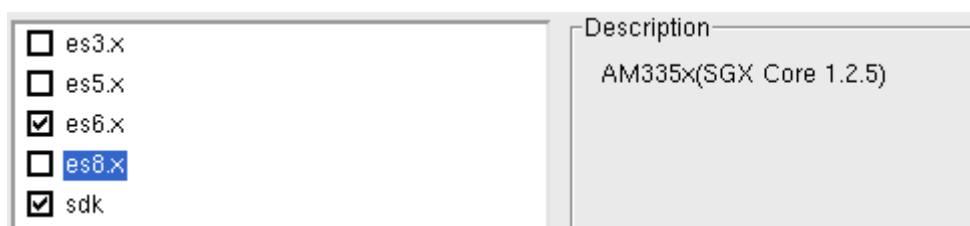


图 3-8-2

安装 Demo 工程后需要修改 Rules.make 配置文件，该文件位于：graphics-sdk\_4.04.00.02 根目录下。修改内容如下：

- 1) 修改 Graphics SD、syslink、ipc 等组件根目录路径：  
HOME=/home/user/ti-ezsdk\_dm816x-evm\_5\_05\_01\_04/component-source  
s
- 2) 修改 GNU 根目录路径：  
CSTOOL\_DIR=/home/user/CodeSourcery/Sourcery\_G++\_Lite
- 3) 修改 EZSDK 内核根目录路径：  
KERNEL\_INSTALL\_DIR=/home/user/ti-ezsdk\_dm816x-evm\_5\_05\_01\_04/board-support/linux-2.6.37-psp04.04.00.01
- 4) 修改 SDK 包版本号：  
GRAPHICS\_INSTALL\_DIR=ubuntu#(HOME)/graphics-sdk\_4.04.00.02
- 5) 修改编译器前缀：  
CSTOOL\_PREFIX=arm-none-linux-gnueabi-  
注：填写的路径应与当前环境一致。

Graphics SDK 依赖于当前环境下的 Linux 内核和 syslink，因此在编译 graphics SDK 之前需要先编译内核和 syslink。编译内核模块，在 EZSDK 包根目录下执行如下命令：

```
ubuntu# cd /home/user/ti-ezsdk_dm816x-evm_5_05_01_04
```

```
ubuntu# make linux
```

编译 HLOS：syslink、Linux 内核模块和库文件，在 EZSDK 根目录下执行：

```
ubuntu# make syslink_hlos
```

上述编译完成后，即可编译 Graphics SDK Demo 生成可执行文件。生成的可执行文件位于<graphics-sdk\_4.04.00.02>/gfxsdkdemos 目录下，编译工程执行：

```
ubuntu# cd <graphics-sdk_4.04.00.02>
```

```
ubuntu# gmake BUILD=release OMAPES=6.x FBDEV=yes SUPPORT=1 all
```

## 4 基于 MCFW 的应用开发

DVR RDK 是应用于 TI816x, TI814x, TI810x 平台的多核处理器软件开发框架，是一个充分优化过的多通道视频应用，如监控 DVR, NVR, Hybrid-DVR, HD-DVR。这个软件框架允许用户创建不同的多通道视频数据流，包括视频捕获，视频处理（反交叠、缩放、滤波、软件马赛克、软件 OSD、Tamper 检测、运动检测），视频编码（H264、MJPEG），视频解码（H264、MPEG4、MJPEG）和视频显示（HDMI、HDDAC、CVBS、Graphics）。DVR RDK 包含所有的组件，这些组件需要可就各自的视频应用而进行重新配置和编译。

### 4.1 修改 DEMO 示例

在 DVRRDK 框架下包含多个不同的应用示例，包括捕获+显示、高清解码+显示、捕获+编码+解码+显示等等，为适应当前 DES8168-EVM 板的配置状况需做必要的修改。这些示例源码位于<DVRRDK\_04.00.00.03>/dvr\_rdk 目录下的 demo 和 MCFW 文件夹中。

在 DVRRDK 框架默认情况下为：16 路视频输入，4 片 TVP5158 视频捕获芯片，DM8168 视频输入口为 VIP0~VIP3。而目前 EVM 板配置为：4 路视频输入，1 片 TVP5158 视频捕获芯片，DM8168 视频输入口为 VIP1\_A。

#### 4.1.1 标清直通示例

标清直通示例包括 4 路视频输入，主要组件为：视频捕获、标清显示和 HDMI 显示。修改内容主要包括：mian 函数界面中添加标清直通示例选项，捕获和显示通道数量修改，视频输入口修改，HDMI 中 4 路马赛克显示，使能标清显示。

具体修改步骤如下：

1. <DVRRDK\_04.00.00.03>/dvr\_rdk/demos/mcfw\_api\_demos/mcfw\_demo/demo.c 文件 char gDemo\_mainMenu[] 字符数组中添加标清直通示例选项：  
Ln42: "\r\n 5: VCAP + VDIS - NO Encode or Decode + 4 Channel"
2. <DVRRDK\_04.00.00.03>/dvr\_rdk/demos/mcfw\_api\_demos/mcfw\_demo/demo\_vcap\_vdis.c 文件 Void VcapVdis\_start () 函数：  
Ln50: gDemo\_info.maxVcapChannels = 4;//16  
Ln51: gDemo\_info.maxVdisChannels = 4;//16  
Ln55: vcapParams.numChn = 4;//16  
Ln56: vdisParams.numChannels = 4;//16
3. 在 Vdis\_start () 和 Vcap\_start () 函数之间添加函数调用：  
Ln103: Demo\_displaySettings\_4channel ();
4. <DVRRDK\_04.00.00.03>/dvr\_rdk/mcfw/src\_linux/mcfw\_api/usecases/ti816x/multich\_vcap\_vdis.c 文件 Void MultiCh\_createVcapVdis () 函数：

```

Ln190: enableSdtv = TRUE;//FALSE
Ln194: capturePrm.numVipInst = 1; //2 * numSubChains;
Ln203: VIP0 修改为 VIP1
pCaptureInstPrm->vipInstId
=(SYSTEM_CAPTURE_INST_VIP1_PORTA+vipInstId)%SYSTEM_CAPTU
RE_INST_MA;
Void MultiCh_deleteVcapVdis () 函数:
Ln404: UInt32 enableSdtv = TRUE;//FALSE;
5. DVRRDK_04.00.00.03/dvr_rdk/demos/mcfw_api_demos/mcfw_demo 目录下
demo.h 文件中添加函数声明:
int Demo_displaySettings_4channel () ;
6. DVRRDK_04.00.00.03/dvr_rdk/mcfw/src_linux/mcfw_api 目录下 ti_vdis.c 文
件:
Ln391: //OSA_assert(status >= 0);
Ln404: //OSA_assert(status >= 0);
Ln421: //OSA_assert(status >= 0);
7. DVRRDK_04.00.00.03/dvr_rdk/demos/mcfw_api_demos/mcfw_demo/demo_di
splay.c 文件,添加函数:
int Demo_displaySettings_4channel ()
{
 int demoId = 5;
 static VDIS_MOSAIC_S vdMosaicParam;
 UInt32 startChId;
 int prevLayoutId;
 VSYS_PARAMS_S sysContextInfo;
 VDIS_DEV devId;
 Bool forceLowCostScale = FALSE;
 Vsys_getContext(&sysContextInfo);
 if(gDemo_info.maxVdisChannels<=0)
 {
 printf(" \n");
 printf(" WARNING: Display NOT enabled, this menu is NOT
valid !!!\n");
 return -1;
 }
 prevLayoutId = layoutId;
 {
layoutId = DEMO_LAYOUT_MODE_4CH;
 /***** For devID = VDIS_DEV_HDMI *****/
 devId = VDIS_DEV_HDMI;
 startChId = 0;
 Demo_swMsGenerateLayout(
 devId,
 startChId,

```

```

 gDemo_info.maxVdisChannels,
 layoutId,
 &vdMosaicParam,
 forceLowCostScale,
 gDemo_info.Type,
 Vdis_getSwMsLayoutResolution(devId));
 Vdis_setMosaicParams(devId,&vdMosaicParam);
target#if defined (TI_816X_BUILD) || defined (TI_814X_BUILD)
target#if DEMO_SCD_ENABLE_MOTION_TRACKING
target#if USE_FBDEV
 {
 Bool drawGrid = TRUE;
target#if defined (TI_816X_BUILD)
 if(sysContextInfo.systemUseCase !=
VSYS_USECASE_MULTICHN_PROGRESSIVE_VCAP_VDIS_VENC_VDEC)
 {
 drawGrid = FALSE;
 }
target#endif
 if(sysContextInfo.enableScd == TRUE && drawGrid)
 {
 Scd_mosaicUpdated () ;
 }
 }
target#endif
target#endif
target#endif

 /****** For devID = VDIS_DEV_HDCOMP
 *****/
target#ifdef TI_816X_BUILD
if (Vdis_isSupportedDisplay(VDIS_DEV_HDCOMP))
{
 devId = VDIS_DEV_HDCOMP;
 startChId = 0;
 /* if the number of channels being display are more than
that can fit in 4x4 then
 make the other channels appear on the second HD
Display.
 Otherwise show same channels on the other HD
Display
 */

 if(gDemo_info.maxVdisChannels>VDIS_MOSAIC_WIN_MAX)
 startChId = VDIS_MOSAIC_WIN_MAX;

```



```

 Demo_swMsGenerateLayout(
 devId,
 startChId,
 gDemo_info.maxVdisChannels,
 layoutId,
 &vdMosaicParam,
 forceLowCostScale,
 gDemo_info.Type,
 Vdis_getSwMsLayoutResolution(devId)
);
 if ((demoId ==
DEMO_VCAP_VENC_VDEC_VDIS_PROGRESSION) ||
 (demoId == DEMO_VCAP_VENC_VDIS))
 {

Demo_displayChangeFpsForLayout(&vdMosaicParam,layoutId);
 }
 Vdis_setMosaicParams(devId,&vdMosaicParam);
 }
 else
 {
 printf("No HDCOMP present. Usecase id[%d]\n",demoId);
 }
target#endif
/***** For devID = VDIS_DEV_SD *****/
devId = VDIS_DEV_SD;
startChId = 0;
Demo_swMsGenerateLayout(
 devId,
 startChId,
 gDemo_info.maxVdisChannels,
 layoutId,
 &vdMosaicParam,
 forceLowCostScale,
 gDemo_info.Type,
 Vdis_getSwMsLayoutResolution(devId)
);
target#ifdef TI_816X_BUILD
 if ((demoId ==
DEMO_VCAP_VENC_VDEC_VDIS_PROGRESSION) ||
 (demoId == DEMO_VCAP_VENC_VDIS))
 {

Demo_displayChangeFpsForLayout(&vdMosaicParam,layoutId);

```

```

 }
 target#endif
 /* wait for the info prints to complete */
 OSA_waitMsecs(500);
}
return 0;
}

```

### 4.1.2 标清编解码示例

标清编解码示例包括 4 路视频输入，主要组件为：视频捕获、视频编码、视频解码、标清显示和 HDMI 显示。修改内容主要包括：捕获、显示和编解码通道数量修改，视频输入口修改，HDMI 中 4 路马赛克显示，场景检测通道数量修改。

具体修改步骤如下：

1. DVRRDK\_04.00.00.03/dvr\_rdk/demos/mcfw\_api\_demos/mcfw\_demo 目录下 demo\_vcap\_venc\_vdec\_vdis.c 文件，修改如下：  
 Ln122: gDemo\_info.maxVcapChannels = 4; //16  
 Ln123: gDemo\_info.maxVdisChannels = 4; //32  
 Ln124: gDemo\_info.maxVencChannels = 4; //16  
 Ln125: gDemo\_info.maxVdecChannels = 4; //16  
 Ln126: Demo\_info.VsysNumChs = 4; //16  
 Ln551: 修改 DEMO\_LAYOUT\_MODE\_16CH 为 DEMO\_LAYOUT\_MODE\_4CH  
 Demo\_swMsGenerateLayout(VDIS\_DEV\_HDMI, 0, gDemo\_info.maxVdisChannels, DEMO\_LAYOUT\_MODE\_4CH, &vdisParams.mosaicParams[i], forceLowCostScale, gDemo\_info.Type, Vdis\_getSwMsLayoutResolution(VDIS\_DEV\_HDMI));
2. DVRRDK\_04.00.00.03/dvr\_rdk/mcfw/src\_linux/mcfw\_api/usecases/ti816x/multich\_progressive\_vcap\_venc\_vdec\_vdis.c 文件，  
 Void MultiCh\_createProgressiveVcapVencVdecVdis ( ) 函数：  
 Ln748: VIP0 修改为 VIP1  
 pCaptureInstPrm->vipInstId = (SYSTEM\_CAPTURE\_INST\_VIP1\_PORTA+ vipInstId)%SYSTEM\_CAPTURE\_INST\_MAX;  
 Ln1200: dspAlgPrm[1].scdCreateParams.numValidChForSCD = 4; //16

### 4.1.3 标清编码示例

标清编码示例包括 4 路视频输入，主要组件为：视频捕获、视频编码、标

清显示和 HDMI 显示。修改内容主要包括：捕获、显示和编码通道数量修改，视频输入口修改，HDMI 中 4 路马赛克显示，场景检测通道数量修改。

具体修改步骤如下：

1. DVRRDK\_04.00.00.03/dvr\_rdk/demos/mcfw\_api\_demos/mcfw\_demo 目录下 demo\_vcap\_venc\_vdis.c 文件，修改如下：  
Ln48: gDemo\_info.maxVcapChannels = 4;//16  
Ln49: gDemo\_info.maxVdisChannels = 4;//16  
Ln50: Demo\_info.maxVencChannels = 4;//16  
Ln53: vcapParams.numChn = 4;//16  
Ln54: vencParams.numPrimaryChn = 2;//16  
Ln55: vencParams.numSecondaryChn = 2;//16  
Ln56: vdisParams.numChannels = 4;//16  
Ln96: 修改 DEMO\_LAYOUT\_MODE\_7CH\_1CH 为 DEMO\_LAYOUT\_MODE\_4CH  
Demo\_swMsGenerateLayout(VDIS\_DEV\_HDMI, 0, gDemo\_info.maxVdisChannels, DEMO\_LAYOUT\_MODE\_4CH, &vdisParams.mosaicParams[0], TRUE, gDemo\_info.Type, Vdis\_getSwMsLayoutResolution(VDIS\_DEV\_HDMI));
2. DVRRDK\_04.00.00.03/dvr\_rdk/mcfw/src\_linux/mcfw\_api/usecases/ti816x/multich\_vcap\_venc\_vdis.c 文件 Void MultiCh\_createVcapVencVdis ( ) 函数中，修改如下：  
Ln374: VIP0 修改为 VIP1，并添加求余操作如下：  
pCaptureInstPrm->vipInstId = (SYSTEM\_CAPTURE\_INST\_VIP1\_PORTA+ vipInstId)%SYSTEM\_CAPTURE\_INST\_MAX;  
Ln816: dspAlgPrm.scdCreateParams.numValidChForSCD = 4;//16;

#### 4.1.4 高清解码示例

高清解码示例包括 1 路高清解码显示，主要组件为：高清解码和 HDMI 显示。修改内容主要包括：显示和解码通道数量修改，HDMI 中 1 路马赛克显示，ini 配置文件。

具体修改步骤如下：

1. DVRRDK\_04.00.00.03/dvr\_rdk/demos/mcfw\_api\_demos/mcfw\_demo/demo\_vdec\_vdis.c 文件,Void VdecVdis\_start ( ) 函数：  
Ln73: gDemo\_info.maxVdisChannels = 1;//gVdecVdis\_config.fileNum;  
Ln75: gDemo\_info.maxVdecChannels = 1;//gVdecVdis\_config.fileNum;  
Ln77: vdecParams.numChn = 1;//gVdecVdis\_config.fileNum;  
Ln78: vdisParams.numChannels = 1;//gVdecVdis\_config.fileNum;  
Ln108: vdecParams.numChn = 1;//gVdecVdis\_config.fileNum;  
Ln165: vdisParams.numChannels = 1;//gVdecVdis\_config.fileNum;  
Ln201: DEMO\_LAYOUT\_MODE\_4CH\_4CH 修改为

```

DEMO_LAYOUT_MODE_1CH
Demo_swMsGenerateLayout(VDIS_DEV_HDMI, 0, vdecParams.numChn,
 DEMO_LAYOUT_MODE_1CH,
 &vdisParams.mosaicParams[i],
 forceLowCostScale,
 gDemo_info.Type,

Vdis_getSwMsLayoutResolution(VDIS_DEV_HDMI));
Ln238: if(vdecParams.numChn < 4)//16

```

## 4.2 示例编译

各个应用示例修改后需要重新编译、连接生成可执行文件，编译后系统会自动将生成的文件自动拷贝到 DVRRDK 文件系统目录

<DVRRDK\_04.00.00.03>/target/rfs\_816x/opt/dvr\_rdk/ti816x。编译步骤如下：

1. 在 Linux 终端中切换工作目录到<DVRRDK\_04.00.00.03>/dvr\_rdk 目录下：  
target# cd /home/userDVRRDK\_04.00.00.03/dvr\_rdk/
2. 执行如下两个命令之一对所有 DEMO 示例进行编译：  
target# make all (clean 整个 dvrrdk 包然后再编译)  
target# make dvr\_rdk (仅进行增量编译)

## 4.3 示例运行

示例运行需要挂载 DVRRDK 文件系统，可以从 NandFlash 挂载，也可以通过网络挂载。在使用网络挂载文件及系统方式时须确保按照 [4.1](#) 章节修改各个示例并按照 [4.2](#) 章节编译生成可执行程序。

### 4.3.1 标清编解码示例

标清编解码示例包括 4 路视频输入、1 路 HDMI 显示输出和 1 路标清显示输出。其中在 HDMI 显示器中可同时看到 4 路画面显示，标清显示器中可以看到视频输入 1 输入视频画面。示例运行过程如下：

1. 连接 HDMI 显示器、标清显示器、串口线、网线、电源和 4 路标清视频输入(也可以只连接 1 路，在运行时切换各个通道)；
2. PC 机打开串口终端，设置波特率为 115200；
3. 开启 EVM 板电源，若 NAND 中已经烧写内核和文件系统，则只需等待系统启动完成；若未烧写文件系统，则需按照 [2.5.2](#) 章节搭建 DVRRDK 网络文件系统；
4. 进入 Demo 示例文件夹，在串口终端中执行：  
target# cd /opt/dvr\_rdk/ti816x
5. 运行 init.sh 脚本文件，初始化共享区域和载入必须的内核模块，在串口终端中执行：

```
target# ./init.sh
```

6. 运行 load.sh 脚本文件，使用固件下载功能载入 M3\_VPSS、M3\_Video 及 DSP 可执行文件，在串口终端中执行：

```
target# ./load.sh
```

**注：**当 EVM 板正在运行时，如果需要再次载入新编译的工程需要按下面顺序执行：

```
target# ./unload.sh
```

```
target# ./load.sh
```

7. 运行 run.sh 脚本文件，进入示例选择界面，在串口终端中执行：

```
target# ./run.sh
```

8. 在串口终端中输入 1 并按下回车键，选择标清编解码示例，等待执行完毕后可在 HDMI 显示器中看到 4 路输入视频，在标清显示器上看到频输入口 1 输入视频。

### 4.3.2 标清编码示例

标清编码示例包括 4 路视频输入和 1 路 HDMI 显示输出，其中在 HDMI 显示器中可同时看到 4 路画面显示。示例运行过程如下：

1. 连接 HDMI 显示器、标清显示器、串口线、网线、电源和 4 路标清视频输入(也可以只连接 1 路，在运行时切换各个通道)；
2. PC 机打开串口终端，设置波特率为 115200；
3. 开启 EVM 板电源，若 NAND 中已经烧写内核和文件系统，则只需等待系统启动完成；若未烧写文件系统，则需按照 [2.5.2](#) 章节搭建 DVRRDK 网络文件系统；
4. 进入 Demo 示例文件夹，在串口终端中执行：

```
target# cd /opt/dvr_rdk/ti816x
```

5. 运行 init.sh 脚本文件，初始化共享区域和载入必须的内核模块，在串口终端中执行：

```
target# ./init.sh
```

6. 运行 load.sh 脚本文件，使用固件下载功能载入 M3\_VPSS、M3\_Video 及 DSP 可执行文件，在串口终端中执行：

```
target# ./load.sh
```

**注：**当 EVM 板正在运行时，如果需要再次载入新编译的工程需要按下面顺序执行：

```
target# ./unload.sh
```

```
target# ./load.sh
```

7. 运行 run.sh 脚本文件，进入示例选择界面，在串口终端中执行：

```
target# ./run.sh
```

8. 在串口终端中输入 2 并按下回车键，选择标清编码示例，等待执行完毕后可在 HDMI 显示器中看到 4 路输入视频。

### 4.3.3 高清解码示例

高清解码示例包括 1 路视频解码和 1 路 HDMI 显示输出。运行示例，使用 NFS 方式挂载文件系统，需要先拷贝《experiment》文件夹中《Video》文件夹中的文件到虚拟机<DVRRDK\_04.00.00.03>/target/rfs\_816x/opt/dvr\_rdk/ti816x 目录下；示例运行过程如下：

1. 连接 HDMI 显示器、标清显示器、串口线、网线、电源和 4 路标清视频输入(也可以只连接 1 路，在运行时切换各个通道)；
2. PC 机打开串口终端，设置波特率为 115200；
3. 开启 EVM 板电源，若 NAND 中已经烧写内核和文件系统，则只需等待系统启动完成；若未烧写文件系统，则需按照 [2.5.2](#) 章节搭建 DVRRDK 网络文件系统；
4. 进入 Demo 示例文件夹，在串口终端中执行：  
`target# cd /opt/dvr_rdk/ti816x`
5. 运行 init.sh 脚本文件，初始化共享区域和载入必须的内核模块，在串口终端中执行：  
`target# ./init.sh`
6. 运行 load.sh 脚本文件，使用固件下载功能载入 M3\_VPSS、M3\_Video 及 DSP 可执行文件，在串口终端中执行：  
`target# ./load.sh`  
注：当 EVM 板正在运行时，如果需要再次载入新编译的工程需要按下面顺序执行：  
`target# ./unload.sh`  
`target# ./load.sh`
7. 运行 run.sh 脚本文件，进入示例选择界面，在串口终端中执行：  
`target# ./run.sh`
8. 在串口终端中输入 4 并按下回车键，选择解码示例；
9. 按照提示执行 ini 文件，在串口终端中执行：  
`target# ./demo.ini`
10. 终端中提示是否生成头文件，若未生成输入 y，否则输入 n；
11. 等待执行完毕后可在 HDMI 显示器中看到解码后的视频显示。

### 4.3.4 标清直通示例

标清编解码示例包括 4 路视频输入、1 路 HDMI 显示输出和 1 路标清显示输出。其中在 HDMI 显示器中可同时看到 4 路画面显示，标清显示器中可以看到视频输入口 1 输入视频画面。示例运行过程如下：

1. 连接 HDMI 显示器、标清显示器、串口线、网线、电源和 4 路标清视频输入(也可以只连接 1 路，在运行时切换各个通道)；
2. PC 机打开串口终端，设置波特率为 115200；
3. 开启 EVM 板电源，若 NAND 中已经烧写内核和文件系统，则只需等待系统启动完成；若未烧写文件系统，则需按照 [2.5.2](#) 章节搭建 DVRRDK 网络文件系统；

4. 进入 Demo 示例文件夹，在串口终端中执行：  
`target# cd /opt/dvr_rdk/ti816x`
5. 运行 init.sh 脚本文件，初始化共享区域和载入必须的内核模块，在串口终端中执行：  
`target# ./init.sh`
6. 运行 load.sh 脚本文件，使用固件下载功能载入 M3\_VPSS、M3\_Video 及 DSP 可执行文件，在串口终端中执行：  
`target# ./load.sh`  
**注：**当 EVM 板正在运行时，如果需要再次载入新编译的工程需要按下面顺序执行：  
`target# ./unload.sh`  
`target# ./load.sh`
7. 运行 run.sh 脚本文件，进入示例选择界面，在串口终端中执行：  
`target# ./run.sh`
8. 在串口终端中输入 5 并按下回车键，选择标清直通示例，等待执行完毕后可在 HDMI 显示器中看到 4 路输入视频，在标清显示器上看到频输入口 1 输入视频。

## 4.4 视频直通示例加入 sobel 算法

### 4.4.1 源码修改

在视频直通的基础上加入算法需要加入新的 Link，ipcFramesOutVpss、ipcFramesInDsp、dspAlg，结构链路如 4-2-1 所示：

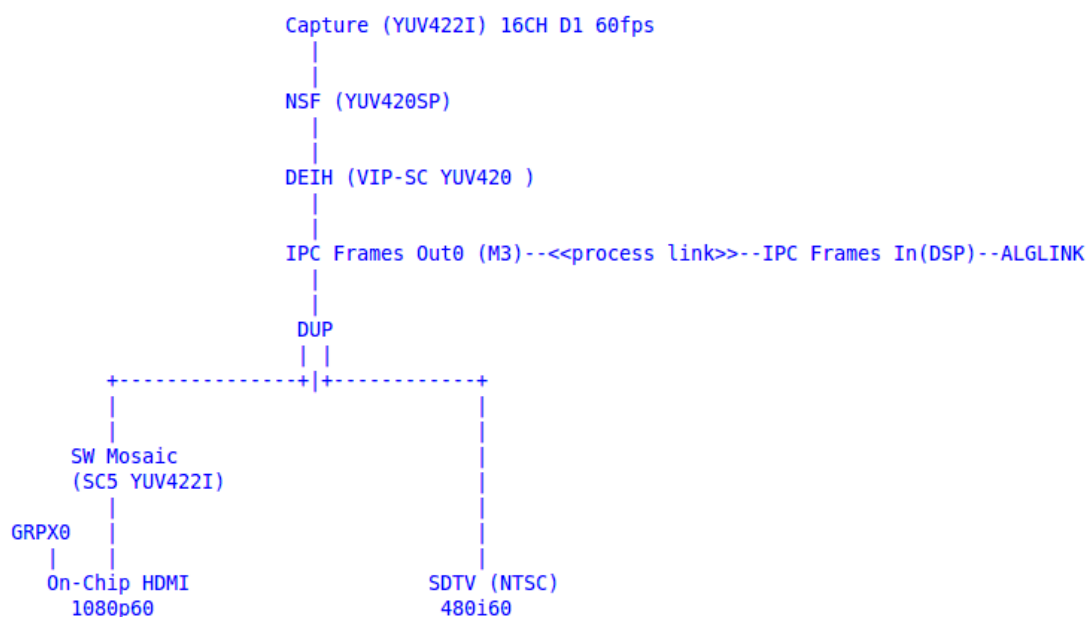


图 4-2-1

主要需要修改的文件为：multich\_vcap\_vdis.c 和 algLink\_pric.c。修改步骤如下：

1. multich\_vcap\_vdis.c 中 MultiCh\_createVcapVdis 函数：
  - 1) 注释掉 nullSrc 和 merge 的声明：
 

```
Ln122: //NullSrcLink_CreateParams nullSrcPrm;
Ln123: //MergeLink_CreateParams mergePrm;
```
  - 2) 注释掉 nullId 和 mergeId 声明：
 

```
Ln134: //UInt32 nullId;
Ln135: UInt32 /*mergeId, */dupId;
```
2. 在 dsp 端加入 sobel 边缘处理算法，首先使用 ipcFramesOutVpss Link 将视频流送到 dsp 端，然后使用 dspAlg Link 调用 dsp 端的算法处理函数，并将处理的结果返回。
  - 1) 在变量声明处加入如下的申明：
 

```
Ln146: IpcFramesOutLinkRTOS_CreateParams
ipcFramesOutVpssPrm[1];
Ln147: IpcFramesInLinkRTOS_CreateParams ipcFramesInDspPrm[1];
Ln148: AlgLink_CreateParams dspAlgPrm[1];
```
  - 2) 在结构体初始化处加入如下的 link 结构体初始化：
 

```
Ln156:
MULTICH_INIT_STRUCT(IpcFramesOutLinkRTOS_CreateParams,ipcFramesOutVpssPrm[0]);
Ln157:
MULTICH_INIT_STRUCT(IpcFramesInLinkRTOS_CreateParams,ipcFramesInDspPrm[0]);
Ln158:
MULTICH_INIT_STRUCT(AlgLink_CreateParams, dspAlgPrm[0]);
```
  - 3) 给新加入的 link 赋 ID 号：
 

```
Ln196: gVcapModuleContext.dspAlgId[0] = SYSTEM_LINK_ID_ALG_0;
Ln197: gVcapModuleContext.ipcFramesOutVpssId[0] =
SYSTEM_VPSS_LINK_ID_IPC_FRAMES_OUT_0;
Ln198: gVcapModuleContext.ipcFramesInDspId[0] =
SYSTEM_DSP_LINK_ID_IPC_FRAMES_IN_0;
```
3. 根据 Link 流程图，配置各个 Link 的前后 Link 链接关系和参数：
 

```
deiPrm[i].outQueParams[deiOutQue].nextLink=
gVcapModuleContext.ipcFramesOutVpssId[0];

ipcFramesOutVpssPrm[0].baseCreateParams.inQueParams.prevLinkId =
gVcapModuleContext.deiId[0];

ipcFramesOutVpssPrm[0].baseCreateParams.inQueParams.prevLinkQueId =
deiOutQue;

ipcFramesOutVpssPrm[0].baseCreateParams.notifyPrevLink = TRUE;

ipcFramesOutVpssPrm[0].baseCreateParams.numOutQue = 1;
```



```

ipcFramesOutVpssPrm[0].baseCreateParams.outQueParams[0].nextLink =
mergeId;

ipcFramesOutVpssPrm[0].baseCreateParams.notifyNextLink = TRUE;

ipcFramesOutVpssPrm[0].baseCreateParams.processLink =
gVcapModuleContext.ipcFramesInDspId[0];

ipcFramesOutVpssPrm[0].baseCreateParams.notifyProcessLink = TRUE;

ipcFramesOutVpssPrm[0].baseCreateParams.noNotifyMode = FALSE;

//prevLink->processLink->nextLink

ipcFramesInDspPrm[0].baseCreateParams.inQueParams.prevLinkId =
gVcapModuleContext.ipcFramesOutVpssId[0];

ipcFramesInDspPrm[0].baseCreateParams.inQueParams.prevLinkQueId = 0;

ipcFramesInDspPrm[0].baseCreateParams.numOutQue = 1;

ipcFramesInDspPrm[0].baseCreateParams.outQueParams[0].nextLink =
gVcapModuleContext.dspAlgId[0];

ipcFramesInDspPrm[0].baseCreateParams.notifyPrevLink = TRUE;

ipcFramesInDspPrm[0].baseCreateParams.notifyNextLink = TRUE;

ipcFramesInDspPrm[0].baseCreateParams.noNotifyMode = FALSE;

 dspAlgPrm[0].inQueParams.prevLinkId =
gVcapModuleContext.ipcFramesInDspId[0];
 dspAlgPrm[0].inQueParams.prevLinkQueId = 0;
 if(TRUE)
 {
 //int chId;
 dspAlgPrm[0].enableOSDAlg = TRUE;
 dspAlgPrm[0].enableSCDAlg = FALSE;

dspAlgPrm[0].outQueParams[ALG_LINK_SCD_OUT_QUE].nextLink =
SYSTEM_LINK_ID_INVALID;
 }

```

```

mergePrm.numInQue = 1;//numSubChains;
 if(i==0) {
 mergePrm.inQueParams[i].prevLinkId =
gVcapModuleContext.ipcFramesOutVpssId[0];
 mergePrm.inQueParams[i].prevLinkQueId = 0;
 }

```

#### 4. 创建新添加的 Link

```

System_linkCreate(gVcapModuleContext.ipcFramesOutVpssId[0],
&ipcFramesOutVpssPrm[0], sizeof(ipcFramesOutVpssPrm[0]));
System_linkCreate(gVcapModuleContext.ipcFramesInDspId[0],
&ipcFramesInDspPrm[0], sizeof(ipcFramesInDspPrm[0]));
System_linkCreate(gVcapModuleContext.dspAlgId[0] , &dspAlgPrm[0],
sizeof(dspAlgPrm[0]));

```

#### 5. algLink\_priv.c 在 AlgLink\_algProcessData 函数:

```

1) 在函数前加临时 buf 定义 unsigned char *tmp_buffer;
 unsigned char *frameBuffer1=(unsigned char *)malloc(720*480);
 unsigned char *frameBuffer2=(unsigned char *)malloc(720*480);
2) 加入 IMG_sobel_3x3_8 函数;
 if (frameList.frames[frameId]->channelNum == 0) {
 for (i = 0; i < 720 * 480; i++) {
 tmp_buffer = pFrame->addr[pFrame->fid][0];

 frameBuffer1[i] =

 (unsigned char) (tmp_buffer
 +i*2);

 }
 IMG_sobel_3x3_8(frameBuffer1, frameBuffer2, 720, 480);
 tmp_buffer = pFrame->addr[pFrame->fid][0];
 for (i = 0; i < 720 * 480; i++) {

 tmp_buffer[i * 2] = frameBuffer2[i];//////////
 }
 }

```

#### 6. rules\_c674.mk 文件

Sobel 算法函数需要库支持, 在 dvr\_rdk\_PATH/makerules/rules\_c674.mk 中可以进行库的连接。

```

LIB_PATHS += ubuntu#(RTSLIB_PATH) \
ubuntu#(dvr_rdk_PATH)/mcfw/src_bios6/alg/imglib/lib/target/imglib2_elf.lib
\
ubuntu#(dvr_rdk_PATH)/mcfw/src_bios6/links_c6xdsp/alg_link/lib_fu/c64plus
/imglib2_rebuild.l64P \

```

## 4.4.2 源码编译

修改完毕后，编译 dvrrdk 包依次执行：

```
target# cd ~ /DVRRDK_04.00.00.03/dvr_rdk
target# make dvr_rdk
```

## 4.4.3 运行算法 demo

参照 4.3.4 章节使用 NFS 方式挂载文件系统运行直通示例。运行后可看到第一路视频输入中的图像边缘亮度发生改变。

## 4.5 M3\_VPSS 测试源码编译

### 4.5.1 修改配置文件

路径：<DVRRDK\_04.00.00.03>/ti\_tools/hdvpss/dvr\_rdk\_hdvpss/rules.make，主要修改路径参数，应与实际环境配置。修改环境参数如下(黑体部分)：

```
hdvpss_RELPATH = dvr_rdk_hdvpss
ifeq (ubuntu#(OS),Windows_NT)
 CODEGEN_PATH_M3 := C:/PROGRA~1/TEXASI~1/TMS470~1.2
 hdvpss_PATH := C:/PROGRA~1/TEXASI~1/ubuntu#(hdvpss_RELPATH)
 bios_PATH := C:/PROGRA~1/TEXASI~1/bios_6_33_02_27_eng
 xdc_PATH := C:/PROGRA~1/TEXASI~1/xdctools_3_23_01_37_eng
 ipc_PATH := C:/PROGRA~1/TEXASI~1/ipc_1_24_01_24
else
 CODEGEN_PATH_M3:=
/home/user/DVRRDK_04.00.00.03/ti_tools/cgt_m3/cgt470_4_9_5
 hdvpss_PATH:=
/home/user/DVRRDK_04.00.00.03/ti_tools/hdvpss/ubuntu#(hdvpss_RELPATH)
 bios_PATH :=
/home/user/DVRRDK_04.00.00.03/ti_tools/bios/bios_6_33_05_46
 xdc_PATH :=
/home/user/DVRRDK_04.00.00.03/ti_tools/xdc/xdctools_3_23_03_53
 ipc_PATH := /home/user/DVRRDK_04.00.00.03/ti_tools/ipc/ipc_1_24_03_32
endif
target# Default platform
ifeq (ubuntu#(PLATFORM),)
 PLATFORM := ti8168-evm
Endif
```

## 4.5.2 编译 M3\_VPSS 测试源码

M3\_VPSS 测试包括捕获和显示两部分。编译生成可执行文件需要使用到 `gmake` 命令，位于 <DVRSDK\_04.00.00.03>/ti\_tools/xdc/xdctools\_3\_23\_03\_53 目录下。在我们首次使用时需要在 /usr/bin 目录下创建这个链接，在虚拟机中执行：

```
target# ln -s <DVRSDK_04.00.00.03>/ti_tools/xdc/xdctools_3_23_03_53/gmake
/usr/bin/gmake
```

在编译源码前，由于当前 EVM 板在硬件上有所改动，因此需修改部分源代码。修改地方包括：

1. M3\_VPSS\_CAPTURE:

```
/home/user/DVRSDK_04.00.00.03/ti_tools/hdvpss/dvr_rdk_hdvpss/packages/ti/
/psp/platforms/src/vps_platform.c
Ln368:devAddr = 0x5C;//Vps_platformTI816xGetVidDecI2cAddr(vidDecId,
vipInstId);
```

2. M3\_VPSS\_DISPLAY:

```
/home/user/DVRSDK_04.00.00.03/ti_tools/hdvpss/dvr_rdk_hdvpss/packages/ti/
psp/examples/utility/src/vpsutils_app.c
Ln413:retVal =
0;//Vps_ths7360SetSdParams(VPS_THSFILTER_ENABLE_MODULE);
Ln588:retVal =
0;//Vps_ths7360SetSdParams(VPS_THSFILTER_DISABLE_MODULE);
```

修改完成并保存文件后，编译源码，在虚拟机终端中依次执行：

```
target# cd <DVRSDK_04.00.00.03>/ti_tools/hdvpss/dvr_rdk_hdvpss
target# gmake -s sdDisplay OS=Linux
target# gmake -s captureVip OS=Linux
```

等待编译完成后会在 `dvr_rdk_hdvpss` 目录下生成 `build` 文件夹，包含编译生成的可执行文件。