

# 我的 Beaglebone 开发之路

最近在开发 Beaglebone 板，在这里我分享一下自己的开发过程及经验，希望大家共同学习，共同进步。我的经验分享主要包括三个部分：Beaglebone 嵌入式平台搭建、D\_CAN 开发、MCSPi 开发。下面我将开始分享我的经验总结，自己的摸索难免有误，也希望各位专家以及开发者给予指导！

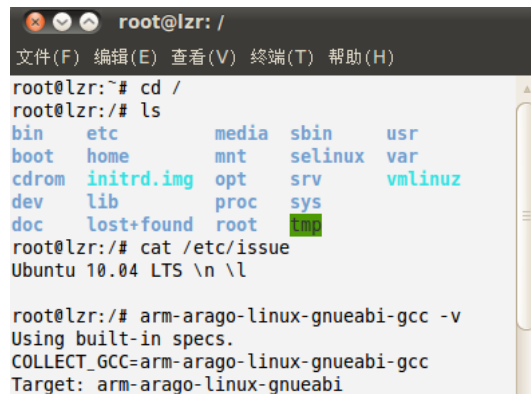
根据公司的开发需求，我们选择了 Beaglebone 板，鉴于它具有如下的优点：

- 1、采用 linux 系统开发，源代码开源，可以从官网上下载到相关的软件包；
- 2、Arm 的高普及率提供了很多可借鉴的资料；
- 3、BeagleBone 拥有 USB、Ethernet 和 JTAG 等完整的开发接口
- 4、BeagleBone 具丰富的外设接口，如 2 个 I2C 总线、DCAN 以及 SPI 等，为扩展开发提供了方便，提供了一个低成本，高扩展性的 ARM 平台。
- 5、BeagleBoard 社区是开放原始码硬件社群之一，可提供持续不断的支援。

## 1 beaglebone 嵌入式平台搭建

我使用的 beaglebone 板是 RevA5，虚拟机是 Ubuntu10.04。

开发的第一步就是搭建嵌入式平台，从最初的安装虚拟机到相应的应用包的安装等过程，在 Software Developer's Guide 里面都有详细的指导，该文档可以从官网下载到。Beaglebone 开发需要的交叉编译工具链是 arm-arago-linux-gnueabi-gcc，安装交叉编译工具是搭建平台最重要的一步。安装完成后在终端串口运行命令可查看相应的信息，如图所示：



```
root@lzt: /
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
root@lzt:~# cd /
root@lzt:/# ls
bin      etc          media       sbin        usr
boot    home        mnt         selinux     var
cdrom    initrd.img  opt         srv         vmlinuz
dev      lib          proc        sys
doc      lost+found  root        Emu
root@lzt:/# cat /etc/issue
Ubuntu 10.04 LTS \n \l

root@lzt:/# arm-arago-linux-gnueabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-arago-linux-gnueabi-gcc
Target: arm-arago-linux-gnueabi
```

买的 Beaglebone 板自带安装了 Angstrom 系统的 SD 卡，考虑到开发的需要，我自己重新制作了 335xevm 的系统。内核版本：[linux-3.1.0-psp04.06.00.03.sdk](#)，uboot 版本：[u-boot-2011.09-psp04.06.00.03](#)，参考文档是 [sitara-linuxsdk-sdg-05.03.03.00Software Developer's Guide](#)。

制作系统首先需要编译 uboot 和内核，得到需要的文件，也可以到 beagleboard 社区下载 <http://downloads.angstrom-distribution.org/demo/beaglebone/>。我自己重新编译了 uboot 及内核。

uboot 编译命令：

```
$ make CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm distclean //清空原有记录
```

```
$make0=am335xCROSS_COMPILE=arm-arago-linux-gnueabi-ARCH=armmake_target_from_table_above
```

编译完成可生成我们需要的 MLO 和 u-boot. img 文件。

kernel 编译命令：

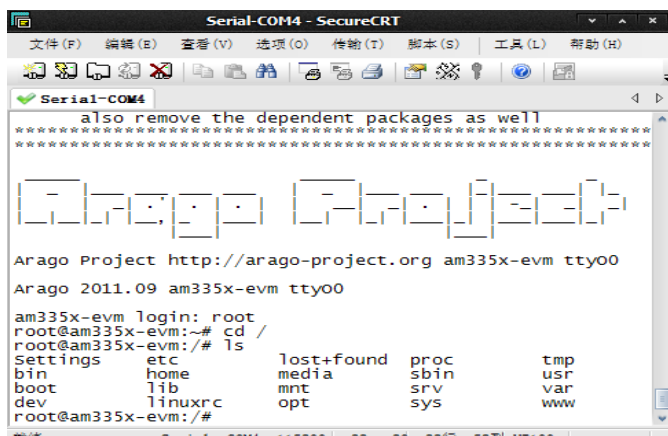
- (1) make ARCH=arm CROSS\_COMPILE=arm-arago-linux-gnueabi- mrproper
- (2) make ARCH=arm CROSS\_COMPILE=arm-arago-linux-gnueabi- tisdsk\_am335x-evm\_defconfig
- (3) make ARCH=arm CROSS\_COMPILE=arm-arago-linux-gnueabi- uImage

(4) make ARCH=arm CROSS\_COMPILE=arm-arago-linux-gnueabi- modules

在没有进行开发之前，采用默认方式进行编译，不用配置内核。编译完成后，在源码目录/arch/arm/boot/下可以生成我们需要的 uImage 文件。

最后一步就是制作 SD 卡映像，将已经编译好的文件 MLO、u-boot.img、uImage、以及 rootfs 压缩包放在一个文件夹下。SD 卡通过读卡器与 PC 机上的 ubuntu 虚拟机直接相连。打开虚拟机终端通过 mksdcard 命令来创建 SD 映像。

制作过程需要十分钟左右，完成后将 SD 卡插入板子，上电启动即可看到启动信息：



## 2 DCAN 的移植

DCAN 的移植包括激活驱动，扩展收发器，以及测试程序编译等。

### 2.1 激活驱动

根据 TI 的开发思路，TI 会将某些不用的功能屏蔽掉，在我们开发的过程中，我们只需要根据官网发布的各种 patch 包，找到我们移植需要添加的注册语句等就可激活驱动。

对于 DCAN 驱动激活，我只做了最基本的改动。在内核 mach-omap2 的 board-am335xevm.c 中进行修改，改动代码如下所示：

配置服用管脚：

```
static structpinmux_configd_can_bbs_pin_mux[] = {
    {"uart1_ctsn.d_can0_tx", OMAP_MUX_MODE2 | AM33XX_PULL_ENBL},
    {"uart1_rtsn.d_can0_rx", OMAP_MUX_MODE2 |
AM33XX_PIN_INPUT_PULLUP},
    {NULL, 0},
};

case LOW_COST_EVM:
    setup_pin_mux(d_can_bbs_pin_mux);
    /* Instance Zero */
    am33xx_d_can_init(0);
    break;

default:
    break;
}
```

在初始化命令中添加 DCAN

```
{usb0_init,      DEV_ON_BASEBOARD, PROFILE_NONE},
{usb1_init,      DEV_ON_BASEBOARD, PROFILE_NONE},
```

```

+     {mmc0_init,      DEV_ON_BASEBOARD, PROFILE_NONE},
      {d_can_init,    DEV_ON_BASEBOARD, PROFILE_NONE},
      {NULL, 0, 0},
};

```

DCAN1 和 DCAN0 的方式相同，修改完之后重新编译，在内核配置中选择 DCAN 驱动。

完成后，在 CRT 终端执行 `ifconfig -a` 命令可以看到两路 CAN。

```

Serial-COM4
root@am335x-evm:~# ifconfig -a
can0      Link encap:UNSPEC  Hwaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:52

can1      Link encap:UNSPEC  Hwaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:55

eth0      Link encap:Ethernet  Hwaddr 00:18:32:28:96:9A
          UP BROADCAST ALLMULTI MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0

```

## 2.2 扩展收发器

Beaglebone 板集成了 DCAN 设备，只有驱动器，没有收发器，我们需要外扩收发器，我采用带隔离的 CTM8251AT 作为收发器。相关电路的设计可以在很多网站上找到，这里就不细讲了，相当简单。

## 2.3 测试

CAN 的测试有 loopback, listen-only, tripple-sampling, one-shot, berr-reporting 多种方式，在两块板子通信中可以用测试程序 `cansend`、`candump`、`cansequence` 等进行测试。

下面是测试过程截图：

我采用 CAN0 发，CAN1 收的方式：

发：`cansend can0 -i 0xFFFF 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 -e --loop=20`

收：

```

root@am335x-evm:~# candump can1
interface = can1, family = 29, type = 3, proto = 1
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88
<0x000fffff> [8] 11 22 33 44 55 66 77 88

```

发：`cansend can0 -i 0x10 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88`

收：标准帧

```

root@am335x-evm:~# candump can1
interface = can1, family = 29, type = 3, proto = 1
<0x010> [8] 11 22 33 44 55 66 77 88

```

发：`cansend can0 -i 0x800 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 -e`

收：扩展帧

```

root@am335x-evm:~# candump can1
interface = can1, family = 29, type = 3, proto = 1
<0x00000800> [8] 11 22 33 44 55 66 77 88

```

### 3、MCSPI 开发

SPI 的移植于 DCAN 移植步骤相同，首先在内核 mach-omap2 的 board-am335xevm.c 中进行修改，在相应部分增加或修改以下程序：

```

1) static struct spi_board_info am335x_spi0_slave_info[] = {
    .modalias = "spidev",
    .max_speed_hz = 24000000,
    .bus_num = 1,
    .chip_select = 0,
    .mode = SPI_MODE_1,
},

```

```

2) static void spi0_init(int evm_id, int profile)
    setup_pin_mux(spi0_pin_mux);
    spi_register_board_info(bone_spi0_info,
        ARRAY_SIZE(bone_spi0_info));
    return;
}

```

3) beaglebone\_dev\_cfg[] struct 中增加：  
 {spi0\_init, DEV\_ON\_BASEBOARD, PROFILE\_NONE}

上面是针对 spi0 的修改，spi1 的修改相同。

重新编译内核，将 spi 编译进去。在 CRT 终端可以查看到 spi 设备

```

./sys/devices/platform/omap/omap2_mcspi.1/spi1.0/spidev
./sys/devices/platform/omap/omap2_mcspi.1/spi1.0/spidev/spidev1.0
./sys/devices/platform/omap/omap2_mcspi.2/spi2.0/spidev
./sys/devices/platform/omap/omap2_mcspi.2/spi2.0/spidev/spidev2.0
./sys/bus/spi/drivers/spidev
./sys/class/spidev
./sys/class/spidev/spidev1.0
./sys/class/spidev/spidev2.0
./sys/module/spidev
./dev/spidev2.0
./dev/spidev1.0
./dev/.udev/names/spidev2.0
./dev/.udev/names/spidev1.0
root@am335x-evm:~#

```

最后进行测试：

短接 spi0 的 D0 和 D1 端（在 P9 的 18 和 21 管脚）

```

root@am335x-evm:/opt# ./spidevtest -D /dev/spidev1.0
spi mode: 0
bits per word: 8
max speed: 500000 Hz (500 KHz)

FF FF FF FF FF FF
40 00 00 00 00 95
FF FF FF FF FF FF
FF FF FF FF FF FF
FF FF FF FF FF FF
DE AD BE EF BA AD
F0 0D
root@am335x-evm:/opt#

```

短接 spi1 的 D0 和 D1 端（在 P9 的 29 和 30 管脚）

```
root@am335x-evm:/opt# ./spidevtest -D /dev/spidev2.0
spi mode: 0
bits per word: 8
max speed: 500000 Hz (500 KHz)

FF FF FF FF FF FF
40 00 00 00 00 95
FF FF FF FF FF FF
FF FF FF FF FF FF
FF FF FF FF FF FF
DE AD BE EF BA AD
F0 0D
root@am335x-evm:/opt# █
```

测试通过！完成最基本的 SPI 开发。