



# **Linux Add-in for CrossCore Embedded Studio**

# Contents

1	Introduction	9
2	Quick Start Guide	10
2.1	SC5xx EZ-Kit Linux Quick Start Guide	10
2.1.1	Introduction	10
2.1.2	Environment set up	10
2.1.3	Flashing U-Boot for the First Time	14
2.1.4	Booting Linux Overview	18
2.1.5	Boot method 1: Copying the Linux image across the network	18
2.1.6	Boot method 2: Booting flashed Linux with persistent file system on SD Card	26
2.1.7	Boot Method 3: Booting the kernel from the SD Card (Quicker Boot)	30
2.1.8	Troubleshooting	32
2.2	Configure and build from source code	33
2.2.1	Introduction	33
2.2.2	Extract the source code	33
2.2.3	Configure and build u-boot	34
2.2.4	Configure and build Buildroot and Linux kernel	34
2.2.5	Customize the Buildroot and Linux kernel	35
2.2.6	Generate Kernel Image for Booting From Non-Volatile Storage	36
2.2.7	More tips about buildroot	37
3	Das U-boot	38
3.1	Ethernet Driver in U-Boot on SC5xx-EZKIT	38
3.1.1	Overview	38
3.1.2	Hardware Setup	38
3.1.3	Build U-Boot with emac0 or emac1	38
3.1.4	Using EMAC Driver	38
3.2	Creating and Booting Linux Using the New U-Boot "fitImage"	40
3.2.1	Introduction	40
3.2.2	Hardware Setup	40
3.2.3	Builroot and kernel Configuration	40
3.2.4	U-Boot Configuration	41
3.2.5	Create fitImage device tree	42
3.2.6	Boot Linux	45
3.3	Loading file from USB storage in u-boot	48
3.3.1	Overview	48
3.3.2	Hardware Setup	48
3.3.3	Test method	49
3.4	Mobile Storage Interface (MSI)	55
3.4.1	Overview	55
3.4.2	Hardware Setup	55
3.4.3	Software Configuration	56

3.4.4	Build and Load Uboot	56
3.4.5	Usage of MSI Driver	57
4	Linux buildroot distribution and applications	58
4.1	Bluetooth and headset audio example	58
4.1.1	Introduction	58
4.1.2	Hardware setup	58
4.1.3	Software configuration	58
4.1.4	Run headset audio example	59
4.2	Build application outside the buildroot framework	61
4.2.1	Introduction	61
4.2.2	Build Application outside Buildroot Framework	61
4.3	OProfile for performance benchmark	62
4.3.1	Introduction	62
4.3.2	Hardware Setup	62
4.3.3	Software Configuration	63
4.3.4	Simple OProfile usage	63
4.3.5	Examples	65
4.4	FTP Server (ftpd)	68
4.4.1	Introduction	68
4.4.2	Operation	68
4.5	Touchscreen Library	69
4.5.1	Introduction	69
4.5.2	Hardware Setup	69
4.5.3	Software Configuration	70
4.5.4	Example	71
4.6	MPlayer	74
4.6.1	Hardware Setup	74
4.6.2	Software Configuration	75
4.6.3	Build and Load Buildroot	76
4.6.4	Run MPlayer	76
4.7	IEEE 1588 and Linux PTP	77
4.7.1	PTP Introduction	77
4.7.2	PTP Configuration	78
4.7.3	Build	79
4.7.4	Example	79
4.7.5	More information	81
4.8	QT example for GUI	82
4.8.1	Introduction	82
4.8.2	Hardware Setup	82
4.8.3	Software Configuration	82
4.8.4	Build and Load Buildroot	85
4.8.5	Example	85
4.9	SQLite example for database	87
4.9.1	Introduction	87

4.9.2	Build and Load Buildroot	87
4.9.3	Run SQLite	88
4.10	Watchdog	88
4.10.1	Introduction	88
4.10.2	Software Configuration	89
4.10.3	Build and Load Buildroot	90
4.10.4	Test	90
5	Linux kernel and drivers	94
5.1	CAN Bus Data Transaction	94
5.1.1	Introduction	94
5.1.2	Hardware Setup	94
5.1.3	Software Configuration	94
5.1.4	Test Example	95
5.2	CPU Frequency utility	96
5.2.1	Introduction	96
5.2.2	Hardware Setup	96
5.2.3	Enabling CPU Frequency Driver in Linux Kernel	96
5.2.4	How to Change the CPU cpufreq	97
5.2.5	Change Core Clock Frequency via cpufreq-utils	99
5.3	CRC Crypto Driver Guide	100
5.3.1	Introduction	100
5.3.2	Hardware Setup	101
5.3.3	Software Configuration	101
5.3.4	Example	101
5.3.5	Linux Kernel Crypto API	102
5.4	Ethernet driver and performance	102
5.4.1	Introduction	102
5.4.2	Hardware Requirement	102
5.4.3	Software Configuration	102
5.4.4	Performance Benchmark Example	104
5.5	Linux LCD device driver	105
5.5.1	Introduction	105
5.5.2	Hardware Setup	105
5.5.3	Software Configuration	106
5.5.4	Example	107
5.6	Linux Sound Driver	108
5.6.1	Introduction	108
5.6.2	Hardware Setup	108
5.6.3	Software Configuration	109
5.6.4	Example	111
5.7	GPIO operation	115
5.7.1	Introduction	115
5.7.2	Hardware	115
5.7.3	Application space GPIO support	116

5.7.4	Paths in Sysfs	116
5.7.5	Dump the GPIO configuration	118
5.8	I2C Bus	118
5.8.1	Introduction	118
5.8.2	I2C in the Linux Kernel	119
5.8.3	I2C in User Space	121
5.9	Link Port driver	123
5.9.1	Introduction	123
5.9.2	Hardware Setup	123
5.9.3	Software configuration	124
5.9.4	Test Example	124
5.10	USB interface	125
5.10.1	Introduction	125
5.10.2	Hardware Setup	125
5.10.3	Software Configuration	125
5.10.4	USB HOST Example	126
5.11	General Bluetooth Dongle via USB	133
5.11.1	Introduction	133
5.11.2	Hardware Setup	133
5.11.3	Software Configuration	133
5.11.4	Example	134
5.12	Kernel API for DMA operation	137
5.12.1	Introduction	137
5.12.2	Linux DMA Framework	137
5.13	Linux MTD Driver	143
5.13.1	Introduction	143
5.13.2	Hardware Required	144
5.13.3	Software Configuration	144
5.13.4	Example	146
5.14	Linux PCIE device driver	147
5.14.1	Introduction	147
5.14.2	Hardware Setup	147
5.14.3	Software Configuration	147
5.14.4	Example	149
5.15	Linux Video Driver	150
5.15.1	Introduction	150
5.15.2	Hardware Required	151
5.15.3	Software Configuration	154
5.15.4	Example	159
5.16	Rotary driver	163
5.16.1	Introduction	163
5.16.2	Hardware Setup	163
5.16.3	Software Configuration	163
5.16.4	Example	165

5.17	SPI Driver	167
5.17.1	Introduction	167
5.17.2	Hardware Setup	169
5.17.3	Software Configuration	169
5.17.4	Example	172
5.18	Mobile Storage Interface for MMC/SD	172
5.18.1	Hardware Setup	172
5.18.2	Software Configuration	173
5.18.3	Build and Load Buildroot	174
5.18.4	Usage of MSI	174
5.19	Real Time Clock on ADSP-SC58x	178
5.19.1	Introduction	178
5.19.2	Hardware Required	178
5.19.3	Software Configuration	178
5.19.4	Build and Load Buildroot	179
5.19.5	Example	179
5.20	UART hardware flow control	180
5.20.1	Introduction	180
5.20.2	Hardware Setup	180
5.20.3	Software Configuration	180
5.20.4	Build and Load Buildroot	181
5.20.5	Example	181
6	Multicore support	182
6.1	Multi-Core Communication	182
6.1.1	MCAPI Supported Functions	182
6.1.2	Multi-core Development	184
6.2	Run the MCAPI demo	184
6.2.1	Introduction	184
6.2.2	Hardware Requirement	185
6.2.3	Configure and build	185
6.2.4	Running MCAPI MSG Test Example	188
6.2.5	Troubleshooting	190
6.3	Enable and Disable SHARC Cores	191
6.3.1	Introduction	191
6.3.2	Method 1: Enable SHARC cores with u-boot ICC command	191
6.3.3	Method 2: Enable SHARC Cores with corecontrol Utility in Linux	192
6.4	Introduction of MCAPI examples	194
6.4.1	Introduction	194
6.4.2	Performance of MCAPI	196
6.5	Run Linux on ARM and bare-metal application on SHARC	197
6.5.1	Introduction	197
6.5.2	MCAPI Test Example	197
6.6	Reserve Pinmux Functions in Linux for SHARC Applications	205
6.6.1	Introduction	205

6.6.2	Linux Kernel Configuration	205
6.6.3	Pinmux Reservation Example	207
6.7	SEC driver and multicore development	211
6.7.1	Introduction	211
6.7.2	Linux kernel Configuration	211
6.7.3	Coordinate the SEC Initialization for multicore Development	212
7	Generic Linux Documents	215
7.1	Generic U-Boot Documents	215
7.2	Generic Linux Kernel Documents	215
7.3	Generic Buildroot Documents	215
8	Developing Linux Applications Using CrossCore Embedded Studio	216
8.1	Create and build a project using the Linux targeting toolchain	216
8.1.1	Install CCES and Linux Add-in on Linux host	216
8.1.2	Start CCES IDE on Linux Host	216
8.1.3	Creating a new project using CCES	216
8.2	Debug a Linux application from within the CCES IDE	220
9	Appendix A: Hardware Used During Testing	224

## **Copyright Information**

© 2017 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

## **Disclaimer**

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

## **Trademark and Service Mark Notice**

The Analog Devices logo, CrossCore, A<sup>2</sup>B, EngineerZone, EZ-Board, EZ-KIT Lite and VisualDSP++ are registered trademarks of Analog Devices, Inc. Blackfin, Blackfin+, SHARC, SHARC+ and SigmaStudio are trademarks of Analog Devices, Inc. All other brand and product names are trademarks or service marks of their respective owners.



# 1 Introduction

The Analog Devices ADSP-SC5xx family is a series of heterogeneous multi-core processors. These systems-on-a-chip contain a single ARM Cortex-A5 processor core accompanied by one or two Analog Devices SHARC+ digital signal processor cores and a wide variety of memory and peripheral interfaces. The ADSP-SC5xx family provides ADI customers with rich and flexible software stacks on ARM and high performance DSP audio processing codecs on SHARC. The Linux Add-In for CrossCore Embedded Studio provides users with a full development environment allowing them to target Linux running on the Cortex-A5 core of the ADSP-SC5xx processor.

The Linux Add-In is only available for Linux hosts. It is not available for Windows host systems.

The Linux Add-In provides the following features:

- Analog Devices enhanced Linaro GCC compiler toolchain
- GDB debugger for Linux applications
- Eclipse-based CrossCore Embedded Studio integrated development and debugging environment for Linux applications
- Pre-configured Linux images including u-Boot, Kernel and filesystem for the ADSP-SC5xx EZ-Kits(including ADSP-SC589, ADSP-SC584 and ADSP-SC573)
- Sources and build environment for U-Boot bootloader
- Sources and 'Buildroot' build environment for Linux kernel and filesystem
- Documentation specific to Linux on the ADSP-SC5xx family

## 2 Quick Start Guide

### 2.1 SC5xx EZ-Kit Linux Quick Start Guide

#### 2.1.1 Introduction

The Analog Devices release of the Linux Add-In for CrossCore Embedded Studio provides both sources and pre-built binaries for the u-Boot bootloader, linux kernel and filesystem.

This Getting Started Guide begins by providing instructions that allow users to quickly deploy pre-built images on the ADSP-SC5xx EZ-Kits in order to quickly evaluate Linux on the ADSP-SC5xx. The document then follows up by providing step-by-step instruction for building and installing u-Boot and Linux from source.

It is strongly recommended that the audience has a basic grounding in embedded Linux system development. Some useful background reading may be found at <http://elinux.org/>

#### 2.1.2 Environment set up

##### Configure the Linux Host Machine

The Linux Add-In tools need to be installed on a host PC running a supported version of Linux. The supported distribution is Ubuntu x86 32-bit, version 14.04, however experienced Linux users will find few issues when installing on newer versions of Ubuntu.

##### **i** Running via virtual machine

While it is possible to use this product when installed on a virtual machine running Linux, we have encountered issues when connecting to some EZ-Kits when the developer is running on a virtual machine. Problems can occur on a machine by machine basis and may not be consistently reproducible.

We strongly recommend that this product is installed on a native PC rather than a virtual machine.

Networking, both local and internet-wide will be required to complete the steps below.

Several packages are required on the host machine in order to complete. These can be installed on your host system with the following command:

```
$ sudo apt-get install build-essential minicom tftpd-hpa git-all
subversion openssh-server ncurses-dev
```

## Installing the Linux Add-In

Download and install the latest versions of the CrossCore Embedded Studio and Linux Add-In for CrossCore Embedded Studio.

Both products are provided as Linux Debian (.deb) packages.

Both packages are available from <http://www.analog.com/en/design-center/processors-and-dsp/evaluation-and-development-software/LinuxAddin.html>

Install the packages by executing the commands below. Note that the version numbers of the packages may differ from those shown on the command lines below.

```
$ sudo dpkg -i adi-CrossCoreEmbeddedStudio-linux-x86-2.6.0.deb
$ sudo dpkg -i adi-LinuxAddinForCCES-linux-x86-1.2.0.deb
```

The products will be installed in the /opt/analog directory on your host system.

## Useful file locations

The files and directories below are required for the following sections:

Location	Description
/opt/analog/cces/2.6.0/ARM/arm-none-eabi	ARM bare-metal toolchain
/opt/analog/cces-linux-add-in/1.2.0/ARM/arm-linux-gnueabi	ARM Linux-targeting toolchain
/opt/analog/cces-linux-add-in/1.2.0/uboot-sc5xx-1.2.0/bin/init-sc589-ezkit.elf	Init ELF, needed for JTAG loading
/opt/analog/cces-linux-add-in/1.2.0/uboot-sc5xx-1.2.0/bin/u-boot-sc589-ezkit	U-Boot ELF, needed for JTAG loading
/opt/analog/cces-linux-add-in/1.2.0/uboot-sc5xx-1.2.0/bin/u-boot-sc589-ezkit.ldr	U-Boot LDR file, for SPI boot

Location	Description
/opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0 /images/sc589-ezkit.dtb	Board DTB file for Linux booting
/opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0 /images/uImage-sc589-ezkit	Boot image file, including kernel and RAM rootfs
/opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0 /images/vmImage-sc589-ezkit	Linux kernel file
/opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0 /images/rootfs.ext3	Root filesystem image

Here we take ADSP-SC589-EZKIT board (ADSP-SC589 EZ-Board) as our working example. For the SC584-EZKIT board and SC573-EZKIT board the steps required are similar except for a different file /dtb load address and different file names in some cases which are showing as follows.

Board	File name	File load address	dtb name	dtb load address
ADSP-SC589 EZKIT	uImage-sc589-ezkit	0xC2000000	sc589-ezkit.dtb	0xC4000000
ADSP-SC584 EZKIT	uImage-sc584-ezkit	0x89000000	sc584-ezkit.dtb	0x8b000000
ADSP-SC573 EZKIT	uImage-sc573-ezkit	0x82000000	sc573-ezkit.dtb	0x84000000

We frequently use variables directly that u-boot console recognizes in following sections, for example use \${loadaddr} instead of 0xC2000000 for ADSP-SC589 EZKIT.

## Set Up the TFTP Server

The ADSP-SC5xx EZ-Kit do not ship with Linux or the u-Boot bootloader pre-installed. In order to transfer files to the EZ-Kit we use TFTP. TFTP is a simplified file transfer protocol that is supported by u-Boot.

The following commands will configure and start a TFTP server on your Linux host:

```
$ sudo mkdir /tftpboot
$ sudo chmod -R 777 /tftpboot
```

Edit the `/etc/default/tftpd-hpa` file. Sudo is required to write the contents of the file. Change the contents to be the following:

```
TFTP_USERNAME="tftp"  
TFTP_DIRECTORY="/tftpboot"  
TFTP_ADDRESS="0.0.0.0:69"  
TFTP_OPTIONS="--secure"
```

Now restart the TFTP service:

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

## Configure the Toolchains

Development for Linux requires the use of two toolchains.

The **Bare Metal** toolchain is provided as part of CrossCore Embedded Studio. It produces bare metal applications that run directly on the hardware without an underlying operating system.

The bare metal toolchain is used to build the u-Boot bootloader.

The **Linux Targeting** toolchain is provided as part of the Linux Add-In for CrossCore Embedded Studio. It produces applications that run under Linux on the ADSP-SC5xx EZ-Kits.

The following instructions in the User Guide assume that the **bin** directories for both toolchains are included in your path.

To add the paths to your **PATH** environment variable, add the following line to the end of the `~/ .bashrc` file:

```
export PATH=$PATH:/opt/analog/cces-linux-add-in/1.2.0/ARM/arm-linux-gnueabi/bin:/opt/analog/cces/2.6.0/ARM/arm-none-eabi/bin
```

Then run the following command, to update your path in your current shell:

```
$ source ~/.bashrc
```

## Set Up Hardware

To complete this step you will require:

- ADSP-SC5xx EZ-Kit: ADSP-SC589/SC584/SC573 EZ-Board, ICE1000 or ICE2000 emulator, 2x micro USB cable (as provided with the EZ-Kit), 1x ethernet cable

- Host PC running the supported version of Linux (Ubuntu 14.04 LTS, 32-Bit)

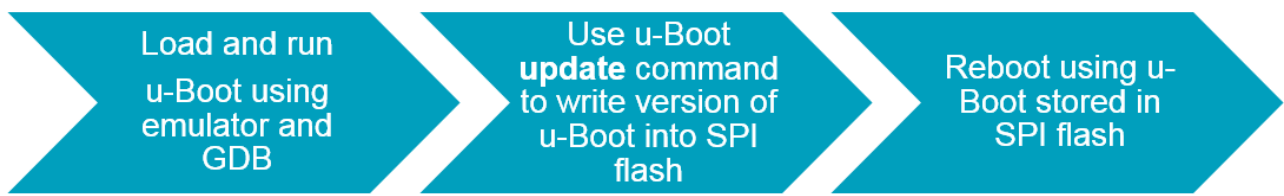
The EZ-Kit should have four connections:

- Power cable
- ICE-1000 or ICE-2000 connected to the Linux Host
- MicroUSB-to-USB cable connected from the EZ-Kit's USB-to-UART port to the Linux Host
- Ethernet cable connected to the 10/100/1000 Ethernet port of the board, another to the same local network as Linux Host

### 2.1.3 Flashing U-Boot for the First Time

Before booting Linux we need to use the **ICE-1000** or **ICE-2000** to load the U-Boot bootloader on to the EZ-Kit.

Once U-Boot is loaded and running, we use it to write another copy of u-Boot into non-volatile SPI Flash.



### Run OpenOCD

```
$ cd /opt/analog/cces/2.6.0/ARM/openocd/share/openocd/scripts
$ sudo su
$ /opt/analog/cces/2.6.0/ARM/openocd/bin/openocd -f interface/ice1000.
cfg -f target/adspsc58x.cfg
```

When successful you should see a message similar to the console output below:

```
Open On-Chip Debugger (Analog Devices CCEs 2.6.0 OpenOCD 0.9.0-
g6aca937) 0.9.0
Licensed under GNU GPL v2
Report bugs to <processor.tools.support@analog.com>
adapter speed: 1000 kHz
Info : auto-selecting first available session transport "jtag". To
override use 'transport select <transport>'.
halt and restart using CTI
trst_only separate trst_push_pull
Info : ICE-1000 firmware version is 1.0.2
Info : clock speed 1000 kHz
Info : JTAG tap: adspsc58x.adjc tap/device found: 0x228080cb (mfg: 0x06
5, part: 0x2808, ver: 0x2)
Info : JTAG tap: adspsc58x.dap enabled
Info : adspsc58x.dap: hardware has 3 breakpoints, 2 watchpoints
Info : adspsc58x.dap: but you can only set 1 watchpoint
```



If you are working from a virtual machine, make sure the ICE1000 or ICE2000 usb device on your computer is passed-through to the virtual machine.

For example, if you are using VirtualBox, Select the **USB Devices** option from the **Devices** menu. Ensure that the **Analog Devices Inc. Blackfin USB Device** is checked.

If you have problem with this step please refer to "**Trouble Shooting**" section of this chapter.

## U-Boot Console Output

The output from u-Boot is transmitted to the host PC using the micro USB cable connected from the Host PC to the USB-to-UART port of the EZ-Kit.

The following instructions in this guide use the **minicom** application to interact with the serial port.

Here we need set up **minicom** to the ADSP-SC5xx serial console. Execute the following commands on the host PC:

1. `sudo minicom -s`
2. Within minicom: "Serial port setup"
3. Select **Serial port setup**
  - a. Set **Serial Device** to `/dev/ttyUSB0`
  - b. Set **Bps/Par/Bits** to `57600 8N1`
  - c. Set **Hardware Flow Control** to **No**
  - d. Close the Serial port setup option

4. Select **Save setup as dfl**
5. Select **Exit**

Once **minicom** has been configured, launch the application:

```
$ sudo minicom
```



If you are running your Host PC within a virtual machine you will need to configure the virtual machine to allow access to USB-to-UART connection.

### Loading U-Boot With GDB

Open a second terminal on the host PC and enter the following commands:

```
$ cd /opt/analog/cces-linux-add-in/1.2.0/u-boot-sc5xx-1.2.0/bin
$ arm-none-eabi-gdb u-boot-sc589-ezkit
(gdb) target remote :3333
```

Next, load the **init stub**. This is an application that initializes the CPU and DDR on the EZ-Kit:

```
(gdb) load init-sc589-ezkit.elf
(gdb) c
<Ctrl + C>
(gdb)
```

Load u-boot and run:

```
(gdb) load u-boot-sc589-ezkit
(gdb) c
```

U-Boot will now be running in RAM on your target board. In the terminal running Minicom you will see the output from the Minicom console.

### Flash U-Boot to SPI Flash

Here we use the u-Boot console to TFTP a version of u-Boot into RAM, and then write this application into SPI flash.

On the Host PC:



```
$ cd /opt/analog/cces-linux-add-in/1.2.0/u-boot-sc5xx-1.2.0/bin
$ cp u-boot-sc589-ezkit.ldr /tftpboot/u-boot.ldr
```

Ensure that the EZ-Kit board is connected to the same LAN network as your Host PC

Execute the following commands in the minicom terminal connected to the EZ-Kit:

```
$ dhcp
$ set serverip $your_host_ip
$ run update
$ save
```

**i** It may take a few seconds to get the IP address with DHCP. Do not interrupt the request. Writing the u-Boot image to SPI flash may also take some time.

Now you should have U-Boot flashed successfully to the on board SPI Flash chip. Switch the boot mode to SPI Master boot (**BMODE** switch set to position 1).

From this point you do not need to use the ICE emulator unless u-Boot fails to boot. If it does repeat the steps above to re-flash a new version of u-Boot.

Remove power from the EZ-Kit, and remove the ICE emulator from the board.

When you connect the power to the EZ-Kit, u-Boot should boot and the output should be visible in the minicom:

```
U-Boot 2015.01 ADI-1.2.0-00103-ge854404-dirty (Jun 07 2017 - 07:43:07)
CPU: ADSP ADSP-SC589-0.1 (Detected Rev: 1.1) (spi flash boot)
VCO: 450 MHz, Cclk0: 450 MHz, Sclk0: 112.500 MHz, Sclk1: 112.500 MHz,
DCLK: 450 MHz
OCLK: 150 MHz
I2C: ready
DRAM: 224 MiB
MMC: SC5XX SDH: 0
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB,
total 16 MiB
In: serial
Out: serial
Err: serial
other init
Net: dwmac.3100c000
Hit any key to stop autoboot
```

## 2.1.4 Booting Linux Overview

### Supported methods for booting Linux

The Linux Add-In supports three methods for booting Linux:

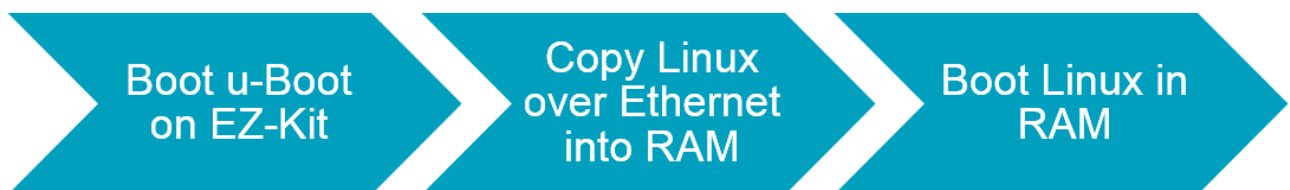
1. u-Boot bootloader downloads the Linux kernel and filesystem from a local TFTP server and then boots this image.  
This is the version of Linux provided in the pre-built images provided in the Linux Add-In.
2. u-Boot copies a version of Linux out of SPI flash to RAM and then boots using a file system located on an SD Card connected to the EZ-Kit
3. u-Boot boots Linux using a kernel located on the SD Card, along with a file system located on the SD Card



The pre-built binary files that are shipped as part of the Linux Add-In support booting using all these 3 boot methods. If you want to use your local binary from your own source, please refer to [Configure and build from source code](#).

### 2.1.5 Boot method 1: Copying the Linux image across the network

This method of booting Linux copies the kernel and filesystem from the Host PC to the target EZ-Kit via Ethernet. This method of booting is useful for development as it is easy to set up. However, since the filesystem is not permanently stored on the EZ-Kit, and it relies on connection to the TFTP server, this method is not suitable for production.



First we copy the pre-built Linux image to the TFTP directory. We copy two files onto the board. The Linux image itself (uImage) and a .dtb (device tree blob) file which contains information about the hardware.

On the Host PC:





```

Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:   26154400 Bytes = 24.9 MiB
Load Address: c2008000
Entry Point:  c2008000
Verifying Checksum ... OK
## Flattened Device Tree blob at c4000000
   Booting using the fdt blob at 0xc4000000
   Loading Kernel Image ... OK
   Loading Device Tree to cfe5b000, end cfe62853 ... OK
Starting kernel ...
Booting Linux on physical CPU 0x0
Linux version 4.0.0-ADI-1.2.0-00059-g27e643c (test@cindy.ad.analog.com) (gcc version 4.8.3 (Analog Devices Inc. ARM Tools (d9f69c572e3368746d6d95caf93dae57c9a689d4)). Distri7
CPU: ARMv7 Processor [410fc051] revision 1 (ARMv7), cr=10c53c7d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: ADI sc589-ezkit
bootconsole [earlycon0] enabled
Memory policy: Data cache writeback
dump init clock rate
CGU0_PLL 450 MHz
CGU0_SYSCLK 225 MHz
CGU0_CCLK 450 MHz
CGU0_SYS0 112 MHz
CGU0_DCLK 450 MHz
CGU0_OCLK 150 MHz
CGU0_SYS0 112 MHz
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 56896
Kernel command line: root=/dev/mtdblock2 rw rootfstype=jffs2 clk_in_hz=(25000000) earlyprintk=serial,uart0,57600 console=ttySC0,57600 mem=224M ip=10.100.4.50:10.100.4.174:19f
PID hash table entries: 1024 (order: 0, 4096 bytes)
Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
Memory: 201596K/229376K available (3820K kernel code, 125K rwdma, 1428K rodata, 20164K init, 88K bss, 27780K reserved, 0K cma-reserved)
Virtual kernel memory layout:
   vector   : 0xffff0000 - 0xffff1000   ( 4 kB)
   fixmap   : 0xffc00000 - 0xfff00000   (3072 kB)
   vmalloc  : 0xce800000 - 0xff000000   ( 776 MB)
   lowmem   : 0xc0000000 - 0xce000000   ( 224 MB)
   modules  : 0xbf000000 - 0xc0000000   ( 16 MB)
   .text    : 0xc0008000 - 0xc0528528   (5250 kB)
   .init    : 0xc0529000 - 0xc18da000   (20164 kB)
   .data    : 0xc18da000 - 0xc18f95a0   ( 126 kB)
   .bss     : 0xc18f95a0 - 0xc190f92c   ( 89 kB)
NR_IRQS:16 nr_irqs:16 16
GIC CPU mask not found - kernel will fail to boot.
GIC CPU mask not found - kernel will fail to boot.

```

```

sched_clock: 32 bits at 112MHz, resolution 8ns, wraps every
38177486839ns
Console: colour dummy device 80x30
Calibrating delay loop... 297.98 BogoMIPS (lpj=595968)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0xc23a2bd0 - 0xc23a2c04
devtmpfs: initialized
do_initcall_level level 0
do_initcall_level level 1
VFP support v0.3: implementor 41 architecture 2 part 30 variant 5 rev 1
pinctrl core: initialized pinctrl subsystem
NET: Registered protocol family 16
do_initcall_level level 2
DMA: preallocated 256 KiB pool for atomic coherent allocations
do_initcall_level level 3
L2C: device tree omits to specify unified cache
L2C-310 dynamic clock gating enabled, standby mode enabled
L2C-310 cache controller enabled, 8 ways, 256 kB
L2C-310: CACHE_ID 0x410000c9, AUX_CTRL 0x06040000
sc58x_init: registering device resources
sec init...
enabled
hw-breakpoint: Failed to enable monitor mode on CPU 0.
ADI DMA2 Controller
do_initcall_level level 4
vgaarb: loaded
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
i2c-bfin-twi 31001400.twi: Blackfin on-chip I2C TWI Contoller,
regs_base@f4001400
i2c-bfin-twi 31001500.twi: Blackfin on-chip I2C TWI Contoller,
regs_base@f4001500
i2c-bfin-twi 31001600.twi: Blackfin on-chip I2C TWI Contoller,
regs_base@f4001600
pps_core: LinuxPPS API ver. 1 registered
pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti
<giometti@linux.it>
PTP clock support registered
Advanced Linux Sound Architecture Driver Initialized.
do_initcall_level level 5
Switched to clocksource cs_gptimer
NET: Registered protocol family 2
TCP established hash table entries: 2048 (order: 1, 8192 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP: reno registered

```

```

UDP hash table entries: 256 (order: 0, 4096 bytes)
UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
NET: Registered protocol family 1
do_initcall_level level 6
hw perfevents: enabled with armv7_cortex_a5 PMU driver, 3 counters
available
futex hash table entries: 256 (order: -1, 3072 bytes)
jffs2: version 2.2. (NAND) 2001-2006 Red Hat, Inc.
External imprecise Data abort at addr=0xb6f4c880, fsr=0x1c06 ignored.
io scheduler noop registered (default)
APP_STAT=0,RSCKPHY_STAT=20000,LINK_CTRL_STATUS_REG2=10001
sc58x-pcie 310b8000.pcie: Link training failed
sc58x-pcie 310b8000.pcie: PORT_LINK_DEBUG0 = 85c700, PORT_LINK_DEBUG1
= 8200000
sc58x-pcie: probe of 310b8000.pcie failed with error -110
ADI serial driver
adi-uart4.0: ttySC0 at MMIO 0x31003000 (irq = 20, base_baud = 7031250)
is a ADI-UART4
console [ttySC0] enabled
console [ttySC0] enabled
bootconsole [earlycon0] disabled
bootconsole [earlycon0] disabled
adi-uart4.2: ttySC2 at MMIO 0x31003800 (irq = 84, base_baud = 7031250)
is a ADI-UART4
loop: module loaded
adi-spi3 31042000.spi: registered ADI SPI controller spi0
adi-spi3 31043000.spi: registered ADI SPI controller spi1
m25p80 spi2.38: found w25q128, expected w25q32
m25p80 spi2.38: w25q128 (16384 Kbytes)
3 ofpart partitions found on MTD device spi2.38
Creating 3 MTD partitions on "spi2.38":
0x000000000000-0x000000080000 : "uboot (spi)"
0x000000080000-0x000000600000 : "kernel (spi)"
0x000000600000-0x000001000000 : "root file system (spi)"
adi-spi3 31044000.spi: registered ADI SPI controller spi2
libphy: Fixed MDIO Bus: probed
CAN device driver interface
bfin_can 31000200.can: bfin_can device registered(&reg_base=f4000200,
rx_irq=22, tx_irq=23, err_irq=24, sclk=112500000)
bfin_can 31000a00.can: bfin_can device registered(&reg_base=f4000a00,
rx_irq=93, tx_irq=94, err_irq=95, sclk=112500000)
adi-spi3 31042000.spi: chipselect 44 already in use
bfin_can 31000a00.can: Can't start can phy
Trying to free nonexistent resource <0000000031000a00-0000000031000ffe>
bfin_can: probe of 31000a00.can failed with error -22
stmmaceth 3100c000.ethernet: no reset control found
stmmac - user ID: 0x10, Synopsys ID: 0x37
Ring mode enabled
DMA HW capability register supported
Enhanced/Alternate descriptors
Enabled extended descriptors

```

```

RX Checksum Offload Engine supported (type 2)
TX Checksum insertion supported
Wake-Up On Lan supported
Enable RX Mitigation via HW Watchdog Timer
libphy: stmmac: probed
eth0: PHY ID 20005c7a at 1 IRQ POLL (stmmac-0:01) active
force_sf_dma_mode is ignored if force_thresh_dma_mode is set.
stmmaceth 3100e000.ethernet: no reset control found
stmmac - user ID: 0x11, Synopsys ID: 0x37
Ring mode enabled
DMA HW capability register supported
Enhanced/Alternate descriptors
    Enabled extended descriptors
RX Checksum Offload Engine supported (type 2)
Wake-Up On Lan supported
Enable RX Mitigation via HW Watchdog Timer
libphy: stmmac: probed
eth1: PHY ID 20005c90 at 1 IRQ POLL (stmmac-1:01) active
usbcore: registered new interface driver usb-storage
musb-hdrc musb-hdrc.1.auto: MUSB HDRC host driver
musb-hdrc musb-hdrc.1.auto: new USB bus registered, assigned bus
number 1
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
musb-hdrc musb-hdrc.3.auto: MUSB HDRC host driver
musb-hdrc musb-hdrc.3.auto: new USB bus registered, assigned bus
number 2
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 1 port detected
mousedev: PS/2 mouse device common for all mice
rtc (null): invalid alarm value: 1900-1-15 0:0:0
rtc-adi2 310c8000.rtc: rtc core: registered 310c8000.rtc as rtc0
i2c /dev entries driver
adi_wdt: initialized: timeout=20 sec (nowayout=0)
Driver 'mmcblk' needs updating - please use bus_type methods
Synopsys Designware Multimedia Card Interface Driver
dwmmc_adi mmc.0: num-slots property not found, assuming 1 slot is
available
dwmmc_adi mmc.0: IDMAC supports 32-bit address mode.
dwmmc_adi mmc.0: Using internal DMA controller.
dwmmc_adi mmc.0: Version ID is 270a
dwmmc_adi mmc.0: DW MMC controller at irq 102, 32 bit host data width,
1024 deep fifo
dwmmc_adi mmc.0: No vmmc regulator found
dwmmc_adi mmc.0: No vqmmc regulator found
dwmmc_adi mmc.0: 1 slots initialized
Blackfin hardware CRC crypto driver
bfin-hmac-crc 31001200.crc: initialized
bfin-hmac-crc 31001300.crc: initialized
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver

```









## Formatting the SD Card

In order to use an SD Card with Linux we need to prepare it by formatting it and copying over the file system contents, in the correct format.

The SD Card provided with the EZ-Kit has been tested and is suitable for use with Linux.

### ⚠ Danger! Disk Formatting!

This section of instructions requires you to correctly identify the SD Card and format the card. If you select the wrong drive you may cause irreversible damage to your Host PC. Please be confident in your actions before proceeding with this section.

To begin installing the file system, insert the SD Card into the host PC directly or via a USB card reader and identify the device which refers to your SD Card.

### ℹ Identifying the SD Card on the HOST PC

The easiest way to identify the SD Card is to view the latest information logged to the device message log.

This can be done by executing the command: **dmesg | tail**

The output from the command should mention a newly connected device. The partition name should be listed and should be of the format **/dev/sd[a-z][0-9]**.

For example the partition **/dev/sdb1** would be a partition on the device **/dev/sdb**.

To double check that this is the correct device you can use the **fdisk** command to display information regarding the device, including the size of partitions.

To view information on the device, issue the command: **fdisk -l <device>** where device is some device, for example **/dev/sdb**.

### ❗ Unmount the SD Card before starting

By default, Ubuntu will auto-mount any SD Card that is inserted. Before formatting the SD Card you will need to un-mount the card.

To check whether your SD Card has been mount, issue the **mount** command and search for your device in the output of the command.

You can then un-mount the command using the unmount command. For example if the SD Card was inserted and contained a /dev/sdb1 partition that was mount at /media/testuser/sd\_card, you can unmount it by issuing the command:

```
sudo unmount /media/testuser/sd_card
```

To format the SD Card, follow the commands below. The example code in this section assumes that the SD Card device is reported to be /dev/sdb. Ensure that you change these commands to use your device.

```
$ sudo fdisk /dev/sdb
/* Create primary partition 1, 256M size*/
Command (m for help): n
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-3887103, default 2048): PRESS ENTER
Last sector, +sectors or +size{K,M,G} (2048-3887103, default 3887103):
PRESS ENTER

/* Save partition */
Command (m for help): w
```

Once fdisk is closed, your Host PC may mount the SD Card again. If it does, un-mount it following the instructions above.

### Writing the file system to the SD Card

Next, we need to copy the pre-built Linux file system to the SD Card. We install this on to the SD Card by mounting the file system on to the local Host PC and copying the contents on to the SD Card.

```
$ sudo mkfs.ext3 /dev/sdb1
$ sudo mount -t ext3 /dev/sdb1 /mnt
$ sudo mkdir ~/sc5xx_rootfs
$ sudo mount /opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0/
images/rootfs.ext3 ~/sc5xx_rootfs
$ sudo cp ~/sc5xx_rootfs/* -fr /mnt
$ sudo umount /mnt
```

The file system is now installed on to the SD Card. The SD Card can now be safely removed from the Host PC.

### Loading and flashing the DTB File

Before proceeding ensure that u-Boot has been flashed into SPI Flash on the EZ-Kit as per the instructions in the section **Flashing U-Boot for the First Time**. Ensure that the **serverip** variable is correctly set in the u-Boot environment.

On the Host PC:

```
$ cp /opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0/images
/sc589-ezkit.dtb /tftpboot/
```

Using the minicom terminal:

```
$ sf probe 2:1
$ tftp ${loadaddr} sc589-ezkit.dtb
$ sf erase 0x100000 0x40000
$ sf write ${loadaddr} 0x100000 ${filesize}
```

### Loading and flashing the Linux kernel

On the Host PC:

```
$ cp /opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0/images
/vmImage-sc589-ezkit /tftpboot/
```

Using the minicom terminal:

```
$ tftp ${loadaddr} vmImage-sc589-ezkit
$ sf erase 0x800000 0x800000
$ sf write ${loadaddr} 0x800000 ${filesize}
```

## Configure u-Boot to use the SD Card file system

Next we need to configure u-Boot to boot using the SD Card as our file system.

Issue the commands below to change the default boot behaviour of u-Boot to always boot using the SPI flash kernel and SD Card file system.

```
$ set sdargs set bootargs root=/dev/mmcblk0p1 rw rootwait
rootfstype=ext3 clk_in_hz=(2500000) earlyprintk console=ttySC0,57600
$ set sdboot 'run sdargs; sf probe 2:1;sf read ${dtbaddr} 0x100000
0x40000;sf read ${loadaddr} 0x800000 0x800000;bootm ${loadaddr} -
${dtbaddr}'
$ set bootcmd run sdboot
$ save
```

To complete the process press the **reset** button on the EZ-Kit.

### **i** Resorting to Ethernet booting

If for some reason you wish to revert to booting via Boot mode 1: Copying Linux over Ethernet, you can issue the following commands in uBoot:

```
$ set bootcmd run ramboot
$ save
```

## 2.1.7 Boot Method 3: Booting the kernel from the SD Card (Quicker Boot)

The final boot method improves on **Boot Method 2** by installing the Linux kernel image into the file system on the SD Card. This provides us with some performance improvement as the SD Card provides better performance than the EZ-Kit SPI flash.



## Formatting the SD Card

Ensure that the SD Card has been formatted using the instructions in the section **Formatting the SD Card** from the **Boot Method 2** chapter, above.

## Writing the file system and kernel to the SD Card

We then copy the file system to the SD Card along with the Linux kernel image. On the Host PC:

```
$ sudo mkfs.ext3 /dev/sdb1
$ sudo mount -t ext3 /dev/sdb1 /mnt
$ sudo mkdir ~/sc5xx_rootfs
$ sudo mount /opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0/
images/rootfs.ext3 ~/sc5xx_rootfs
$ sudo cp ~/sc5xx_rootfs/* -fr /mnt
$ sudo cp /opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0/ima
ges/vmImage-sc589-ezkit /mnt
$ sudo umount /mnt
```

## Load and Flash the DTB File

We still need to flash the kernel DTB file to the EZ-Kit. On the Host PC:

```
$ cp /opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0/images
/sc589-ezkit.dtb /tftpboot/
```

Using Minicom:

```
$ tftp ${loadaddr} sc589-ezkit.dtb
$ sf probe 2:1
$ sf erase 0x100000 0x40000
$ sf write ${loadaddr} 0x100000 ${filesize}
```

## Configure u-Boot to boot using Linux and the file system on the SD Card

```
$ set sdargs set bootargs root=/dev/mmcblk0p1 rw rootwait
rootfstype=ext3 clk_in_hz=(25000000) earlyprintk console=ttySC0,57600
$ set sdboot 'run sdargs; sf probe 2:1;sf read ${dtbaddr} 0x100000
0x40000;ext2load mmc 0:1 ${loadaddr} vmImage-sc589-ezkit;bootm
${loadaddr} - ${dtbaddr}'
$ set bootcmd run sdboot
$ save
```

To complete the process press the **reset** button on the EZ-Kit.

### **i** Resorting to Ethernet booting

If for some reason you wish to revert to booting via Boot mode 1: Copying Linux over Ethernet, you can issue the following commands in uBoot:

```
$ set bootcmd run ramboot
$ save
```

## 2.1.8 Troubleshooting

This document has been very carefully verified, yet we are trying to list some of the usual trouble that we might encounter by following it.

### **Have problem GDB loading and run u-boot**

The board you received may already have some bootable image in the on board SPI Flash, which may run immediately after you power on. In this case, try to set the boot mode switch to position 0 before performing the initial GDB loading and running of u-boot.

### **Get error message "bad format" when booting from Non-Volatile storage**

When do flash erasing and flashing u-boot, dtb file and kernel file into SPI Flash, double confirm the size of the actual files, and the space you reserve for them by command in the flash operation commands, to make sure files do not get overwritten by each other. Meanwhile, double check whether dtb file and kernel file have been wrote into SPI Flash successfully, from the prompt message on u-boot console after you issue each command. In addition, you only need to do the "sf probe " only once everytime you reset your board.



## 2.2 Configure and build from source code

### 2.2.1 Introduction

We will need customized configuration and building in product development, this document describes how to extract the source code for u-boot and for Linux and build each of them using the default configuration. It also shows how to customize features of the Linux kernel, applications and libraries in Buildroot.

At first follow the [quick start guide document](#) to set up the Ubuntu host environment and the toolchain, and to install CCES and the CCES Linux add in. After that we will copy the source code for u-boot and Buildroot to a working location.

### 2.2.2 Extract the source code

In this document we use `~/sc5xx_dev` as the working path:

```
$ mkdir ~/sc5xx_dev
```

#### Extract the u-boot source code

Run the following commands in sequence to copy the u-boot source code tarfile to the working directory and unpack it:

```
$ cd ~/sc5xx_dev
$ cp /opt/analog/cces-linux-add-in/1.2.0/u-boot-sc5xx-1.2.0/src/u-boot-
sc5xx-1.2.0.tar.gz ./
$ tar -zxvf u-boot-sc5xx-1.2.0.tar.gz
```

Now you have the source code for u-boot in `~/sc5xx_dev/u-boot`.

#### Extract the buildroot and Linux kernel source code

Run the following commands in sequence to copy the Buildroot and Linux source code tarfile to the working directory and unpack it:

```
$ cd ~/sc5xx_dev
$ cp /opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0/src
/buildroot-sc5xx-1.2.0.tar.gz ./
$ tar -zxvf buildroot-sc5xx-1.2.0.tar.gz
```

Now you have the source code for Buildroot and the Linux kernel in `~/sc5xx_dev/buildroot`.

### 2.2.3 Configure and build u-boot

Follow the steps below to configure and build the u-boot, here we take the ADSP-SC589 EZ-KIT Board as an example.

#### Configure the u-boot

```
$ cd ~/sc5xx_dev/uboot
$ make distclean
$ make sc589-ezkit_defconfig
```

#### Build the u-boot

```
$ make
```

#### List of generated files

File	Description
u-boot.ldr	bootable u-boot image
u-boot	u-boot executable ELF

### 2.2.4 Configure and build Buildroot and Linux kernel

Follow below steps to use the default configuration to configure and build the buildroot and Linux kernel, here we take the ADSP-SC589 EZ-KIT board as an example.

#### Default Configuration

```
$ cd ~/sc5xx_dev/buildroot
$ make distclean
$ make sc589-ezkit_defconfig
```

## Build

The final step is to make all changes in buildroot and linux-kernel.

```
$ cd ~/sc5xx_dev/buildroot
$ make
```

It may take many minutes to build for the first time, as buildroot will try to download all the source code for packages from the internet in the initial building.

## 2.2.5 Customize the Buildroot and Linux kernel

Follow below steps to customize features of Linux kernel and Buildroot.

### Customize the Linux kernel

```
$ cd ~/sc5xx_dev/buildroot
$ make linux-menuconfig
```

Above command will invoke Linux kernel configuration menu, which is exactly the same as the general make menuconfig for Linux kernel. In this way you can navigate to different sections of the kernel to enable/disable some features. But never try to go to the source of Linux to do make menuconfig, this will lead to a mess of disaster.

```
$ make linux-rebuild
```

Above command will rebuild the whole buildroot with updated Linux kernel configuration.

### Enable new packages in buildroot

Follow below steps if we want to enable new packages in Buildroot and build them into the generated file system.

```
$ cd ~/sc5xx_dev/buildroot
$ make menuconfig
```

With command above you can navigate to different sections of the buildroot to enable/disable some features for application and libraries integrated in it.

```
$ make
```

## List of generated files

Here is a list of generated files that are frequently used after building.

Location	Description
output/images/uImage	generated Linux boot image
output/images/vmImage	Linux kernel image
output/images/sc589-ezkit.dtb	compiled device tree data file
output/images/rootfs.ext3	ext3 format filesystem image of the root filesystem
output/target/	contents of the rootfs
output/build/linux-custom/	build output of the Linux source tree
output/build/<pkg>	build output of various packages

### 2.2.6 Generate Kernel Image for Booting From Non-Volatile Storage

In the quick start guide we introduced how to boot from SD card with prebuilt images, extra operation is required, however, if you want to boot from SD card with your own built images, file uImage generated with customized configuration as described below will be used as the kernel image file instead of the vmImage.

Execute the **make menuconfig** command and then an interactive configuration application will be launched. In the application:

1. Select **Filesystem images**
2. In the **Filesystem images** menu move the cursor down until **initial RAM filesystem linked into linux kernel** is highlighted

3. Ensure that the option is **not selected** (Press space to toggle selection)
4. Move the cursor across to **Save** and press **Enter**
5. Select **Exit** and press **Enter**
6. Again select **Exit** and press **Enter** to leave the interactive menu

Now we can rebuild the kernel and file system:

```
$ make
```

## 2.2.7 More tips about buildroot

Read full document for Buildroot from <http://buildroot.uclibc.org/docs.html>. Here we are trying to cover some frequently used tips during development.

### Clean the buildroot

Sometimes we want to run **make clean** or **make distclean** for a clean build from scratch, they will remove all the output during last build, as well as the dl folder, it's a good practice to backup the dl folder since it takes long time to download them.

### Rebuild an updated package

Sometimes we make changes on the configuration of a package, or source of a package from output /package. It does not work if we simply run **make** after the changes. Instead, we need to run **make <pkg>-rebuild**

## 3 Das U-boot

### 3.1 Ethernet Driver in U-Boot on SC5xx-EZKIT

#### 3.1.1 Overview

Ethernet driver provides driver for the MAC controller present in ADI processors. The ethernet driver package also has code to interface with the PHYs on the ADI released boards.

There are 2 network interfaces on ADSP- SC5xx : EMAC0 and EMAC1

- EMAC0 is configurable as 10/100 Mbps, interfacing via RMII, or 10/100/1000 Mbps, interfacing via RGMII. On the SC5xx-EZ-Board the RGMII interface is used, providing 10/100/1000 Mbps (gigabit) capability
- EMAC1 is a fixed 10/100 Mbps EMAC, interfacing via RMII.

#### 3.1.2 Hardware Setup

- An ADSP-SC5xx EZ-Kit: ADSP-SC589/SC584/SC573 EZ-Board

#### 3.1.3 Build U-Boot with emac0 or emac1

U-Boot only supports one network port. You have to select one emac port in include/configs/sc589-ezkit.h at build time.(For SC584/SC573 EZ-Board , we should do it in file sc584-ezkit.h or sc573-ezkit.h)

```
#define CONFIG_DW_PORTS          1 // enable EMAC0
or
#define CONFIG_DW_PORTS          2 // enable EMAC1
```

#### 3.1.4 Using EMAC Driver

U-boot provides a set of basic net commands:

- dhcp - invoke a DHCP client request to obtain IP/boot params
- ping - send a ICMP ECHO\_REQUEST to the network host
- tftpboot - boot an image via network using the TFTP protocol

For example:

```

CPU:   ADSP ADSP-SC589-0.0 (Detected Rev: 1.1) (spi flash boot)
VCO:  450 MHz, Cclk0: 450 MHz, Sclk0: 112.500 MHz, Sclk1: 112.500 MHz,
DCLK: 225 MHz
OCLK: 150 MHz
I2C:   ready
DRAM:  112 MiB
MMC:   SC5XX SDH: 0
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB,
total 16 MiB
In:    serial
Out:   serial
Err:   serial
other init
Net:   dwmac.3100c000
Hit any key to stop autoboot:  0
sc #
sc # dhcp
Speed: 100, full duplex
BOOTP broadcast 1
BOOTP broadcast 2
DHCP client bound to address 10.99.24.200 (260 ms)
sc # ping 10.99.24.94
Speed: 100, full duplex
Using dwmac.3100c000 device
host 10.99.24.94 is alive
sc # tftp ${loadaddr} ${ramfile}
Speed: 100, full duplex
Using dwmac.3100c000 device
TFTP from server 10.99.24.94; our IP address is 10.99.24.200
Filename 'uImage'.
Load address: 0x89000000
Loading:
#####
#####
.....
#####
      2 MiB/s
done
Bytes transferred = 10502080 (a03fc0 hex)

```

## 3.2 Creating and Booting Linux Using the New U-Boot "fitImage"

### 3.2.1 Introduction

Currently, the most common way of booting Linux through U-Boot is to create a uImage and dtb in Buildroot and then supply these two images to the bootm command, eg. bootm \${loadaddr} - \${dtbaddr}. Recently, U-Boot has implemented a new image format modelled on the Linux kernel device tree, the Flattened Image Tree or "fitImage". This new format allows for multiple configurations of kernel, ramdisk and dtb images to be combined into a single image file. It also provides a mechanism to easily specify authentication options, critical for implementing a chain of trust while booting Linux. This page describes how to configure, implement and boot the new fitImage format.

Before users start to try the fitImage, please make sure the following tools have already been installed.

- CCES (for Linux) is installed on the host machine and the location is exported to \$PATH
- SC5xx ARM toolchains are installed and exported to \$PATH
- Buildroot and U-Boot sources are installed on the host machine
- The host machine has already started a tftp service
- Device Tree Compiler (DTC) is installed (from git) and added to \$PATH
- The 'serverip' U-Boot environment variable is set to the IP address of the tftp server
- Run "sudo apt-get install libssl-dev" to install libssl

### 3.2.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board

### 3.2.3 Buildroot and kernel Configuration

For Buildroot, two changes to our current configuration are needed. Here we take SC589-EZKIT board as example, for SC584 and SC573 EZ-Board the steps required are similar except for different file names.

In Buildroot's menuconfig:

```
$ make clean
$ make sc589-ezkit_defconfig (For SC584 or SC573, it should be make
sc584-ezkit_defconfig/make sc573-ezkit_defconfig )
$ make menuconfig
```



## Change kernel format to vmlinux

```
Kernel --->
  Kernel binary format --->
    [*] vmlinux
```

## Deselect the linked initial RAM filesystem

```
Filesystem images --->
  [ ] initial RAM filesystem linked into linux kernel
```

## Compile Linux as usual with "make"

```
$ make
```

After the compiling is done, you need to compress the kernel binary and the rootfs using gzip.

## Create compressed kernel binary

```
$ cd output/images/
$ arm-linux-gnueabi-objcopy -O binary vmlinux vmlinux.bin
$ gzip --best --force vmlinux.bin
$ gzip --best --force rootfs.cpio
```

## Copy all images to common location (Use /tftpboot as the example)

```
$ cp vmlinux.bin.gz rootfs.cpio.gz sc589-ezkit.dtb /tftpboot
```

## 3.2.4 U-Boot Configuration

### Enable fitImage Support

Compatibility for Flattened Image Tree must be added to U-Boot. In U-Boot's directory, set the default config and **make menuconfig**:

```
$ make clean
$ make sc589-ezkit_defconfig
$ make menuconfig
```

Select the following options:

```
Boot images --->
  [*] Support Flattened Image Tree
  [*]   Display verbose messages on FIT boot
  [*]   Enable signature verification of FIT uImage
Device Drivers --->
  [*] Enable Driver Model
```

Compile U-Boot like normal with "make" and then boot into U-Boot on the ezkit.

```
$ make
```

Don't forget to **run update** in your uboot console. For more information about how to boot from U-Boot, please refer to [SC5xx ezkit Linux quick start guide](#) .

### 3.2.5 Create fitImage device tree

U-Boot's device tree is based on the Linux device tree, so the similarities aren't surprising. Images are defined as nodes within an "images" block and can be anything from kernel images, dtb images, ramdisk images to u-boot images, firmware images and standalone executables. Configurations are also defined as nodes within a "configurations" block and a default configuration can be specified. Below is a sample device tree using our previously built images:

In your U-Boot source directory, create a file named **fitImage.its** and add the following content:

```

/dts-v1/;
/ {
    description = "SC5xx Linux Distribution";
    #address-cells = <1>;
    images {
        kernel@1 {
            description = "SC5xx Linux kernel";
            data = /incbin("/tftpboot/vmlinux.
bin.gz");
            arch = "arm";
            os = "linux";
            type = "kernel";
            compression = "gzip";
            load = <0x89008000>;
            entry = <0x89008000>;
        };
        fdt@1 {
            description = "Flattened Device Tree blob";
            data = /incbin("/tftpboot/sc589-ezkit.dtb");
            type = "flat_dt";
            arch = "arm";
            compression = "none";
        };
        ramdisk@1 {
            description = "SC5xx Linux Rootfs";
            data = /incbin("/tftpboot/rootfs.cpio.gz");
            type = "ramdisk";
            arch = "arm";
            os = "linux";
            compression = "gzip";
            load = <0x8d000000>;
            entry = <0x8d000000>;
        };
    };
    configurations {
        default = "conf@1";
        conf@1 {
            description = "Boot Linux";
            kernel = "kernel@1";
            ramdisk = "ramdisk@1";
            fdt = "fdt@1";
        };
    };
};

```



Don't forget to change the dtb file name and kernel/ramdisk load address according to your board type (i.e. SC589-ezkit or SC584-ezkit or SC573-ezkit), otherwise the fitimage may not boot properly. About the difference among these boards you could refer to [SC5xx ezkit Linux quick start guide](#).

### Compile the device tree into a .itb "blob"

Please add the Linux kernel dtc script location to your PATH environment variable before building the fitImage. For example:

```
$ export PATH=/home/test/workspace/buildroot/output/build/linux-custom  
/scripts/dtc:$PATH
```

In your U-Boot source directory root:

```
$ ./tools/mkimage -f fitImage.its fitImage.itb
```

You will get a fitImage.itb file and see the the following output if the compile succeed.

```

FIT description: SC5xx Linux Distribution
Created:        Wed Mar 22 10:43:47 2017
Image 0 (kernel@1)
  Description:  SC5xx Linux kernel
  Created:     Wed Mar 22 10:43:47 2017
  Type:       Kernel Image
  Compression: gzip compressed
  Data Size:  2869587 Bytes = 2802.33 kB = 2.74 MB
  Architecture: ARM
  OS:         Linux
  Load Address: 0xc2008000
  Entry Point: 0xc2008000
Image 1 (fdt@1)
  Description:  Flattened Device Tree blob
  Created:     Wed Mar 22 10:43:47 2017
  Type:       Flat Device Tree
  Compression: uncompressed
  Data Size:  18516 Bytes = 18.08 kB = 0.02 MB
  Architecture: ARM
Image 2 (ramdisk@1)
  Description:  SC5xx Linux Rootfs
  Created:     Wed Mar 22 10:43:47 2017
  Type:       RAMDisk Image
  Compression: gzip compressed
  Data Size:  17137994 Bytes = 16736.32 kB = 16.34 MB
  Architecture: ARM
  OS:         Linux
  Load Address: 0xc6000000
  Entry Point: 0xc6000000
Default Configuration: 'conf@1'
Configuration 0 (conf@1)
  Description:  Boot Linux
  Kernel:      kernel@1
  Init Ramdisk: ramdisk@1
  FDT:        fdt@1

```

Then also copy the fitImage.itb file to the common location. Still use /tftpboot as the example

```
$ cp fitImage.itb /tftpboot/
```

### 3.2.6 Boot Linux

Boot the ezkit into the newly compiled u-boot with fitImage support. Once booted, follow the steps below to boot Linux:

#### Tftp the fitImage into RAM

```
sc # tftp 0xc4000000 fitImage.itb
```

Here is the console output

```
Speed: 1000, full duplex
Using dwmac.3100c000 device
TFTP from server 10.100.4.174; our IP address is 10.100.4.50
Filename 'fitImage.itb'.
Load address: 0xc4000000
Loading:
#####
#####
###
#####
###
.....
#####
###
#####
###
#####
###
3 MiB/s
done
Bytes transferred = 20027491 (1319863 hex)
```

**Run the boot args (assuming that they have already been flashed to your board)**

```
sc # run ramargs; run addip;
```

**Boot**

```
sc # bootm
```

Here is the console output for booting Linux.

```
## Loading kernel from FIT Image at c4000000 ...
Using 'conf@1' configuration
Verifying Hash Integrity ... OK
Trying 'kernel@1' kernel subimage
Description: SC5xx Linux kernel
Type: Kernel Image
```

```

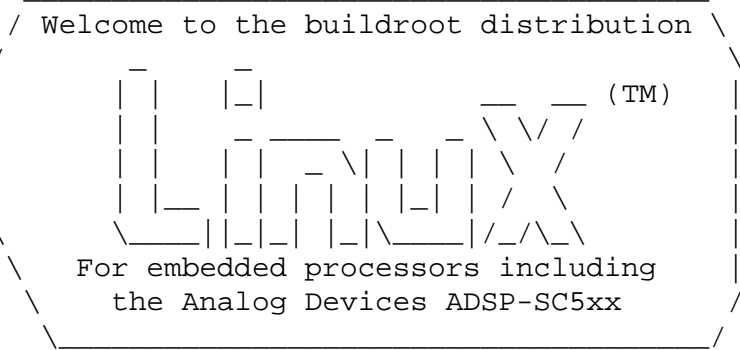
    Compression:  gzip compressed
    Data Start:   0xc40000d0
    Data Size:   2869587 Bytes = 2.7 MiB
    Architecture: ARM
    OS:          Linux
    Load Address: 0xc2008000
    Entry Point:  0xc2008000
    Verifying Hash Integrity ... OK
## Loading ramdisk from FIT Image at c4000000 ...
    Using 'conf@1' configuration
    Trying 'ramdisk@1' ramdisk subimage
        Description:  SC5xx Linux Rootfs
        Type:         RAMDisk Image
        Compression:  gzip compressed
        Data Start:   0xc42c13a0
        Data Size:   17137994 Bytes = 16.3 MiB
        Architecture: ARM
        OS:           Linux
        Load Address: 0xc6000000
        Entry Point:  0xc6000000
        Verifying Hash Integrity ... OK
    Loading ramdisk from 0xc42c13a0 to 0xc6000000
## Loading fdt from FIT Image at c4000000 ...
    Using 'conf@1' configuration
    Trying 'fdt@1' fdt subimage
        Description:  Flattened Device Tree blob
        Type:         Flat Device Tree
        Compression:  uncompressed
        Data Start:   0xc42bcad4
        Data Size:   18516 Bytes = 18.1 KiB
        Architecture: ARM
        Verifying Hash Integrity ... OK
    Booting using the fdt blob at 0xc42bcad4
    Uncompressing Kernel Image ... OK
    Loading Ramdisk to cee00000, end cfe5814a ... OK
    Loading Device Tree to cedf8000, end cedff853 ... OK
Starting kernel ...
Booting Linux on physical CPU 0x0
Linux version 4.0.0-ADI-1.1.0-00048-gd20cee5 (hfeng@hfeng) (gcc
version 4.8.3 (Analog Devices Inc. ARM Tools
(4972be04c874843c07fba1554f4271c8dae0b9fb). Dist7
CPU: ARMv7 Processor [410fc051] revision 1 (ARMv7), cr=10c53c7d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction
cache
Machine model: ADI sc589-ezkit
bootconsole [earlycon0] enabled
Memory policy: Data cache writeback
.
.
.
Welcome to Buildroot

```

```
buildroot login: root
Password:
```

```

a8888b.
d888888b.
8P"YP"Y88
8|o||o|88
8' .88
8`._.' Y8.
d/      `8b.
dP      .   Y8b.
d8:' "   `::88b
d8"      'Y88b
:8P      '   :888
8a.      :   _a88P
._/"Yaa_:_| 88P|
\      YP"  `| 8P `
/      \.____.d| .
`--..__.)8888P`._.' jgs/a:f
Have a lot of fun...
#
```



```
For further information, check out:
- http://buildroot.org/
- http://www.analog.com/
```

For more information on fitImage, please see the documentation in the UBoot directory tree at \$(UBOOT\_DIR)/doc/uImage.FIT/

### 3.3 Loading file from USB storage in u-boot

#### 3.3.1 Overview

This document talks about how we can load files on USB memory stick, into system RAM from u-boot. Here we take ADSP-SC573 board as example.

#### 3.3.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board
- A USB memory stick
- A USB adapter cable (provided in the ezkit box)

Connect USB stick to the USB OTG port, via the USB adapter cable, as following, and reset the board





### 3.3.3 Test method

#### Formatting the USB stick

Insert the USB memory stick into a Linux PC, you will see new items show up in `/dev/sd*`, as following:

```
$ ls /dev/sd*  
/dev/sda /dev/sda1 /dev/sda2 /dev/sda5 /dev/sdb /dev/sdb1
```

In this case the `/dev/sdb` is for the USB stick we just plugged in.

**Caution :** Please double check the device node newly created for your memory stick, otherwise serious damage like system permanent crash down happens!

The format a vfat partiton on it:

```
$ sudo mkfs.vfat /dev/sdb1  
[sudo] password for ...  
mkfs.vfat 3.0.12 (29 Oct 2011)
```

#### Copy files in it

```
$ sudo mount /dev/sdb1 /mnt  
$ cd /opt/analog/cces-linux-add-in/1.2.0/buildroot-sc5xx-1.2.0/images  
$ sudo cp uImage-sc573-ezkit /mnt/uImage  
$ sudo cp sc573-ezkit.dtb /mnt  
$ sudo umount /mnt
```

Then plug the USB stick to the USB OTG port in board, via the USB adapter cable, and reset the board

## Start the USB

Run "start usb" In the u-boot console:

```
sc # usb start
(Re)start USB...
USB0:   scanning bus 0 for devices... 1 USB Device(s) found
        scanning usb for storage devices... 1 Storage Device(s) found
```

Run "fatls usb 0:1" In the u-boot console:

```
sc # fatls usb 0:1
12452384  uimage
10306    sc573-ezkit.dtb
2 file(s), 0 dir(s)
```

This shows the FAT files information in USB device 0 partition 1, with files we copied.

## Load file into RAM

As example we load both the dtb and the uImage file into proper location.



Don't forget to change the file/dtb name and load address according to your board type (i.e. SC589-ezkit or SC584-ezkit or SC573-ezkit), otherwise it may not boot properly.

```
sc # fatload usb 0:1 0x84000000 sc573-ezkit.dtb
reading sc573-ezkit.dtb
10306 bytes read in 30 ms (335 KiB/s)
sc # fatload usb 0:1 0x82000000 uImage
reading uImage
12452384 bytes read in 7772 ms (1.5 MiB/s)
```

Verify the read operation:

```
sc # bootm 0x82000000 - 0x84000000
## Booting kernel from Legacy Image at 82000000 ...
Image Name:   Linux-4.0.0-ADI-1.1.0-00027-g5c7
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    12452320 Bytes = 11.9 MiB
Load Address: 82008000
Entry Point:  82008000
```

```

Verifying Checksum ... OK
## Flattened Device Tree blob at 84000000
   Booting using the fdt blob at 0x84000000
   Loading Kernel Image ... OK
   Loading Device Tree to 8fe5d000, end 8fe62841 ... OK
Starting kernel ...
Booting Linux on physical CPU 0x0
Linux version 4.0.0-ADI-1.1.0-00027-g5c77b60 (i7@i7-OptiPlex-9010)
(gcc version 4.8.3 (Analog Devices Inc. ARM Tools
(e596c0da91703b047df99f5513e6
CPU: ARMv7 Processor [410fc051] revision 1 (ARMv7), cr=10c53c7d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction
cache
Machine model: ADI sc57x-ezkit
bootconsole [earlycon0] enabled
Memory policy: Data cache writeback
dump init clock rate
CGU0_PLL 450 MHz
CGU0_SYSCLK 225 MHz
CGU0_CCLK 450 MHz
CGU0_SYS0 112 MHz
CGU0_DCLK 225 MHz
CGU0_OCLK 150 MHz
CGU0_SYS0 112 MHz
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 56
896
Kernel command line: root=/dev/mtdblock2 rw rootfstype=jffs2 clk_hz=(
25000000) earlyprintk=serial,uart0,57600 console=ttySC0,57600 mem=224M
PID hash table entries: 1024 (order: 0, 4096 bytes)
Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
Memory: 215024K/229376K available (3139K kernel code, 96K rwdana,
1184K rodata, 7736K init, 71K bss, 14352K reserved, 0K cma-reserved)
Virtual kernel memory layout:
   vector   : 0xffff0000 - 0xffff1000   (   4 kB)
   fixmap   : 0xfffc0000 - 0xffff0000  (3072 kB)
   vmalloc  : 0xce800000 - 0xff000000   ( 776 MB)
   lowmem   : 0xc0000000 - 0xce000000   ( 224 MB)
   modules  : 0xbf000000 - 0xc0000000   (   16 MB)
   .text    : 0xc0008000 - 0xc044113c   (4325 kB)
   .init    : 0xc0442000 - 0xc0bd0000   (7736 kB)
   .data    : 0xc0bd0000 - 0xc0be81e0   (   97 kB)
   .bss     : 0xc0be81e0 - 0xc0bfa098   (   72 kB)
NR_IRQS:16 nr_irqs:16 16
GIC CPU mask not found - kernel will fail to boot.
GIC CPU mask not found - kernel will fail to boot.
sched_clock: 32 bits at 112MHz, resolution 8ns, wraps every
38177486839ns
Console: colour dummy device 80x30
Calibrating delay loop... 297.98 BogoMIPS (lpj=595968)
pid_max: default: 32768 minimum: 301

```

```

Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0x822fdd48 - 0x822fdd7c
devtmpfs: initialized
do_initcall_level level 0
do_initcall_level level 1
VFP support v0.3: implementor 41 architecture 2 part 30 variant 5 rev 1
pinctrl core: initialized pinctrl subsystem
NET: Registered protocol family 16
do_initcall_level level 2
DMA: preallocated 256 KiB pool for atomic coherent allocations
do_initcall_level level 3
L2C: device tree omits to specify unified cache
L2C-310 dynamic clock gating enabled, standby mode enabled
L2C-310 cache controller enabled, 8 ways, 256 kB
L2C-310: CACHE_ID 0x410000c9, AUX_CTRL 0x06040000
sc57x_init: registering device resources
sec init...
enabled
hw-breakpoint: Failed to enable monitor mode on CPU 0.
ADI DMA2 Controller
adi-dma2 31026180.dma: Invalid ADI DMA channel.
adi-dma2: probe of 31026180.dma failed with error -2
adi-dma2 31026100.dma: Invalid ADI DMA channel.
adi-dma2: probe of 31026100.dma failed with error -2
adi-dma2 30fff080.dma: Invalid ADI DMA channel.
adi-dma2: probe of 30fff080.dma failed with error -2
adi-dma2 31026280.dma: Invalid ADI DMA channel.
adi-dma2: probe of 31026280.dma failed with error -2
adi-dma2 31026200.dma: Invalid ADI DMA channel.
adi-dma2: probe of 31026200.dma failed with error -2
adi-dma2 3109a000.dma: Invalid ADI DMA channel.
adi-dma2: probe of 3109a000.dma failed with error -2
adi-dma2 3109a080.dma: Invalid ADI DMA channel.
adi-dma2: probe of 3109a080.dma failed with error -2
adi-dma2 3109b000.dma: Invalid ADI DMA channel.
adi-dma2: probe of 3109b000.dma failed with error -2
adi-dma2 3109b080.dma: Invalid ADI DMA channel.
adi-dma2: probe of 3109b080.dma failed with error -2
DMA channel 40 is not registered
DMA channel 39 is not registered
do_initcall_level level 4
SCSI subsystem initialized
i2c-bfin-twi 31001400.twi: Blackfin on-chip I2C TWI Contoller,
regs_base@f4001400
i2c-bfin-twi 31001500.twi: Blackfin on-chip I2C TWI Contoller,
regs_base@f4001500
i2c-bfin-twi 31001600.twi: Blackfin on-chip I2C TWI Contoller,
regs_base@f4001600
pps_core: LinuxPPS API ver. 1 registered

```

```

pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti
<giometti@linux.it>
PTP clock support registered
do_initcall_level level 5
Switched to clocksource cs_gptimer
NET: Registered protocol family 2
TCP established hash table entries: 2048 (order: 1, 8192 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP: reno registered
UDP hash table entries: 256 (order: 0, 4096 bytes)
UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
NET: Registered protocol family 1
do_initcall_level level 6
hw perfevents: enabled with armv7_cortex_a5 PMU driver, 3 counters
available
futex hash table entries: 256 (order: -1, 3072 bytes)
jffs2: version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
io scheduler noop registered (default)
ADI serial driver
adi-uart4.0: ttySC0 at MMIO 0x31003000 (irq = 20, base_baud = 7031250)
is a ADI-UART4
console [ttySC0] enabled
console [ttySC0] enabled
bootconsole [earlycon0] disabled
bootconsole [earlycon0] disabled
adi-uart4.2: ttySC2 at MMIO 0x31003800 (irq = 60, base_baud = 7031250)
is a ADI-UART4
loop: module loaded
adi-spi3 3102e000.spi: registered ADI SPI controller spi0
adi-spi3 3102f000.spi: registered ADI SPI controller spi1
adi-spi3 31044000.spi: registered ADI SPI controller spi2
stmmaceth 3100c000.ethernet: no reset control found
stmmac - user ID: 0x10, Synopsys ID: 0x37
Ring mode enabled
DMA HW capability register supported
Enhanced/Alternate descriptors
    Enabled extended descriptors
RX Checksum Offload Engine supported (type 2)
TX Checksum insertion supported
Wake-Up On Lan supported
Enable RX Mitigation via HW Watchdog Timer
libphy: stmmac: probed
eth0: PHY ID 20005c7a at 1 IRQ POLL (stmmac-0:01) active
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
adi_wdt: initialized: timeout=20 sec (nowayout=0)
TCP: cubic registered
NET: Registered protocol family 17
do_initcall_level level 7
ThumbEE CPU extension supported.

```

```
Freeing unused kernel memory: 7736K (c0442000 - c0bd0000)
Starting logging: OK
Starting mdev...
Starting watchdog...
Initializing random number generator... random: dd urandom read with 8
bits of entropy available
done.
Starting network...
/bin/sh: run-parts: not found
ssh-keygen: generating new host keys: ED25519
Starting sshd: OK
Starting inetd: OK
Welcome to Buildroot
```

## More information

There are more command available for USB in u-boot:

```
sc # usb info
1: Mass Storage, USB Revision 2.0
- Generic USB Storage 000000009451
- Class: (from Interface) Mass Storage
- PacketSize: 64 Configurations: 1
- Vendor: 0x05e3 Product 0x0723 Version 148.81
Configuration: 1
- Interfaces: 1 Bus Powered 500mA
Interface: 0
- Alternate Setting 0, Endpoints: 2
- Class Mass Storage, Transp. SCSI, Bulk only
- Endpoint 1 In Bulk MaxPacket 512
- Endpoint 2 Out Bulk MaxPacket 512
```

```
sc # usb storage
Device 0: Vendor: Generic Rev: 9451 Prod: STORAGE DEVICE
Type: Removable Hard Disk
Capacity: 1897.0 MB = 1.8 GB (3885056 x 512)
```

```
sc # usb tree
USB device tree:
1 Mass Storage (12 Mb/s, 500mA)
Generic USB Storage 000000009451
```

```
sc # fatinfo usb 0:1
Interface:  USB
  Device 0:  Vendor:  Generic   Rev:  9451  Prod:  STORAGE DEVICE
            Type:  Removable Hard Disk
            Capacity: 1897.0 MB = 1.8 GB (3885056 x 512)
Filesystem: FAT32  "          "
```

## 3.4 Mobile Storage Interface (MSI)

### 3.4.1 Overview

Some of the ADSP-SC5xx processors provide a mobile storage interface(MSI). MSI is a fast, synchronous controller that uses various protocols to communicate with MMC, SD, and SDIO cards to address the growing storage need in embedded systems, handheld and consumer electronics applications requiring low power. The MSI is compatible with the following protocols.

- MMC (Multimedia Card) bus protocol
- SD (Secure Digital) bus protocol
- SDIO (Secure Digital Input Output) bus protocol

All of these storage solutions use similar interface protocols. The main difference between MMC and SD support is the initialization sequence. The main difference between SD and SDIO support is the use of interrupt and read wait signals for SDIO.

### 3.4.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC573 EZ-Board(the SC584 processor does not include the MSI interface)
- SD card

The SD/MMC card slot is **J18** on the SC589-EZKIT and SC573-EZKIT board. This slot accepts full-size SD and MMC cards, or microSD cards with an adapter.



### 3.4.3 Software Configuration

To enable the MSI driver add the following MSI-related config macros in **include/configs/sc589-ezkit.h** or **include/configs/sc573-ezkit.h**

```
#define CONFIG_GENERIC_MMC
#define CONFIG_MMC
#define CONFIG_SC5XX_DWMMC
#define CONFIG_DWMMC
#define CONFIG_BOUNCE_BUFFER
```

### 3.4.4 Build and Load Uboot

A UBoot image can now be built and loaded onto the target board. See [SC5xx ezkit Linux quick start guide](#) for details.



### 3.4.5 Usage of MSI Driver

#### Initialize MSI(MMC/SDIO) Sub-System

```
# mmc info
Device: SC5XX SDH
Manufacturer ID: 3
OEM: 5344
Name: SD01G
Tran Speed: 25000000
Rd Block Len: 512
SD version 1.10
High Capacity: No
Capacity: 968.8 MiB
Bus Width: 4-bit
```

#### Get More MMC Command Usage

```
# mmc help
or
# help mmc
```

## 4 Linux buildroot distribution and applications

### 4.1 Bluetooth and headset audio example

#### 4.1.1 Introduction

Bluetooth is low-cost, low-power, short-range wireless technology. It was designed as a replacement for cables and other short-range technologies like IrDA. Bluetooth operates in personal area range that typically extends up to 10 meters. More information about Bluetooth can be found at [www.bluetooth.com](http://www.bluetooth.com).

The Linux Bluetooth subsystem consists of several layers:

- Bluetooth Core (HCI device and connection manager, scheduler)
- HCI Device drivers (interface to the hardware)
- RFCOMM module (RFCOMM Protocol)
- BNEP module (Bluetooth Network Encapsulation Protocol)
- HIDP module (Human Interface Device Protocol)

The most common use of Bluetooth audio is to make a wireless connection to a cellphone. A special transport, Synchronous Connection-Oriented, or SCO, was invented to provide this low-delay monophonic audio. An ALSA plug-in library is designed to reroute the sound stream from ALSA player to Bluetooth SCO device. Following example shows the steps to configure a Bluetooth SCO headset through USB Bluetooth controller.

#### 4.1.2 Hardware setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board
- A Bluetooth USB dongle
- A Bluetooth headset

#### 4.1.3 Software configuration

##### Linux Kernel Configuration

For the Linux Kernel, users need to configure the Bluetooth socket stack.

```

[*] Networking support --->
    [*] Bluetooth subsystem support --->
        [*] Bluetooth Classic (BR/EDR) features
        <*> RFCOMM protocol support
            Bluetooth device drivers --->
                <*> HCI USB driver
    Device Drivers --->
        HID support --->
            [*] USB support --->
                MUSB Mode Selection (Host only mode)--->
                    (X) Host only mode
                    ( ) Gadget only mode
                    ( ) Dual Role only mode

```

## Buildroot Configuration

For Buildroot, user needs configure the following applications:

```

Target Packages --->
    Libraries --->
        Jason/XML --->
            -*-- expact
        Audio/Sound --->
            -*-- alsa-lib
    Hardware handling --->
        -*-- dbus
    Audio and video applications --->
        [*] alsa-utils
        ALSA utils selection --->
            [*] amixer
            [*] aplay/arecord
        [*] play
    Networking application --->
        [*] bluez-utils
        [*] audio support

```

### 4.1.4 Run headset audio example

Boot the SC5xx EZkit board with the image just compiled (see [SC5xx EZ-Kit Linux Quick Start Guide](#) for details) and plug in the Bluetooth USB dongle. By default USB works on HOST mode, the device should be detected automatically:

```
usbhid: USB HID core driver
usb 1-1: new full-speed USB device number 2 using musb-hdrc
```

Scan Bluetooth devices and set up the headset address in asound.conf

**Example**

```
# hciconfig hci0 up
# hcitool scan
Scanning ...
        30:F9:ED:E1:9E:A9          DR-BT140Q
# vi /etc/asound.conf
```

Add the following content to /etc/asound.conf

**Example**

```
#####
pcm.bluetooth {
    type bluetooth
    device "30:F9:ED:E1:9E:A9"
}
```

Enable bluez Socket in audio.conf:

**Example**

```
# vi /etc/bluetooth/audio.conf
```

The content of audio.conf is:

**Example**

```
[General]
Enable=Socket,Control
```

Start Bluetooth daemons and create devices

### Example

```
# /etc/init.d/S30dbus start &
Starting system message bus: done
# bluetoothd &
# agent 0000 &
[1]- Done bluetoothd
# export BTADAPTER=`dbus-send --system --dest=org.bluez --print-reply
/ org.bluez.Manager.DefaultAdapter | tail -n 1 | sed 's/^\.*"\(.*\)".*$/\1/'`
# dbus-send --system --print-reply --dest=org.bluez $BTADAPTER org.
bluez.Adapter.CreateDevice string:30:F9:ED:E1:9E:A9
method return sender=:1.0 -> dest=:1.3 reply_serial=2
object path "/org/bluez/560/hci0/dev_30_F9_ED_E1_9E_A9"
# dbus-send --system --print-reply --dest=org.bluez $BTADAPTER
/dev_30_F9_ED_E1_9E_A9 org.bluez.Audio.Connect
Pincode request for device /org/bluez/560/hci0/dev_30_F9_ED_E1_9E_A9
method return sender=:1.0 -> dest=:1.6 reply_serial=2
```

Now user can play the music file and hear the voice from the headset.

### Example

```
# aplay -D bluetooth test.wav
Playing WAVE 'test.wav' : Signed 16 bit Little Endian, Rate 44100 Hz,
Stereo
```

## 4.2 Build application outside the buildroot framework

### 4.2.1 Introduction

This page describes how to build application outside the buildroot framework. Buildroot will take care the build options for components of the buildroot, such as Linux Kernel, and different kinds of packages. For standalone applications which outside buildroot framework, developers should deal with build options by themselves.

### 4.2.2 Build Application outside Buildroot Framework

The Buildroot framework appends many compilation and link options in the external toolchain wrapper when building Buildroot packages.

If you wish to build your application with the GNU Linux toolchain for ADSP-SC5xx, outside the Buildroot framework, you have to add the following options to your Makefile or command line. Analog Devices GNU Toolchain for ARM processors accepts switches to select the Analog Devices processor (See **-mproc**) and silicon revision (See **-msi-revision**). Use of these switches will ensure that the correct pre-processor macros and run-time libraries are used for the processor. The switches will also ensure that any silicon anomaly workarounds relevant to the processor are enabled at compile, assemble and link time. For more information about the switches, please see ARM® Development Tools Documentation -> Cortex-A -> Analog Devices ARM Toolchain Manual -> GCC C/C++ Compiler Additional Features -> Additional Compiler Switches on the CrossCore Embedded Studio IDE Help, or provide the **--help** or **--target-help** options to the **arm-linux-gnueabi-gcc**.

For example to compile for the *ADSP-SC589* processor, with a silicon revision of 0.0:

```
-mabi=aapcs-linux -mproc=ADSP-SC589 -msi-revision=0.0 -marm -pipe -  
mfloat-abi=hard -std=gnu99 -D_GNU_SOURCE -D_LARGEFILE_SOURCE -  
D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64
```

Take the package **version** as an example to show the build process:

```
$ arm-linux-gnueabi-gcc -mabi=aapcs-linux -mproc=ADSP-SC589 -msi-  
revision=0.0 -marm -pipe -mfloat-abi=hard -std=gnu99 -D_GNU_SOURCE -  
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE  
-D_FILE_OFFSET_BITS=64 -Os -I/home/test/projects/buildroot/output/host  
/usr/arm-buildroot-linux-gnueabi/hf/sysroot/usr/include/tirpc/ -o  
version version.c stamp.c
```

## 4.3 OProfile for performance benchmark

### 4.3.1 Introduction

OProfile is one of the packages of the Buildroot distribution. This section will help guide you through the process of compiling OProfile and running it on your target system. Please find more OProfile details in its user manual at: <http://oprofile.sourceforge.net/>.

### 4.3.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board

### 4.3.3 Software Configuration

#### Configure Linux Kernel

First you have to enable OProfile support in Linux kernel.

```
General setup --->
  [*] Profiling support
  <[*> OProfile system profiling
```

This will enable the system timer profiling. If you want to use the ARM PMU hardware monitor feature, please enable following options as well.

```
Kernel Features --->
  [*] Enable hardware performance counter support for perf events
```

#### Configure Buildroot

At the top of your buildroot user configuration, you have to turn on the OProfile application and the Whetstone benchmark.

```
Target packages --->
  Debugging, profiling and benchmark --->
    [*] oprofile
    [*] whetstone
```

### 4.3.4 Simple OProfile usage

Login to your target Linux serial console as root. Copy the kernel image “vmlinux” to your target file system via Ethernet and ssh:

```
$ scp vmlinux root@target:/'
```

Make sure that there are the following files under the /usr/bin directory:

- `opannotate` -- output annotated source file with samples. The executable file needs to contain debug information, the source file needs to be copied to the target system from the development system.
- `oparchive` -- backup the oprofiling data files.
- `opcontrol` -- control the oprofile system, you need to use `bfin_opcontrol` for blackfin.
- `opgprof` -- create gprof style report.
- `ophelp` -- help.
- `opreport` -- create summary report from data files.
- `oprofiled` -- the oprofiling daemon.

It's important to note that "**opcontrol**" and "**oprofiled**" are no longer used in Oprofile 1.0.0 and later version, please ignore the "**opcontrol**" operations in the following examples if you are using Oprofile 1.0.0 and later version.

If you use the latest version Oprofile, please refer to the following "Profile Whetstone Only by OPerf and ARM PMU Events " section as the example.

Export the OPDIR environment variable first:

```
# export OPDIR=/usr/bin/
```

You can get the supported events by executing:

```
# opcontrol --list-events
```

Use following commands to start OProfile:

```
# opcontrol --vmlinux=kernel_image_path
# opcontrol --event=EVENT_TYPE:MAX_COUNT:START_COUNT:PROF_KERNEL:
PROF_USER
# opcontrol --start --kernel-range=start,end
```

This initiates the profiling. Please run any application you want to profile.

Then we can run the following to force a flush of the collected profiling data:

```
# opcontrol --dump
```



The sample data file will be generated in `/var/lib/oprofile/samples/`. Then we can run:

```
# oprofile -l
```

Some user-readable output is produced, as in the following example:

```
Using /var/lib/oprofile/samples/ for samples directory.
CPU: CPU with timer interrupt
Profiling through timer interrupt
warning: could not check that the binary file /vmlinux has not been
modified since the profile was.
samples  %      app name      symbol name
411480   88.9696  vmlinux      cpu_v7_name
195      0.0422   vmlinux      sg_miter_get_next_page
136      0.0294   vmlinux      zap_page_range
132      0.0285   vmlinux      kill_pid_info_as_cred
118      0.0255   vmlinux      try_to_free_buffers
```

You can stop profiling by:

```
# opcontrol --shutdown
```

You can reset profiling data by:

```
# opcontrol --reset
```

## 4.3.5 Examples

### Profile Linux Kernel and Whetstone by OPControl and the Timer Interrupt

```
# export OPDIR=/usr/bin/
# opcontrol --vmlinux=/vmlinux
oprofile: hardware counters not available
oprofile: using timer interrupt.
#
# opcontrol --start --kernel-range=0xc0000000,0xffffffff
Using 2.6+ OProfile kernel interface.
Reading module info.
Using log file /var/lib/oprofile/samples/oprofiled.log
Daemon started.
Profiler running.
```

```

#
# whetstone 50000
Loops: 50000, Iterations: 1, Duration: 24 sec.
C Converted Double Precision Whetstones: 208.3 MIPS
# opcontrol --dump
# oprofile -l
Using /var/lib/oprofile/samples/ for samples directory.
CPU: CPU with timer interrupt
Profiling through timer interrupt
samples  %          app name          symbol name
616      34.4134 whetstone          /usr/bin/whetstone
580      32.4022 libm-2.18.so      /lib/libm-2.18.so
558      31.1732 vmlinux           set_memory_rw
8         0.4469  libc-2.18.so      /lib/libc-2.18.so
6         0.3352  ld-2.18.so        /lib/ld-2.18.so
2         0.1117  vmlinux           bdi_dirty_limit
1         0.0559  busybox           /bin/busybox
1         0.0559  vmlinux           Sys_select
1         0.0559  vmlinux           __d_lookup
1         0.0559  vmlinux           __ptrace_unlink
1         0.0559  vmlinux           allocate_resource
1         0.0559  vmlinux           bdi_set_min_ratio
1         0.0559  vmlinux           d_materialise_unique
1         0.0559  vmlinux           d_prune_aliases
1         0.0559  vmlinux           dentry_lru_isolate_shrink
1         0.0559  vmlinux           dma_pool_alloc
1         0.0559  vmlinux           do_sys_truncate
1         0.0559  vmlinux           get_cmdline
1         0.0559  vmlinux           rmap_walk
1         0.0559  vmlinux           set_cpu_itimer
1         0.0559  vmlinux           sg_miter_get_next_page
1         0.0559  vmlinux           traverse
1         0.0559  vmlinux           unlink_anon_vmas
1         0.0559  vmlinux           vfs_fstatat
1         0.0559  vmlinux           vsscanf
1         0.0559  vmlinux           wait_consider_task

```

## Profile Linux Kernel and Whetstone by OPcontrol and the ARM PMU Events

```

# export OPDIR=/usr/bin/
# opcontrol --vmlinux=/vmlinux
# opcontrol --event=CPU_CYCLES:100000:0:1:1
# opcontrol --start --kernel-range=0xc0000000,0xffffffff
ATTENTION: Use of opcontrol is discouraged. Please see the man page fo
r operf.
Using 2.6+ OProfile kernel interface.
Reading module info.
Using log file /var/lib/oprofile/samples/oprofiled.log

```

```

Daemon started.
Profiler running.
# whetstone 50000
Loops: 50000, Iterations: 1, Duration: 24 sec.
C Converted Double Precision Whetstones: 208.3 MIPS
# opcontrol --dump
# oprofile -l
Using /var/lib/oprofile/samples/ for samples directory.
warning: /oprofile could not be found.
CPU: ARM Cortex-A5
Counted CPU_CYCLES events (CPU cycle) with a unit mask of 0x00 (No
unit mask) count 100000
samples  %          app name          symbol name
4953     47.2705 whetstone          /usr/bin/whetstone
4645     44.3310 libm-2.18.so      /lib/libm-2.18.so
126      1.2025  vmlinux           get_signal_to_deliver
39       0.3722  libc-2.18.so     /lib/libc-2.18.so
35       0.3340  busybox          /bin/busybox
34       0.3245  ld-2.18.so       /lib/ld-2.18.so
30       0.2863  vmlinux           print_prefix
29       0.2768  vmlinux           input_handle_event
27       0.2577  vmlinux           set_cpu_itimer
25       0.2386  vmlinux           __bdi_update_bandwidth
22       0.2100  vmlinux           kfifo_out_copy_r
21       0.2004  vmlinux           vsscanf
20       0.1909  vmlinux           evdev_handle_set_keycode_v2
19       0.1813  vmlinux           allocate_resource
18       0.1718  vmlinux           bdi_dirty_limit
15       0.1432  vmlinux           ack_bad
14       0.1336  vmlinux           __arm_dma_free.isra.23
14       0.1336  vmlinux           bdi_set_min_ratio
14       0.1336  vmlinux           flush_workqueue
14       0.1336  vmlinux           unlink_anon_vmas
14       0.1336  vmlinux           wait_consider_task
13       0.1241  vmlinux           anon_vma_clone
13       0.1241  vmlinux           vfs_fstatat
12       0.1145  vmlinux           drain_workqueue
10       0.0954  oprofile         /oprofile
...

```

## Profile Whetstone Only by OPerf and ARM PMU Events

```

# operf whetstone 10000
operf: Profiler started

Loops: 10000, Iterations: 1, Duration: 10 sec.

```

```

C Converted Double Precision Whetstones: 100.0 MIPS

Profiling done.
#
# oprofile -l
Using //oprofile_data/samples/ for samples directory.
warning: /no-vmlinux could not be found.
CPU: ARM Cortex-A5
Counted CPU_CYCLES events (CPU cycle) with a unit mask of 0x00 (No
unit mask) count 100000
samples  %      image name      symbol name
992      50.6639  whetstone      /usr/bin/whetstone
944      48.2125  libm-2.18.so   /lib/libm-2.18.so
18       0.9193   no-vmlinux     /no-vmlinux
4        0.2043   ld-2.18.so     /lib/ld-2.18.so

```

## 4.4 FTP Server (ftpd)

### 4.4.1 Introduction

This document describes how to enable the FTP server (ftpd) on Linux running on an ADSP-SC5xx EZ-Kit.

### 4.4.2 Operation

#### Enable the FTPD Manually in Busybox

- a) **make busybox-menuconfig** (in the Buildroot source root)
- b) Select to enable "Networking utilities -> ftpd"
- c) **make**

#### Test the FTP Server

You need to transfer the generated uImage to your target board, using the steps described in [SC5xx EZ-Kit Linux Quick Start Guide](#), and boot it. You should now be able to use the ftp server daemon running on the SC5xx-ezkit to transfer data to or from a remote PC running Linux (or another OS).

#### Connect to the target board from the Linux host:

```
host: /> ftp $YOUR_BOARD_IP
```

## Upload a file from the Linux host to the target board

```
ftp:/> put $FILE
```

## Check the uploaded file is there (on the sc5xx-ezkit):

```
root:/> ls /tmp
```

## Download file from the target board

```
ftp:/> get $FILE
```

Run **ls** on the Linux host's console after this. You should see the file downloaded is present.

The ftp daemon (ftpd) is automatically launched on boot by the file **/etc/inetd.conf** on the target SC5xx-ezkit board, with the following command:

```
root:/> ftp      stream tcp nowait root /sbin/ftpd ftpd -w -S /tmp
```

## 4.5 Touchscreen Library

### 4.5.1 Introduction

The touchscreen library (tslib) is an abstraction layer for touchscreen panel events, as well as a filter stack for the manipulation of those events (such as jitter smoothing and calibration transform). It provides the user space API to interacting with touchscreens under Linux. It is the only method supported anymore for accessing touchscreen devices under Linux on the SC58x EZ-kits. The upstream homepage can be found at [github.com/kergoth/tslib](http://github.com/kergoth/tslib)

### 4.5.2 Hardware Setup

- An ADSP-SC58x EZ-Board: ADSP-SC589/SC584 EZ-Board
- A WVGA/LCD EI3 LCD extender board (For more information about this board, you could refer to the doc in ADI Web Site <http://www.analog.com/EX3-LCD>.)

Connect the **J1** socket of EI3 LCD extender to the **P1A** socket of the SC589-EZKIT board.

Note: This case does not apply to SC573 EZ-kit due board level compatibility issue between it and the WVGA/LCD EI3 LCD extender board.

### 4.5.3 Software Configuration

#### Configure Buildroot

The tslib package can be enabled under your user/vendor settings in the buildroot distribution.

```
Target Packages --->
  Libraries --->
    Hardware Handling --->
      [*] tslib
```

You should also enable the event test program to assist if you're setting things up for the first time.

```
Target Packages ---->
  Miscellaneous ---->
    [*] event test
```

#### Configure Linux Kernel

There are two things to enable in the Linux kernel. First, you need the common code for the event interface. Then you need the specific driver for the touchscreen device you're using.

```
Device Drivers --->
  Input device support --->
    [*] Generic input layer (needed for keyboard, mouse, ...)
    <*> Event interface
```

The touchscreen drivers will be in the same place. Enable the driver for the device you're using.

```
Device Drivers --->
  Input device support --->
```

```
[*] Touchscreens --->
    <*> Analog Devices AD7879-1/AD7889-1 touchscreen interface
    <*> support SPI bus connection
```

Don't forget to enable LCD and disable CAN driver for touch screen calibration.

```
Device Drivers --->
    Graphics support --->
        Frame buffer Devices --->
            <*> Support for frame buffer devices --->
            <*> NEC NL8048HL WVGA LCD for BF609
[*] Networking support --->
    < > CAN bus subsystem support ----
```

## 4.5.4 Example

### Runtime Config

Before you can use the touchscreen, you need to export a few environment variables. You may want to put these into a file so you can easily source it at runtime.

```
# cat /etc/tslib.env
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_CONSOLEDEVICE=none
export TSLIB_CONFFILE=/etc/ts.conf
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_TSDEVICE=/dev/input/event0
# . /etc/tslib.env
```

### Enable LCD backlight

Set up GPIO 102 to enable the LCD backlight on the EI3 LCD expander

```
# echo 102 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio102/direction
```

## Calibration

Before you start using the touchscreen, you should calibrate it. This can be easily accomplished with the `ts_calibrate` application on the EI3 LCD extender. Just follow the on-screen directions.



```
# ts_calibrate
xres = 800, yres = 480
Took 10 samples...
Top left : X = 661 Y = 392
Took 7 samples...
Top right : X = 654 Y = 3782
Took 6 samples...
Bot right : X = 3356 Y = 3801
Took 6 samples...
Bot left : X = 3409 Y = 412
Took 5 samples...
Center : X = 2100 Y = 2102
-30.183826 -0.001486 0.206507
-46.264732 0.139335 0.001229
Calibration constants: -1978127 -97 13533 -3032005 9131 80 65536
```

## Testing

Once you're up and running and you've loaded all the right drivers, we can check to make sure things are OK.

## Device Nodes

Make sure you have some event input device nodes.

```
# ls -al /dev/input/
drwxr-xr-x  2 root  root      100 Jan  1 00:00 .
drwxr-xr-x  7 root  root     2480 Jan  1 01:18 ..
crw-rw----  1 root  root      13,  64 Jan  1 00:00 event0
crw-rw----  1 root  root      13,  63 Jan  1 00:00 mice
crw-rw----  1 root  root      13,  32 Jan  1 00:00 mouse0
```



## Event Test

Assuming you enabled the input event test application, you can run it to test the hardware. After you give it the right path to the event device, start using the touchscreen. The application should decode the input immediately.

```
# event_test /dev/input/event0
Input driver version is 1.0.1
Input device ID: bus 0x1c vendor 0x0 product 0x7a version 0x3
Input device name: "AD7879 Touchscreen"
Supported events:
  Event type 0 (Sync)
  Event type 1 (Key)
    Event code 330 (Touch)
  Event type 3 (Absolute)
    Event code 0 (X)
      Value 2077
      Min 0
      Max 4095
    Event code 1 (Y)
      Value 2090
      Min 0
      Max 4095
    Event code 24 (Pressure)
      Value 0
      Min 0
      Max 10000
Testing ... (interrupt to exit)
Event: time 5070.168834, type 1 (Key), code 330 (Touch), value 1
Event: time 5070.168834, type 3 (Absolute), code 0 (X), value 2491
Event: time 5070.168834, type 3 (Absolute), code 1 (Y), value 1422
Event: time 5070.168834, type 3 (Absolute), code 24 (Pressure), value 666
Event: time 5070.168834, ----- Report Sync -----
Event: time 5070.181232, type 3 (Absolute), code 0 (X), value 2514
Event: time 5070.181232, type 3 (Absolute), code 1 (Y), value 1401
Event: time 5070.181232, type 3 (Absolute), code 24 (Pressure), value 714
Event: time 5070.181232, ----- Report Sync -----
Event: time 5070.193622, type 3 (Absolute), code 0 (X), value 2531
Event: time 5070.193622, type 3 (Absolute), code 1 (Y), value 1383
Event: time 5070.193622, type 3 (Absolute), code 24 (Pressure), value 704
Event: time 5070.193622, ----- Report Sync -----
Event: time 5070.206022, type 3 (Absolute), code 0 (X), value 2554
Event: time 5070.206022, type 3 (Absolute), code 1 (Y), value 1369
Event: time 5070.206022, type 3 (Absolute), code 24 (Pressure), value 696
Event: time 5070.206022, ----- Report Sync -----
Event: time 5070.218426, type 3 (Absolute), code 0 (X), value 2560
Event: time 5070.218426, type 3 (Absolute), code 1 (Y), value 1368
Event: time 5070.218426, type 3 (Absolute), code 24 (Pressure), value 691
Event: time 5070.218426, ----- Report Sync -----
```

```

Event: time 5070.230829, type 3 (Absolute), code 0 (X), value 2562
Event: time 5070.230829, type 3 (Absolute), code 1 (Y), value 1369
Event: time 5070.230829, type 3 (Absolute), code 24 (Pressure), value 685
Event: time 5070.230829, ----- Report Sync -----
Event: time 5070.243233, type 3 (Absolute), code 0 (X), value 2564
Event: time 5070.243233, type 3 (Absolute), code 1 (Y), value 1374
Event: time 5070.243233, type 3 (Absolute), code 24 (Pressure), value 669
Event: time 5070.243233, ----- Report Sync -----
Event: time 5070.255633, type 3 (Absolute), code 0 (X), value 2562
Event: time 5070.255633, type 3 (Absolute), code 1 (Y), value 1387
Event: time 5070.255633, ----- Report Sync -----
Event: time 5070.268030, type 3 (Absolute), code 0 (X), value 2549
Event: time 5070.268030, type 3 (Absolute), code 1 (Y), value 1405
Event: time 5070.268030, type 3 (Absolute), code 24 (Pressure), value 668
Event: time 5070.268030, ----- Report Sync -----
Event: time 5070.280430, type 3 (Absolute), code 0 (X), value 2537
Event: time 5070.280430, type 3 (Absolute), code 1 (Y), value 1423
Event: time 5070.280430, type 3 (Absolute), code 24 (Pressure), value 674
Event: time 5070.280430, ----- Report Sync -----
Event: time 5070.292829, type 3 (Absolute), code 0 (X), value 2531
Event: time 5070.292829, type 3 (Absolute), code 1 (Y), value 1443
Event: time 5070.292829, type 3 (Absolute), code 24 (Pressure), value 677
Event: time 5070.292829, ----- Report Sync -----
Event: time 5070.344532, type 3 (Absolute), code 24 (Pressure), value 0
Event: time 5070.344532, type 1 (Key), code 330 (Touch), value 0
Event: time 5070.344532, ----- Report Sync -----

```

## 4.6 MPlayer

This page describes the steps required to build and use MPlayer to play video files on Linux using an ADSP-SC58x-EZKIT board. Here we take ADSP-SC589 EZKIT board as our example. Note this case does not apply to SC573 EZ-kit due board level compatibility issue between it and the WVGA/LCD EI3 LCD extender board. If you want to use mplayer command to play a specified sound, you can refer to [Linux Sound Driver](#).

### 4.6.1 Hardware Setup

- An ADSP-SC58x EZ-Board: ADSP-SC589/SC584 EZ-Board
- A WVGA/LCD EI3 Extender Board

Connect the **J1** connector on the LCD EI3 Extender Board to the **P1A** connector on the SC589-EZKIT.

## 4.6.2 Software Configuration

All configuration operations are conducted within the Buildroot source directory.

Buildroot can be configured using the following command:

```
$ make menuconfig
```

The Linux kernel can be configured using the following command:

```
$ make linux-menuconfig
```

### Default Config

For the SC589-EZKIT board, Buildroot must be configured using the default config file before further customisation can be made:

```
$ make sc589-ezkit_defconfig
```

### Configure Buildroot

#### 1) Compile video test program into linux image

```
Target packages --->
  Miscellaneous --->
    [*] video test program
```

#### 2) Compile mplayer into linux image

```
Target packages --->
  Audio and video applications --->
    [*] mplayer
    [*] Build and install mplayer
```

## Configure the Linux Kernel

### 1) Enable touchscreen and backlight

```
Device Drivers --->
  Input device support --->
    [*] Touchscreens --->
      <*> Analog Devices AD7879-1/AD7889-1 touchscreen interface
      <*> support SPI bus connection
```

### 2) Enable NL8048HL WVGA LCD for ADSP-SC589

```
Device Drivers --->
  Graphics support --->
    Frame buffer Devices --->
      <*> Support for frame buffer devices --->
      <*> NEC NL8048HL WVGA LCD for BF609
```

### 3) Avoid LCD driver probe failure by disabling CAN bus support

```
[*] Networking support --->
  < > CAN bus subsystem support ----
```

## 4.6.3 Build and Load Buildroot

A Buildroot image can now be built and loaded onto the target board. See [SC5xx EZ-Kit Linux Quick Start Guide](#) for details.

## 4.6.4 Run MPlayer

The following commands are executed on the target board.

### Enable LCD Back-light

Configure soft switch to enable the back-light:

```
# echo 102 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio102/direction
```

## Soft Switch Configuration

The settings in this section is only apply to ADSP-SC584 EZ-Board.

```
# echo 484 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio484/direction
# echo 491 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio491/direction
```

## Play Video File

```
# mplayer -nodouble -nosound video.avi
```

## 4.7 IEEE 1588 and Linux PTP

### 4.7.1 PTP Introduction

#### Precision Time Protocol(PTP)

The Precision Time Protocol (PTP) is a high-precision time synchronization protocol for networked measurement and control systems. It is defined in the IEEE 1588 standard, which is designed for local systems requiring very high accuracies - beyond those attainable using NTP (Network Time Protocol). PTP makes it possible to synchronize distributed clocks with sub-microsecond accuracy via Ethernet networks, with relatively low demands on the local clocks and the network and computing capacity.

ADSP-SC573, SC584 and SC589 all support PTP. But only EMAC0 (the 10/100/1000 Mbps port) supports PTP, EMAC1 (the 10/100 Mbps port) is not capable of PTP operation.

## 4.7.2 PTP Configuration

### Device tree configuration

The timestamps that are the basis of PTP can be acquired with greater accuracy when they are captured by the ethernet PHY hardware. The RMII PHY interface supports hardware timestamps but the RGMII PHY interface does not. In order to obtain a higher accuracy of time synchronization the RMII PHY is the better choice, even though this means being limited to 10/100 Mbps operation.

The **phy-mode** can be changed to "rmii" in the device tree header in the linux kernel source **arch/arm/boot/dts/sc58x-ezkit.dts** or **arch/arm/boot/dts/sc57x.dtsi**



... : ellipsis, means other properties in EMAC0 node stay the same

- : minus, means delete this property

+ : plus, means add this property

```
emac0 : ethernet@0X3100c000{
...
- phy-mode = "rgmii";
+ phy-mode = "rmii";
...
};
```

### Buildroot configuration

Buildroot can be configured using the following command, within the Buildroot source directory.:

```
$ make menuconfig
```

```
Target packages --->
  Networking applications --->
    [*] linuxptp
```

## Kernel Configuration

The Linux kernel can be configured using the following command, within the Buildroot source directory.:

```
$ make linux-menuconfig
```

```
General Setup --->
  *- Configure standard kernel features (expert users) --->
    [*] Enable eventpoll support
    [*] Enable timerfd() system call

[*] Networking support --->
  Networking options --->
    [*] Timestamping in PHY devices

Device Drivers --->
  PPS support --->
  PTP clock support --->
  [*] Network device support --->
    [*] Ethernet driver support --->
      [*] STMicroelectronics devices
      <*> STMicroelectronics 10/100/1000 Ethernet driver
      <*> STMMAC Platform bus support
```

### 4.7.3 Build

A Buildroot image can now be built, ready to be loaded onto the target board. See [SC5xx EZ-Kit Linux Quick Start Guide](#) for details.

### 4.7.4 Example

#### Preliminary work

##### 1) Hardware Setup

Two ADSP-SC5xx boards are required. One board act as a master, and the other act as a slave. The two boards are connected by their respective EMAC0 ports using a standard crossover network cable.

## 2) Enable PTP in U-Boot

Add "**stmmac.debug=11**" to the kernel boot arguments in U-Boot.

### Example

```
$ set bootargs root=/dev/mtdblock0 rw stmmac.debug=11 console=ttyBF0,57600
```

## 3) Master's MAC address must be different from slave's

In order to make the master's and the slave's MAC address different, change the slave's address in **U-Boot**.

### Example

```
$ set ethaddr 00:20:22:fe:85:29
```

## 4) Master's ip address must be different from slave ip

Reset IP address after **linux boot up**.

### Example: master

```
# ifconfig eth0 10.100.4.50 up
```

### Example: slave

```
# ifconfig eth0 10.100.4.51 up
```

## Run Example

### 1) Master

**Note: Master board should be configured first.**

```
# ifconfig eth0 10.100.4.51  
# date -s 2010.01.01-13:30
```



```
Fri Jan 1 13:30:00 UTC 2010
# testptp -g
clock time: 0.000000000 or Thu Jan 1 00:00:00 1970
# testptp -s
set time okay
# hwstamp_ctl -i eth0 -r 6 -t 1
tx_type 1
rx_filter 6
# ptp4l -i eth0 &
# date
Fri Jan 1 13:30:51 UTC 2010
```

Date on master board is 2010.1.1-13:30:51.

## 2) Slave

```
# hwstamp_ctl -i eth0 -r 6 -t 1
tx_type 1
rx_filter 6
# ptp4l -i eth0 -s &
# date
Mon Jan 1 00:02:24 UTC 2007
# phc2sys -s /dev/ptp0 -O 0 &
# date
Mon Jan 1 00:02:24 UTC 2007
...
# date
Fri Jan 1 13:30:58 UTC 2010
```

Date on the slave board is 2007.1.1-00:02:24 before synchronization, and changes to 2010.1.1-13:30:58 after a few seconds of synchronization with the master.

### 4.7.5 More information

[Linux PTP project](#)

[IEEE1588 standard](#)

[PTP User guide](#)

## 4.8 QT example for GUI

### 4.8.1 Introduction

Qt is a cross-platform application framework created and maintained by Trolltech. Using Qt, you can develop applications and user interfaces once and deploy them across many desktop and embedded operating systems without rewriting the source code. Visit the QT homepage at [www.qtsoftware.com](http://www.qtsoftware.com) for more information.

Qt provides different licensing methods depending on your needs. For people whose project is open source, they can use the free version. For people whose project is closed source, a commercial version is available for you to license. See the QT licensing page for more details at [www.qtsoftware.com/products/licensing](http://www.qtsoftware.com/products/licensing).

The Qt documentation can be found at [doc.qt.io](http://doc.qt.io), and every Qt source tree includes full documentation as well. Simply browse to the `doc/html/` sub directory in the QT distribution.

This page describes the steps required to build and use qt with WVGA/LCD EI3 Extender Board on Linux using an ADSP-SC589-EZKIT board.

### 4.8.2 Hardware Setup

- An ADSP-SC589-EZKIT board
- A WVGA/LCD EI3 Extender Board, an EZ-Extender product

Connect the **J1** connector on the LCD EI3 Extender Board to the **P1A** connector on the SC589-EZKIT.

Note this case does not apply to SC573 EZ-kit due board level compatibility issue between it and the WVGA/LCD EI3 LCD extender board.

### 4.8.3 Software Configuration

All configuration operations are conducted within the Buildroot source directory.

Buildroot can be configured using the following command:

```
$ make menuconfig
```

The Linux kernel can be configured using the following command:

```
$ make linux-menuconfig
```

## Default Configuration

For the SC589-EZKIT board, Buildroot must be configured using the default config file before further customization can be made:

```
$ make sc589-ezkit_defconfig
```

## Buildroot Configuration

### 1) Enable Video Test

```
Target packages --->
  Miscellaneous --->
    [*] video test program
```

### 2) Enable QT

```
Target packages --->
  Graphic libraries and applications (graphic/text) --->
    [*] Qt --->
      [*] Compile and install Qt demos (with code)
      [*] Compile and install Qt examples (with code)
      [*] Approve free license
      *- Gui Module
          Pixel depths --->
              [*] 24 bpp, rgb 8-8-8
              [ ] 32 bpp, argb 8-8-8-8 and rgb
Mouse drivers --->
  [*] linux input
  [*] tslib
Keyboard drivers --->
  [*] linux input
```

### 3) Enable Tslib Support

```
Target packages --->
  Libraries --->
    Hardware handling --->
      [*] tslib
```

## Linux Kernel Configuration

### 1) Enable Touchscreen and Back-light

```
Device Drivers --->
  Input device support --->
    [*] Touchscreens --->
      <*> Analog Devices AD7879-1/AD7889-1 touchscreen interface
      <*> support SPI bus connection
```

### 2) Enable NL8048HL WVGA LCD for ADSP-SC589

```
Device Drivers --->
  Graphics support --->
    Frame buffer Devices --->
      <*> Support for frame buffer devices --->
      <*> NEC NL8048HL WVGA LCD for BF609
```

### 3) Avoid LCD driver probe failure by disabling CAN bus support

```
[*] Networking support --->
  < > CAN bus subsystem support ----
```

## 4) Input Configuration

```
Device Drivers --->
  Input device support --->
    *- Generic input layer (needed for keyboard, mouse, ...)
      <*> Mouse interface
      (800) Horizontal screen resolution
      (480) Vertical screen resolution
      <*> Event interface
      [*] Keyboards --->
      [*] Mice --->
  HID support --->
    USB HID support --->
      <*> USB HID transport layer
```

## 5) USB Configuration

In order to use USB mouse and keyboard, configure USB to HOST mode or OTG mode. Please refer "[USB interface](#)" for details.

### 4.8.4 Build and Load Buildroot

After the configuration, user can make and boot the image on SC589. Please refer "[SC5xx EZ-Kit Linux Quick Start Guide](#)" for details.

### 4.8.5 Example

The following commands are executed on the target board.

This example shows a qt calendar demo running on Linux on SC589-EZKIT board.

### Enable LCD Back-light

```
# echo 102 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio102/direction
```

## Runtime Configuration for QT



The selected event device in /dev/input must corresponds to selected input device.

## 1) USB Mouse

```
# export QWS_MOUSE_PROTO=USB:/dev/input/event0
```

## 2) USB Keyboard

```
# export QWS_KEYBOARD=USB:/dev/input/event0
```

## 3) Touchscreen

```
# export QWS_MOUSE_PROTO=tslib  
# export TSLIB_FBDEVICE=/dev/fb0  
# export TSLIB_CONSOLEDEVICE=none  
# export TSLIB_CONFFILE=/etc/ts.conf  
# export TSLIB_CALIBFILE=/etc/pointercal  
# export TSLIB_TSDEVICE=/dev/input/event0
```

### Touchscreen calibration

```
# ts_calibrate
```

The calibration program will be displayed on LCD after executing the `ts_calibrate` program. Before you start using the touchscreen, you should calibrate it.

### Run Qt Example

```
# /usr/share/qt/examples/widgets/calendarwidget/calendarwidget -qws
```

User can operate a calendar widget displayed on LCD through the selected device.

## 4.9 SQLite example for database

### 4.9.1 Introduction

SQLite is a powerful, embedded relational database management system in a compact C library. It offers support for multiple tables and indexes, transactions, views, triggers and a wide array of client interfaces and drivers. The library is self-contained and implemented in less than 25,000 lines of ANSI C, which is free to use for any purpose. It is fast, efficient and scalable, and it runs on a wide variety of platforms and hardware architectures. Its database format is binary-compatible between machines with different byte orders and scales up to 2 terabytes ( $2^{41}$  bytes) in size.

### SQLite Configuration

In order to use SQLite database, the user needs to select the SQLite package (in Buildroot) and also enable command-line editing for SQLite.

```
Target packages --->
  Libraries --->
    Database --->
      [*] sqlite
      [*] Command-line editing
```

The user can optionally also select the SQLite test as an example.

```
(Optional)
Target packages --->
  Miscellaneous --->
    [*] sqlite test
```

### 4.9.2 Build and Load Buildroot

After the configuration, user can make and boot the image on SC5xx. See [SC5xx EZ-Kit Linux Quick Start Guide](#) for details.

### 4.9.3 Run SQLite

Please refer to [SQLite Embedded Database](#) for detailed information about how to use SQLite on embedded Linux.

Here we give an example on how to run an SQLite test on SC5xx.

After selecting the "sqlite test", an SQL file **wishlist.sql** is built into the /usr directory.

The user can create a database using this file:

```
# cd /usr
# sqlite3 wishlist.db < wishlist.sql
# sqlite3 wishlist.db ".dump"
# sqlite_test "foo"
```

After configuring the database, run the following command to check the result.

```
# sqlite_test "foo"
```

If you see output as follows, the test passed.

```
+-----+
W I S H L I S T
+-----+
* foo <foo@test> *
```

## 4.10 Watchdog

### 4.10.1 Introduction

This page describes how watchdog is used in Linux on ADSP-SC5xx board, and we can test it manually.

A watchdog timer is a hardware device that will reset the system when certain fault conditions are met, we set a time out period , for example T seconds, when initially setup and enable the watchdog, this means after a time period of T, it will time out and the system get reset by hardware. If we clear the counter from software before time of T, it will never get time out and the system don't reset.



In Linux, the counter clearing job is done by a daemon process. In normal condition, the daemon will clear the counter every a few seconds before the timer out happens, so the system never get reset, if something bad happens on the system, software including the watchdog daemon stops working normally, time out will be triggered and system get reset.

Watchdog also is supported in Uboot, the counter clearing job is done in many parts of uboot by calling "WATCHDOG\_RESET()", such as "putc", nand w/r, etc.

## 4.10.2 Software Configuration

To enable the watchdog feature in Linux, we need to configure Buildroot and Linux kernel properly.

### Configure Buildroot

```
Target packages --->
  [*] Install the watchdog daemon startup script
```

After enabling the following option, a script entry will be added to the inittab file in the target board root file system, which will start and run the watchdog daemon automatically after the board is up, this daemon writes to watchdog device fd every 5 seconds to keep the system from resetting.

### Configure Linux Kernel

```
Device Drivers --->
  [*] Watchdog Timer Support --->
    *- Watchdog Timer Driver Core
    [ ] Disable watchdog shutdown on close
    [*] adi watchdog
```

To enable the watchdog feature in Uboot, we need to configure Uboot properly.

### Configure Uboot

Please refer to following patch to enable and configure hardware watchdog in the uboot source code.

```
diff --git a/include/configs/sc_adi_common.h b/include/configs
/sc_adi_common.h
```

```

index cfb5bd4..4cfc0b3 100644
--- a/include/configs/sc_adi_common.h
+++ b/include/configs/sc_adi_common.h
@@ -287,11 +287,9 @@
 # endif
 #endif

-/*
 #define CONFIG_HW_WATCHDOG
 #define CONFIG_ADI_WATCHDOG
 #define CONFIG_WATCHDOG_TIMEOUT_MSECS 10000
-*/

#define CONFIG_RSA

```

### 4.10.3 Build and Load Buildroot

A Buildroot image can now be built and loaded onto the target board. See [SC5xx EZ-Kit Linux Quick Start Guide](#) for details.

### 4.10.4 Test

#### Test in Linux

If we run "ps -a" on the target console, we will see something like following:

```

244 root      0 SW<   [kblockd]
365 root      0 SW     [kworker/0:1]
378 root      0 SW     [kswapd0]
401 root      0 SW<   [kworker/u3:0]
402 root      0 SW<   [kloopd]
425 root      0 SW     [spi0]
433 root      0 SW     [spi1]
435 root      0 SW     [spi2]
494 root      0 SW<   [kpsmoused]
516 root      0 SW     [iccqd]
517 root      0 SW     [iccqd]
540 root      0 SW<   [deferwq]
555 root      2140 S     /sbin/syslogd -n
558 root      2140 S     /sbin/klogd -n
566 root      2140 S     watchdog -t 5 /dev/watchdog
577 dbus      2044 S     dbus-daemon --system
587 root      4036 S     /usr/sbin/sshd
593 root      2208 S     sbin/inetd
595 root      2212 S     -sh
599 root      2212 R     ps -aux
#

```

This means a default watchdog daemon is running, kill the daemon and set the reset timeout time(-T) less than the reset time(-t), the system will reboot soon.

```
424 root      0 SW    [spi0]
432 root      0 SW    [spi1]
434 root      0 SW    [spi2]
490 root      0 SW<  [kpsmoused]
512 root      0 SW    [iccqd]
513 root      0 SW    [iccqd]
536 root      0 SW<  [deferwq]
552 root      2140 S    /sbin/syslogd -n
555 root      2140 S    /sbin/kload -n
563 root      2140 S    watchdog -t 5 /dev/watchdog
574 dbus      2044 S    dbus-daemon --system
584 root      4036 S    /usr/sbin/sshd
590 root      2208 S    sbin/inetd
592 root      2212 S    -sh
597 root      2212 R    ps -aux
# kill 563
# random: nonblocking pool is initialized
# watchdog -T 10 -t 20 /dev/watchdog
#
U-Boot 2015.01 ADI-1.1.0-00044-ged51f15-dirty (Nov 09 2016 - 15:19:47)

CPU:  ADSP ADSP-SC589-0.0 (Detected Rev: 1.1) (spi flash boot)
VCO:  450 MHz, Cclk0: 450 MHz, Sclk0: 112.500 MHz, Sclk1: 112.500 MHz, DCLK: 225z
OCLK: 150 MHz
I2C:  ready
DRAM:  224 MiB
MMC:  GRIFFIN SDH: 0
SF:  Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB, total 16 MiB
In:    serial
Out:   serial
Err:   serial
other init
Net:   dwmac.3100c000
Hit any key to stop autoboot:  5
```

## Test in Uboot

If we configure uboot as the configuration section, we will find watchdog enabled information after booting uboot.

```

U-Boot 2015.01 ADI-1.1.0-00046-g1fb70c7 (Jan 13 2017 - 13:31:10)

CPU:   ADSP ADSP-SC589-0.0 (Detected Rev: 1.1) (spi flash boot)
VCO: 450 MHz, Cclk0: 450 MHz, Sclk0: 112.500 MHz, Sclk1: 112.500 MHz, DCLK: 225 MHz
OCLK: 150 MHz
I2C:   ready
DRAM:  224 MiB
MMC:   GRIFFIN SDH: 0
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB, total 16 MiB
In:    serial
Out:   serial
Err:   serial
other init
Net:   dwmac.3100c000
Hit any key to stop autoboot:  0
sc # run update
Speed: 100, full duplex
Using dwmac.3100c000 device
TFTP from server 10.99.24.159; our IP address is 10.99.24.195
Filename 'u-boot.ldr'.
Load address: 0xc2000000
Loading: #####
          1.8 MiB/s
done
Bytes transferred = 282844 (450dc hex)
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB, total 16 MiB
SF: 524288 bytes @ 0x0 Erased: OK
SF: 282844 bytes @ 0x0 Written: OK
sc # save
Saving Environment to SPI Flash...
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB, total 16 MiB
Erasing SPI flash...Writing to SPI flash...done
sc # reset
resetting ...

U-Boot 2015.01 ADI-1.1.0-00046-g1fb70c7-dirty (Jan 13 2017 - 13:32:41)

CPU:   ADSP ADSP-SC589-0.0 (Detected Rev: 1.1) (spi flash boot)
VCO: 450 MHz, Cclk0: 450 MHz, Sclk0: 112.500 MHz, Sclk1: 112.500 MHz, DCLK: 225 MHz
OCLK: 150 MHz
      Watchdog enabled
I2C:   ready
DRAM:  224 MiB
MMC:   GRIFFIN SDH: 0
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB, total 16 MiB
In:    serial
Out:   serial
Err:   serial
other init
Net:   dwmac.3100c000
Hit any key to stop autoboot:  0
sc #

```

Then we can check the watchdog feature in uboot by accessing address illegally, and there will be data-abort exception which will cause system error, after the time you set to init the watchdog counter (refer to CONFIG\_WATCHDOG\_TIMEOUT\_MSECS in configuration section), the watchdog timeout interrupt happens, and watchdog will reboot the CPU automatically.

```

U-Boot 2015.01 ADI-1.1.0-00046-g1fb70c7-dirty (Jan 13 2017 - 13:32:41)

CPU:   ADSP ADSP-SC589-0.0 (Detected Rev: 1.1) (spi flash boot)
VCO: 450 MHz, Cclk0: 450 MHz, Sclk0: 112.500 MHz, Sclk1: 112.500 MHz, DCLK: 225 MHz
OCLK: 150 MHz
      Watchdog enabled
I2C:   ready
DRAM:  224 MiB
MMC:   GRIFFIN SDH: 0
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB, total 16 MiB
In:    serial
Out:   serial
Err:   serial
other init
Net:   dwmac.3100c000
Hit any key to stop autoboot:  0
sc # md.w 0xff30ffff
ff30ffff:data abort
pc : [<cff8c7b8>]      lr : [<cff8c778>]
sp : cfe62d10  ip : ff30ffff      fp : 00000004
r10: ff30ffff  r9 : cfe62f08      r8 : ff30ffff
r7 : 00000008  r6 : 00000040      r5 : 00000002  r4 : 00000008
r3 : 00000000  r2 : 00000000      r1 : cfe62d24  r0 : 00000000
Flags: nZCv  IRQs off  FIQs off  Mode SVC_32
Resetting CPU ...

### ERROR ### Please RESET the board ###

U-Boot 2015.01 ADI-1.1.0-00046-g1fb70c7-dirty (Jan 13 2017 - 13:32:41)

CPU:   ADSP ADSP-SC589-0.0 (Detected Rev: 1.1) (spi flash boot)
VCO: 450 MHz, Cclk0: 450 MHz, Sclk0: 112.500 MHz, Sclk1: 112.500 MHz, DCLK: 225 MHz
OCLK: 150 MHz
      Watchdog enabled
I2C:   ready
DRAM:  224 MiB
MMC:   GRIFFIN SDH: 0
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB, total 16 MiB
In:    serial
Out:   serial
Err:   serial
other init
Net:   dwmac.3100c000
Hit any key to stop autoboot:  5

```

## 5 Linux kernel and drivers

### 5.1 CAN Bus Data Transaction

#### 5.1.1 Introduction

This document describes how to do a data transaction test via CAN bus on SC5xx EZ-Board.

#### 5.1.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board
- Another ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board
- An RJ-11 crossover cable

Connect the RJ-11 port labelled "CAN0" on each board together with the crossover cable. Note only CAN0 is verified in this test, CAN1 is not.

Connect a serial cable for each of the board to PC so we have a serial console for each of the boards.

#### 5.1.3 Software Configuration

##### Configure Linux Kernel

Run **make linux-menuconfig** and configure the kernel as follows:

```
Networking support --->
  <*> CAN bus subsystem support --->
    <*> Raw CAN Protocol (raw access with CAN-ID filtering)
    <*> Broadcast Manager CAN Protocol (with content filtering)
  CAN Device Drivers --->
    <*> Platform CAN drivers with Netlink support
    [*] CAN bit-timing calculation
    <M> Analog Devices on-chip CAN
```

Note: For SC573-ezkit, we also need to disable the spidev by from the device tree file arch/arm/boot/dts/sc573-ezkit.dts, due to SPI chip select line conflict. Please refer to [SPI Driver](#) .

##### Configure Buildroot

Run **make menuconfig**, configure the application as following

```
Network Applications --->
  [*] can-utils
  [*] iproute2 --->
```

## Build buildroot

Firstly rebuild linux kernel with the command **make linux-rebuild** and then rebuild buildroot with **make**.

### 5.1.4 Test Example

#### Power Up and Boot The Boards

Power up and boot both boards with the same Buildroot image generated above

#### Bring Up CAN0 Interface On Both Boards

Run the following command on both of the boards:

```
root:/> modprobe bfin-can
bfin_can bfin_can.0: bfin_can device registered (reg_base=ffc02a00,
rx_irq=22, tx_irq=23, err_irq=43, sclk=125000000)
root:/> ip link set can0 type can bitrate 125000
bfin_can bfin_can.0: setting can bitrate:125000 brp:125 prop_seg:3
phase_seg1:3 phase_seg2:1
root:/> ifconfig can0 up
```

#### Data Send & Receive Test

Run the data receiving program on one of the boards:

```
root:/> candump can0
```

Run the data sending program on the other board:

```
root:/> cansend can0 123#AABBCCDD
root:/> cansend can0 123#R
root:/> cansend can0 1F334455#1122334455667788
root:/> cansend can0 1F334455#R
```

You should now see the transmitted data printing on the console of the receiving board:

```

can0  123  [4] AA BB CC DD
can0  123  [0] remote request
can0  1F334455  [8] 11 22 33 44 55 66 77 88
can0  1F334455  [0] remote request

```

## 5.2 CPU Frequency utility

### 5.2.1 Introduction

CPU frequency scaling enables the operating system to scale the CPU frequency up or down in order to save power. Clock scaling allows you to change the clock speed of the CPUs on the fly.

### 5.2.2 Hardware Setup

- ADSP-SC589 Ezkit v1.1 and above, or, ADSP-SC584 Ezkit v1.0 and above, or, ADSP-SC573 Ezkit v0.1 and above

The ADSP-SC5xx processors have Clock Generation Unit (CGU) support. The CGU allows program to change the PLL clock frequency and the CCLKn, SYSCLK, SCLKn, and OUTCLK clock scaling.

### 5.2.3 Enabling CPU Frequency Driver in Linux Kernel

Run **make linux-menuconfig** and configure the kernel as follows:

```

CPU Power Management --->
  CPU Frequency scaling --->
    [*] CPU Frequency scaling
    <*> CPU frequency translation statistics
    [*] CPU frequency translation statistics details
    Default CPUFreq governor (performance) --->
    *- 'performance' governor
    < > 'powersave' governor
    <*> 'userspace' governor for userspace frequency scaling
    < > 'ondemand' cpufreq policy governor
    < > 'conservative' cpufreq governor
    *** CPU frequency scaling drivers ***
    < > Generic DT based cpufreq driver
    [*] SC58X CPUFreq support

```



## 5.2.4 How to Change the CPU cpufreq

### Preferred Interface: sysfs

The preferred interface is located in the sysfs filesystem. If you mounted it at /sys, the cpufreq interface is located in a subdirectory "cpufreq" within the cpu-device directory (e.g. /sys/devices/system/cpu/cpu0/cpufreq/ for the first CPU).

```
cpuinfo_min_freq :          This file shows the minimum operating
frequency the processor can run at (in kHz)

cpuinfo_max_freq :          This file shows the maximum operating
frequency the processor can run at (in kHz)

scaling_driver :            This file shows what cpufreq driver is used to
set the frequency on this CPU

scaling_available_governors :
    This file shows the CPUfreq governors available in this
kernel. You can see the
    currently activated governor in scaling_governor, and
by "echoing" the name of another
    governor you can change it. Please note    that some
governors won't load - they only
    work on some specific architectures or processors.

scaling_min_freq and scaling_max_freq :
    These files show the current "policy limits" (in kHz).
By echoing new values into these
    files, you can change these limits.    NOTE: when
setting a policy you need to
    first set scaling_max_freq, then scaling_min_freq.

scaling_setspeed :
    By "echoing" a new frequency into this file you can
change the speed of the CPU,
    but only within the limits of scaling_min_freq and
scaling_max_freq.
```

### What Is A CPUFreq Governor?

### Governors In the Linux Kernel

## Performance

The CPUfreq governor “performance” sets the CPU statically to the highest frequency within the borders of `scaling_min_freq` and `scaling_max_freq`.

## Powersave

The CPUfreq governor “powersave” sets the CPU statically to the lowest frequency within the borders of `scaling_min_freq` and `scaling_max_freq`.

## Userspace

The CPUfreq governor “userspace” allows the user, or any userspace program running with UID “root”, to set the CPU to a specific frequency by making a sysfs file “`scaling_setspeed`” available in the CPU-device directory.

## Ondemand

The CPUfreq governor “ondemand” sets the CPU depending on the current usage. To do this the CPU must have the capability to switch the frequency very quickly. There are a number of sysfs file accessible parameters:

`sampling_rate`: Measured in  $\mu\text{S}$  ( $10^{-6}$  seconds), this is how often you want the kernel to look at the CPU usage and to make decisions on what to do about the frequency. Typically this is set to values of around '10000' or more.

`show_sampling_rate_(min|max)`: The minimum and maximum sampling rates available that you may set 'sampling\_rate' to.

`up_threshold`: Defines what the average CPU usage between the samplings of 'sampling\_rate' needs to be for the kernel to make a decision on whether it should increase the frequency. For example when it is set to its default value of '80' it means that between the checking intervals the CPU needs to be on average more than 80% in use to then decide that the CPU frequency needs to be increased.

`sampling_down_factor`: This parameter controls the rate that the CPU makes a decision on when to decrease the frequency. When set to its default value of '5' it means that at 1/5 the `sampling_rate` the kernel makes a decision to lower the frequency. Five “lower rate” decisions have to be made in a row before the CPU frequency is actually lowered. If set to '1' then the frequency decreases as quickly as it increases, if set to '2' it decreases at half the rate of the increase.

ignore\_nice\_load: This parameter takes a value of '0' or '1'. When set to '0' (its default), all processes are counted towards the 'cpu utilisation' value. When set to '1', the processes that are run with a 'nice' value will not count (and thus are ignored) in the overall usage calculation. This is useful if you are running a CPU intensive calculation on your laptop that you do not care how long it takes to complete, as you can 'nice' it and prevent it from taking part in the deciding process of whether to increase your CPU frequency.

## Conservative

The CPUfreq governor “conservative”, much like the “ondemand” governor, sets the CPU depending on the current usage. It differs in behaviour in that it gracefully increases and decreases the CPU speed rather than jumping to max speed the moment there is any load on the CPU. This behaviour is more suitable in a battery powered environment. The governor is tweaked in the same manner as the “ondemand” governor through sysfs with the addition of:

freq\_step: This describes what percentage steps the cpu freq should be increased and decreased smoothly by. By default the cpu frequency will increase in 5% chunks of your maximum CPU frequency. You can change this value to anywhere between 0 and 100 where '0' will effectively lock your CPU at a speed regardless of its load whilst '100' will, in theory, make it behave identically to the “ondemand” governor.

down\_threshold: Same as the 'up\_threshold' found for the “ondemand” governor but for the opposite direction. For example when set to its default value of '20' it means that if the CPU usage needs to be below 20% between samples to have the frequency decreased.

## 5.2.5 Change Core Clock Frequency via cpufreq-utils



In order to modify frequency settings with cpufreq-utils, you need to have the **userspace** governor enabled.

```
# cpufreq-info
cpufrequtils 005: cpufreq-info (C) Dominik Brodowski 2004-2006
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: sc58x cpufreq
  CPUs which need to switch frequency at the same time: 0
  hardware limits: 113 MHz - 450 MHz
  available frequency steps: 450 MHz, 225 MHz, 113 MHz
  available cpufreq governors: performance
  current policy: frequency should be within 113 MHz and 450 MHz.
                    The governor "performance" may decide which speed to
use
```

```
        within this range.
current CPU frequency is 450 MHz (asserted by call to hardware).
cpufreq stats: 450 MHz:0.00%, 225 MHz:0.00%, 113 MHz:0.00%
#
```

```
# cpufreq-set -f 225000
# cpufreq-info
cpufrequtils 005: cpufreq-info (C) Dominik Brodowski 2004-2006
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: sc58x cpufreq
  CPUs which need to switch frequency at the same time: 0
  hardware limits: 113 MHz - 450 MHz
  available frequency steps: 450 MHz, 225 MHz, 113 MHz
  available cpufreq governors: userspace, performance
  current policy: frequency should be within 113 MHz and 450 MHz.
                    The governor "userspace" may decide which speed to
use
        within this range.
current CPU frequency is 225 MHz (asserted by call to hardware).
cpufreq stats: 450 MHz:0.00%, 225 MHz:0.00%, 113 MHz:0.00% (1)
```

## 5.3 CRC Crypto Driver Guide

### 5.3.1 Introduction

The CRC peripheral in ADSP-SC5xx processors is a hardware block used to compute the CRC of a block of data. It is a CRC32 engine which computes the CRC value of 32-bit data words presented to it. For data words of < 32-bit in size, it is the responsibility of application to pack the data into 32-bit data units.

The main features of the CRC peripheral are:

- Memory Scan mode
- Memory Transfer mode
- Data Verify mode
- Data Fill mode
- 32b CRC polynomial (Programmable polynomials)
- Bit/Byte Mirroring option
- Fault/Error interrupt mechanisms

The Linux CRC driver is based on the Linux crypto driver framework. The Scatterlist Crypto API takes page vectors (scatterlists) as arguments, and works directly on memory pages. The CRC driver implements the asynchronous hash interface. The CRC results are calculated via array-descriptor-based DMA operation, which is generated at run time. If the input buffer length is not a multiple of 32 bits this driver appends zeroes automatically. The framework only supports Memory Scan mode. This driver can be found at **Linux-kernel/drivers/crypto/bfin\_crc.c** . The SC5xx and the BF609 share the same Linux CRC driver because the same CRC peripheral IP is used in both processors, the only difference is that the driver platform data is defined in the board file for BF609 but in the device tree file for SC5xx.

### 5.3.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board

### 5.3.3 Software Configuration

#### Configure Linux Kernel

Run **make linux-menuconfig** and configure the kernel as follows:

```
Cryptographic API --->
  *- Cryptographic algorithm manager
  <*> HMAC support
  [*] Hardware crypto devices --->
      <*> Support for Blackfin CRC hardware accelerator
```

To include a CRC test suite module in the kernel:

```
Cryptographic API --->
  [ ] Disable run-time self tests
  <*> Testing module
```

### 5.3.4 Example

When the kernel is booting up on SC5xx, you can check if the Crypto CRC driver probes devices correctly by looking for the following two lines of output:

```
Blackfin hardware CRC crypto driver
bfin-hmac-crc bfin-hmac-crc.0: initialized
```

You can also run a full test with **tcrypt** kernel module. This test module always exits with the warning "can't load module tcrypt" no matter whether the test passes or fails. However, if any test error occurs, "tcrypt: one or more tests failed" is printed out.

```
root: /> modprobe tcrypt mode=110
modprobe: can't load module tcrypt (kernel/crypto/tcrypt.ko): Resource
temporarily unavailable
```

### 5.3.5 Linux Kernel Crypto API

If you want to use this CRC driver via the Linux kernel crypto API, please refer to the generic [Linux Kernel Crypto API document](#).

## 5.4 Ethernet driver and performance

### 5.4.1 Introduction

This document describes how to enable the gigabit and 10/100 Ethernet devices on the ADSP-SC5xx Ezkit board in Linux. The performance benchmark data of the gigabit Ethernet device are provided for reference. The data is collected when running **netperf** testing with an Ubuntu host PC.

### 5.4.2 Hardware Requirement

- ADSP-SC589 Ezkit v1.1 and above, or,
- ADSP-SC584 Ezkit v1.0 and above, or,
- ADSP-SC573 Ezkit v0.1 and above
- 1 Ubuntu PC with Gigabit Ethernet port
- 1 Gigabit Ethernet switch

### 5.4.3 Software Configuration

#### Configure Buildroot

The **netperf** test utility is optional. You can build it for both the target board and your host Linux system.

```
Target packages --->
  Debugging, profiling and benchmark --->
    [*] netperf
Host utilities --->
  [*] host netperf
```

## Configure Linux Kernel

The on-chip Ethernet devices on the ADSP-SC5xx serial processors can be enabled in the Linux kernel configuration system using **make linux-menuconfig**.

```
Device Drivers --->
  [*] Network device support --->
    [*] Ethernet driver support --->
      [*] STMicroelectronics devices
          <*> STMicroelectronics 10/100/1000 Ethernet driver
          <*> STMMAC Platform bus support
```

## Configure Device Tree

### Gigabit Ethernet

The GMAC device is based on the STM MAC IP. MAC specific features can be tuned in the device tree Ethernet node. Node properties start with "snps" can be configured according to the requirement of the customer. See the **Documentation/devicetree/bindings/net/stmmac.txt** document in the Linux sources for details.

```
emac0: ethernet@0x3100C000 {
    compatible = "snps,dwmac", "snps,dwmac-3.710";
    reg = <0x3100C000 0x2000>;
    interrupt-parent = <&gic>;
    interrupts = <0 148 0>;
    interrupt-names = "macirq";
    snps,reset-gpio = <&gpb 14 0>;
    snps,fixed-burst;
    snps,burst_len = <0x8>; /* BLEN16 */
    snps,pbl = <2>;
    snps,force_sf_dma_mode;
```

```

        phy-mode = "rgmii";
        clock-names = "stmmaceth";
        pinctrl-names = "default";
        pinctrl-0 = <&eth0_default>;
    };

```

## 100M Ethernet

The 100M EMAC device is based on the STM MAC IP. MAC specific features start with "snps" in device node can be configured according to the [Documentation/devicetree/bindings/net/stmmac.txt](#) document in the Linux sources.

```

    emac1: ethernet@0x3100E000 {
        compatible = "snps,dwmac", "snps,dwmac-3.710";
        reg = <0x3100E000 0x2000>;
        interrupt-parent = <&gic>;
        interrupts = <0 149 0>;
        interrupt-names = "macirq";
        snps,fixed-burst;
        snps,burst_len = <0x4>; /* BLEN8 */
        snps,pbl = <1>;
        snps,force_thresh_dma_mode;
        phy-mode = "rmii";
        clock-names = "stmmaceth";
        pinctrl-names = "default";
        pinctrl-0 = <&eth1_default>;
    };

```

### 5.4.4 Performance Benchmark Example

The Ethernet performance is tested with the **netperf** utility running on a Linux host on one end, and on the ADSP-SC5xx EZKIT board on the other end. Please select **netperf** package in Buildroot for both target and host.



## GMAC Ethernet Result

Test case	No Preemption Client (250HZ)	No Preemption Server (250HZ)	Voluntary Preemption Client (250HZ)	Voluntary Preemption Server (250HZ)	Preemptible Client (250HZ)	Preemptible Server (250HZ)
TCP_STREAM	306.95 Mbps	200.38 Mbps	285.79 Mbps	195.34 Mbps	289.40 Mbps	176.93 Mbps
TCP_RR	1300.05 rps	1305.18rps	1266.92 rps	1293.63 rps	1273.37 rps	1275.15 rps
UDP_STREAM	304.04 Mbps	604.73 Mbps	304.40 Mbps	601.08 Mbps	277.51 Mbps	597.69 Mbps
UDP_RR	1323.08 rps	1341.38 rps	1321.42 rps	1338.78 rps	1298.85 rps	1297.38 rps

- **Client** means the netperf tool is running as a test client on the ADSP-SC5xx EZKIT, while **Server** means it is running as a test server on the ADSP-SC5xx EZKIT.
- **No Preemption, Voluntary Preemption and Preemptible** are 3 different kernel schedule policies.
- The ARM A5 core runs at 450M clock while the DDR is at 225M clock during the test.

## 5.5 Linux LCD device driver

### 5.5.1 Introduction

This section describes the steps required to build and use LCD device driver on Linux using an ADSP-SC58x board and a WVGA/LCD EI3 extender board.

WVGA/LCD EI3 Extender LCD board is not supported on the ADSP-SC573 EZ-KIT due to some board level hardware conflict, the SC573 processor itself is capable of doing this.

### 5.5.2 Hardware Setup

- ADSP-SC58x EZ-KIT: ADSP-SC589/SC584 EZ-Board
- A WVGA/LCD EI3 Extender Board

Connect the **J1** connector on the LCD EI3 Extender Board to the **P1A** connector on the SC58x-EZKIT

### 5.5.3 Software Configuration

The following configuration should be done on top of the sc589-ezkit/sc584-ezkit default configuration.

#### Configure Buildroot

Run **make menuconfig** and enable video test program in the Linux image:

```
Target packages --->
  Miscellaneous --->
    [*] video test program
```

#### Configure Linux Kernel

Run **make linux-menuconfig** and configure the kernel as follows:

#### Enable touchscreen and backlight

```
Device Drivers --->
  Input device support --->
    [*] Touchscreens --->
      <*> Analog Devices AD7879-1/AD7889-1 touchscreen interface
      <*> support SPI bus connection
```

#### Enable NL8048HL WVGA LCD for ADSP-SC58x

```
Device Drivers --->
  Graphics support --->
    Frame buffer Devices --->
      <*> Support for frame buffer devices --->
      <*> NEC NL8048HL WVGA LCD for BF609
```

#### Avoid LCD driver probe failure by disabling CAN bus support

```
[*] Networking support --->
```

```
< > CAN bus subsystem support ----
```

## 5.5.4 Example

### Enable LCD Back-light

```
# echo 102 > /sys/class/gpio/export  
# echo high > /sys/class/gpio/gpio102/direction
```

Make sure your LCD screen becomes white at this point.

### Soft Switch Configuration

The settings in this section is only apply to ADSP-SC584 EZ-Board.

```
# echo 484 > /sys/class/gpio/export  
# echo high > /sys/class/gpio/gpio484/direction  
# echo 491 > /sys/class/gpio/export  
# echo high > /sys/class/gpio/gpio491/direction
```

### Run video\_test Program

You will see a crossing curve like an "8" in the LCD window.

```
# video_test
```

### How to Install Modules

Select these drivers as modules:

```
<M> Analog Devices AD7879-1/AD7889-1 touchscreen interface  
<M> support SPI bus connection  
  
<M> Support for frame buffer devices ---->  
<M> NEC NL8048HL WVGA LCD for BF609
```

If AD7879 module has already been installed, removed it first:

```
# lsmod
Module                Size  Used by
ad7879_spi            2222  0
ad7879                3763  1 ad7879_spi
# modprobe -r ad7879_spi
# lsmod
Module                Size  Used by
```

Then install the modules as follows:

```
# modprobe bf609_n18048
bf609_n18048 31040000.lcd: LCD fb0 registered@f4040000,dma=28,irq=34,
cs=39
# modprobe ad7879_spi
input: AD7879 Touchscreen as /devices/platform/scb/31042000.spi
/spi_master/spi0/spi0.32/input/input1
```

Now you can run LCD test as above.

## 5.6 Linux Sound Driver

### 5.6.1 Introduction

This section describes the steps required to build and use sound driver to record and play audio files on Linux using an ADSP-SC5xx board.

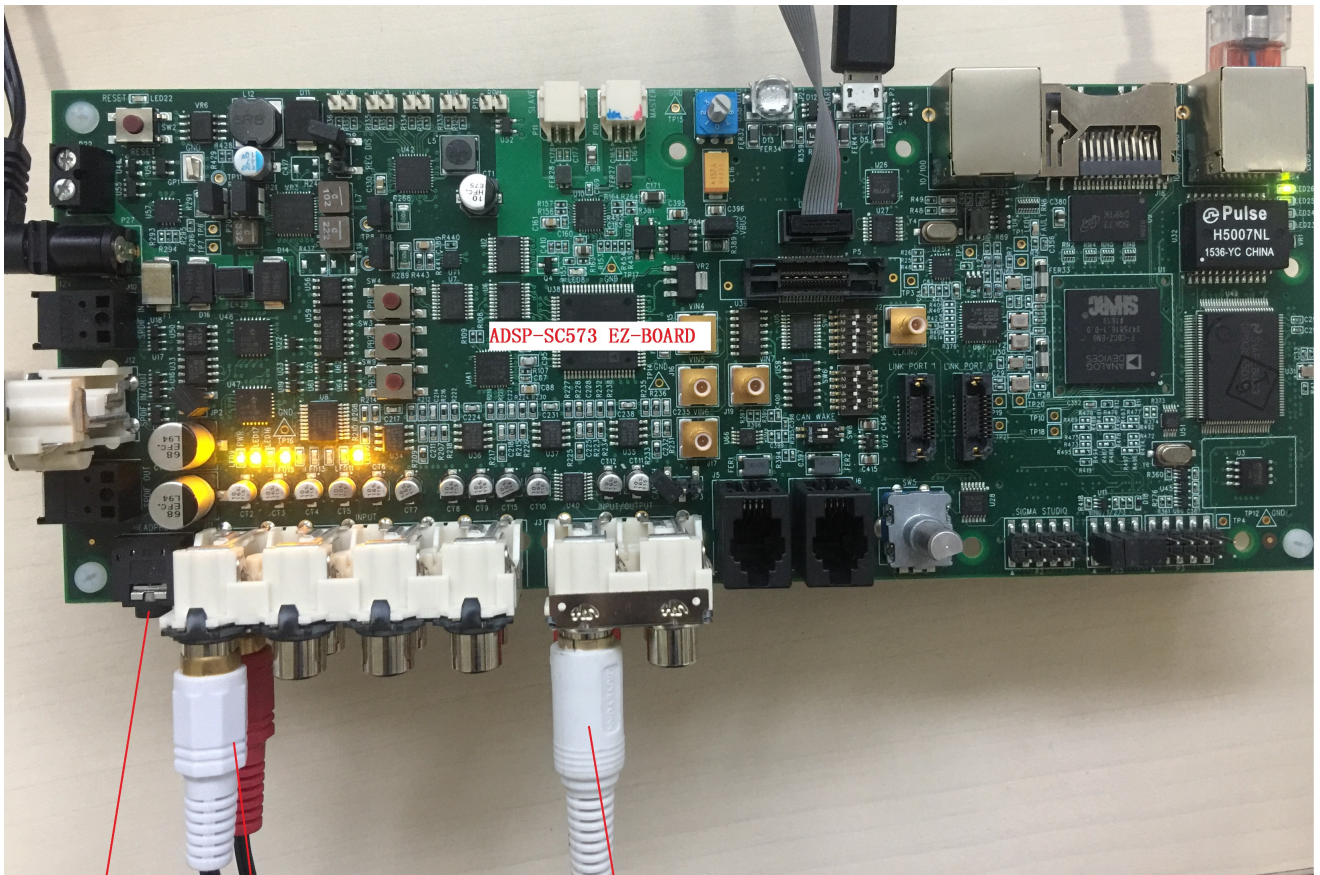
The Advanced Linux Sound Architecture (ALSA) provides audio and MIDI functionality to the Linux operating system.

For more information about the Advanced Linux Sound Architecture (ALSA), please refer to [http://www.alsa-project.org/main/index.php/Main\\_Page](http://www.alsa-project.org/main/index.php/Main_Page).

### 5.6.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board
- Two line cables, a line-in and line-out cables

Take the ADSP-SC573 EZ-Board as an example, connect line-in cables to J3 and line-out cables to J11, or you can plug one headphone into the Headphone JACK directly for DAC.



Headphone JACK  
Line outs from DAC  
To Headphone

Line-in to ADC  
from MIC

### 5.6.3 Software Configuration

The following configuration should be done on top of the SC589-ezkit/SC584-ezkit/SC573-ezkit default configuration.

#### Configure Buildroot

Run **make menuconfig** and configure the image as follows:

#### Enable alsa-lib support

```
Target packages --->
  Libraries --->
    Audio/Sound --->
      *- alsa-lib
```

## Compile alsa-utils into Linux image

The ALSA utility provides many simple and powerful tools for testing the ALSA drivers to make sure they are working correctly, such as **arecord**, **aplay**, **amixer**, **speaker-test**, etc. **arecord** is a command-line soundfile recorder for the ALSA soundcard driver, it supports several file formats and multiple soundcards with multiple devices, **aplay** is much the same, only it plays instead of recording; **amixer** allows command-line control of the mixer for the ALSA soundcard driver; **speaker-test** is a command-line speaker test tone generator for ALSA, it can be used to test the speakers of a device.

```
Target packages --->
  Audio and video applications --->
    [*] alsa-utils
        ALSA utils selection --->
          [*] amixer
          [*] aplay/arecord
          [*] speaker-test
    [*] mplayer
    [*] Build and install mplayer (NEW)
```

## Configure Linux Kernel

Run **make linux-menuconfig** and configure the kernel as follows:

### Enable ADAU1962 sound card driver

```
Device Drivers --->
  <*> Sound card support --->
    <*> Advanced Linux Sound Architecture --->
      <*> ALSA for SoC audio support --->
        <*> SoC Audio for the ADI SC5XX chip
        <*> Support for the ADAU1962 board on
SC5XX ezkit board
```

### Enable ADAU1979 sound card driver

```
Device Drivers --->
  <*> Sound card support --->
    <*> Advanced Linux Sound Architecture --->
      <*> ALSA for SoC audio support --->
```

```
<*> SoC Audio for the ADI SC5XX chip
<*> Support for the ADAU1979 board on SC5XX
ezkit board
```

## 5.6.4 Example

### Find the device

After the configuration in above steps, you are able to find the audio device information in boot log:

```
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
01,2a,a0,00
00,00,01,00
06,00,00,00
00,00,00,00
00,00,00,00
00,00,00,00
00,00,00,00
00,aa,aa,aa
sc5xx-i2s-dai 31002000.i2s: SPORT create success
snd-sc5xx scb:sound: adau1962-hifi <-> 31002000.i2s mapping ok
snd-sc5xx scb:sound: adau1977-hifi <-> 31002000.i2s mapping ok
TCP: cubic registered
NET: Registered protocol family 17
do_initcall_level level 7
ThumbEE CPU extension supported.
IP-Config: Complete:
    device=eth0, hwaddr=02:80:ad:20:31:e8, ipaddr=10.99.24.186, mask=255.255.255.0, gw=10.99.24.1
    host=sc57x, domain=, nis-domain=(none)
    bootserver=10.99.24.159, rootserver=10.99.24.159, rootpath=
ALSA device list:
  #0: sc5xx-asoc-card
Freeing unused kernel memory: 9200K (c04de000 - c0dda000)
Starting logging: OK
Starting mdev...
Starting watchdog...
Initializing random number generator... random: dd urandom read with 18 bits of entropy available
done.
Starting network...
/bin/sh: run-parts: not found
ssh-keygen: generating new host keys: ED25519
Starting sshd: OK
Starting inetd: OK

Welcome to Buildroot
buildroot login: stmmaceth 3100c000.ethernet eth0: Link is Up - 100Mbps/Full - flow control rx/tx

Welcome to Buildroot
buildroot login: root
Password:
```

You are able to use **arecord -l** and **aplay -l** to find the card number.

```
# arecord -l
**** List of CAPTURE Hardware Devices ****
card 0: sc5xxasoccard [sc5xx-asoc-card], device 1: ADAU1979 adau1977-
hifi-1 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
# aplay -l
**** List of PLAYBACK Hardware Devices ****
```

```
card 0: sc5xxasoccard [sc5xx-asoc-card], device 0: ADAU1962 adau1962-
hifi-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

In **capture** case, you can find "0" is the **card number**, and "1" is the **device number**;

In **playback** case, you can find "0" is the **card number**, and "0" is the **device number**.

## Amixer setting

If you want to unmute the ADC and DAC or adjust the volume, you can use the **amixer** tool.

```
# amixer -c 0 controls
numid=41,iface=MIXER,name='ADC1 Capture Volume'
numid=49,iface=MIXER,name='ADC1 DC Substraction Capture Switch'
numid=45,iface=MIXER,name='ADC1 Highpass-Filter Capture Switch'
numid=42,iface=MIXER,name='ADC2 Capture Volume'
numid=50,iface=MIXER,name='ADC2 DC Substraction Capture Switch'
numid=46,iface=MIXER,name='ADC2 Highpass-Filter Capture Switch'
numid=43,iface=MIXER,name='ADC3 Capture Volume'
numid=51,iface=MIXER,name='ADC3 DC Substraction Capture Switch'
numid=47,iface=MIXER,name='ADC3 Highpass-Filter Capture Switch'
numid=44,iface=MIXER,name='ADC4 Capture Volume'
numid=52,iface=MIXER,name='ADC4 DC Substraction Capture Switch'
numid=48,iface=MIXER,name='ADC4 Highpass-Filter Capture Switch'
numid=3,iface=MIXER,name='DAC Deemphasis Switch'
numid=40,iface=MIXER,name='DAC Oversampling Rate'
numid=2,iface=MIXER,name='DAC Playback Switch'
numid=1,iface=MIXER,name='DAC Playback Volume'
numid=5,iface=MIXER,name='DAC1 Playback Switch'
numid=4,iface=MIXER,name='DAC1 Playback Volume'
numid=6,iface=MIXER,name='DAC1 Power Adjust'
numid=32,iface=MIXER,name='DAC10 Playback Switch'
numid=31,iface=MIXER,name='DAC10 Playback Volume'
numid=33,iface=MIXER,name='DAC10 Power Adjust'
numid=35,iface=MIXER,name='DAC11 Playback Switch'
numid=34,iface=MIXER,name='DAC11 Playback Volume'
numid=36,iface=MIXER,name='DAC11 Power Adjust'
numid=38,iface=MIXER,name='DAC12 Playback Switch'
numid=37,iface=MIXER,name='DAC12 Playback Volume'
numid=39,iface=MIXER,name='DAC12 Power Adjust'
numid=8,iface=MIXER,name='DAC2 Playback Switch'
numid=7,iface=MIXER,name='DAC2 Playback Volume'
numid=9,iface=MIXER,name='DAC2 Power Adjust'
numid=11,iface=MIXER,name='DAC3 Playback Switch'
numid=10,iface=MIXER,name='DAC3 Playback Volume'
numid=12,iface=MIXER,name='DAC3 Power Adjust'
numid=14,iface=MIXER,name='DAC4 Playback Switch'
```



```

numid=13,iface=MIXER,name='DAC4 Playback Volume'
numid=15,iface=MIXER,name='DAC4 Power Adjust'
numid=17,iface=MIXER,name='DAC5 Playback Switch'
numid=16,iface=MIXER,name='DAC5 Playback Volume'
numid=18,iface=MIXER,name='DAC5 Power Adjust'
numid=20,iface=MIXER,name='DAC6 Playback Switch'
numid=19,iface=MIXER,name='DAC6 Playback Volume'
numid=21,iface=MIXER,name='DAC6 Power Adjust'
numid=23,iface=MIXER,name='DAC7 Playback Switch'
numid=22,iface=MIXER,name='DAC7 Playback Volume'
numid=24,iface=MIXER,name='DAC7 Power Adjust'
numid=26,iface=MIXER,name='DAC8 Playback Switch'
numid=25,iface=MIXER,name='DAC8 Playback Volume'
numid=27,iface=MIXER,name='DAC8 Power Adjust'
numid=29,iface=MIXER,name='DAC9 Playback Switch'
numid=28,iface=MIXER,name='DAC9 Playback Volume'
numid=30,iface=MIXER,name='DAC9 Power Adjust'

```

If you want to increase the volume of ADC1, you can set a bigger value.

```

# amixer -c 0 cget numid=41
numid=41,iface=MIXER,name='ADC1 Capture Volume'
; type=INTEGER,access=rw-R-,values=1,min=0,max=255,step=0
: values=95
| dBminmaxmute-min=-35.62dB,max=60.00dB

```

```

# amixer -c 0 cset numid=41 200
numid=41,iface=MIXER,name='ADC1 Capture Volume'
; type=INTEGER,access=rw---R-,values=1,min=0,max=255,step=0
: values=200
| dBminmaxmute-min=-35.62dB,max=60.00dB

```

'DAC Playback Switch' is the DAC master mute.

'DAC Playback Volume' is the master volume control.

'DAC1 Playback Switch' is the DAC channel 1 mute control.

```

# amixer -c 0 -cget numid=5
numid=5,iface=MIXER,name='DAC1 Playback Switch'
; type=BOOLEAN,access=rw-----,value=1
: values=on

```

You can set 0 or 1 to mute or unmute DAC channel 1.

'DAC1 Playback Volume' is the DAC channel 1 volume control.

```
# amixer -c 0 cget numid=4
numid=4,iface=MIXER,name='DAC1 Playback Volume'
; type=INTEGER,access=rw---R-,values=1,min=0,max=255,step=0
: values=255
| dBminmaxmute-min=-95.62dB,max=0.00dB
```

You can set value 0 - 255. The range is from -95.625 dB to 0 dB. Each 1-bit step corresponds to a 0.375dB change in volume.

'DAC1 Power Adjust' is the DAC channel 1 power adjust control.

```
# amixer -c 0 cget numid=6
numid=6,iface=MIXER,name='DAC1 Power Adjust'
; type=ENUMERATED,access=rw-----,values=1,items=4
; Item #0 'Low Power'
; Item #1 'Lowest Power'
; Item #2 'Best Performance'
; Item #3 'Good Performance'
: values=2
```

You can choose a power state for this DAC channel.

## Record audio file

You can use arecord tool to record audio stream, and we already know "0" is the card number, "1" is the device number for capture case in "Find the device" section, so you can use "-D hw:<card number>, <device number>" to select the pcm device.

For example, record stereo 48KHz 32bits stream:

```
# arecord -D hw:0,1 -c 2 -r 48000 -f S32_LE -t wav 1.wav
```

## Play audio file

If you have a sample wav file, you can use **aplay** tool or **speaker-test** tool or **mplayer** tool to test the playback, and we already know "0" is the card number, "0" is the device number for playback case, so you can use "-D hw:<card number>, <device number>" to select the pcm device. Also you can use "-D plug:dmix:<card number>" for dmix plugin to test the playback.

```
# aplay -D hw:0,0 sample.wav
# aplay -D plug:dmix:0 sample.wav
```

Or you can use **speaker-test** tool:

```
# speaker-test -D hw:0,0 -c 2 -t sine -F S32_LE
# speaker-test -D plug:dmix:0 -c 2 -t sine -f 1000 -r 4800
```

Or you can use **mplayer** tool:

```
# mplayer sample.wav
```

## Loopback

If you want to hear the audio at the ADC input, you can use following command:

```
# arecord -D hw:0,1 -c 2 -r 48000 -f S32_LE | aplay -D hw:0,0
```

## 5.7 GPIO operation

### 5.7.1 Introduction

A General Purpose Input/Output (GPIO) is a flexible software-controlled digital signal. They are provided from many kinds of chip, and are familiar to Linux developers working with embedded and custom hardware. Each GPIO represents a bit connected to a particular pin, or “ball” on Ball Grid Array (BGA) packages. Board schematics show which external hardware connects to which GPIOs. Drivers can be written generically, so that board setup code passes such pin configuration data to drivers.

Often different aspects of the GPIO need to be controlled, such as:

- direction: input or output
- value: set or unset
- polarity: high or low
- edge: rising or falling

### 5.7.2 Hardware

- ADSP-SC5xx EZKIT Board: ADSP-SC589/SC584/SC573 EZKIT Board

In the ADSP-SC58x processors there are 7 gpio blocks, called PORTA..PORTG. Each PORTx interface has 16 GPIO pins. PORT(A-G) pins are brought out on the 110 pins of SC589-EZKIT(PA0-PG13) and 80 pins of SC584-EZKIT(PA0-PD16). While in the ADSP-SC573 processors there are only 6 gpio blocks, called PORTA..PORTF.

### 5.7.3 Application space GPIO support

Standard Linux kernel have inside a special interface allow to access to GPIO pins. You can use kernel menuconfig (linux-menuconfig) to verify that these interfaces are active in your kernel and, if necessary, enable them.

The kernel tree path is the following:

```
Symbol: GPIO_SYSFS [=y]
  Prompt: /sys/class/gpio/... (sysfs interface)
  Defined at drivers/gpio/Kconfig:51
  Depends on: GPIOLIB && SYSFS && EXPERIMENTAL
  Location:
    -> Kernel configuration
    -> Device Drivers
    -> GPIO Support (GPIOLIB [=y])
```

### 5.7.4 Paths in Sysfs

There are three kinds of entry in /sys/class/gpio:

- Control interfaces used to get userspace control over GPIOs
- GPIOs themselves
- GPIO controllers ("gpio\_chip" instances)

This is in addition to standard files including the "device" symlink.

The control interfaces are write-only:

```
/sys/class/gpio/
```

```
"export" ... Userspace may ask the kernel to export control of
a GPIO to userspace by writing its number to this file.
```

```
Example: "echo 19 > export" will create a "gpio19" node
for GPIO #19, if that's not requested by kernel code.
```

```
"unexport" ... Reverses the effect of exporting to userspace.
```

Example: "echo 19 > unexport" will remove a "gpio19" node exported using the "export" file.

GPIO signals have paths like /sys/class/gpio/gpio42/ (for GPIO #42) and have the following read/write attributes:

/sys/class/gpio/gpioN/

"direction" ... reads as either "in" or "out". This value may normally be written. Writing as "out" defaults to initializing the value as low. To ensure glitch free operation, values "low" and "high" may be written to configure the GPIO as an output with that initial value.

Note that this attribute *will not exist* if the kernel doesn't support changing the direction of a GPIO, or it was exported by kernel code that didn't explicitly allow userspace to reconfigure this GPIO's direction.

"value" ... reads as either 0 (low) or 1 (high). If the GPIO is configured as an output, this value may be written; any nonzero value is treated as high.

If the pin can be configured as interrupt-generating interrupt and if it has been configured to generate interrupts (see the description of "edge"), you can poll(2) on that file and poll(2) will return whenever the interrupt was triggered. If you use poll(2), set the events POLLPRI and POLLERR. If you use select(2), set the file descriptor in exceptfds. After poll(2) returns, either lseek(2) to the beginning of the sysfs file and read the new value or close the file and re-open it to read the value.

"edge" ... reads as either "none", "rising", "falling", or "both". Write these strings to select the signal edge(s) that will make poll(2) on the "value" file return.

This file exists only if the pin can be configured as an interrupt generating input pin.

"active\_low" ... reads as either 0 (false) or 1 (true). Write any nonzero value to invert the value attribute both for reading and writing. Existing and subsequent

```
poll(2) support configuration via the edge attribute
for "rising" and "falling" edges will follow this
setting.
```

GPIO controllers have paths like `/sys/class/gpio/gpiochip42/` (for the controller implementing GPIOs starting at #42) and have the following read-only attributes:

```
/sys/class/gpio/gpiochipN/

"base" ... same as N, the first GPIO managed by this chip

"label" ... provided for diagnostics (not always unique)

"ngpio" ... how many GPIOs this manages (N to N + ngpio - 1)
```

## 5.7.5 Dump the GPIO configuration

```
# cat /sys/kernel/debug/gpio
GPIOs 0-15, platform/31004000.gport, adi-gpio:
GPIOs 16-31, platform/31004080.gport, adi-gpio:
  gpio-30 (mdio-reset          ) out hi
GPIOs 32-47, platform/31004100.gport, adi-gpio:
GPIOs 48-63, platform/31004180.gport, adi-gpio:
GPIOs 64-79, platform/31004200.gport, adi-gpio:
GPIOs 80-95, platform/31004280.gport, adi-gpio:
GPIOs 96-101, platform/31004300.gport, adi-gpio:
GPIOs 480-495, i2c/0-0022, mcp23017, can sleep:
GPIOs 496-511, i2c/0-0021, mcp23017, can sleep:
```

## 5.8 I2C Bus

### 5.8.1 Introduction

The Inter-Integrated Circuit (I<sup>2</sup>C) bus is a two wire multi-master/slave low speed serial bus. Multiple slave devices may be accessed over the same bus, using a unique 7-bit addresses for each slave. Communication on the bus is half-duplex, and slaves do not transmit any data unless a master has addressed it first.

From the Linux point of view the driver for I2C hardware controller is the adapter driver, while drivers for the peripheral I2C devices are the client drivers. The adapter driver is provided by ADI, and most of the work that a product developer needs to do is in implementing the client driver to connect a specific I2C slave device to applications running under Linux on the SC5xx processor.

This document focuses on explaining the programming interface for the I2C client driver, it talks about how to create the client driver from both kernel and user space, to guide the audience to develop the client driver of their own.

## 5.8.2 I2C in the Linux Kernel

This section talks about the I2C kernel driver framework and how to implement the client I2C driver from kernel space.

### I2C Kernel Driver

The main source code for the SC5xx I2C adapter driver is in `drivers/i2c/busses/i2c-bfin-twi.c`. The "Blackfin" name is used here because the ADSP-SC5xx processors use the same hardware IP for the I2C interface as the ADSP-BFxxx series. The device tree description for the 3 controllers, `i2c0`, `i2c1` and `i2c2` is in file `arch/arm/boot/dts/sc58x.dtsi` or `arch/arm/boot/dts/sc57x.dtsi`. Select from the following options to enable the I2C adapter driver, and you can set the clock speed from there:

```
Linux Kernel Configuration
  Device Drivers --->
    I2C support --->
      I2C Hardware Bus support --->
        <*> Blackfin TWI I2C support
          (50) Blackfin TWI I2C clock (kHz)
```

### I2C Client Driver Example

In this section we take audio codec driver for ADAU1977 as an example to show the typical code structure for the I2C client driver, demonstrating how users normally initialize a client's driver, register it to the system, then use the registered method to do data read/write via the I2C bus. The main source code file for this example includes `sound/soc/codecs/adau1977-i2c.c`, `sound/soc/codecs/adau1977.c`, `drivers/base/regmap/regmap-i2c.c` and device tree file `arch/arm/boot/dts/sc589-ezkit.dts`.

### Client driver instance

The following C code defines a client driver instance, which is later registered to the Linux I2C subsystem. See the source code for details on the implementation of each method.

```
static struct i2c_driver adau1977_i2c_driver = {
```

```

.driver = {
    .name = "adau1977",
    .owner = THIS_MODULE,
    .of_match_table = of_match_ptr(adau1977_dt_ids),
},
.probe = adau1977_i2c_probe,
.remove = adau1977_i2c_remove,
.id_table = adau1977_i2c_ids,
};

```

## Register

The following C code registers the above client driver to the Linux I2C sub-system

```

module_i2c_driver(adau1977_i2c_driver);

```

## Data read/write method

This audio codec driver uses the regmap programming interface, which is built upon the lower level of the I2C data read/write interface, to do the data read/write immediately with code of the following style in the file `sound/soc/codecs/adau1977.c`

```

ret = regmap_read(adau1977->regmap, ADAU1977_REG_PLL, &val);

ret = regmap_write(adau1977->regmap, ADAU1977_REG_POWER,
                  ADAU1977_POWER_RESET);

```

This regmap interface for the I2C bus is implemented in file `drivers/base/regmap/regmap-i2c.c`. In this file, we can see the I2C bus data read/write is carried out via either the raw I2C transfer interface or the SMBus interface. Take the SMBus for example:

Read data:

```

static int regmap_smbus_byte_reg_read(void *context, unsigned int reg,
                                     unsigned int *val)
{
    struct device *dev = context;
    struct i2c_client *i2c = to_i2c_client(dev);
    int ret;
    if (reg > 0xff)
        return -EINVAL;
    ret = i2c_smbus_read_byte_data(i2c, reg);
    if (ret < 0)

```



```

        return ret;
    *val = ret;
    return 0;
}

```

Write data:

```

static int regmap_smbus_byte_reg_write(void *context, unsigned int reg,
                                       unsigned int val)
{
    struct device *dev = context;
    struct i2c_client *i2c = to_i2c_client(dev);
    if (val > 0xff || reg > 0xff)
        return -EINVAL;
    return i2c_smbus_write_byte_data(i2c, reg, val);
}

```

SMBus protocol and its APIs is a subset of the I2C protocol and is widely used for the I2C device driver in Linux, get more details on the SMBus protocol from the kernel source: [Documentation/i2c/smbus-protocol](#).

### 5.8.3 I2C in User Space

Usually I2C devices are controlled by the device driver in kernel space, but we can also implement this from user space.

#### Kernel configuration

Linux kernel introduced an i2c-dev layer. It exports the adapter driver for the I2C controller as device nodes to user space, so we can implement the driver for a specific I2C client in user space. To take advantage of this feature, we need to enable the i2c-dev interface from the kernel menu configuration

```

Linux Kernel Configuration
  Device Drivers  --->
    I2C support  --->
      <*> I2C device interface

```

Get more details about i2c-dev interface from [Documentation/i2c/dev-interface](#).

#### Example

The TWI LCD I2C driver code is a good example for an I2C user space driver. Enable the following package in Buildroot:

```
Buildroot Configuration
  Target packages --->
    Miscellaneous --->
      [*] twi lcd test
```

Once rebuilt, we will have the source code in `output/build/twi_lcd_test`. The file `twilcd_userspace_test.c` is the main source code for this example. We can see the general steps for implementing a user space I2C client driver in function **main**, code skeleton as follows:

```
int main (int argc, char *argv[])
{
    int file;
    int adapter_nr = 2; /* probably dynamically determined */
    char filename[20];
    snprintf(filename, 19, "/dev/i2c-%d", adapter_nr);

    /** Open the I2C controller device node **/
    file = open(filename, O_RDWR);

    if (file < 0) {
        /* ERROR HANDLING; you can check errno to see what went
wrong */
        exit(1);
    }

    int addr = 0x40; /* The I2C address */

    /** Set the address for the I2C peripheral device **/
    if (ioctl(file, I2C_SLAVE, addr) < 0) {
        /* ERROR HANDLING; you can check errno to see what went
wrong */
        exit(1);
    }
    /* Using SMBus commands for data transaction*/
    res = i2c_smbus_read_word_data(file, reg);
    if (res < 0) {
        /* ERROR HANDLING: i2c transaction failed */
    } else {
        /* res contains the read word */
    }
}
}
```

## 5.9 Link Port driver

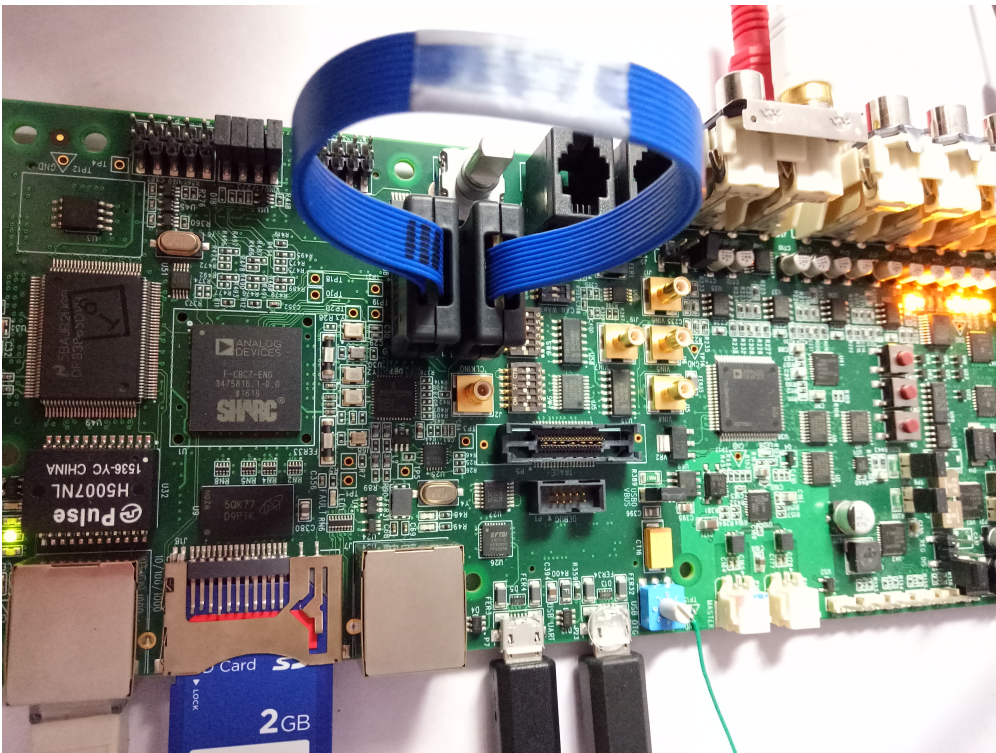
### 5.9.1 Introduction

Link ports allow the processor to connect to other processors or peripheral link ports using a simple communication protocol for high-speed parallel data transfer. This peripheral allows a variety of I/O peripheral interconnection schemes to I/O peripheral devices as well as co-processing and multiprocessing schemes. This document describes how to do a data transaction test via link ports on SC5xx EZ-Board.

### 5.9.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board
- A linkport cable

Connect the LINK PORT 0 and LINK PORT 1 with the matched cable in the target board.



Note: You need to run this test on a board version 1.1 or above version if you are using ADSP-SC584 EZ-Kit Board.

### 5.9.3 Software configuration

The following configuration should be done on top of the SC589-ezkit/SC584-ezkit/SC573-ezkit default configuration.

#### Buildroot configuration

```
Target packages --->
  Miscellaneous --->
    [*] bfin linkport test
```

#### Linux kernel configuration

```
[*] Networking support --->
  <N> CAN bus subsystem support ----
Device Drivers --->
  <N> Memory Technology Device (MTD) support ----
  [N] SPI support ----
  Character devices --->
    <*> Blackfin LINKPORT driver
```

#### Build and Load Buildroot

Now build the buildroot and load the image onto the target board. See [SC5xx EZ-Kit Linux Quick Start Guide](#) for more details.

### 5.9.4 Test Example

linkport\_test will send data to linkport1, and receive data from linkport0, then verify the data.

```
# linkport_test
linkport test passed
```

## 5.10 USB interface

### 5.10.1 Introduction

This document introduces usage of USB interface under Linux. This document apply to ADSP-SC573, SC584 and SC589. Take ADSP-SC589 as example in following sections.

### 5.10.2 Hardware Setup

An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board.

The SC5xx-EZKIT boards each have the following two USB connectors:

- USB OTG HOST/DEVICE port: can act as host and device, supports USB host mode, USB gadget mode, and USB On-The-Go (dynamic mode switching)
- USB to UART port: for serial console support. This connector is not part of the discussion below.

The SC589-EZKIT board has an additional USB connector:

- USB HS DEVICE port: can act as device, supports USB gadget mode only

The connector receptacle (socket type) used by all of the above is USB Micro-A/B. The cable types which may be used with each are as follows:

- Host mode and On-The-Go: USB Micro-A male plug
- Device mode (and USB to UART): USB Micro-B male plug

### 5.10.3 Software Configuration

All configuration operations are conducted within the Buildroot source directory.

Buildroot can be configured using the following command:

```
# make menuconfig
```

The Linux kernel can be configured using the following command:

```
# make linux-menuconfig
```

### Default Config

The following configuration should be done on top of the SC589-ezkit/SC584-ezkit/SC573-ezkit default configuration.

```
# make sc589-ezkit_defconfig (for SC584/SC573,it should be "make sc584-  
ezkit_defconfig/sc573-ezkit_defconfig")
```

## Build and Load Buildroot

After the configuration, users can make and boot the image on SC5xx. Please refer [SC5xx EZ-Kit Linux Quick Start Guide](#) for details.

### 5.10.4 USB HOST Example

Take usage of USB memory stick as example. ADSP\_SC5xx acts as HOST in this example.

- Supported Port: USB OTG port.
- Supported USB Mode: OTG and Host.
- Supported Cable: USB Micro-A male plug.

#### 1) Config as HOST

Please select in order:

```
Device Drivers  --->  
  [*] USB support  --->  
    <*> Support for Host-side USB  
          USB Physical Layer drivers  --->  
            <*> NOP USB Transceiver Driver  
    <*> USB Gadget Support  --->
```

ADI USB specific configuration depends on selection of the "Support for Host-side USB" and "USB Gadget Support" options. ADI USB specific configuration will not appear until "Support for Host-side USB" and "USB Gadget Support" options are selected at the same time:

```
Device Drivers  --->  
  [*] USB support  --->  
    <*> Inventra Highspeed Dual Role Controller (TI, ADI,  
    <*> ADI  
          MUSB DMA mode (Inventra)  --->
```

```

Device Drivers --->
  [*] USB support --->
    <*>      Inventra Highspeed Dual Role Controller (TI, ADI,
    MUSB Mode Selection (Host only mode) --->

```

## 2) Mass Storage Support Configuration

```

Device Drivers
  SCSI device support --->
    <*> SCSI device support
    <*> SCSI disk support
Device Drivers --->
  USB support
    <*>      USB Mass Storage support

```

If USB interface is configured as Dual-Role, don't forget to enable a USB gadget driver, such as `g_serial`. Gadget serial driver can be compiled as a module:

```

Device Drivers --->
  [*] USB support --->
    <*>      USB Gadget Support --->
    <M>      Serial Gadget (with CDC ACM and CDC OBEX s

```

After Linux boot up, plug a USB memory stick into USB OTG HOST/DEVICE port.

If USB interface is configured Dual-Role, make sure that the gadget serial driver is loaded first.

```
# modprobe g_serial
```

Kernel outputs messages like shown below:

```

usb 1-1: new high-speed USB device number 8 using musb-hdrc
usb-storage 1-1:1.0: USB Mass Storage device detected
scsi2 : usb-storage 1-1:1.0
scsi 2:0:0:0: Direct-Access      SanDisk  Cruzer Blade      1.27 PQ: 0
ANSI:
sd 2:0:0:0: [sda] 15633408 512-byte logical blocks: (8.00 GB/7.45 GiB)
sd 2:0:0:0: [sda] Write Protect is off

```

```
sd 2:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't
support DPO or FUA
sda:
sd 2:0:0:0: [sda] Attached SCSI removable disk
```

At this point you are ready to mount the file system on USB Flash Disk. This is done as follows:

```
# mount -t vfat /dev/sda /mnt
EXT2-fs (sda): warning: mounting unchecked fs, running e2fsck is
recommended
```

You can check the file in USB Flash Disk by accessing the /mnt directory.

## USB Device Example

This example shows how to use ram of ADSP-SC5xx boards as a mass storage device to HOST PC. In this example, ADSP\_SC5xx acts as DEVICE and PC as HOST.

- Supported Port: USB OTG Port and USB HS Port.
- Support USB Mode: OTG and gadget.
- Supported Cable: Micro-B male plug.

### 1) Configure as Device

Please select in order:

```
Device Drivers  --->
  [*] USB support  --->
    <*> Support for Host-side USB
          USB Physical Layer drivers  --->
            <*> NOP USB Transceiver Driver
    <*> USB Gadget Support  --->
```

ADI USB specific configuration depends on selection of the "Support for Host-side USB" and "USB Gadget Support" options. ADI USB specific configuration will not appear until "Support for Host-side USB" and "USB Gadget Support" options are selected at the same time:

```
Device Drivers  --->
  [*] USB support  --->
    <*> Inventra Highspeed Dual Role Controller (TI, ADI,
          MUSB Mode Selection (Gadget only mode)  --->
          *** Platform Glue Layer ***
```



```
<*>          ADI
              MUSB DMA mode (Inventra)  --->
```

## 2) Enable Mass Storage Gadget support

```
Device Drivers  --->
  [*] USB support  --->
    <*>  USB Gadget Support  --->
        <M>      Mass Storage Gadget
```

## 3) Build and Boot to Linux

## 4) Create mount directory on HOST PC

```
$ mkdir -p /mnt/usb
```

## 5) Insmo mass storage gadget driver on DEVICE board

**When plugged into your host pc by USB cable (the plug connects with the board musb by Micro-B type), the kernel outputs messages as shown below:**

```
# dd if=/dev/zero of=fsg.block bs=1M count=16
16+0 records in
16+0 records out
# modprobe g_mass_storage file=/fsg.block stall=0
Number of LUNs=8
Mass Storage Function, version: 2009/09/11
LUN: removable file: (no medium)
Number of LUNs=1
LUN: file: /fsg.block
Number of LUNs=1
g_mass_storage gadget: Mass Storage Gadget, version: 2009/09/11
g_mass_storage gadget: userspace failed to provide iSerialNumber
g_mass_storage gadget: g_mass_storage ready
```

When plugged into your host pc by USB cable (the plug connects with the board musb by Micro-B type), the kernel outputs messages as shown below:

```
# g_mass_storage gadget: high-speed config #1: Linux File-Backed
Storage
```

At the same time, HOST PC will create a new /dev/sdb(not fixed, just a example) device for DEVICE board automaticlly.

## 6) Partition mass storage device which is actually an ADSP-SC5xx board on HOST PC

Please input commands which was shown in below:

```
$ time fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI or
OSF disklabel
Building a new DOS disklabel with disk identifier 0x5926a9e1.
Changes will remain in memory only, until you decide to write them.

After that, of course, the previous content won't be recoverable.
Warning: invalid flag 0x0000 of partition table 4 will be corrected by
w(rite)

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-32767, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-32767, default 32767):
Using default value 32767

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
0.00user 0.09system 0:31.02elapsed 0%CPU (0avgtext+0avgdata
2576maxresident)k
144inputs+8outputs
(0major+111minor)pagefaults 0swaps
```

Then a new device /dev/sdb1 will be created.

## 7) Format mass storage device to ext2 type on HOST PC

```
$ mke2fs /dev/sdb1
mke2fs 1.42.9 (4-Feb-2014)
Filesystem label= OS
type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
```

```
Stride=0 blocks, Stripe width=0 blocks
3840 inodes, 15360 blocks
768 blocks (5.00%) reserved for the super user
First data block=1 Maximum filesystem blocks=15728640
2 block groups
8192 blocks per group, 8192 fragments per group
1920 inodes per group Superblock backups stored on blocks:
    8193
Allocating group tables: done

Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

## 8) Mount formatted /dev/sdb1 to /mnt/usb

```
$ mount -t ext2 -o sync /dev/sdb1 /mnt/usb
```

lost+found folder can be found in /mnt/usb if mount successfully.

Then more operations can be done to this mass storage device which was actually the ram of ADSP-SC5xx board as wanted.

9) Last, don't forget to unmount the mass storage device before unplug the USB cabel.

```
$ umount /mnt/usb
```

## USB Dual-Role Example

Dual-Role(Only apply to USB OTG HOST/DEVICE port, known as On-The-Go) means Linux kernel can support ADSP-SC573, SC584 and SC589 based EZ-KIT boards act as both HOST and DEVICE roles. Role can be switched by cable types, USB Micro-A male plug will turn this port to HOST role and USB Micro-B male plug will turn this port to DEVICE role.

This section will shows how USB OTG HOST/DEVICE port acts in Dual-Role mode. It will includes USB mass storage example and USB mass storage device example. ADSP-SC5xx acts as Dual-Role (OTG) device in this example.

- Supported Port: USB OTG port.
- Supported USB Mode: OTG and Host.
- Supported Cable: Micro-A male plug.



In order to use the USB port in OTG mode, one of gadget drivers must be select.  
The USB port won't work until the gadget driver has been loaded (via insmod).

## 1) Dual-Role configuration

```
Device Drivers --->
  [*] USB support --->
    <*> Support for Host-side USB
          USB Physical Layer drivers --->
            <*> NOP USB Transceiver Driver
    <*> USB Gadget Support --->
```

ADI USB specific configuration depends on selection of the "Support for Host-side USB" and "USB Gadget Support" options. ADI USB specific configuration will not appear until "Support for Host-side USB" and "USB Gadget Support" options are selected at the same time:

```
Device Drivers --->
  [*] USB support --->
    <*> Inventra Highspeed Dual Role Controller (TI, ADI,
          MUSB Mode Selection (Dual Role mode) --->
            *** Platform Glue Layer ***
    <*> ADI
          MUSB DMA mode (Inventra) --->
```

## 2) Mass storage configuration

Refer to configuration in "USB HOST Example".

## 3) Mass storage device configuration

Refer to configuration in "USB DEVICE Example".

## 4) Test Dual-Role

Board can act as a HOST or a DEVICE depends on the type of USB connect are plugged in.

If plug in the USB memory stick to USB OTG HOST/DEVICE port, board can act as HOST, more details please refer to "USB HOST Example" section.

If plug in USB Micro-B male plug which was connect to HOST PC, board can act as DEVICE, more details please refer to "USB DEVICE Example" section.

## USB HS DEVICE Port Usage

A secondary USB HS port is available on the SC589-EZKIT, and this port only supports the device (DEVICE mode) function. This port is only available when the USB OTG HOST/DEVICE port is in Dual-Role or DEVICE mode.

Each port has its own gadget driver, so make sure the two different gadget drivers are loaded after kernel boot up. The first loaded driver is adopted by the USB OTG HOST/DEVICE port, and second loaded driver is adopted by HS DEVICE port.

## 5.11 General Bluetooth Dongle via USB

### 5.11.1 Introduction

Bluetooth is a low-cost, low-power, short-range wireless technology. It was designed as a replacement for cables and other short-range technologies like IrDA. Bluetooth operates in "personal area" range, that typically extends up to 10 meters. More information about Bluetooth can be found at <http://www.bluetooth.com/>.

Linux has support for almost any Bluetooth USB dongle. This document will guide users on how to set up a Bluetooth USB dongle on Linux.

### 5.11.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board
- A Bluetooth USB dongle

### 5.11.3 Software Configuration

#### Configure Linux Kernel

Please enable Bluetooth Support and the HCI USB driver.

```
[*] Networking support --->
  <*> Bluetooth subsystem support --->
  [*] Bluetooth Classic (BR/EDR) features --->
    <*> RFCOMM protocol support --->
      [*] RFCOMM TTY support --->
    <*> BNEP protocol support
      [*] Multicast filter support
      [*] Protocol filter support
```

```
<*> HIDP protocol support
[*] Bluetooth device drivers
<*> HCI USB driver
```

Note that by the default configuration, the USB port works on OTG mode. Users need to probe the USB Bluetooth dongle before it can work.

Otherwise users can select USB host mode for the USB dongle. For detailed information about how to configure USB, please refer to [User manual for usb in linux kernel](#)

```
(Optional)
Device Drivers --->
  HID support --->
  [*] USB support --->
    MUSB Mode Selection (Host only mode)--->
      (X) Host only mode
      ( ) Gadget only mode
      ( ) Dual Role only mode
```

## Configure Buildroot

Users need to select bluez-utilities to set up the Bluetooth devices.

```
Target packages --->
  Networking applications --->
    [*] bluez-utils
```

### 5.11.4 Example

Plug in the USB dongle. If the USB interface is in host mode the device should be detected automatically:

```
usbhid: USB HID core driver
usb 1-1: new full-speed USB device number 2 using musb-hdrc
```

If the USB interface is in OTG mode then the device must be probed first:

```
# modprobe g_serial
```

Bring up the interface:

```
# hciconfig hci0 up
# hciconfig
hci0:   Type: BR/EDR   Bus: USB
        BD Address: 00:18:E4:08:CC:30  ACL MTU: 192:8  SCO MTU: 64:8
        UP RUNNING PSCAN
        RX bytes:383 acl:0 sco:0 events:16 errors:0
        TX bytes:300 acl:0 sco:0 commands:15 errors:0
```

Users can use "hcidtool" to set the configuration of Bluetooth connections.

```
# hcidtool

hcidtool - HCI Tool ver 4.101
Usage:
    hcidtool [options] <command> [command parameters]

Options:
    --help    Display help
    -i dev    HCI device

Commands:
    dev       Display local devices
    inq       Inquire remote devices
    scan      Scan for remote devices
    name      Get name from remote device
    info      Get information from remote device
    spinq     Start periodic inquiry
    epinq     Exit periodic inquiry
    cmd       Submit arbitrary HCI commands
    con       Display active connections
    cc        Create connection to remote device
    dc        Disconnect from remote device
    sr        Switch master/slave role
    cpt       Change connection packet type
    rssi      Display connection RSSI
    lq        Display link quality
    tpl       Display transmit power level
    afh       Display AFH channel map
    lp        Set/display link policy settings
    lst       Set/display link supervision timeout
    auth      Request authentication
    enc       Set connection encryption
    key       Change connection link key
    clkoff    Read clock offset
    clock     Read local or remote clock
```

```
lescan Start LE scan
lewlladd Add device to LE White List
lewlrml Remove device from LE White List
lewllsz Read size of LE White List
lewllclr Clear LE White list
lecc Create a LE Connection
ledc Disconnect a LE Connection
lecup LE Connection Update
For more information on the usage of each command use:
hcitool <command> --help
```

For more information on the usage of each command use:

```
# hcitool <command> --help
```

Take some commands as example, such as show HCI devices

```
# hcitool -i hci0 dev
Devices:
hci0 00:18:E4:08:CC:30
```

Scan and inquire for Bluetooth devices in the area:

```
# hcitool -i hci0 scan
Scanning ...
30:F9:ED:E1:9E:A9 DR-BT140Q
34:80:B3:4D:5F:5A mi4
# hcitool -i hci0 inq
Inquiring ...
34:80:B3:4D:5F:5A clock offset: 0x2555 class: 0x5a020c
30:F9:ED:E1:9E:A9 clock offset: 0x57f5 class: 0x240408
```

Get some information from these devices:

```
# hcitool -i hci0 info 30:F9:ED:E1:9E:A9
Requesting information ...
BD Address: 30:F9:ED:E1:9E:A9
Device Name: DR-BT140Q
LMP Version: 2.0 (0x3) LMP Subversion: 0x1225
Manufacturer: Cambridge Silicon Radio (10)
Features: 0xff 0xff 0x8f 0x7e 0x98 0x19 0x00 0x80
          <3-slot packets> <5-slot packets> <encryption> <slot
offset>
```



```

mode> <timing accuracy> <role switch> <hold mode> <sniff
packets> <park state> <RSSI> <channel quality> <SCO link> <HV2
scheme> <HV3 packets> <u-law log> <A-law log> <CVSD> <paging
RSSI> <power control> <transparent SCO> <broadcast encrypt>
<EDR ACL 2 Mbps> <EDR ACL 3 Mbps> <enhanced iscan>
<interlaced iscan> <interlaced pscan> <inquiry with
<AFH cap. slave> <AFH class. slave> <3-slot EDR ACL>
<5-slot EDR ACL> <AFH cap. master> <AFH class. master>
<extended features>

```

Ping a device:

```

# l2ping -i hci0 -f 30:F9:ED:E1:9E:A9
Ping: 30:F9:ED:E1:9E:A9 from 00:18:E4:08:CC:30 (data size 44) ...
4 bytes from 30:F9:ED:E1:9E:A9 id 0 time 30.38ms
4 bytes from 30:F9:ED:E1:9E:A9 id 1 time 8.70ms
4 bytes from 30:F9:ED:E1:9E:A9 id 2 time 8.48ms
4 bytes from 30:F9:ED:E1:9E:A9 id 3 time 8.90ms
4 bytes from 30:F9:ED:E1:9E:A9 id 4 time 8.51ms

```

## 5.12 Kernel API for DMA operation

### 5.12.1 Introduction

The Direct Memory Access (DMA) controller in the ADSP-SC5xx processor allows automated data transfers with minimal overhead for the core. DMA transfers can occur between any of the DMA capable peripherals (such as the SPORT or PPI) and the memory in L2 SRAM or external DDR.

### 5.12.2 Linux DMA Framework

There are two aspects of the Linux DMA framework.

- The generic Linux DMA mapping API
- The DMA API for the SC5xx onchip DMA controller

#### Linux DMA Mapping API

- Document: **linux-kernel/Documentation/DMA-API-HOWTO.txt, Linux Device Driver (3rd) - chapter 15.**

- API definition: **linux-kernel/include/linux/dma-mapping.h**, **linux-kernel/arch/arm/include/asm/dma-mapping.h**

DMA operations allocate a buffer and pass bus addresses to your device. A DMA mapping is a combination of allocating a DMA buffer and generating an address for that buffer that is accessible by the device.

DMA mappings must also address the issue of cache coherency. Modern processors keep copies of recently accessed memory areas in a fast, local cache. Without this cache, reasonable performance is not possible. If your device changes an area of main memory it is imperative that any processor caches covering that area be invalidated. Otherwise the processor may work with an incorrect image of main memory and data corruption may result. Similarly, when your device uses DMA to read data from main memory any changes to that memory residing in processor caches must be flushed out first.

On SC5xx, DMA mapping is done in the same way as other ARM processors. **dma\_alloc\_coherent()** can be called to allocate a DMA buffer in the drivers. A block of 256k bytes DDR pool is reserved for DMA atomic, coherent usage while the normal coherent DMA memory is reserved from DDR without a size limit. New VM areas and page table entries of the allocated page structures are created with the uncacheable page attribute before the area address pointer is returned.

If you are writing a portable device driver, make sure to use the generic DMA APIs (for a full list please refer to the documentation):

```
void *dma_alloc_coherent(struct device *dev, size_t size, dma_addr_t
*dma_handle, gfp_t gfp);

void dma_free_coherent(struct device *dev, size_t size, void *vaddr,
dma_addr_t dma_handle);

dma_addr_t dma_map_single(struct device *dev, void *ptr, size_t size, e
num dma_data_direction dir)

dma_addr_t dma_map_page(struct device *dev, struct page *page,
unsigned long offset, size_t size, enum dma_data_direction dir)

int dma_map_sg(struct device *dev, struct scatterlist *sg, int nents, e
num dma_data_direction dir);
```

## What is a bus address

When the CPU (say with the MMU turned off) wants to access physical memory it puts that address on its output pins. This a **physical Address**.

When a peripheral device wants to access the same physical memory (as in a DMA function) it may have to use a different address to get to the same physical location. This is a **bus address**.

So a **bus address** is the address used by a peripheral to access a certain **physical address**.

## Generic DMA mapping guide

Please refer to the Linux kernel document [DMA API HOWTO](#) for details.

### DMA APIs for SC5xx

The SC5xx processor offers a wide array of DMA capabilities.

- 44 Different DMA channels
- Memory to Memory and IO to Memory Channel transfers
- Dual X and Y indexing Address counters
- Register base configuration
- Flexible Descriptor Based Configuration
- Memory interface supporting 8, 16, 32, 64, 128 and 256 bit data transfers
- Peripheral interface supports 8, 16 and 32 bit data transfers
- Interrupt on each DMA packet completion
- Flexible DMA Priority

### Flow Types and Descriptor

There are 6 different ways the DMA controller can be set up. These are called Flow types

- FLOW\_STOP - Stop after the current job
- FLOW\_AUTO - Autobuffer, Repeat the current transfer until stopped
- FLOW\_LIST - Use a linked list of descriptors
- FLOW\_ARRAY - Use a sequential list of descriptors
- FLOW\_LIST\_DEMAND - Use a linked list of descriptors and fetch the next only after the DMA channel detects an incoming trigger event
- FLOW\_ARRAY\_DEMAND - Use a sequential list of descriptors and fetch next only after the DMA channel detects an incoming trigger event

The flow type can be defined in a CONFIG word in a descriptor so the modes can be mixed and the operation quite complex.

Descriptors are used to control the DMA channel and allow a complex stream of data packets to be assembled if required.

- Array Descriptor - Simple Sequential array of descriptors in memory
- List Descriptor - Descriptors chained via address word in memory

For descriptor list mode, at a minimum the DMA\_DSCPTR\_NXT register must be written prior to write to the DMA\_CFG register, which is the special action required to start the DMA channel. For descriptor array mode, at a minimum the DMA\_DSCPTR\_CUR register must be written prior to writing to the DMA\_CFG register, which is the special action required to start the DMA channel.

One other slight complexity in the descriptor is the fact the DMA controller does not have to read ALL of the words in the descriptor array. The NDSIZE part of the CONFIG Register contains the number of elements to read into the DMA controller for this operation.

## Descriptor Memory Layout

List Descriptor:

```
struct dmasg {
    void *next_desc_addr;
    unsigned long start_addr;
    unsigned long cfg;
    unsigned long x_count;
    long x_modify;
    unsigned long y_count;
    long y_modify;
} __packed;
```

Array Descriptor:

```
struct dma_desc_array {
    unsigned long start_addr;
    unsigned long cfg;
    unsigned long x_count;
    long x_modify;
} __packed;
```

## 2-D DMA

2-D DMA can be roughly viewed as:

```
/* Correct me if the boundary check is wrong */
for ( ; Y_COUNT > 1; Y_COUNT--)
{
    for ( ; X_COUNT > 1; X_COUNT--)
        DMAx_CURR_ADDR += X_MODIFY;
    DMAx_CURR_ADDR += Y_MODIFY;
}
```

---

In some video application, 2-D DMA is more convenient to use than 1-D DMA.

## MDMA Copy Wrapper for Linux Drivers

As an alternative to setting up MDMA by yourself there exist APIs to use MDMA. See API implementation in **arch/arm/mach-sc58x/dma.c** or **arch/arm/mach-sc57x/dma.c**.

```
void *dma_memcpy(void *dest, const void *src, size_t count)
void *dma_memcpy_nocache(void *dest, const void *src, size_t count)
void *safe_dma_memcpy(void *dest, const void *src, size_t count)
void early_dma_memcpy(void *dest, const void *src, size_t count)
void early_dma_memcpy_done(void);
```

## DMA Operation for Linux Drivers

Please refer to: **arch/arm/mach-sc58x/include/mach/dma.h**, and **arch/arm/mach-sc58x/dma.c** or **arch/arm/mach-sc57x/include/mach/dma.h**, and **arch/arm/mach-sc57x/dma.c**.

The DMA channel management API:

```
int request_dma(unsigned int channel, const char *device_id)
void free_dma(unsigned int channel)
void enable_dma(int channel)
void disable_dma(int channel)
```

The extended DMA manipulation API allows for increased flexibility in SC5xx:

```
unsigned long gen_dma_config2(char direction, char flow_mode, char
intr_mode, char dma_mode, char mem_width, char syncmode, char
peri_width)
unsigned long gen_dma_config(char direction, char flow_mode, char
intr_mode, char dma_mode, char mem_width, char syncmode)
void set_dma_start_addr(unsigned int channel, unsigned long addr)
void set_dma_next_desc_addr(unsigned int channel, unsigned long addr)
void set_dma_x_count(unsigned int channel, unsigned short x_count)
void set_dma_x_modify(unsigned int channel, short x_modify)
void set_dma_y_count(unsigned int channel, unsigned short y_count)
void set_dma_y_modify(unsigned int channel, short y_modify)
void set_dma_config(unsigned int channel, unsigned short config)
unsigned short set_bfin_dma_config(char direction, char flow_mode, char
intr_mode, char dma_mode, char width)
unsigned short get_dma_curr_irqstat(unsigned int channel)
unsigned short get_dma_curr_xcount(unsigned int channel)
unsigned short get_dma_curr_ycount(unsigned int channel)
```

```

void set_dma_sg(unsigned int channel, struct dmasg_t *sg, int nr_sg)
void dma_disable_irq(unsigned int channel)
void dma_disable_irq_nosync(unsigned int channel)
void dma_enable_irq(unsigned int channel)
void clear_dma_irqstat(unsigned int channel)
int set_dma_callback(unsigned int channel, dma_interrupt_t callback, void
id *data)

```

## DMA Example

This is a simple DMA example taken from the `adsp-spidad.c` driver. This is getting 8-bit data from the SPI device into `mybuffer`.

```

#define SPI0_RX_DMA_CH 23
#define BUF_SIZE 1024 * 32
static unsigned char mybuffer[BUF_SIZE];

int mydmatest(struct device *dev)
{
    int ret;
    dma_addr_t dma_addr;

    // Ask for the DMA channel
    ret = request_dma(SPI0_RX_DMA_CH, "SPI RX Test");
    if ( ret < 0 ) {
        printk(" Unable to get SPI0 RX DMA channel\n");
        return 1;
    }

    // Turn off the DMA channel
    disable_dma(SPI0_RX_DMA_CH);

    // Set the IRQ callback
    set_dma_callback(SPI0_RX_DMA_CH, myirq, mydata);

    // Map the memory for DMA device access
    dma_addr = dma_map_single(dev, mybuffer, BUF_SIZE,
DMA_FROM_DEVICE);
    if (dma_mapping_error(dev, dma_handle)) {
        free_dma(SPI0_RX_DMA_CH);
        printk(" Unable to map DMA region\n");
        return 1;
    }

    // Set up the dma config
    // WNR We are going to write to memory
    // RESTART throw away any old data in the fifo
    // Enable Interrupts

```

```

set_dma_config(SPI0_RX_DMA_CH, ( WNR | RESTART | DI_EN ));

// Set address to drop data into
set_dma_start_address(SPI0_RX_DMA_CH, dma_addr);

// Set the transfer size in bytes
set_dma_x_count(SPI0_RX_DMA_CH, size);

// Set the X modify ( dont worry about Y for this one )
set_dma_x_modify(SPI0_RX_DMA_CH, 1);

// Off we go
enable_dma(SPI0_RX_DMA_CH);
}

```

The IRQ routine could look like this. It simply clears the IRQ status.

```

static irqreturn_t myirq( int irq, void *data)
{
    unsigned short mystat;
    struct device *dev = (struct device*)data;

    mystat = get_dma_curr_irqstat(SPI0_RX_DMA_CH);
    clear_dma_irqstat(SPI0_RX_DMA_CH);

    // Unmap the DMA memory for processor access
    dma_unmap_single(dev, mybuffer, BUF_SIZE, DMA_TO_DEVICE);
    free_dma(SPI0_RX_DMA_CH);

    wake_up_interruptible(&mywaiting_task);

    return IRQ_HANDLED;
}

```

## 5.13 Linux MTD Driver

### 5.13.1 Introduction

This section describes the steps required to build and use MTD(Memory Technology Device) subsystem on Linux using an ADSP-SC5xx board.

The MTD software stack looks like below:

```
MTD
```

```
-----  
SPI NOR framework
```

```
-----  
m25p80
```

```
-----  
SPI bus driver
```

```
-----  
SPI NOR chip
```

### 5.13.2 Hardware Required

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board

### 5.13.3 Software Configuration

The following configuration should be done on top of the SC589-ezkit/SC584-ezkit/SC573-ezkit default configuration.

#### Configure Buildroot

Enable MTD tools.

```
Target packages --->  
  Filesystem and flash utilities --->  
    [*] mtd, jffs2 and ubi/ubifs tools  
        [*] flashcp  
        [*] flash_erase  
        [*] mtd_debug  
        [*] mtdinfo  
        [*] mkfs.jffs2
```

#### Configure Linux Kernel

**Enable MTD and SPI NOR flash w25x driver.**

```
Device Drivers --->  
  <*> Memory Technology Device (MTD) support --->
```



```

<*> Command line partition table parsing
<*> Caching block device access to MTD devices
<*> SPI-NOR device support --->
Self-contained MTD device drivers --->
    <*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)

```

## Enable JFFS2 filesystem support.

```

File systems --->
    [*] Miscellaneous filesystems --->
        <*> Journalling Flash File System v2 (JFFS2) support

```

## Configure Device Tree

Add a child device node under spi master node for spi flash. As our current release, SC5xx boards have added spi flash as default, you could easily have a check.

```

spi_2: spi@0x31044000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "adi,spi3";
    reg = <0x31044000 0xFF>;
    .....
    flash: w25q32@0 {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "winbond,w25q32";
        spi-max-frequency = <5000000>;
        reg = <38>;
        spi-cpol;
        spi-cpha;

        partition@0 {
            label = "uboot (spi)";
            reg = <0x0 0x80000>;
        };
        partition@1 {
            label = "kernel (spi)";
            reg = <0x80000 0x580000>;
        };
        partition@2 {
            label = "root file system (spi)";

```

```
};
};
};
reg = <0x600000 0xa00000>;
```

### 5.13.4 Example

#### Get the MTD device info.

```
# cat /proc/mtd
dev:   size  erasesize  name
mtd0: 00080000 00001000 "uboot (spi)"
mtd1: 00580000 00001000 "kernel (spi)"
mtd2: 00a00000 00001000 "root file system (spi)"
# mtdinfo
Count of MTD devices:      3
Present MTD devices:      mtd0, mtd1, mtd2
Sysfs interface supported: yes
# mtdinfo /dev/mtd0
mtd0
Name:                      uboot (spi)
Type:                      nor
Eraseblock size:          4096 bytes, 4.0 KiB
Amount of eraseblocks:    128 (524288 bytes, 512.0 KiB)
Minimum input/output unit size: 1 byte
Sub-page size:            1 byte
Character device major/minor: 90:0
Bad blocks are allowed:   false
Device is writable:      true
```

#### Erase the MTD device.

```
# flash_erase -j -q /dev/mtd1 0 0
```

#### Mount the MTD device.

```
# mount -t jffs2 /dev/mtdblock1 /mnt
```

## Read and Write data to the MTD device.

```
# echo hello > /mnt/test
# cat /mnt/test
hello
```

## Unmount the MTD device.

```
# umount /mnt
```

## 5.14 Linux PCIE device driver

### 5.14.1 Introduction

This section describes the steps required to build and use PCIE device driver on Linux using two ADSP-SC589 boards. ADSP-SC584/ADSP-SC573 does not support PCIE in hardware.

### 5.14.2 Hardware Setup

- One ADSP-SC589 EZ-KIT v1.0 board configured as Root Complex
- One ADSP-SC589 EZ-KIT v1.1 board configured as Endpoint
- One SAMTEC PCIE cable

Connect J5 PCI Express 1x Connector with SAMTEC PCIE cable on two SC589-EZKIT boards.

### 5.14.3 Software Configuration

The following configuration should be done on top of the SC589-ezkit default configuration.

#### Configure Linux Kernel

#### Enable PCIE Bus driver

For both boards:

```
Bus support --->
[*] PCI support
[*] Message Signaled Interrupts (MSI and MSI-X)
PCI host controller drivers --->
  [*] ADI SC58X PCIe controller
```

## Enable PCIe Root Complex Controller driver

For the Root Complex (RC) board:

```
Device Drivers --->
  Character devices --->
    <*> SC58X PCIe Device Driver
```

## Enable PCIe Endpoint driver

For the Endpoint (EP) board:

```
Device Drivers --->
  Character devices --->
    <*> SC58X PCIe Endpoint Driver
```

## Configure u-boot

The PCIe EP/RC soft switch PIN should be pulled up for the SC589 EZ-Kit v1.1 board in the PCIe EP mode. Please edit the software configuration array in u-boot file board/adi/sc589-ezkit/soft\_switch.h, and change U48 PORTB array value1 from 0 to 0x80.

```
{
    .dir0 = 0x0,
    .dir1 = 0x0,
    .value0 = 0x0,
    .value1 = 0x80,    /* 0x0 -> 0x80 */
},
};
```

Don't forget to flash the new built u-boot.ldr to the SPI flash. No change is needed for the SC589 EZ-Kit v1.0 board in the PCIE RC mode.

## 5.14.4 Example

### Bring up Endpoint board

You will see below kernel log. "start link training..." means Endpoint is waiting for Root Complex to start link initialization and training.

```
console [ttySC0] enabled
console [ttySC0] enabled
bootconsole [earlycon0] disabled
bootconsole [earlycon0] disabled
adi-uart4.2: ttySC2 at MMIO 0x31003800 (irq = 91, base_baud = 7031250) is
start link training...
random: nonblocking pool is initialized
```

### Bring up Root Complex board

You will see below kernel log on Root Complex board.

```
sc58x-pcie 310b8000.pcie: 2.5 GT/s : Width x1
sc58x-pcie 310b8000.pcie: PCI host bridge to bus 0000:00
pci_bus 0000:00: root bus resource [mem 0x50000000-0x5effffff]
pci_bus 0000:00: root bus resource [mem 0x60000000-0x7fffffff pref]
pci_bus 0000:00: root bus resource [bus 00-ff]
pci_bus 0000:00: root bus resource [io 0x1000-0xffff]
PCI: bus0: Fast back to back transfers disabled
pci 0000:00:00.0: BAR 0: assigned [mem 0x60000000-0x6fffffff pref]
...
bar[0]=60000000@10000000
bar[1]=0@0
bar[2]=0@0
bar[3]=0@0
bar[4]=0@0
bar[5]=0@0
...
receive msi 0
receive msi 0
receive msi 0
```

```
inbound write finish 1234abcd
receive msi 0
inbound read finish
```

At the same time, you will see below kernel log on Endpoint board.

```
PCIE link is up
waiting rc enable device...
BAR0 = 60000008
send msi 0
out of reset: 8c988000
send msi 0
inbound write 4 bytes to 8c988010
send msi 0
inbound read 1234abcd[55aaaa55]
send msi 0
```

## 5.15 Linux Video Driver

### 5.15.1 Introduction

This section describes the steps required to build and use video driver to capture or display video images on Linux using an ADSP-SC5xx board, a Video Decoder EI3 Extender Board and a Video Encoder EI3 Extender Board.

The Linux Kernel Media Subsystems provides support for devices like webcams, streaming capture and output, analog TV, digital TV, AM/FM radio, Software Digital Radio (SDR) and remote controllers.

The Linux Media Infrastructure API converges the kernel to userspace APIs used on media drivers. It has 4 parts:

- Part I: The V4L2 API
- Part II: The Linux DVB API
- Part III: The Remote Controller API
- Part IV: The Media Controller API

For more information about the Linux Kernel Media Subsystems, please refer to <http://linuxtv.org/>.

## 5.15.2 Hardware Required

- An ADSP-SC5XX EZ-Board (ADSP-SC589 EZ-Board / ADSP-SC573 EZ-Board)
- A Video Decoder EI3 Extender Board
- A Video Encoder EI3 Extender Board
- HDMI Cable and 3RCA Cable

### EPPI on ADSP-SC5XX EZ-Board

The Enhanced Parallel Peripheral Interface (EPPI) is a half-duplex, bidirectional port with a dedicated clock pin and three frame sync (FS) pins directly output from the processor. It can support direct connections to active TFT LCDs, parallel A/D and D/A converters, video encoders and decoders, image sensor modules and other general-purpose peripherals. We can find EPPI port from **P1A** connector on the back of ADSP-SC5XX EZ-Board.



### Video Decoder EI3 Extender Board

The Video Decoder EI3 Extender Board is a separately daughter board that plugs onto the EI3 of an EZ-KIT LITE/EZ-Board, it extends the capabilities of the EZ-KIT LITE/EZ-Board by providing a connection between the enhanced parallel peripheral interface (EPPI) of the processor and the ADV7842 video decoder. For more information about the ADV7842 or Video Decoder EI3 Extender Board, go to [www.analog.com](http://www.analog.com) and search for ADV7842 or Video Decoder EI3 Extender Board.

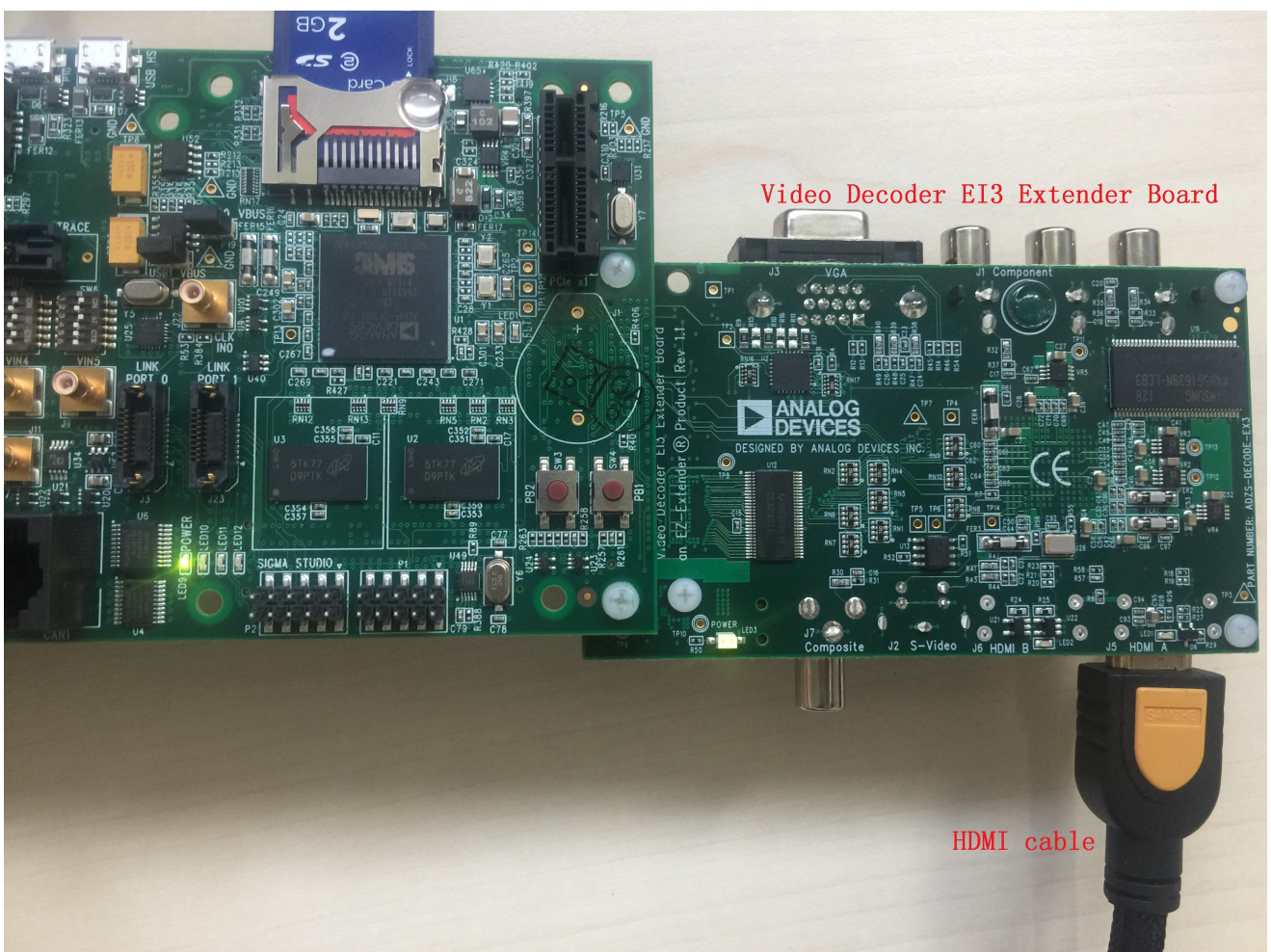
Connect the Video Decoder EI3 Extender Board board to the **P1A** connector on the ADSP-SC5XX EZ-Board.

The following formats are supported:

- 720p60 (HD)
- NTSC (480i60)
- PAL (576i50)

When using 720p60, connect an HDMI cable from your source (e.g. a video player) to the HDMI A port and ensure the video source's resolution is set to 720p.

For SD video, connect video cables to CVBS, S-Video or Component port.



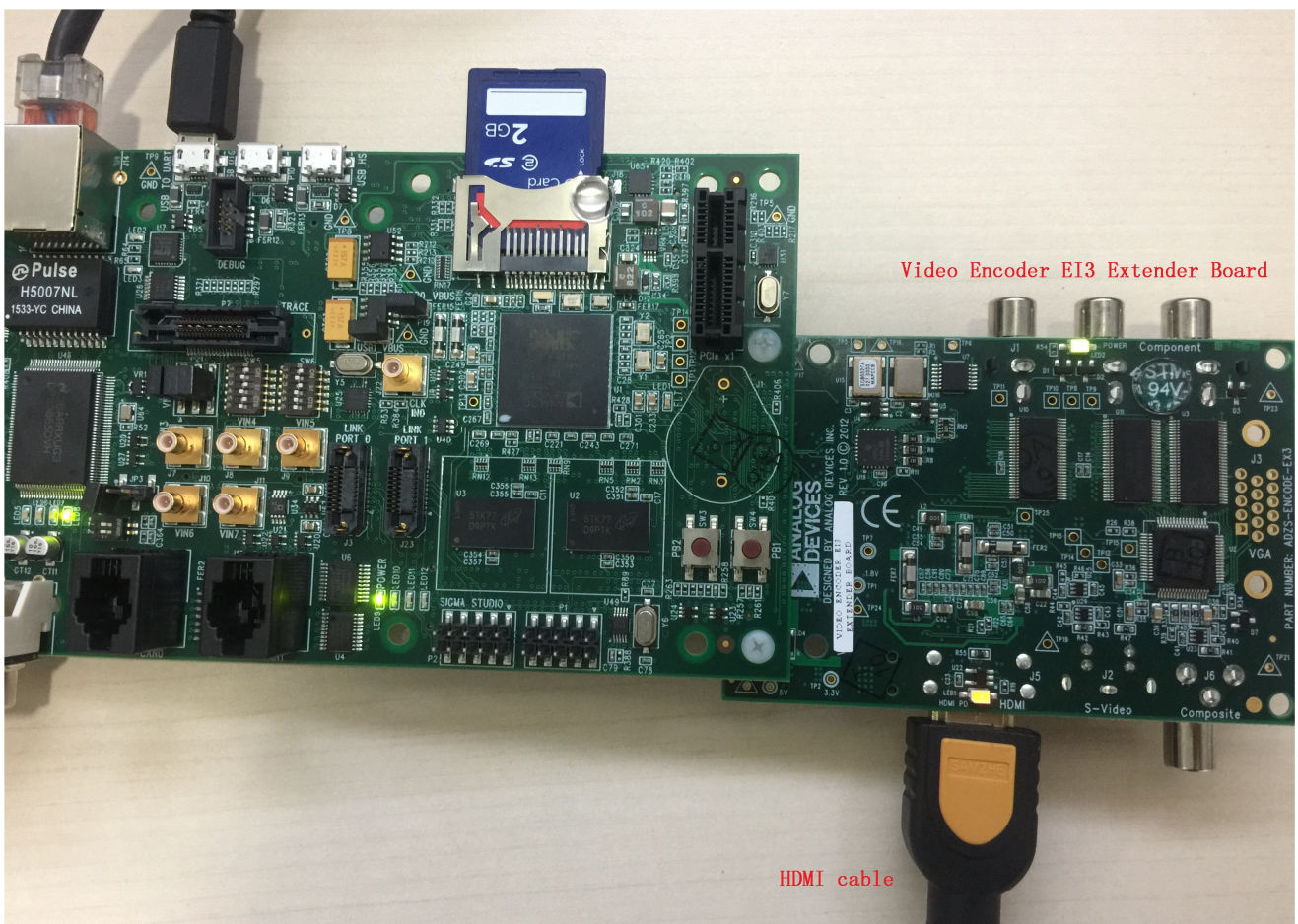


## Video Encoder EI3 Extender Board

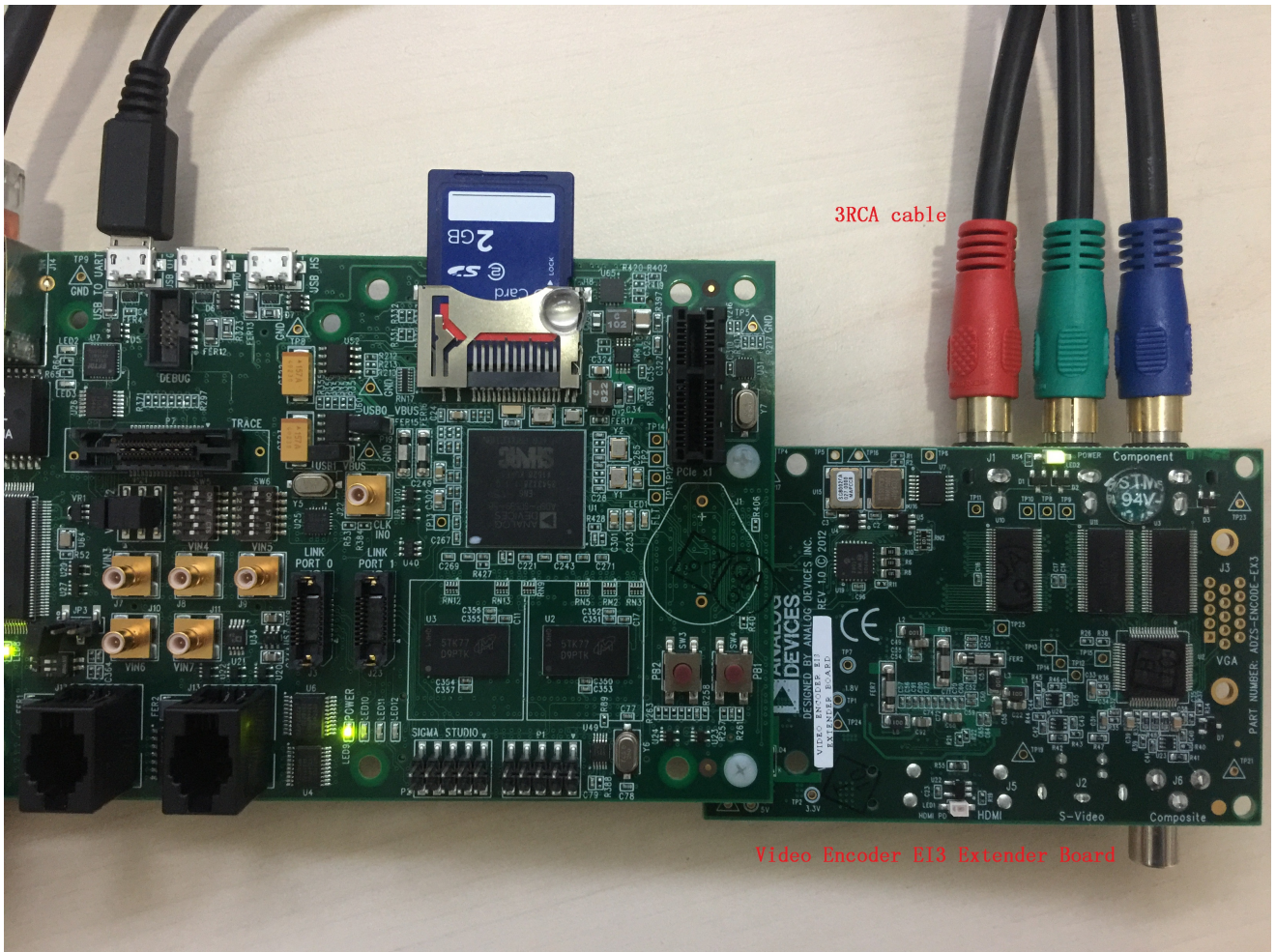
The Video Encoder EI3 Extender Board is a separately daughter board that plugs onto the EI3 of an EZ-KIT LITE/EZ-Board, it extends the capabilities of the EZ-KIT LITE/EZ-Board by providing a connection between the enhanced parallel peripheral interface (EPPI) of the processor and the ADV7511 and ADV7341 video encoder. For more information about the ADV7511, ADV7341 or Video Encoder EI3 Extender Board, go to [www.analog.com](http://www.analog.com) and search for ADV7511 or ADV7341 or Video Encoder EI3 Extender Board.

Connect the Video Encoder EI3 Extender Board board to the **P1A** connector on the ADSP-SC5XX EZ-Board.

Only HD format is supported for ADV7511, so connect an HDMI cable to a TV to display captured HD yuv file (720p).



ADV7341 supports NTSC and PAL format video. Connect the Composite port on the adapter board to TV video port using 3RCA cable.



### 5.15.3 Software Configuration

The following configuration should be done on top of the sc589-ezkit/sc573-ezkit default configuration.

#### Buildroot

Compile V4L2 video test program into Linux image

```
Package Selection for the target --->
  Miscellaneous --->
    [*] V4L2 video test program
```

## Kernel

### Enable I2C support

```
Device Drivers --->
  I2C support --->
    I2C Hardware Bus support --->
      <*> Blackfin TWI I2C support
      (50) Blackfin TWI I2C clock (kHz)
```

### Enable Microchip MCP23xxx I/O expander support

```
Device Drivers --->
  *- GPIO support --->
    <*> Microchip MCP23xxx I/O expander
```

### Enable V4L2 capture platform driver and ADV7842 video decoder driver

```
Device Drivers --->
  <*> Multimedia support --->
    [*] Cameras/video grabbers support
    [*] Media Controller API (EXPERIMENTAL)
    [*] V4L platform devices --->
      <*> Blackfin Video Capture Driver
    Encoders, decoders, sensors and other helper chips --->
      <*> Analog Devices ADV7842 decoder
```

### Enable V4L2 display platform driver and ADV7343 video encoder drivers

As there is only one PPI on the ADSP-SC5xx board, please don't select V4L2 capture and display platform driver at the same time. You can't select ADV7511 and ADV7343 either. If you want to display HD video, please select ADV7511 HDMI transmitter driver. For SD video you should choose ADV7343 video encoder driver, it should be noted that driver for ADV7343 works for ADV7341 as well.

```

Device Drivers --->
  <*> Multimedia support --->
    [*] Cameras/video grabbers support
    [*] Media Controller API (EXPERIMENTAL)
    [*] V4L platform devices --->
      <*> Blackfin Video Display Driver
    Encoders, decoders, sensors and other helper chips --->
      <*> Analog Devices ADV7511 HDMI transmitter
      <*> ADV7343 video encoder

```

As the PPI hardware pin conflicts with SPI on the ADSP-SC573 EZ-Board, you should disable SPI before using ppi, otherwise you will get a pin request error message from pinctrl. So extra configuration only for ADSP-SC573 EZ-Board:

```

Device Drivers --->
  [*] SPI support --->
    <> SPI controller v3 for ADI

```

## Device Tree

### Device node for soft switch on Video Decoder EI3 Extender Board

Please add following child node ssw2 to i2c0 master node in the device tree(sc589-ezkit.dts/sc573-ezkit.dts). We need to setup soft switch before we start capture streaming.

```

i2c0: twi@31001400 {
...
ssw1: gpio@0x22 {
    compatible = "microchip,mcp23017";
    gpio-controller;
    #gpio-cells = <2>;
    reg = <0x22>;
};
+ssw2: gpio@0x26 {
+   compatible = "microchip,mcp23017";
+   gpio-controller;
+   #gpio-cells = <2>;
+   reg = <0x26>;
+};
adau1979@0x11 {

```

```

    compatible = "adi,adaul977";
    reg = <0x11>;
};

```

## Device node for adi video capture driver

Please add following node video\_decoder in the device tree to support adi video capture driver.

For ADSP-SC573 EZ-Board(sc573-ezkit.dts):

```

+video_decoder: cap {
+   compatible = "adi,cap";
+   card-name = "SC57X";
+   type = <2>;
+   dma-channel = <28>;
+   interrupts = <0 83 0>;
+   reg = <0x3102D000 0xfff>;
+   spu_securep_id = <68>;
+   i2c_bus_id = <0>;
+   pinctrl-names = "8bit", "16bit";
+   pinctrl-0 = <&ppi0_8b>;
+   pinctrl-1 = <&ppi0_16b>;
+};

```

For ADSP-SC589 EZ-Board(sc589-ezkit.dts):

```

+video_decoder: cap {
+   compatible = "adi,cap";
+   card-name = "SC58X";
+   type = <2>;
+   dma-channel = <28>;
+   interrupts = <0 83 0>;
+   reg = <0x31040000 0xfff>;
+   spu_securep_id = <95>;
+   i2c_bus_id = <0>;
+   pinctrl-names = "8bit", "16bit", "24bit";
+   pinctrl-0 = <&ppi0_8b>;
+   pinctrl-1 = <&ppi0_16b>;
+   pinctrl-2 = <&ppi0_24b>;
+};

```

## Device node for soft switch on Video Encoder EI3 Extender Board

Please add following child node ssw2 to i2c0 master node in the device tree(sc589-ezkit.dts/sc573-ezkit.dts). We need to setup soft switch before we start display streaming.

```
i2c0: twi@31001400 {
    ...
    ssw1: gpio@0x22 {
        compatible = "microchip,mcp23017";
        gpio-controller;
        #gpio-cells = <2>;
        reg = <0x22>;
    };
    +ssw2: gpio@0x25 {
    +     compatible = "microchip,mcp23017";
    +     gpio-controller;
    +     #gpio-cells = <2>;
    +     reg = <0x25>;
    +};
    adau1979@0x11 {
        compatible = "adi,adau1977";
        reg = <0x11>;
    };
};
```

## Device node for adi video display driver

Please add following node video\_encoder in the device tree to support adi video display driver.

For ADSP-SC573 EZ-Board(sc573-ezkit.dts):

```
+video_encoder: disp {
+     compatible = "adi,disp";
+     card-name = "SC57X";
+     type = <2>;
+     dma-channel = <28>;
+     interrupts = <0 83 0>;
+     reg = <0x3102D000 0xfff>;
+     spu_securep_id = <68>;
+     i2c_bus_id = <0>;
+     pinctrl-names = "8bit", "16bit";
+     pinctrl-0 = <&ppi0_8b>;
+     pinctrl-1 = <&ppi0_16b>;
+};
```

For ADSP-SC589 EZ-Board(sc589-ezkit.dts):

```
+ video_encoder: disp {
+   compatible = "adi,disp";
+   card-name = "SC58X";
+   type = <2>;
+   dma-channel = <28>;
+   interrupts = <0 83 0>;
+   reg = <0x31040000 0xfff>;
+   spu_securep_id = <95>;
+   i2c_bus_id = <0>;
+   pinctrl-names = "8bit", "16bit", "24bit";
+   pinctrl-0 = <&ppi0_8b>;
+   pinctrl-1 = <&ppi0_16b>;
+   pinctrl-2 = <&ppi0_24b>;
+};
```

#### 5.15.4 Example

The state of pins from IO expander is "uncertain" after we enable Soft Switch on the ADSP-SC573 EZ-Board, some pins are also the OE pins (active low) of modules, as the PPI signal pins are reused by many modules, you should disable related modules by setting Soft Switch first, otherwise the signals from ppi will be interfered. So extra configuration only for ADSP-SC573 EZ-Board:

```
# echo 482 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio482/direction
# echo 485 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio485/direction
# echo 505 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio505/direction
```

## Video Decoder EI3 Extender Board

### Setup Soft Switch

```
# echo 464 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio464/direction
# echo 466 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio466/direction
```

### Capture Video Stream

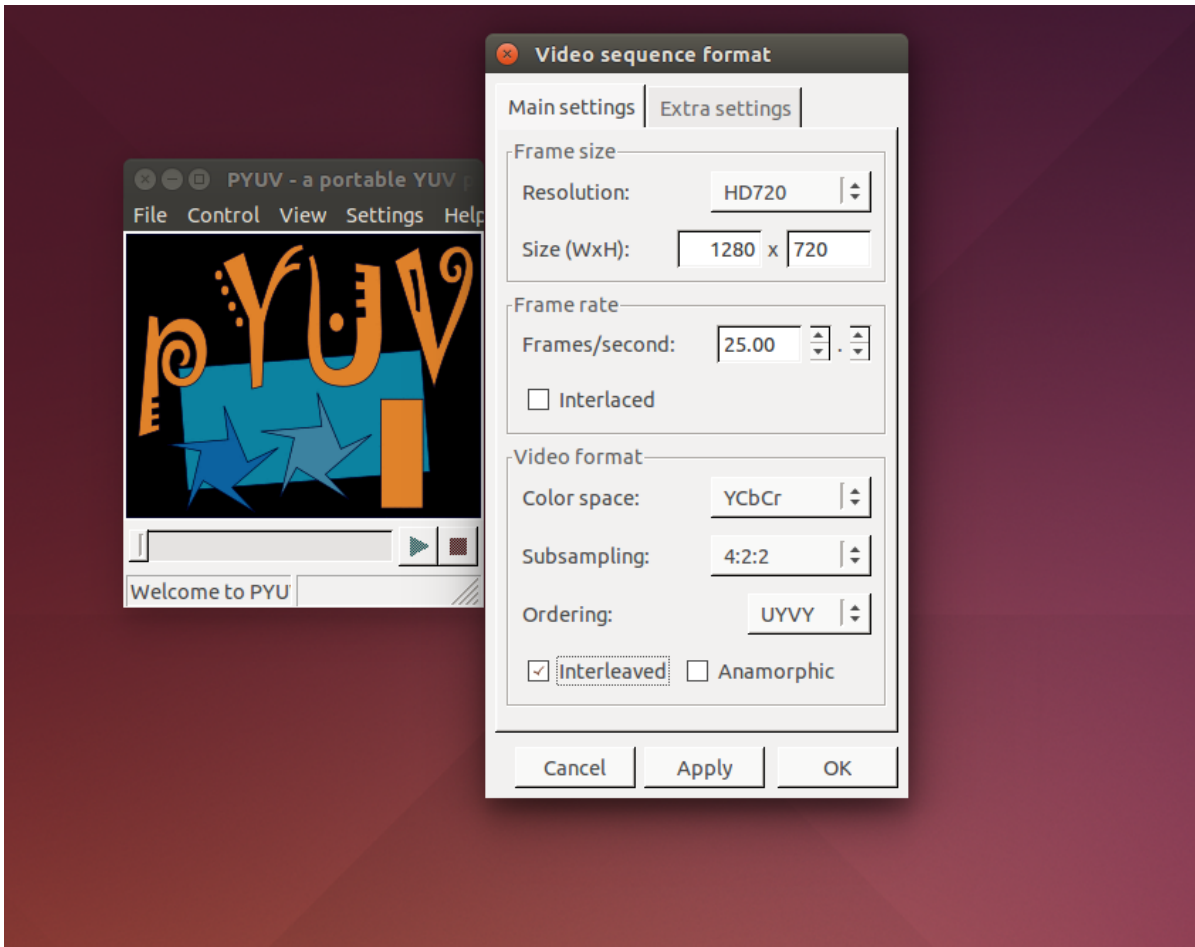
You can use "v4l2\_video\_capture" provided by Buildroot to test video capture function for ADV7842, it shows the decoding of HD signal input and YCbCr pixel output in this example:

```
# v4l2_video_capture -I 4 -F adv7842.yuv
```

The 720p yuv file can be played by a raw video sequence player, for example pYUV. Or it can be saved as the input of the ADV7511. PYUV must be configured as below:

```
Resolution -> HD720
Color space -> YCbCr
Subsampling -> 4:2:2
Ordering -> UYVY
And select Interleaved option.
```





## Video Encoder EI3 Extender Board

### ADV7511 HDMI transmitter

One thing to note here is that the ADSP-SC573 can support up to 56.25MHz ppi clock when transmitting data or frame sync, however, ADSP-SC589 can support up to 75MHz ppi clock, and there are two different ppi clock generators on Video Encoder EI3 Extender Board: 27MHz and 74MHz, so we can enable 27MHz clock both for ADSP-SC573 EZ-Board and ADSP-SC589 EZ-Board, and 74MHz clock only for ADSP-SC589 EZ-Board. For more information, please refer to the data sheets of ADSP-SC573/ADSP-SC589 and the Video Encoder EI3 Extender Board Manual.

### Setup Soft Switch

Only for ADSP-SC589 EZ-Board (74MHz PPI clock):

```
# echo 466 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio466/direction
```

```
# echo 469 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio469/direction
```

Both for ADSP-SC573 EZ-Board and ADSP-SC589 EZ-Board (27MHz PPI clock):

```
# echo 466 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio466/direction
# echo 468 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio468/direction
```

## Display Video Images

You can use "v4l2\_video\_display" provided by Buildroot to play HD yuv file and output the HD signal via ADV7511, the HD signal will be transmitted to TV, then you will see the image on TV.

```
# v4l2_video_display -F 720p60.yuv
```

## ADV7341 Video Encoder

### Setup Soft Switch

```
# echo 468 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio468/direction
# echo 470 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio470/direction
```

## Display Video Images

You can use "v4l2\_video\_display" provided by Buildroot to play pal yuv file and output the component analog signal via ADV7341, the analog signal will be transmitted to TV by 3RCA cable, then you will see the image on TV.

```
# v4l2_video_display -F pal.yuv
```

## 5.16 Rotary driver

### 5.16.1 Introduction

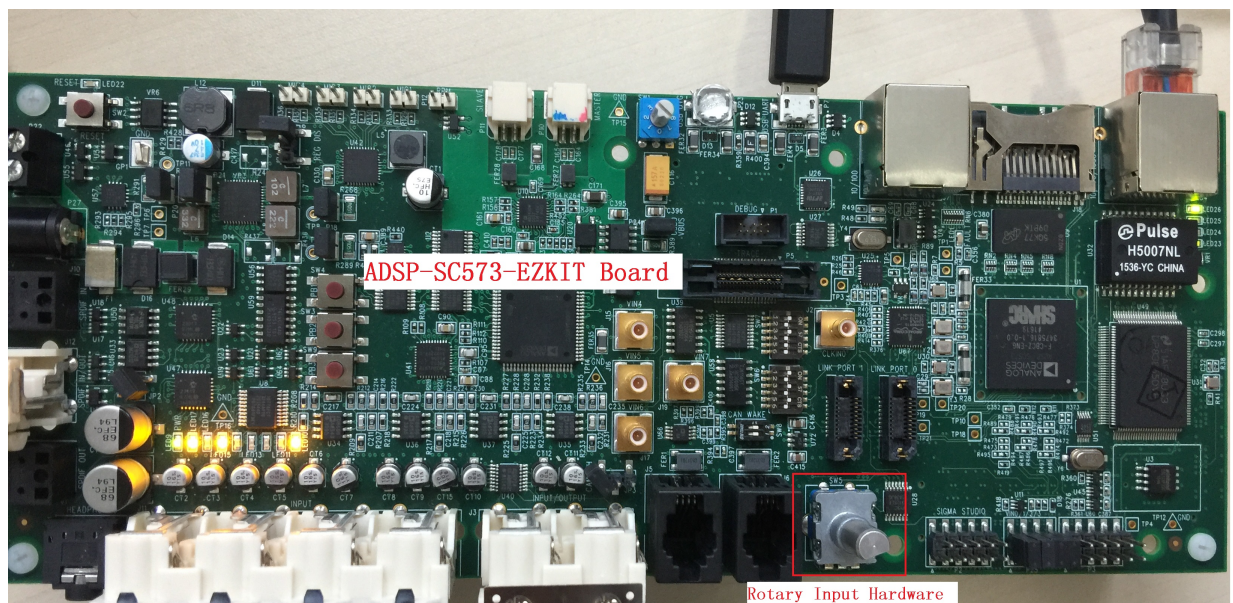
SC5xx processors feature an up/down counter and thumbwheel interface.

A 32-bit up/down counter is provided that can sense 2-bit quadrature or binary codes as typically emitted by industrial drives or manual thumb wheels. The counter can also operate in general-purpose up/down count modes, in which case count direction is either controlled by a level sensitive input pin or by two edge detectors. A third input can provide flexible zero marker support and can alternatively be used to input the push-button signal of thumb wheels. All three pins have a programmable debouncing circuit.

### 5.16.2 Hardware Setup

- An ADSP-SC584/SC573 EZKIT board

The SC589-EZKIT board does not include the rotary input hardware.



### 5.16.3 Software Configuration

#### Configure Linux kernel

You need to enable the ADI rotary driver in Linux kernel.

```

Device Drivers --->
  Input device support --->
    <*>   Event interface
    [*]   Miscellaneous devices --->
      <*>   ADI Rotary support

```

## Configure Buildroot

You should also enable the event test program to assist with testing.

```

Target Packages ---->
  Miscellaneous ---->
    [*] event test

```

## Customization

The driver can be customized via the device tree node, take the SC573-ezkit board as an example, add rotary device and pinctrl node in **linux\_kernel\_source/arch/arm/boot/dts/sc573-ezkit.dts**:



... : ellipsis, means other properties in i2c node stay the same

- : minus, means delete this property

+ : plus, means add this property

```

+rotary@0x3100B000 {
+   #address-cells = <1>;
+   #size-cells = <0>;
+   compatible = "adi,rotary";
+   reg = <0x3100B000 0xFF>;
+   pinctrl-names = "default";
+   pinctrl-0 = <&rotary0_default>;
+   interrupts = <0 118 0>;
+   rotary_rel_code = <0x08>; /* REL_WHEEL */
+   rotary_button_key = <28>; /* KEY_ENTER */
+   debounce = /bits/ 16 <10>; /* 0..17 */
+   debounce_en = /bits/ 16 <1>;
+   cnt_mode = /bits/ 16 <0>; /* CNTMODE_QUADENC */
+   boundary_mode = /bits/ 16 <0>; /* BNDMODE_COMP */
+   invert_czm = /bits/ 16 <0>;

```

```

+     invert_cud = /bits/ 16 <0>;
+     invert_cdg = /bits/ 16 <0>;
+};

pinctrl@0 {
    ...
    mmc {
        ...
        +rotary {
            +     rotary0_default: rotary0@0 {
            +         adi,group = "rotary0grp";
            +         adi,function = "rotary0";
            +     };
            +};
        };
    };
};

```

The driver provides two options. It can either send Key (KEY) or Relative (REL) events.

**Option 1 – send Key events:** Provide two KEY Codes for:

```

rotary_up_key = KEY_PLUS
rotary_down_key = KEY_MINUS

```

**Option 2 – send REL events:** Provide one REL event type:

```

rotary_rel_code = REL_WHEEL

```

In case your CZM input is connected (push-button signal of thumb wheels) Specify the KEY event – this will enable the CZM input. See include/linux/input.h for a full list of supported events.

```

rotary_button_key = KEY_ENTER

```

The debounce prescale value is used to select the noise filtering characteristic of the input pins. Must be in the range of 0..17

```

debounce = 10

```

The driver supports various Counter types

```

cnt_mode = CNTMODE_QUADENC

```

## 5.16.4 Example

You will get following information when the rotary hardware device is turned left, right or pushed.

```

# event_test /dev/input/event0
Input driver version is 1.0.0
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "bfin-rotary"
Supported events:
  Event type 0 (Reset)
    Event code 0 (Reset)
    Event code 1 (Key)
    Event code 2 (Relative)
  Event type 1 (Key)
    Event code 28 (Enter)
  Event type 2 (Relative)
    Event code 8 (Wheel)
Testing ... (interrupt to exit)
Event: time 21755.740000, type 2 (Relative), code 8 (Wheel), value 1
Event: time 21755.740000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21755.780000, type 2 (Relative), code 8 (Wheel), value 1
Event: time 21755.780000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21755.808000, type 2 (Relative), code 8 (Wheel), value 1
Event: time 21755.808000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21755.812000, type 2 (Relative), code 8 (Wheel), value 1
Event: time 21755.812000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21756.040000, type 2 (Relative), code 8 (Wheel), value 1
Event: time 21756.040000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21756.208000, type 2 (Relative), code 8 (Wheel), value -1
Event: time 21756.208000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21757.152000, type 2 (Relative), code 8 (Wheel), value -1
Event: time 21757.152000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21757.176000, type 2 (Relative), code 8 (Wheel), value -1
Event: time 21757.176000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21757.184000, type 2 (Relative), code 8 (Wheel), value -1
Event: time 21757.184000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21757.188000, type 2 (Relative), code 8 (Wheel), value -1
Event: time 21757.188000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21757.380000, type 2 (Relative), code 8 (Wheel), value -1
Event: time 21757.380000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21760.336000, type 1 (Key), code 28 (Enter), value 1
Event: time 21760.336000, type 0 (Reset), code 0 (Reset), value 0
Event: time 21760.336000, type 1 (Key), code 28 (Enter), value 0
Event: time 21760.336000, type 0 (Reset), code 0 (Reset), value 0

```

## 5.17 SPI Driver

### 5.17.1 Introduction

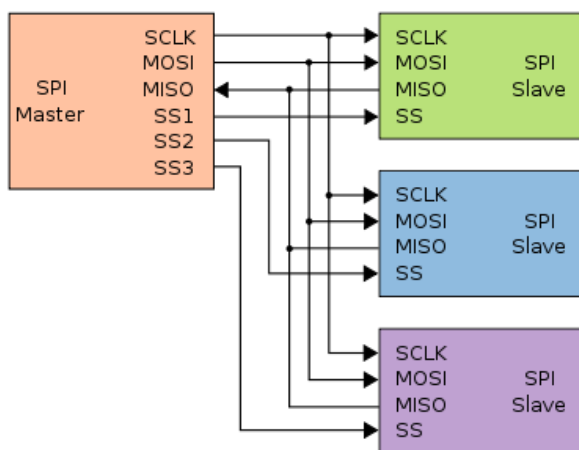
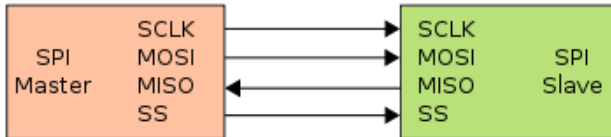
This section describes the steps required to build and use the SPI bus on Linux using an ADSP-SC5xx board.

The Serial Peripheral Interface (SPI) bus is a four wire master/slave full duplex synchronous bus. You can hook up multiple slave devices by utilizing chip select lines.

The bus is composed of two data pins, one clock pin, and one chip select pin:

- SCLK - Serial Peripheral Interface Clock Signal (generated by the master) (also referred to as SCK)
- MOSI - Master Out Slave In data (output from the master)
- MISO - Master In Slave Out (output from the slave)
- CS - Chip Select (also referred to as Slave Select (SS))

It is not uncommon to use the bus with just one master and one slave, but it is certainly possible to use it as a real bus with many devices on it.



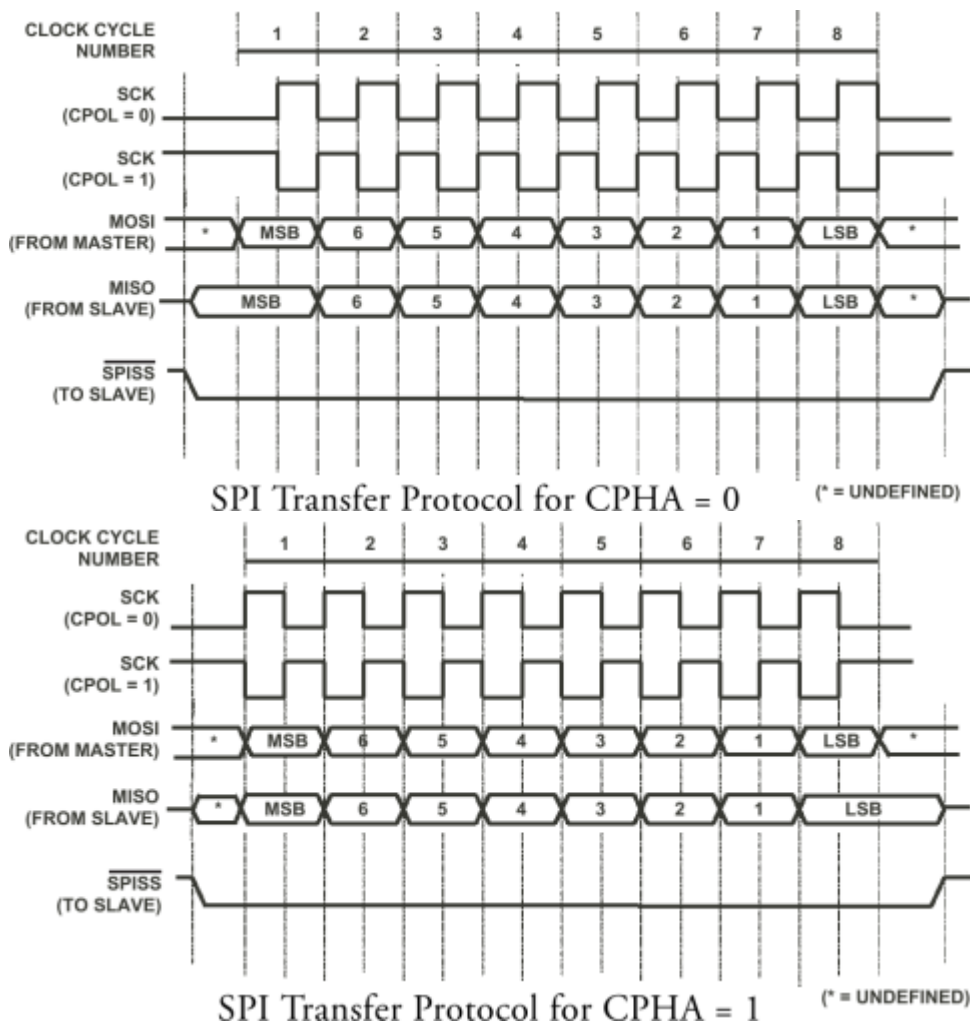
Each slave may operate at different clock frequencies as well as different clock polarities and clock phases with respect to the data. The permutations of polarities and phases are referred to as SPI modes. Below you can see the relationship between modes and the polarity/phase of the clock.

Mode	Polarity	Phase
SPI_MODE_0	0	0
SPI_MODE_1	0	1
SPI_MODE_2	1	0
SPI_MODE_3	1	1

Chip specifications won't always say "uses SPI mode X" in as many words, but their timing diagrams will make the CPOL and CPHA modes clear.

The figures below demonstrate the two basic transfer formats as defined by the CPHA bit. Two waveforms are shown for SPI\_CLK—one for SPI\_CTL.CPOL=0 and the other for SPI\_CTL.CPOL=1.





### 5.17.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board

### 5.17.3 Software Configuration

The following configuration should be done on top of the SC589-ezkit/SC584-ezkit/SC573-ezkit default configuration.

#### Configure Buildroot

Enable `spidev_test` program if you want to use user space SPI test utility.

```
Target packages --->
  Debugging, profiling and benchmark --->
    [*] spidev_test
```

## Configure Linux Kernel

### Enable SPI controller driver for ADSP-SC5xx

```
Device Drivers --->
  [*] SPI support --->
    <*> SPI controller v3 for ADI
```

### Enable SPI slave driver (for example spi flash w25q128)

```
Device Drivers --->
  <*> Memory Technology Device (MTD) support --->
    <*> SPI-NOR device support --->
      Self-contained MTD device drivers --->
        <*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)
```

### Enable spidev driver if you want to use user space API.

```
Device Drivers --->
  [*] SPI support --->
    <*> User mode SPI device driver support
```

## Configure Device tree

### SPI slave node properties

SPI busses can be described with a node for the SPI master device and a set of child nodes for each SPI slave on the bus.

Below is the child node for SPI flash of SPI master2 node.

```

flash: w25q32@0 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "winbond,w25q32";
    spi-max-frequency = <500000>;
    reg = <38>;
    spi-cpol;
    spi-cpha;
    spi-rx-bus-width = <4>;
    dma-mode;
}

```

SPI slave nodes must be children of the SPI master node and can contain the following properties.

reg - (required) Chip select address of device.

compatible - (required) Name of SPI device following generic names recommended practice

spi-max-frequency - (required) Maximum SPI clocking speed of device in Hz

spi-cpol - (optional) Empty property indicating device requires inverse clock polarity (CPOL) mode

spi-cpha - (optional) Empty property indicating device requires shifted clock phase (CPHA) mode

spi-cs-high - (optional) Empty property indicating device requires chip select active high

spi-rx-bus-width - (optional) A value of 4 indicates to setup the SPI controller to receive data in Quad SPI mode.

dma-mode - (optional) Empty property indicating device requires DMA mode transfer

### **spidev device node**

If you want to use spidev, please add following SPI slave node to SPI master0 node.

```

spidev {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "rohm,dh2228fv";
    spi-max-frequency = <5000000>;
    reg = <44>;
};

```

## 5.17.4 Example

```
# spidev_test -D /dev/spidev0.44 -b 8 -H -O
spi mode: 0x3
bits per word: 8
max speed: 500000 Hz (500 KHz)
FF 80 00 00 3F FF
40 00 00 00 02 00
3F FF FF FF FF 80
00 00 3F FF FF FF
FF 80 00 00 3F FF
DE AD BE 80 00 00
30 0D
```

## 5.18 Mobile Storage Interface for MMC/SD

The ADSP-SC5xx processors provide a mobile storage interface (MSI). MSI is a fast, synchronous controller that uses various protocols to communicate with MMC, SD, and SDIO cards to address the growing storage need in embedded systems, handheld and consumer electronics applications requiring low power. The MSI is compatible with the following protocols.

- MMC (Multimedia Card) bus protocol
- SD (Secure Digital) bus protocol
- SDIO (Secure Digital Input Output) bus protocol

All of these storage solutions use similar interface protocols. The main difference between MMC and SD support is the initialization sequence. The main difference between SD and SDIO support is the use of interrupt and read wait signals for SDIO.

### 5.18.1 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC573 EZ-Board (SC584 processor does not include MMC/SD controller)

The SD/MMC card slot is J18 on the SC589/SC573 EZKIT board . This slot accepts full-size SD and MMC cards, or microSD cards with an adapter.



## 5.18.2 Software Configuration

### Configure Linux Kernel

#### 1) MSI Support

```
Device Drivers
  MMC/SD/SDIO card support --->
    [*] Synopsys DesignWare Memory Card Interface
    [*]   Internal DMAC interface
    [*]   Synopsys Designware MCI Support as platform device
    [*]   ADI specific extensions for Synopsys DW Memory Card
    Interface
```

This configuration has been added in `sc589-ezkit_defconfig/sc573-ezkit_defconfig` as the default configuration.

#### 2) File System Support

if you want to mount an SD card in a particular format, you should compile the Linux kernel with the corresponding file-system first.

**Example1**FAT32 SD card, corresponding filesystem is VFAT. Configuration is shown below

```
File systems --->
  -> DOS/FAT/NT Filesystems
    [*] VFAT (Windows-95) fs support
    (437) Default codepage for FAT
    (iso8859-1) Default iocharset for FAT
```

**Example2** : ext2 SD card, corresponding filesystem is ext2. Configuration is shown below

```
File systems --->
  <*> Second extended fs support
  [*]   Ext2 extended attributes
  [*]   Ext2 POSIX Access Control Lists
  [*]   Ext2 Security Labels
```

## Configure Buildroot

Add Bonnie++ to the Buildroot configuration. Bonnie++ is a program for testing filesystem throughput, see [www.coker.com.au/bonnie++](http://www.coker.com.au/bonnie++) for details.

```
Target packages --->
  Debugging, profiling and benchmark --->
    [*] bonnie++
```

### 5.18.3 Build and Load Buildroot

A Buildroot image can now be built and loaded onto the target board. See [SC5xx EZ-Kit Linux Quick Start Guide](#) for details.

### 5.18.4 Usage of MSI

#### Mount

The most typical use of an SD Card in embedded applications is as a removable storage device (disk) that can be easily taken from the embedded target board. In such contexts, the SD Card installed to the embedded target board is typically already formatted with an MS-DOS file system. The Linux kernel must be specially configured to allow mounting the MS-DOS file system. See part 2 in section 2.1 for details. The **mount** utility is also needed. Typically, mount will already be enabled in your busybox configuration.

Insert a pre-formatted card with an MS-DOS (FAT) file system to the SD Card slot on the ADSP-SC5xx. When you boot the uImage on the ADSP-SC5xx, there should be messages similar to the ones shown below. In the below example, Linux has detected an SD Card with a single partition on it:

```
mmc_host mmc0: Bus speed (slot 0) = 500000000Hz (slot req 250000000Hz,
actual 250000000HZ div = 1)
mmc0: new SD card at address b368
mmcblk0: mmc0:b368 FFFFF 1.85 GiB
```

At this point you are ready to mount the MS-DOS file system on the SD Card. This is done as follows:

```
# mount -t vfat -o sync /dev/mmcblk0p1 /mnt
```

Check that the file system has indeed been mounted (refer to the last line in the below output):

```
# mount
rootfs on / type rootfs (rw)
devtmpfs on /dev type devtmpfs (rw,relatime,size=42740k,nr_inodes=10685,mode=755)
proc on /proc type proc (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw,relatime,mode=777)
tmpfs on /tmp type tmpfs (rw,relatime,mode=777)
sysfs on /sys type sysfs (rw,relatime)
debugfs on /sys/kernel/debug type
debugfs (rw,relatime)
/dev/mmcblk0p1 on /mnt type vfat (rw,sync,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)
```

Now you can write something to the SD Card. In the below example, we store the current date and time to a log file, although in real-life applications you will probably want to do something more meaningful:

```
# date > /mnt/log.file
```

Verify the written content by reading the log file back:

```
# cat /mnt/log.file
Wed Aug 12 10:29:00 UTC 2015
```

Now you can remove the card from the embedded target board. Unmount the file system and then extract the card from the SD card slot:

```
# umount /mnt
```

```
mmc0: card b368 removed
```

## Test MSI Performance with Bonnie++

### 1) Test Case 1: Bonnie++ on Ext2

Input the following command on the target board console

```
root: /> mkfs.ext2 /dev/mmcblk0p1
root: /> mount /dev/mmcblk0p1 /mnt/
root: /> bonnie++ -u root -d /mnt/
```

### Result

```
Using uid:0, gid:0.
Writing with putc()...done
Writing intelligently...done
Rewriting...done Reading with getc()...done
Reading intelligently...done
start 'em...done...done...done...
Create files in sequential order...done.
Stat files in sequential order...done.
Delete files in sequential order...done.
Create files in random order...done.
Stat files in random order...done.
Delete files in random order...done.
Version 1.03e          -----Sequential Output----- --Sequential Input--
--Random-
-Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --
Seeks--
Machine          Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP
/sec %CP
buildroot        300M  2568  70  2780  2  4212  6  3734  97 18000  11
1169  16
-----Sequential Create----- -----Random
Create-----
-Create-- --Read--- -Delete-- -Create-- --Read---
-Delete--
files /sec %CP /sec %CP /sec %CP /sec %CP /sec %
CP /sec %CP
          16  958  98 +++++ + + + + + 1032  98 +++++
+++ 2449  99 buildroot,300M,2568,70,2780,2,4212,6,3734,97,18000,11,116
9.2,16,16,958,98,+++++,+,+++++,+,1032,98,+++++,+,2449,99
```



## 2) Test Case 2: Bonnie++ on FAT32

Input the following command on the target board console

```
root:/> mkfs.vfat -F 32 /dev/mmcblk0p1
root:/> mount /dev/mmcblk0p1 /mnt/
root:/> bonnie++ -u root -d /mnt/
```

### Result

```
Using uid:0, gid:0. Writing with putc()...done
Writing intelligently...done
Rewriting...done
Reading with getc()...done
Reading intelligently...done
start 'em...done...done...done...
Create files in sequential order...done
Stat files in sequential order...done.
Delete files in sequential order...done.
Create files in random order...done.
Stat files in random order...done.
Delete files in random order...done.
Version 1.03e          -----Sequential Output----- --Sequential Input-
--Random-
                -Per Chr- --Block-- -Rewrite- -Per Chr- --Block--
--Seeks--
Machine          Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP
/sec %CP
buildroot        300M  2414  64  5529   8  4502   8  3715  96 17941  12
92.0  1
                -----Sequential Create----- -----Random
Create-----
                -Create-- --Read--- -Delete-- -Create-- --Read--- -
Delete--
                files /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP
/sec %CP
                16    12  99 29536 100    156  99    18  99 +++++
+++    41  97
buildroot,300M,2414,64,5529,8,4502,8,3715,96,17941,12,92.0,1,16,12,99,2
9536,100,156,99,18,99,+++++,+++ ,41,97
```

## 5.19 Real Time Clock on ADSP-SC58x

### 5.19.1 Introduction

This page describes the steps required to build and use the RTC on the ADSP-SC589 EZ-board (ADSP-SC584 and ADSP-SC573 don't support RTC).

The Real Time Clock (RTC) serves two purposes: to keep accurate time/date information and to provide wake up alarms (both during runtime and while sleeping). Since the RTC can be externally powered and clocked independently of the processor it can remain running even when the rest of the system is turned off.

### 5.19.2 Hardware Required

- An ADSP-SC589 EZ-Board

### 5.19.3 Software Configuration

#### Configure Buildroot

Enable RTC test:

```
Target packages --->
  Miscellaneous --->
    [*] rtc test
```

#### Configure Linux Kernel

Enable RTC support:

```
Device Drivers --->
  [*] Real Time Clock --->
    <*> ADI On-Chip RTC v2
```

## 5.19.4 Build and Load Buildroot

After configuration, users can make and boot the image on SC589 EZ-Board. Please refer [SC5xx EZ-Kit Linux Quick Start Guide](#) for details.

## 5.19.5 Example

Users just need to run `rtc_test` command on board to test rtc.

Results as shown below:

```
# rtc_test
0. open and release
opened '/dev/rtc0': fd = 3
1. ioctl RTC_UIE_ON
2. RTC read 5 times
RTC read 1
RTC read 2
RTC read 3
RTC read 4
RTC read 5
3. ioctl RTC_UIE_OFF
4. Get RTC Time
Current RTC date/time is 24-4-76, 22:48:15
5. Set RTC Time
Set Current RTC date/time to 31-5-104, 02:30:00
Get RTC time
Current RTC date/time is 31-5-104, 02:30:00
6. Set alarm Time
7. Get alarm Time
Alarm time now set to 02:30:50
Waiting 50 seconds for alarm...
random: nonblocking pool is initialized
Okay. Alarm rang.
Current RTC date/time is 31-5-104, 02:30:50
8. ioctl RTC_AIE_OFF
8.5 test sleep 10
Current RTC date/time is 31-5-104, 02:31:00
Current RTC date/time is 31-5-104, 02:31:00
RTC Tests done !
```

## 5.20 UART hardware flow control

### 5.20.1 Introduction

This section describes the steps required to enable UART hardware flow control on ADSP-SC5xx board.

### 5.20.2 Hardware Setup

- An ADSP-SC5xx EZ-Board: ADSP-SC589/SC584/SC573 EZ-Board

The UART interface is the **USB to UART** port on the ADSP-SC5xx EZ-board.

### 5.20.3 Software Configuration

#### Configure Linux kernel

```
Device Drivers --->
  Character devices --->
    Serial drivers --->
      <*> ADI uart4 serial port support
      [*] Console on ADI uart4 serial port
```

#### Configure Buildroot

```
Target packages --->
  Miscellaneous --->
    [*] UART flow control test utility
```

#### Configure Device tree

Add in uart0 node of '**linux source code directory**'/arch/arm/boot/dts/sc58x.dtsi or '**linux source code directory**'/arch/arm/boot/dts/sc57x.dtsi

```
uart0: uart@0x31003000 {
  ...
```

```
adi,uart-has-rtscts;  
...  
};
```

## 5.20.4 Build and Load Buildroot

A Buildroot image can now be built and loaded onto the target board. See [SC5xx EZ-Kit Linux Quick Start Guide](#) for details.

## 5.20.5 Example

### Preliminary setup

Setting serial port parameter

```
# minicom -s  
Select "Serial port setup"  
Select "F - Hardware Flow Control: Yes"
```

### Test

#### 1) Input invalid before Linux boot up

After powering-off and restarting the board you will find that UBoot doesn't accept the input from the serial console. Instead it automatically boot up kernel according to its predefined parameters. Once the Linux kernel has been booted it will accept input..

#### 2) run rtscts\_test case

```
# rtscts_test ttySC0 -t
```



In the above command the serial device is given as an example, and will not necessarily be ttySC0.

## 6 Multicore support

### 6.1 Multi-Core Communication

The CrossCore Embedded Studio Linux Add-In provides user space APIs that allow applications to easily communicate with the SHARC cores of the ADSP-SC5xx processor.

Communication is performed using the Multi-core Communications API (MCAPI) specification which defines an API and a semantics for communication and synchronization between processing cores in embedded systems. MCAPI is supported on the SHARC cores of the ADSP-SC5xx processors in both bare-metal and RTOS using CrossCore Embedded Studio.

For more information about MCAPI, please refer to the MCAPI Specification document from [MCAPI.org](http://mcaapi.org).

#### 6.1.1 MCAPI Supported Functions

The Linux Add-In supports a subset of the APIs stated in the MCAPI specification, version 2.0.15.

- Inter-core communication is only supported using **messaging** mode. Communication using packet and scalar types is **not** supported.
- Communication can be performed using blocking and non-blocking methods.

The following table summarizes the APIs supported in this release of the Linux Add-In.

Function	Supported APIs	Un-supported APIs
<b>General Functions</b>		
Initialize MCAPI Environment	mcapi_initialize	
Finalize MCAPI Environment	mcapi_finalize	
Get Domain ID	mcapi_domain_id_get	
Get Node ID	mcapi_node_id_get	
Initialize Structure Values	mcapi_node_init_attributes	

Change Default Values	mcapi_node_set_attribute	
Query Node Attributes	mcapi_node_get_attribute	
<b>Endpoint Functions</b>		
Create Endpoint	mcapi_endpoint_create	
Delete Endpoint	mcapi_endpoint_delete	
Get Endpoint Identifier	mcapi_endpoint_get_i	
	mcapi_endpoint_get	
Query Endpoint Attribute		mcapi_endpoint_get_attribute
Assign Endpoint Attribute		mcapi_endpoint_set_attribute
<b>Message Functions</b>		
Send message	mcapi_msg_send_i	
	mcapi_msg_send	
Receive message	mcapi_msg_recv_i	
	mcapi_msg_recv	
Check Message Availability	mcapi_msg_available	
<b>Non-Blocking, Connectionless Message, Packet, and Channel Functions</b>		
Wait for Completion of Operation	mcapi_wait	
Check for Non-blocking Ops Completion	mcapi_test	

Cancel Non-blocking Ops		mcapi_cancel
Wait for Completion of Ops List		mcapi_wait_any
<b>Support Functions</b>		
Display MCAPI Status Message	mcapi_display_status	

## 6.1.2 Multi-core Development

Following Documents covers how to run the dual core data transaction demo with MCAPI, as well as some general multi-core development/debugging introduction.

Run the MCAPI demo

- [Run the MCAPI demo](#)

Debugging multi-core application in general

- [Run Linux on ARM and bare-metal application on SHARC](#)

In a multi-core system such as the ADSP-SC5xx, there are lots of resources that are shared between the ARM and SHARC cores. This guide will introduce two shared resources (pins and SEC). It is important to understand how to avoid inter-core conflicts in these devices. For more information about using these devices in Linux, please refer to:

- [Reserve Pinmux Functions in Linux for SHARC Applications](#)
- [SEC driver and multicore development](#)

## 6.2 Run the MCAPI demo

### 6.2.1 Introduction

This document shows how to run the MCAPI demo on ADSP-SC5xx EZ-Kit board.

One method of loading applications to multiple cores of the ADSP-SC5xx processor is to use the loader functionality in the processor BootROM.



The BootROM supports the loading of multiple binary files onto separate cores from one LDR image which is stored in the SPI flash. In order to achieve this a loader image must be created using the proprietary ELF loader, **elfloader.exe** that is provided in the windows version of CrossCore Embedded Studio.

For more information on the elfloader.exe application see the CrossCore Embedded Studio IDE Help, or provide the **-help** option to the **elfloader.exe** application.

### **i No GNU ARM Loader support**

The GNU ARM loader utility provided with Linux does not currently support the linking of SHARC binary images into a loader image.

## **6.2.2 Hardware Requirement**

- ADSP-SC589 EZ-Kit v1.0 and above, or
- ADSP-SC584 EZ-Kit v1.0 and above, or
- ADSP-SC573 EZ-Kit v1.0 and above
- ICE1000 or ICE2000 JTAG board

Here we take ADSP-SC589 EZ-kit as our example.

## **6.2.3 Configure and build**

This section shows how to configure, build the source code, and generate the dual core combined bootloader.

### **Generate Linux Kernel image**

#### **Configure Linux kernel**

MCAPI lib is built on top of the ICC (Inter-Core Communications) device driver. You should run **make linux-menuconfig** in Linux kernel to enable the ICC driver in Linux kernel first:

```
Device Drivers  --->
  [*] Staging drivers  --->
    [*]   icc driver  --->
      --- icc driver
        [*]   icc protocol
          [ ]   icc debug
```

## Configure Buildroot

Then you need to run **make menuconfig** in buildroot to enable libmcapi in Buildroot:

```
Target packages --->
  Libraries --->
    Other --->
      [*] libmcapi
```

Two methods to control SHARC cores are described in the document [Enable and Disable SHARC Cores](#). Here we use corecontrol tool in Linux to enable and disable SHARC cores.

```
Target packages --->
  Miscellaneous --->
    [*] sc5xx corecontrol tool
```

## Build

Finally, build the buildroot to generate the linux kernel image.

```
$ make
```

Copy the **uImage** file to **/tftpboot** folder.

## Generate Combined u-boot LDR image

Note there is a known issue, check details in "trouble shooting". The elfloader.exe tool in CCES installation directory in windows supports a large number of parameters. In order to generate a combined u-boot LDR image with both the u-boot and SHARC code, you have to use many parameters. Required steps should be done follow the following parts.

### Step1: Copy files from Linux host to Windows host

We should first customize a default u-boot according to your board.

```
$ cd u-boot/  
$ make sc589-ezkit_defconfig  
$ make
```

Then copy three files we need from the Linux Add-In to the Windows host.

1. Copy the **cces\_mcap\_i\_test** example which is located in the **/opt/analog/cces-linux-add-in/1.2.0/examples/cces\_mcap\_i\_test** folder as part of the Linux Add-In to a Windows host PC .
2. Copy your **u-boot** file that is located at the top level of your u-boot sources to the **\$CCES\_DIR** in Windows host PC.
3. Copy the u-boot **init** file that is located in the **arch/arm/cpu/armv7/sc58x** directory of your u-boot sources file to the **\$CCES\_DIR** in Windows host PC.



**\$CCES\_DIR** is the installation directory of CCES which is often in "C:\Analog Devices\CrossCore Embedded Studio 2.6.0".

## Step2: Build project on the Windows host in CrossCore Embedded Studio

Here we build the SHARC core project within CrossCore Embedded Studio

1. From the **File** menu select **Import...**
2. From the Import dialog expand the **General** entry, select **Existing Projects Into Workspace** and click **Next**
3. Browse to the location where you copied the **mcap\_i\_send\_recv\_Core1** project to the Windows PC, and select the **mcap\_i\_send\_recv\_Core1** project for your EZ-Kit, then click **Import**
4. In the **Project Explorer** right click on the **mcap\_i\_send\_recv\_Core\_sc589** project and select the **Build Project** option
5. Copy **mcap\_i\_send\_recv\_Core1\_sc589.dxe** in `$your_project_folder/Debug` to the **\$CCES\_DIR**

## Step3: Generate a combined u-boot LDR using a command line Shell

Here we use the command line CrossCore Embedded Studio **elfloader** utility to produce a single loader image containing u-boot and the SHARC executable.

Before creating the loader image, ensure that command line shell is in the directory where the output executable from the CrossCore Embedded Studio project resides.

Also ensure that the **uBoot** and **init** binaries have been copied to the same directory.

1. Run following command to generate u-boot-mcap\_i.ldr, replacing the **\$CCES\_DIR** with the installation directory for CrossCore Embedded Studio:

```
$CCES_DIR\elfloader.exe -proc ADSP-SC589 -si-revision 1.0 -b spimaster
-f binary -width 8 -bcode 1 -init init -core0=u-boot -
core1=mcapi_send_recv_Core1_sc589.dxe -NoFinalTag=u-boot -o u-boot.ldr
```

2. Copy the resultant **u-boot.ldr** file to the Linux Host PC as the file **/tftpboot/u-boot.ldr**

#### Step 4: Flash your new uboot to the SPI flash

```
sc# run update
```

### 6.2.4 Running MCAPI MSG Test Example

After flashing a combined u-boot, hit the **reset** button on the ezkit board, boot the Linux to the serial console

Then start the sharc core 1

```
$ corecontrol --start 1
Test core 1 start
Test core 1 end: 0
```

Test the MCAPI protocol in Linux

```
# arm_sharc_msg_demo
CHECK_STATUS---init_node_attr: MCAPI_SUCCESS
semget
CHECK_STATUS---initialize: MCAPI_SUCCESS
  node=0, port=101
CHECK_STATUS---create_ep: MCAPI_SUCCESS
ep1 65
CHECK_STATUS---get_ep_i: MCAPI_PENDING
  node=1, port=5
CHECK_STATUS---wait: MCAPI_SUCCESS
ep2 10005
send() start.....
CHECK_STATUS---send_i: MCAPI_PENDING
CHECK_STATUS---test: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of send() - endpoint=101 has sent: [HELLO_MCAPI core0 0]
```

```

coreA: mode(0) message send. The 0 time sending
recv() start.....
session_idx 0
  node=1, port=31
CHECK_STATUS---recv_i: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of recv() - endpoint=101 size 0x9ca8 has received: [hello mcapi
core1]
CoreA: mode(0) message recv. The 0 time receiving
send() start.....
CHECK_STATUS---send_i: MCAPI_PENDING
CHECK_STATUS---test: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of send() - endpoint=101 has sent: [HELLO_MCAPI core0 1]
coreA: mode(0) message send. The 1 time sending
recv() start.....
session_idx 0
  node=1, port=31
CHECK_STATUS---recv_i: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of recv() - endpoint=101 size 0x9ca8 has received: [hello mcapi
core1]
CoreA: mode(0) message recv. The 1 time receiving
....
send() start.....
CHECK_STATUS---send_i: MCAPI_PENDING
CHECK_STATUS---test: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of send() - endpoint=101 has sent: [HELLO_MCAPI core0 99]
coreA: mode(0) message send. The 99 time sending
recv() start.....
session_idx 0
  node=1, port=31
CHECK_STATUS---recv_i: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of recv() - endpoint=101 size 0x9ca8 has received: [hello mcapi
core1]
CoreA: mode(0) message recv. The 99 time receiving
CHECK_STATUS---del_ep: MCAPI_SUCCESS
mcapi_finalize 322
CHECK_STATUS---finalize: MCAPI_SUCCESS
CoreA 100 rounds mode(0) demo Test PASSED!!

```

More details of the introduction of MCAPI examples, please refer to [Introduction of MCAPI examples](#)

## 6.2.5 Troubleshooting

After flashing a **multi-core uboot.ldr** which is combined with mcapi\_send\_recv\_Core1\_sc589.dxe and u-boot and init file , the console gives out a message "**Warning – bad CRC , using default environment**", the board will not boot on next power on or reset.

This issue is mostly because elfloader.exe does not support the "punchit" feature as of now, save command will write into Flash and break the dual core u-boot.ldr image on it. For users prefer to use save command to change the environment on fly in dual core application, we recommend a workaround as following:

Change the u-boot source file as following, note in this example the location for saving environment variable on Flash will start from 0x70000, you will need to increase it more if the size of your combined u-boot.ldr is larger than that, but make sure to update your usage of MTD table in Linux kernel accordingly so that your MTD partion does not get overwritten by accident.

```
diff --git a/arch/arm/cpu/armv7/sc58x/config.mk b/arch/arm/cpu/armv7
/sc58x/config.mk
index 2717269..a506a68 100644
--- a/arch/arm/cpu/armv7/sc58x/config.mk
+++ b/arch/arm/cpu/armv7/sc58x/config.mk
@@ -29,7 +29,7 @@ endif

  ifneq ($(CONFIG_SC_BOOT_MODE), SC_BOOT_UART)
  ifneq ($(CONFIG_SC58X_CHAIN_BOOT), Y)
- LDR_FLAGS-$(CONFIG_ENV_IS_EMBEDDED_IN_LDR) += --punchit
+ $$(( $(CONFIG_ENV_OFFSET) )) : $$(( $(CONFIG_ENV_SIZE) )) : env-ldr.o
+ LDR_FLAGS-$(CONFIG_ENV_IS_EMBEDDED_IN_LDR) :=
  endif
  endif

diff --git a/arch/arm/cpu/armv7/sc58x/init b/arch/arm/cpu/armv7/sc58x
/init
index 46a7d1f..a127bb0 100755
Binary files a/arch/arm/cpu/armv7/sc58x/init and b/arch/arm/cpu/armv7
/sc58x/init differ
diff --git a/include/configs/sc589-ezkit.h b/include/configs/sc589-
ezkit.h
index adde83a..ac99137 100644
--- a/include/configs/sc589-ezkit.h
+++ b/include/configs/sc589-ezkit.h
@@ -169,7 +169,7 @@
 * Env Storage Settings
 */
#define CONFIG_ENV_IS_IN_SPI_FLASH
-#define CONFIG_ENV_OFFSET          0x10000
+#define CONFIG_ENV_OFFSET          0x70000
```

```

#define CONFIG_ENV_SIZE          0x2000
#define CONFIG_ENV_SECT_SIZE    0x10000
#define CONFIG_ENV_IS_EMBEDDED_IN_LDR
diff --git a/include/configs/sc_adi_common.h b/include/configs
/sc_adi_common.h
index da3599c..d2aef92 100644
--- a/include/configs/sc_adi_common.h
+++ b/include/configs/sc_adi_common.h
@@ -258,7 +258,7 @@
 /*
  * Env Storage Settings
  */
-#define CONFIG_ENV_OFFSET        0x10000
+#define CONFIG_ENV_OFFSET        0x70000
#define CONFIG_ENV_SIZE          0x2000
#define CONFIG_ENV_SECT_SIZE    0x10000
/* We need envcrc to embed the env into LDRs */

```

Then run make, and continue with section “**Generate Combined u-boot LDR image**”.

## 6.3 Enable and Disable SHARC Cores

### 6.3.1 Introduction

This document introduces two kind of core control solutions which support ADSP-SC573, SC584 and SC589 EZ-Kits. One method is to control SHARC Cores with u-boot ICC command. Other way includes a kernel device driver and a command-line utility, for enabling and disabling the SHARC cores (Core 1 & 2) from the ARM core (Core 0), which have been added to the Buildroot Linux distribution for ADSP-SC573, SC584 and SC589.

### 6.3.2 Method 1: Enable SHARC cores with u-boot ICC command

#### Configure u-boot to Enable Slave Cores

The slave SHARC core 0 and 1 in SC5xx silicon can be enabled and disabled in u-boot command console. In order to include this ICC command into u-boot, you need to select it in u-boot configuration.

```

$ cd u-boot/
$ make sc589-ezkit_defconfig
$ make menuconfig

```

```
ARM architecture --->
  [*] ICC command to enable and disable slave cores
```

After saving the configuration in u-boot, rebuild the u-boot ldr image for SC58x-EZKIT.

```
$ make
```

Finally, flash the u-boot LDR image into the SPI flash of SC5xx-EZKIT.

### U-boot ICC command usage

The ICC command can enable or disable a specific slave core. The SHARC core ids accepted by this command for SC5xx are 1 and 2, any other id value is ignored. The ICC message queue at the beginning of L2 SRAM is reset before enabling the SHARC cores.

```
sc # icc
icc - Inter core communication interface
Usage:
icc icc enable <coreid>
  - enable coreid
icc icc disable <coreid>
  - disable coreid
```

## 6.3.3 Method 2: Enable SHARC Cores with corecontrol Utility in Linux

### Linux Core Control Driver

#### The corectrl Device

A new corectrl device has been created to allow Linux user to enable and disable the SHARC cores. See icc.h for macro values.

The device, /dev/corectrl, supports the following ioctl requests:



Request	Description	Format
CMD_CORE_START	Start the specified core running from the programmed SVECT address.	int ioctl(<FD>, CMD_CORE_START, <COREID>);
CMD_CORE_STOP	Stop the specified core by putting it back in reset.	int ioctl(<FD>, CMD_CORE_STOP, <COREID>);

<FD> - File descriptor of /dev/corectrl

<COREID> - The number of the core to start or stop. Values accepted are CCTRL\_CORE1 and CCTRL\_CORE2.

Both requests return 0 on success and -1 on failure.

## Usage Example

An example to enable both cores is as follows:

```
#include <icc.h>
...
int fd = open("/dev/corectrl", O_RDWR);
if (fd < 0) {
    perror("Unable to open /dev/corectrl");
    exit(1);
}
...
if (ioctl(fd, CMD_CORE_START, CCTRL_CORE1)) {
    perror("Unable to start Core 1");
    exit(2);
}
if (ioctl(fd, CMD_CORE_START, CCTRL_CORE2)) {
    perror("Unable to start Core 2");
    exit(3);
}
```

## Configuration

The SHARC cores can be controlled from the Linux command line using the corecontrol utility. To use corecontrol, first enable it in Buildroot:

```
[*] sc5xx corecontrol tool
Prompt: sc5xx corecontrol tool
```

```
Location:
-> Target packages
    -> Miscellaneous
```

With the corecontrol utility, sharc cores can be controlled to start or stop.

## Usage Example

```
# corecontrol -h
Valid options
--start CORE_NUM
--stop CORE_NUM
# corecontrol --start 1
# corecontrol --stop 1
```

## 6.4 Introduction of MCAPI examples

### 6.4.1 Introduction

This document gives a more detailed introduction of MCAPI examples.

This example involves two following parts:

- Baremetal: MCAPI Baremetal CCES projects run on SHARC 1 core
- Linux: Linux MCAPI demo examples run on ARM core

There are 3 MCAPI inter-operability demos included in the libmcapitests, **currently we only support msg demo:**

- msg (unconnected message protocol) – supported
- sclchan (connected scalar channel protocol) – unsupported
- pktchan (connected packet channel protocol) – unsupported

There are 2 ways to load the MCAPI baremetal project: loaded by ROM code and loaded by ICE1000 in CCES.

For more information about "loaded by ROM code", please refer to [Run the MCAPI demo](#)

For more information about "loaded by ICE1000 in CCES", please refer to [Run Linux on ARM and bare-metal application on SHARC](#)

## MCAPI Baremetal test examples

You can get the MCAPI Baremetal test examples code after installing the linux kit, and the examples code in "/opt/analog/cces-linux-add-in/1.2.0/examples/cces\_mcap\_i\_test/". You'll find three folders which are for msg, sclchan and pktchan, each folder contains the baremetal mcap\_i cces projects for sc589, sc584 and sc573 boards, "Mcap\_i\_Test\_Core1.c" is the main application c file. You should use CCES to open and build the project, since we only support msg test example, we'll take the MCAPI Baremetal MSG test as the example in the next section.

## MCAPI Linux demos

After buildroot has been compiled, source files of MCAPI-lib, demos can be found in "buildroot /output/build/libmcap\_i-HEAD/", "arm\_sharc\_msg\_demo.c" is the main application c file.

Once the linux boot finished, one command for MCAPI demo test can be found in linux, you are able to use "arm\_sharc\_msg\_demo -h" to get more information about the command:

```
# arm_sharc_msg_demo -h
Usage: arm_sharc_msg_demo <options>
Available options:
    -h, --help                this help
    -m, --mode                select the mode:
                                0 --- nonblocking mode0(default)
                                1 --- nonblocking mode1
                                2 --- nonblocking mode2
                                3 --- blocking mode
    -t, --timeout            timeout value in jiffies(default:5000)
    -r, --round              number of test round(default:100)
```

"arm\_sharc\_msg\_demo" shows the example use of blocking/nonblocking message send/receive between two different endpoints on different nodes, the endpoint of ARM sends a message then receives a message from another endpoint of CORE1 in one round, after comparing the receive data with the data you expect, the number of passed rounds will be increased, only when the passed rounds and the test rounds are equal, it'll give demo passed log.

Each mode means the different ways to do message transaction between two endpoints using different MCAPI APIs we supported([Multi-Core Communication](#)), and you can choose the way by "-m":

options	Description			
-m, --mode	mode 0	nonblocking send	mcap_i_test() / mcap_i_msg_available()	nonblocking recv

options	Description			
	mode 1	nonblocking send	mcapi_test()	nonblocking recv
	mode 2	nonblocking send	mcapi_wait()	nonblocking recv
	mode 3	blocking send		blocking recv
-t,-- timeout	Timeout value is used to set the maximum wait time for wait and blocking function			
-r,-- round	Round is the number of rounds you want to test, <b>since the total number of check round in MCAPI baremetal test example is 100, we should also make sure that the total number of check round in linux is less than or equal to 100, if number equals 100, passed log will be output in CCES.</b>			

## 6.4.2 Performance of MCAPI

The test method of the performance data of MCAPI is to measure the time required of doing message transactions about 1000 rounds between 2 endpoints on ARM and SHARC, and one round includes sending and receiving one message for each endpoint, that's what the MCAPI demo example does, so we'll measure the interval time by running the MCAPI demo example. For more information about the message transaction, please refer to [Multi-Core Communication](#) .

### Dependencies

There are many influences in measuring the MCAPI performance:

- Different platform We'll test on all sc5xx boards(SC589, SC584, SC573)
- System load We assume that the system load should be normal, and can be ingored, we disable all logs of MCAPI(both Linux and Baremetal)
- MCAPI operation mode

### Result

Below table is the interval time of 1000 rounds message transaction under different conditions:

Board	Mode 0	Mode 1	Mode 2	Mode3
sc589-ezkit	190.98ms	186.48ms	174.34ms	150.49ms
sc584-ezkit	188.72ms	180.95ms	176.29ms	145.65ms

Board	Mode 0	Mode 1	Mode 2	Mode3
sc573-ezkit	191.28ms	184.99ms	178.85ms	147.92ms

## 6.5 Run Linux on ARM and bare-metal application on SHARC

### 6.5.1 Introduction

This document introduces steps to run Linux on ARM core and SHARC baremetal application on SHARC cores(Core 1 & 2) by CCES. Take the Linux MCAPI inter-operability demo example as a SHARC baremetal application.

### 6.5.2 MCAPI Test Example

You can get the MCAPI Baremetal test examples code after installing the linux kit, and the examples code in "/opt/analog/cces-linux-add-in/1.2.0/examples/cces\_mcapi\_test/". You'll find three folders which are for msg, sclchan and pktchan, each folder contains the baremetal mcapi cces projects for sc589, sc584 and sc573 boards, "Mcapi\_Test\_Core1.c" is the main application c file. You should use CCES to open and build the project, since we only support msg test example, we'll take the MCAPI Baremetal MSG test as the example.

### Hardware Requirement

- ADSP-SC589 EZ-Kit v1.0 and above, or
- ADSP-SC584 EZ-Kit v1.0 and above, or
- ADSP-SC573 EZ-Kit v1.0 and above
- ICE1000 or ICE2000 JTAG board

Here we take ADSP-SC589 EZ-kit as our example.

### Software Configuration

This section shows how to config Linux to prepare for this test.

### Configure Linux kernel

MCAPI lib is built on top of the ICC (Inter-Core Communications) device driver. You should run **make linux-menuconfig** in Linux kernel to enable the ICC driver in Linux kernel first:



```
Device Drivers --->
  [*] Staging drivers --->
    [*]   icc driver --->
        --- icc driver
          [*]   icc protocol
          [ ]   icc debug
```

## Configure Buildroot

Then you need to run **make menuconfig** in buildroot to enable libmcap and corecontrol in Buildroot:

```
Target packages --->
  Libraries --->
    Other --->
      [*] libmcap
```

Refer to [Enable and Disable SHARC Cores](#), you can enable SHARC cores with ICC command in u-boot or with corecontrol utility in Linux. Here we use corecontrol tool in Linux to enable and disable SHARC cores.

```
Target packages --->
  Miscellaneous --->
    [*] sc5xx corecontrol tool
```

## Build

Finally, build the buildroot to generate the linux kernel image.

```
$ make
```

## Load Linux on ARM and SHARC applications on SHARC by CCES

A brief step to run multicore on the EZ-Kit board is showing as follows:

1. Boot u-boot to console and stop at u-boot;

2. Start CCES to load application to SHARC cores and wait;
  - a. Build baremetal example in CCES;
  - b. Remove the program of Device 0[Core 0];
  - c. Uncheck automatically setted breakpoints and disable semihosting in "Automatic Breakpoints" view of debug configuration;
  - d. Uncheck " Halt core after connecting to target" for ARM core in debug configuration and start debug;
  - e. Click the debug button, it will load the dxr file then waiting for linux to start Core 1 & 2;
3. Enable SHARC cores in Linux
4. Resume Core 1 & 2 Application running in CCES

### Step1: Boot u-boot to Console

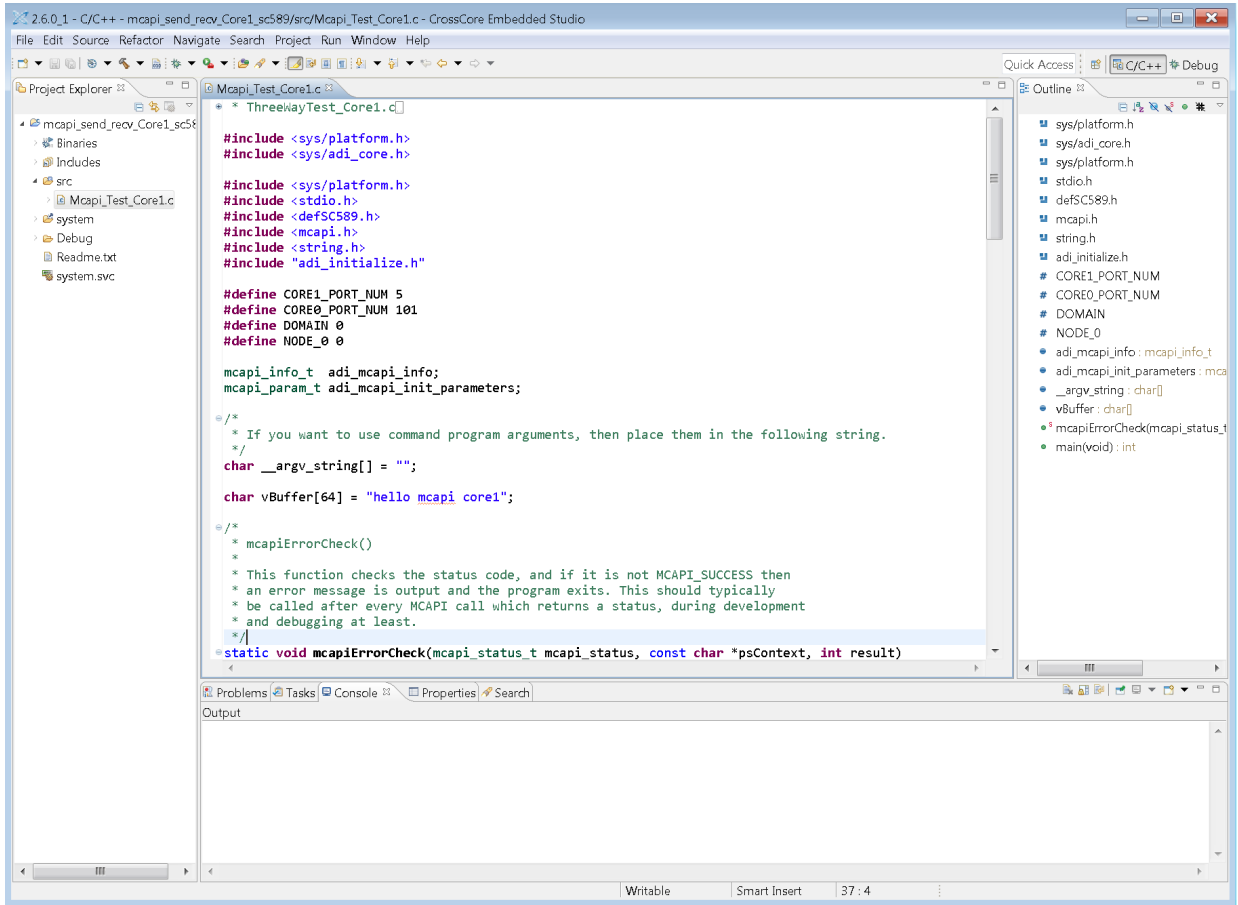
```
U-Boot 2015.01 ADI-1.2.0-00105-g33e00dd-dirty (Jul 21 2017 - 15:38:12)
CPU: ADSP ADSP-SC589-0.1 (Detected Rev: 1.1) (spi flash boot)
VCO: 450 MHz, Cclk0: 450 MHz, Sclk0: 112.500 MHz, Sclk1: 112.500 MHz,
DCLK: 450 MHz
OCLK: 150 MHz
I2C: ready
DRAM: 224 MiB
MMC: SC5XX SDH: 0
SF: Detected W25Q128BV with page size 256 Bytes, erase size 4 KiB,
total 16 MiB
In: serial
Out: serial
Err: serial
other init
Net: dwmac.3100c000
Hit any key to stop autoboot: 0
sc #
```

### Step2: CCES Debug Configuration

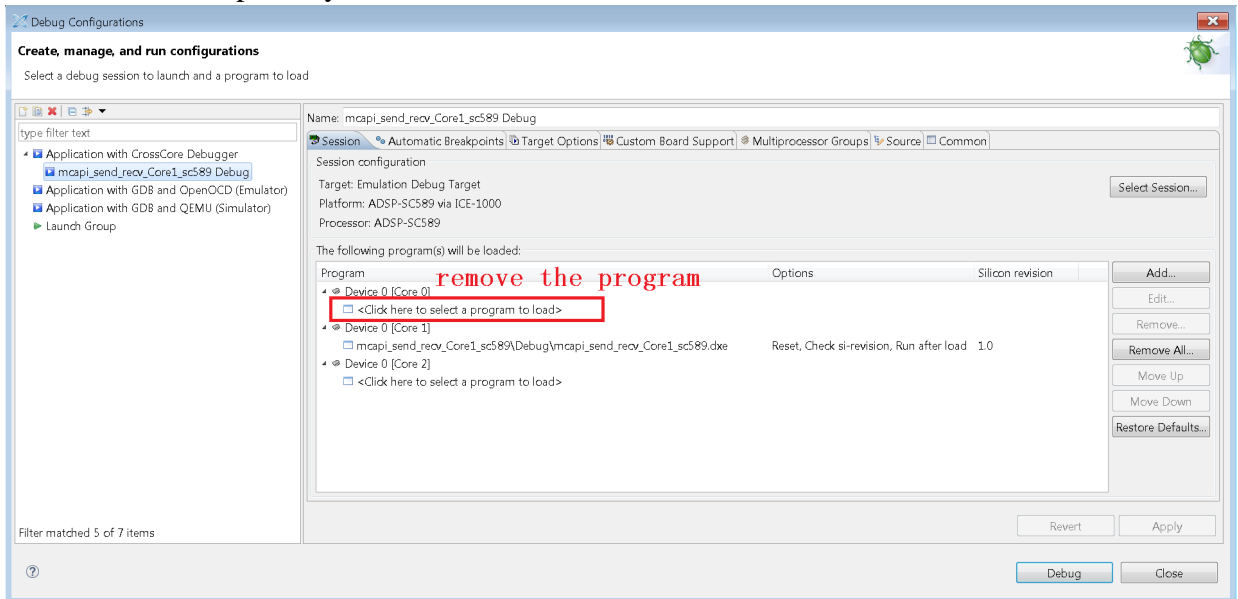
Before running the application we need to first config CCES debug configuration.

1. Connect ICE-1000/2000 JTAG emulator between the SC589-EZKIT and your PC

2. Start CCES (2.6.0 or later) and open the application project, build it and enter into the debug configuration.

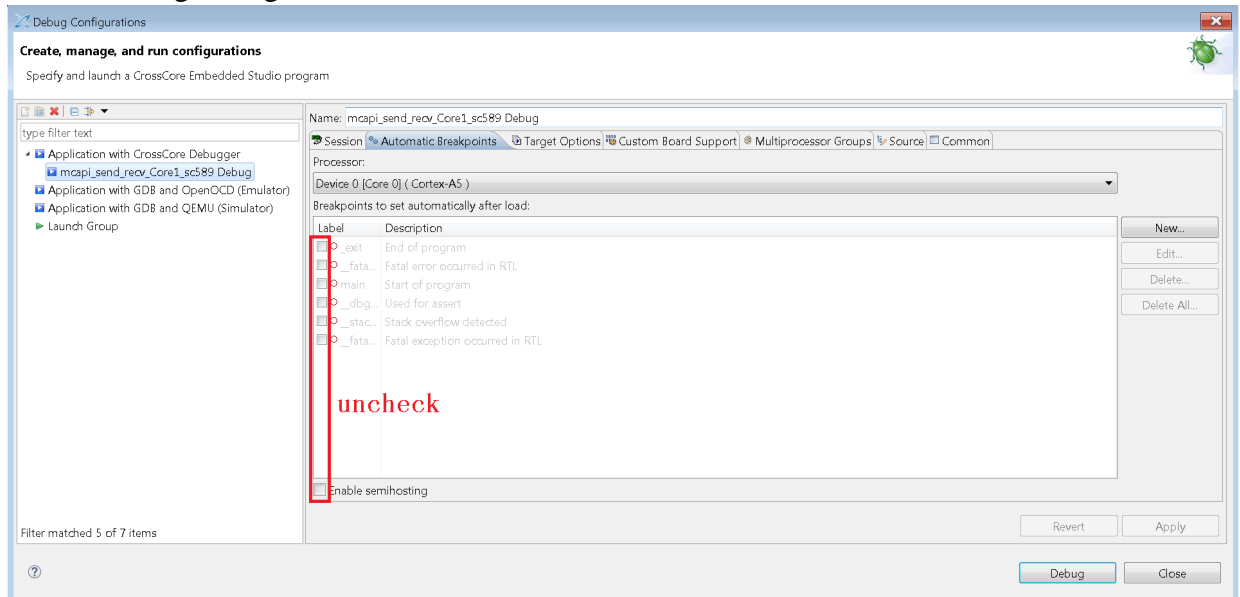


3. Remove the ARM core project from the debug configuration, only load core1 and core2, MCAPI test example only need to load core 1.

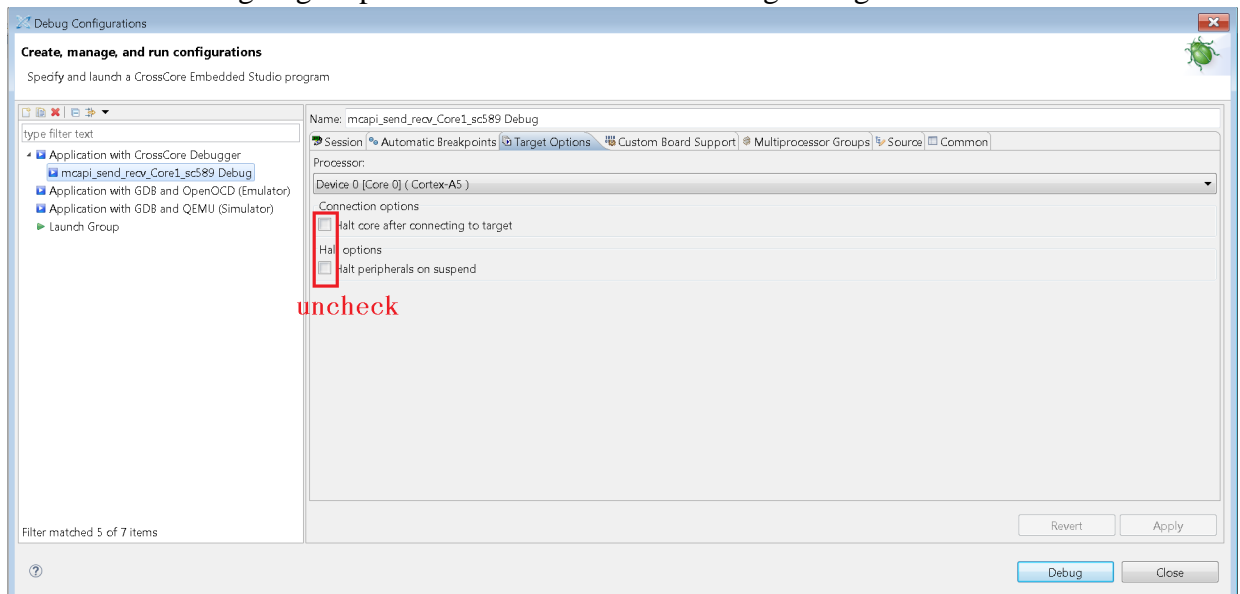




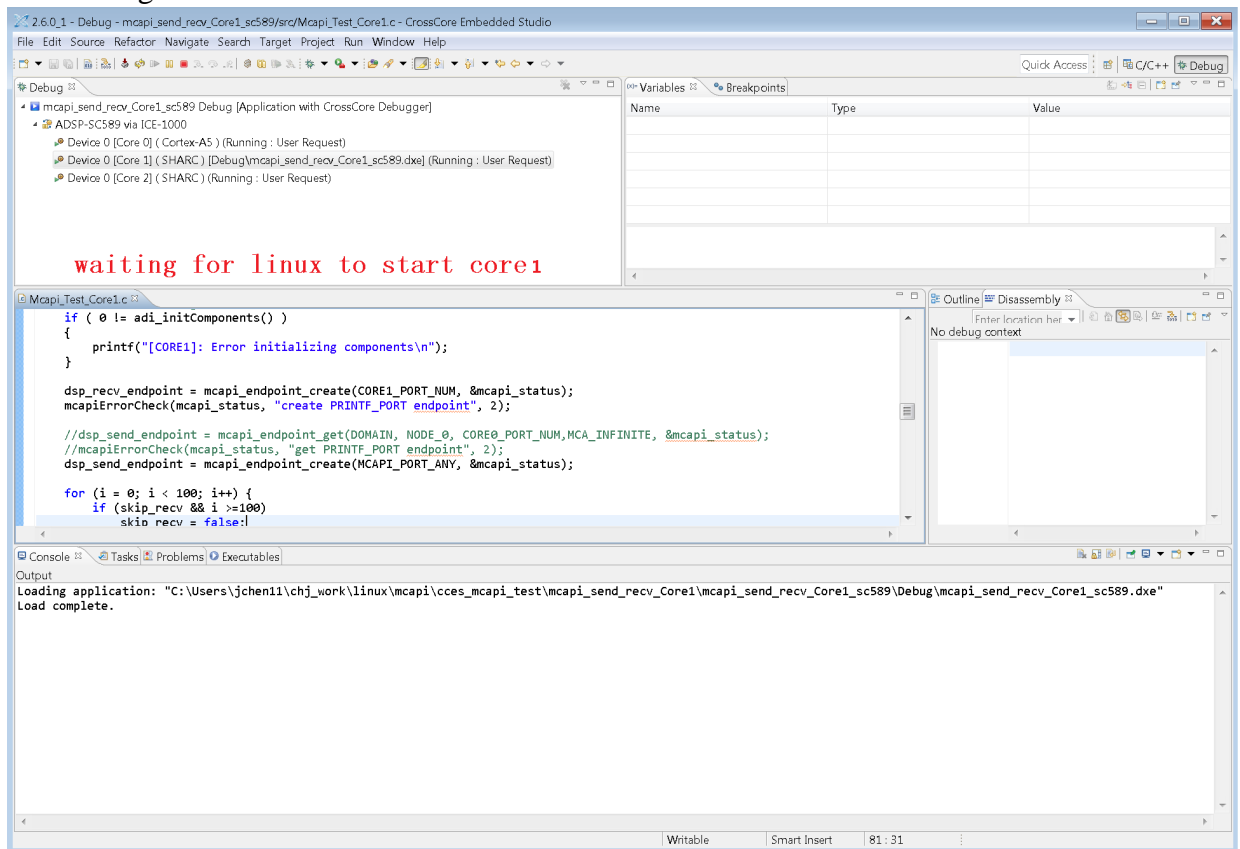
4. Uncheck automatically setted breakpoints and disable semihosting in "Automatic Breakpoints" view of debug configuration.



5. Uncheck the debug target option "Halt core after connecting to target" for ARM core.



## 6. Start debug and wait for Linux to start core1.



### Step3: Enable SHARC cores in Linux

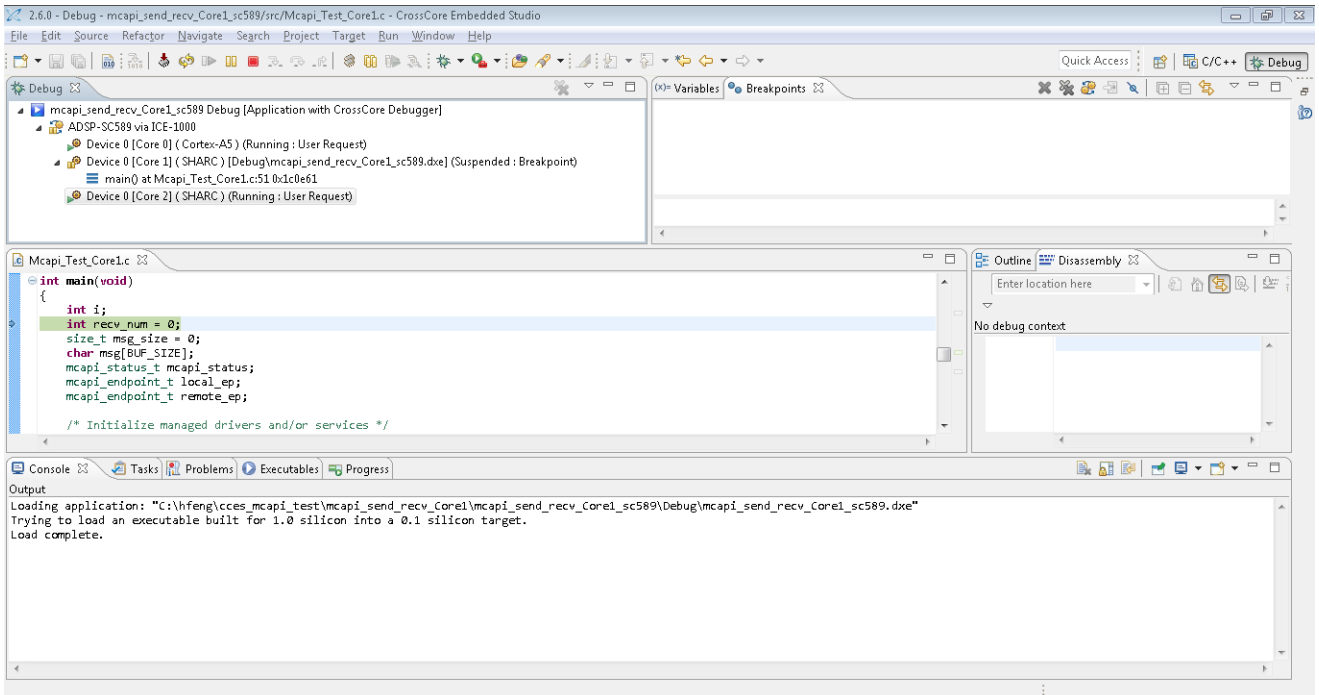
In u-boot, enable SHARC core then boot linux

```
sc # icc enable 1
```

Or boot Linux, and then in Linux use the corecontrol utility to start the SHARC core:

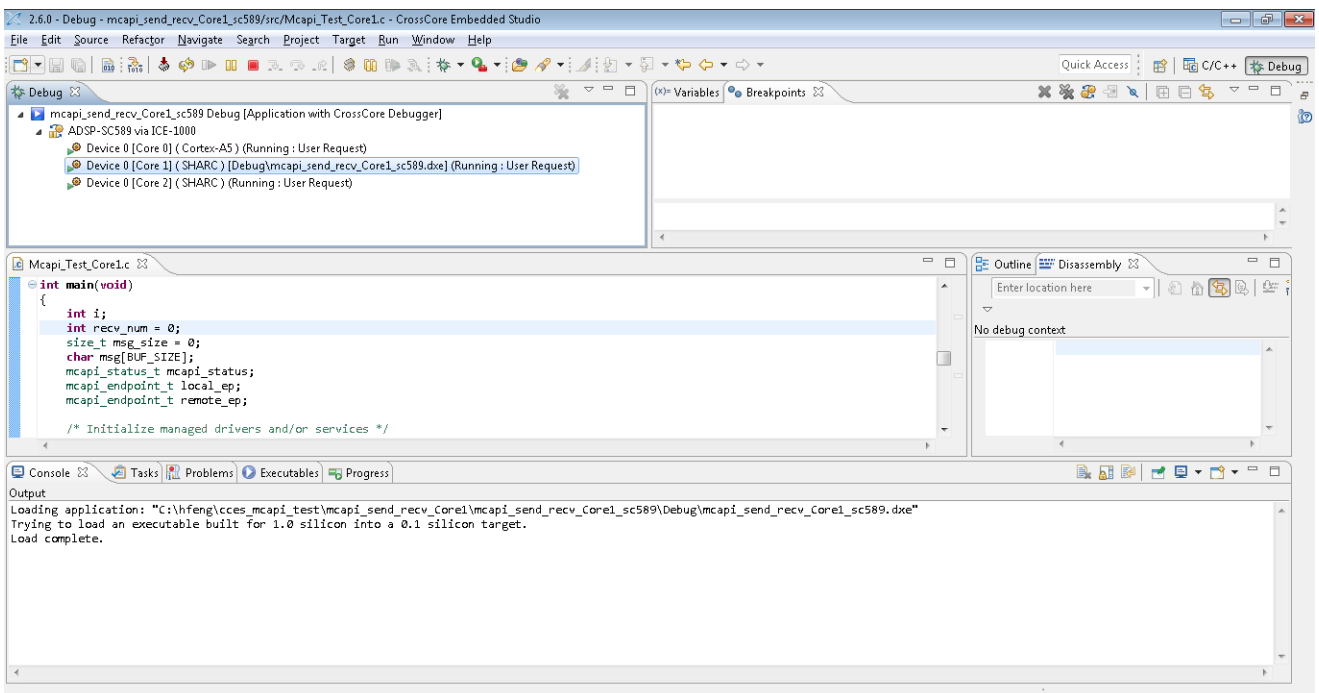
```
# corecontrol --start 1
```

After running SHARC core, CCES halts in the first line of the application code on SHARC core 1.



## Step4: Resume Core 1 & 2 Application running in CCES

Resume(F5) core 1 and continue running the application in CCES.



Now Linux is running on ARM core 0 while SHARC baremetal application is running on core 1.

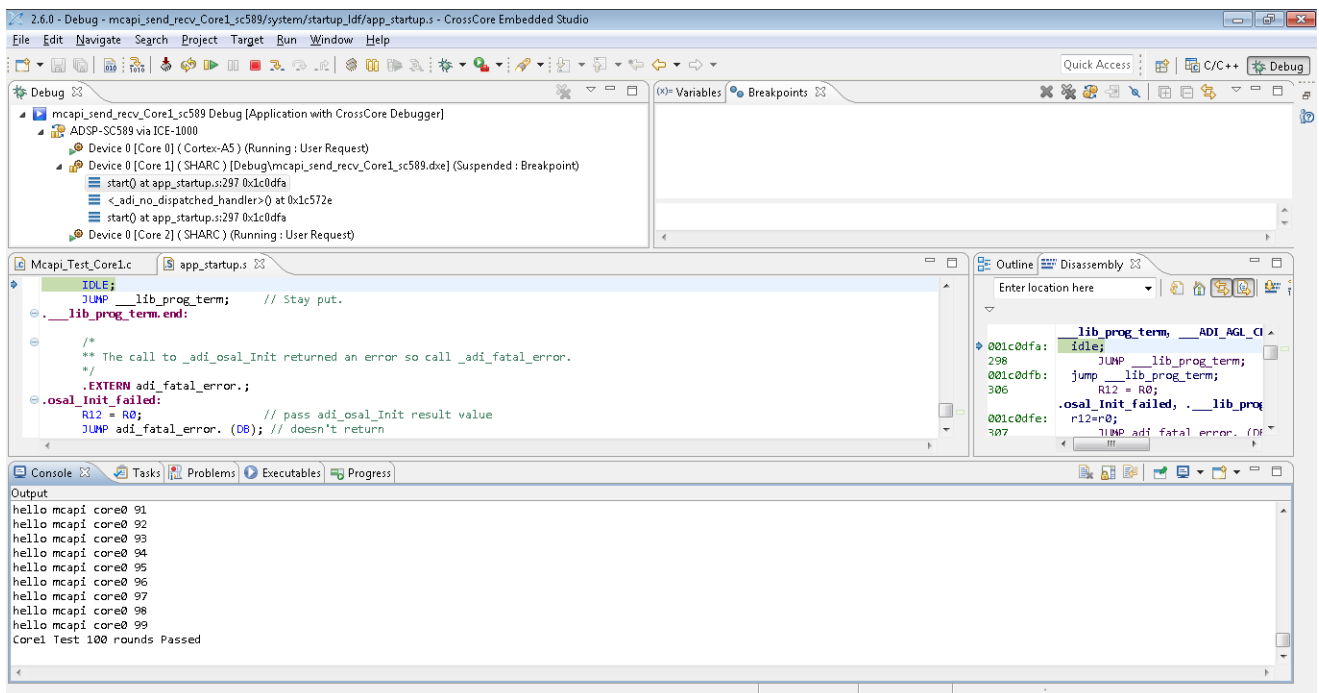
## Run Linux MCAPI MSG Demo Test

Run "arm\_sharc\_msg\_demo" command in Linux and the passed log in linux is showing as the picture.

```
# arm_sharc_msg_demo
```

```
CoreA: mode(0) message send. The 96 time sending
recv() start.....
session_idx 0
node=1, port=31
CHECK_STATUS---recv_i: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of recv() - endpoint=101 size 0x9ca8 has received: [hello mcapi core1]
CoreA: mode(0) message recv. The 96 time receiving
send() start.....
CHECK_STATUS---send_i: MCAPI_PENDING
CHECK_STATUS---test: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of send() - endpoint=101 has sent: [HELLO_MCAPI core0 97]
CoreA: mode(0) message send. The 97 time sending
recv() start.....
session_idx 0
node=1, port=31
CHECK_STATUS---recv_i: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of recv() - endpoint=101 size 0x9ca8 has received: [hello mcapi core1]
CoreA: mode(0) message recv. The 97 time receiving
send() start.....
CHECK_STATUS---send_i: MCAPI_PENDING
CHECK_STATUS---test: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of send() - endpoint=101 has sent: [HELLO_MCAPI core0 98]
CoreA: mode(0) message send. The 98 time sending
recv() start.....
session_idx 0
node=1, port=31
CHECK_STATUS---recv_i: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of recv() - endpoint=101 size 0x9ca8 has received: [hello mcapi core1]
CoreA: mode(0) message recv. The 98 time receiving
send() start.....
CHECK_STATUS---send_i: MCAPI_PENDING
CHECK_STATUS---test: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of send() - endpoint=101 has sent: [HELLO_MCAPI core0 99]
CoreA: mode(0) message send. The 99 time sending
recv() start.....
session_idx 0
node=1, port=31
CHECK_STATUS---recv_i: MCAPI_SUCCESS
CHECK_STATUS---wait: MCAPI_SUCCESS
end of recv() - endpoint=101 size 0x9ca8 has received: [hello mcapi core1]
CoreA: mode(0) message recv. The 99 time receiving
CHECK_STATUS---del_ep: MCAPI_SUCCESS
mcapi_finalize 322
CHECK_STATUS---finalize: MCAPI_SUCCESS
CoreA 100 rounds mode(0) demo Test PASSEDI!
```

The output on CCES console:



More details about the introduction of MCAPI examples, please refer to [Introduction of MCAPI examples](#).



Reset the board then follow the above steps if you want to restart the demo test. Relaunching or restarting the MCAPI application in CCES without resetting the board is not recommended.

## 6.6 Reserve Pinmux Functions in Linux for SHARC Applications

### 6.6.1 Introduction

For ADSP-SC573, SC584 and SC589, most of on-chip peripherals are shared between the ARM core and the SHARC cores. When a peripheral is used in a SHARC application it can't be used by Linux on the ARM core. This can be ensured by compiling configuration statically in both the SHARC application and the Linux kernel. However, different peripherals used on different cores may still share some pins, which are configured through the pinmux controller. In order to avoid Linux drivers altering the pinmux functions set in SHARC applications by mistake, the Linux ICC driver allows developers to reserve the pinmux functions for peripherals used by the SHARC application. Take ADSP-SC589 as example in following sections.

### 6.6.2 Linux Kernel Configuration

#### Set Up ICC Device Tree Node

In order to reserve the pinmux function, it should be defined as a pinctrl configuration node first in the device tree file.

```
pinctrl0: pinctrl@0 {
    compatible = "adi,adi2-pinctrl";
    #address-cells = <1>;
    #size-cells = <1>;
    #interrupt-cells = <2>;
    interrupt-controller;
    reg = <0 0>;
    icc {
        icc_default: icc0@0 {
            adi,group = "ppi0_16bgrp", "lp0grp";
            adi,function = "ppi0", "lp0";
        }
    }
}
```



```

ADI_PIN_GROUP("ppi0_24bgrp", ppi0_24b_pins, ppi0_24b_mux),
ADI_PIN_GROUP("mmc0grp", mmc0_pins, mmc0_mux),
};

static const struct adi_pmx_func adi_pmx_functions[] = {
    ADI_PMX_FUNCTION("uart0", uart0grp),
    ADI_PMX_FUNCTION("uart1", uart1grp),
    ADI_PMX_FUNCTION("uart2", uart2grp),
    ADI_PMX_FUNCTION("spi0", spi0grp),
    ADI_PMX_FUNCTION("spi1", spi1grp),
    ADI_PMX_FUNCTION("spi2", spi2grp),
    ADI_PMX_FUNCTION("can0", can0grp),
    ADI_PMX_FUNCTION("can1", can1grp),
    ADI_PMX_FUNCTION("smc0", smc0grp),
    ADI_PMX_FUNCTION("lp0", lp0grp),
    ADI_PMX_FUNCTION("lp1", lp1grp),
    ADI_PMX_FUNCTION("eth0", eth0grp),
    ADI_PMX_FUNCTION("eth1", eth1grp),
    ADI_PMX_FUNCTION("mmc0", mmc0grp),
    ADI_PMX_FUNCTION("ppi0", ppi0grp),
};

```

### 6.6.3 Pinmux Reservation Example

After Linux boots up with the new device tree binary file, you can find the specified PPI0 and LP0 pinmux functions reservation information from the sysfs.

```

# ls /dev/icc
/dev/icc
#
# cd /sys/kernel/debug/pinctrl/pinctrl-adi2.0/
# cat
gpio-ranges      pinmux-functions pins
pingroups       pinmux-pins
# cat pinmux-pins
Pinmux settings per pin
Format: pin (name): mux_owner gpio_owner hog?
pin 0 (PA0): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 1 (PA1): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 2 (PA2): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 3 (PG3): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 4 (PA4): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp

```

```

pin 5 (PA5): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 6 (PA6): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 7 (PA7): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 8 (PA8): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 9 (PA9): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 10 (PA10): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 11 (PA11): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 12 (PA12): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 13 (PA13): 3100c000.ethernet (GPIO UNCLAIMED) function eth0 group
eth0grp
pin 14 (PA14): (MUX UNCLAIMED) adi-gpio:14
pin 15 (PA15): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 16 (PB0): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 17 (PB1): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 18 (PB2): (MUX UNCLAIMED) adi-gpio:18
pin 19 (PB3): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 20 (PB4): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 21 (PB5): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 22 (PB6): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 23 (PB7): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 24 (PB8): (MUX UNCLAIMED) adi-gpio:24
pin 25 (PB9): 31000a00.can (GPIO UNCLAIMED) function can1 group can1grp
pin 26 (PB10): 31000a00.can (GPIO UNCLAIMED) function can1 group
can1grp
pin 27 (PB11): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 28 (PB12): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 29 (PB13): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 30 (PB14): (MUX UNCLAIMED) adi-gpio:30
pin 31 (PB15): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 32 (PC0): (MUX UNCLAIMED) adi-gpio:32
pin 33 (PC1): 31044000.spi (GPIO UNCLAIMED) function spi2 group spi2grp
pin 34 (PC2): 31044000.spi (GPIO UNCLAIMED) function spi2 group spi2grp
pin 35 (PC3): 31044000.spi (GPIO UNCLAIMED) function spi2 group spi2grp
pin 36 (PC4): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 37 (PC5): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 38 (PC6): (MUX UNCLAIMED) adi-gpio:38
pin 39 (PC7): 31000200.can (GPIO UNCLAIMED) function can0 group can0grp
pin 40 (PC8): 31000200.can (GPIO UNCLAIMED) function can0 group can0grp
pin 41 (PC9): 31042000.spi (GPIO UNCLAIMED) function spi0 group spi0grp

```



```

pin 42 (PC10): 31042000.spi (GPIO UNCLAIMED) function spi0 group
spi0grp
pin 43 (PC11): 31042000.spi (GPIO UNCLAIMED) function spi0 group
spi0grp
pin 44 (PC12): (MUX UNCLAIMED) adi-gpio:44
pin 45 (PC13): adi-uart4.0 (GPIO UNCLAIMED) function uart0 group
uart0grp
pin 46 (PC14): adi-uart4.0 (GPIO UNCLAIMED) function uart0 group
uart0grp
pin 47 (PC15): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 48 (PD0): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 49 (PD1): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 50 (PD2): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 51 (PD3): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 52 (PD4): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 53 (PD5): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 54 (PD6): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 55 (PD7): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 56 (PD8): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 57 (PD9): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 58 (PD10): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 59 (PD11): 20080000.icc (GPIO UNCLAIMED) function lp0 group lp0grp
pin 60 (PD12): adi-uart4.2 (GPIO UNCLAIMED) function uart2 group
uart2grp
pin 61 (PD13): adi-uart4.2 (GPIO UNCLAIMED) function uart2 group
uart2grp
pin 62 (PD14): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 63 (PD15): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 64 (PE0): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 65 (PE1): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 66 (PE2): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 67 (PE3): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 68 (PE4): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 69 (PE5): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 70 (PE6): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 71 (PE7): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 72 (PE8): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 73 (PE9): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp

```

```
pin 74 (PE10): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 75 (PE11): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 76 (PE12): 20080000.icc (GPIO UNCLAIMED) function ppi0 group
ppi0_16bgrp
pin 77 (PE13): 31043000.spi (GPIO UNCLAIMED) function spil group
spilgrp
pin 78 (PE14): 31043000.spi (GPIO UNCLAIMED) function spil group
spilgrp
pin 79 (PE15): 31043000.spi (GPIO UNCLAIMED) function spil group
spilgrp
pin 80 (PF0): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 81 (PF1): (MUX UNCLAIMED) (GPIO UNCLAIMED)
pin 82 (PF2): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 83 (PF3): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 84 (PF4): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 85 (PF5): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 86 (PF6): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 87 (PF7): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 88 (PF8): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 89 (PF9): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 90 (PF10): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 91 (PF11): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 92 (PF12): mmc.0 (GPIO UNCLAIMED) function mmc0 group mmc0grp
pin 93 (PF13): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
pin 94 (PF14): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
pin 95 (PF15): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
pin 96 (PG0): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
pin 97 (PG1): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
pin 98 (PG2): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
pin 99 (PG3): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
pin 100 (PG4): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
pin 101 (PG5): 3100e000.ethernet (GPIO UNCLAIMED) function eth1 group
eth1grp
```

## 6.7 SEC driver and multicore development

### 6.7.1 Introduction

There are two Interrupt Controllers on the ADSP-SC573, SC584 and SC589. The GIC (Generic Interrupt Controller) for the ARM core and the SEC (System Event Controller) for the SHARC cores. The SEC manages the configuration of all system event sources, and also manages the propagation of system events to all connected SHARC cores and the system fault interface.

Linux, running on the ARM core, manages all of its interrupts via the GIC. However, if any Linux driver wants to trigger an interrupt to the SHARC cores directly it should raise an SSI source event in the SEC. In addition, the watchdog device can only reset the cores via the system fault interface. The Linux watchdog driver has to set up its event route in the SEC as well. All of these interactions with the SSI depend on proper initialization of the SEC controller. Because, the SEC controller can be initialized by code on either the ARM core or the SHARC cores, there must be a way to avoid conflicting initialization on different cores. This guide apply to both ADSP-SC573, SC584 and SC589. Take ADSP-SC589 as example in following sections.

### 6.7.2 Linux kernel Configuration

The SHARC slave core number in the ADSP-SC58x serial processors differs among models. So, the slave core count should be set up accordingly in the Linux kernel. The Linux SEC driver initializes the proper SCIs in the SEC controller according to this setting. However, the SEC driver is built into the Linux kernel no matter how many slave cores are configured.

```
System Type --->
  sc58x platform type --->
    (2) Slave core count in sc58x
```

The valid slave core count is 0, 1 or 2.

When the Linux kernel is booting up, following message is printed out.

```
sec init...
enabled
```

### 6.7.3 Coordinate the SEC Initialization for multicore Development

If both Linux SEC driver and the SHARC SEC code initialize the SEC controller independently, the SEC may be in an unknown and wrong state. In multicore development, if the SHARC applications start executing earlier than Linux, the Linux SEC driver should skip the SEC initialization. Meanwhile, if the SHARC applications start executing after Linux boots up, the SHARC SEC driver should not initialize the controller.

#### Allow Linux to skip the SEC initialization

A Linux kernel early parameter "**enable\_sec**" is defined in the SEC driver to get the instruction from user. This parameter should be provided in the kernel boot-up command line either via the U-Boot environment or via the device tree property.

- **In u-boot console**

```
sc# set ramargs bootargs=root=/dev/mtdblock0 rw console=ttySC0,
115200n8 earlyprintk enable_sec=no
```

- **In Linux device tree file**

```
chosen {
    bootargs = "/dev/mtdblock0 rw console=ttySC0,115200n8 earlyprintk
enable_sec=no";
};
```

With parameter **enable\_sec=no** set, the Linux kernel prints out the following message during boot.

```
sec init...
skipped
```

#### Allow SHARC SEC Code to Skip the SEC Reset

By default, the sc5xx SEC library for SHARC cores always resets the SEC registers in function **adi\_sec\_Reset()** and **adi\_sec\_ResetSFI()**. To skip resetting the SEC registers on SHARC core, the API **ADI\_SEC\_RESULT adi\_sec\_Init(void)** should be revised to accept a skip flag.

- **file:SHARC\lib\src\services\Source\int\adi\_int.c**

```

ADI_SEC_RESULT adi_sec_Init(bool skip_reset) {
    ...
        if (!skip_reset)
            adi_sec_Reset();
        /* Initialize SEC data structure */
        gSECData.pfHandler = &DefaultErrorHandler;
        gSECData.pCBParam = NULL;
        /* Register the SEC error handler */
        if(adi_int_InstallHandler((uint32_t)ADI_SEC_ERR_INTR,
                                adi_SEC_ErrorHandler,
                                NULL, true) !=
ADI_INT_SUCCESS)
        {
            return ADI_SEC_REGISTRATION_FAILED;
        }
        /* Reset fault interface */
        if (!skip_reset)
            adi_sec_ResetSFI();
    ...
}

```

The API `adi_initComponents(void)` should be revised to support no SEC reset as well.

- `file:system\adi_initialize.c`

```

int32_t _adi_initComponents(bool skip_reset)
{
    int32_t result = 0;
    result = adi_sec_Init(skip_reset);

    if (result == 0) {
        result = adi_initpinmux(); /* auto-generated code (order:0) */
    }
    if (result == 0) {
        result = adi_SRU_Init(); /* auto-generated code (order:0) */
    }
    if (result == 0) {
        result = adi_mcap_i_Init(); /* auto-generated code (order:6) */
    }
    return result;
}

int32_t adi_initComponents(void)
{
    _adi_initComponents(false);
}

int32_t adi_initComponents_no_sec_reset(void)
{
    _adi_initComponents(true);
}

```

```
}
```

## 7 Generic Linux Documents

### 7.1 Generic U-Boot Documents

- [U-Boot Frequently Asked Questions](#)
- [U-Boot User Manual](#)
- [Subscribe U-Boot Development Mailing List](#)

### 7.2 Generic Linux Kernel Documents

- [Linux Kernel Device Tree Specification](#)
- [Understanding the Linux Kernel, 3rd Edition](#)
- [Linux Device Drivers, 3rd Edition](#)
- [Subscribe To Development Mailing Lists for Linux Kernel and Driver Frameworks](#)

### 7.3 Generic Buildroot Documents

- [Buildroot User Manual](#)
- [Subscribe to Buildroot Development Mailing Lists](#)

# 8 Developing Linux Applications Using CrossCore Embedded Studio

## 8.1 Create and build a project using the Linux targeting toolchain

### 8.1.1 Install CCES and Linux Add-in on Linux host

Firstly we need get CCES and Linux Add-in latest deb package ready and install the debian kits on Linux machine with Ubuntu 14.04 distribution(32 bit).

a. To install Linux Add-in for CCES, run the following command:

```
sudo dpkg -i adi-LinuxAddinForCCES-linux-x86-1.2.0.deb
```

b. If you need reinstall Linux Add-in for CCES later, please remove it completely with the following command before starting installation:

```
sudo dpkg -P adi-LinuxAddinForCCES-linux-x86-1.2.0.deb
```

### 8.1.2 Start CCES IDE on Linux Host

```
// start CCES IDE  
cd /opt/analog/cces/2.6.0/Eclipse  
./cces
```

There would be "Workspace Launcher", use the default directory(e.g /home/test/cces/2.6.0), click OK. Here comes the Welcome page!

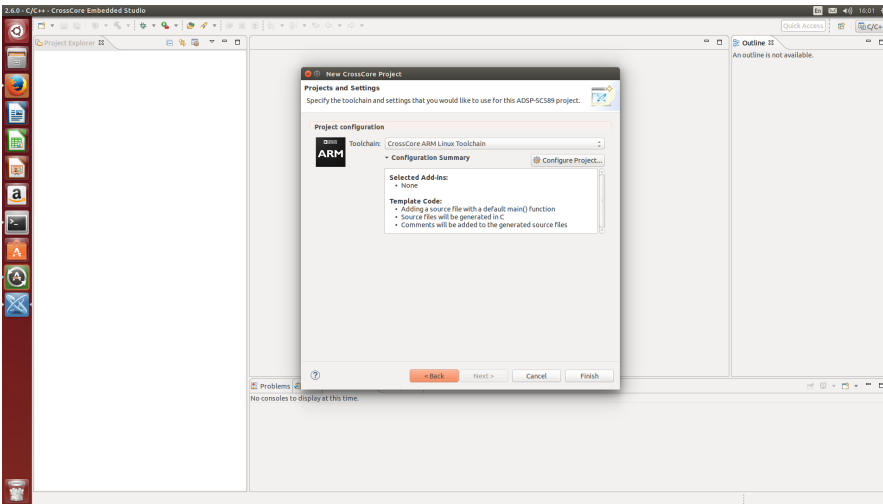
Move the mouse to the top of the screen, there shows up the menu bar of CCES IDE.

### 8.1.3 Creating a new project using CCES

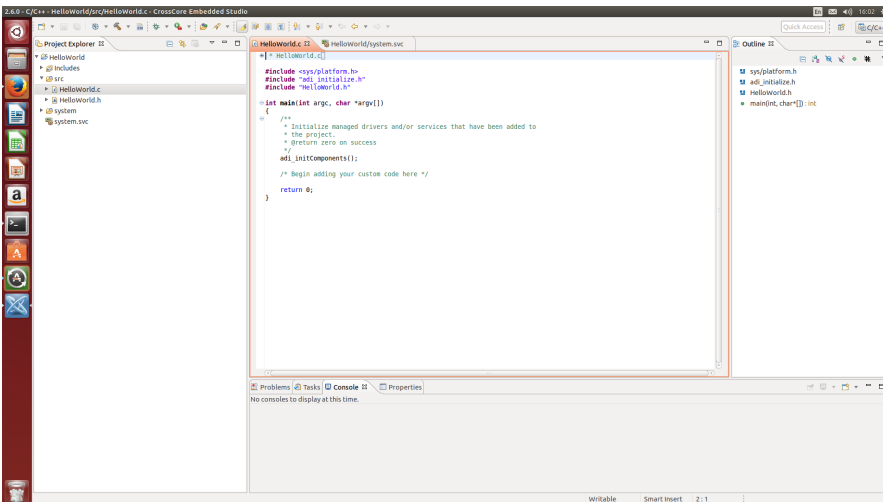
To create a new project:



1. Select **File | New | CrossCore Project** from the main menu.
2. Give your project a name (e.g. HelloWorld). Click Next.
3. Choose your Processor family(e.g SHARC), Processor type(e.g ADSP-SC589) and Silicon revision(e.g 1.0). Click Next.
4. Choose the "CrossCore ARM Linux Toolchain" from the Toolchain drop-down list and click Finish.



Here we see our "HelloWorld" project generated!



In the left-handed window, there lists various component of this projects, like include paths, source files, etc, unfold the Project Name "HelloWorld", you can see all of them.

You can also add your own lines in src/HelloWorld.c to realize desired function.

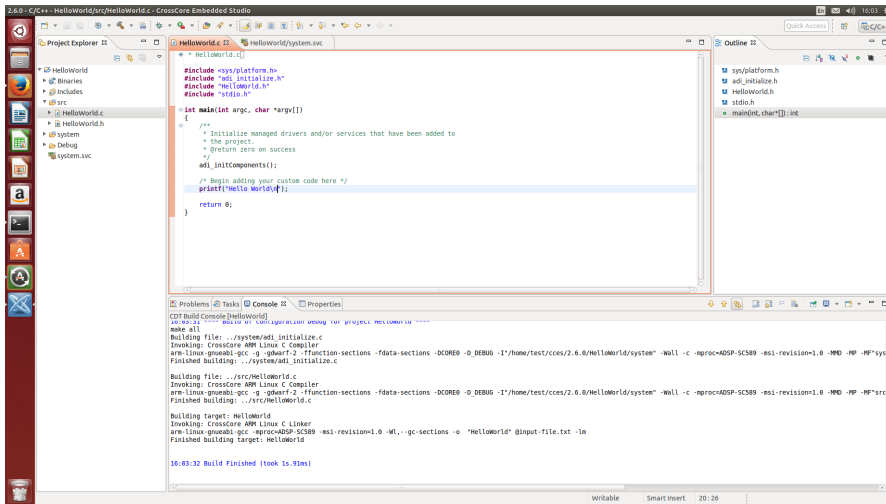
In our example, we write a simple printf command for demonstration:



```
printf("Hello World\n");
```

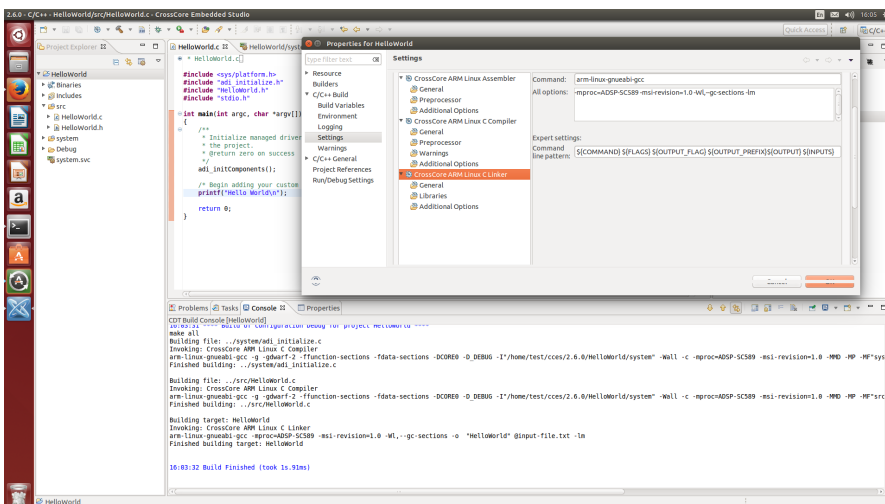
To build your new project:

1. Select **Project | Build All** from the main menu.
2. or Click the hammer icon on the toolbar.



We can see that this building process is carried out by ARM Linux toolchain (arm-linux-gnueabi-gcc) combined with several other params, dynamically linked with -lm.

Surely we can change the setting by right clicking at the HelloWorld project and selecting Properties, the following window comes out.



After modification, build the project again. Finally the building process says: "Build Finished (took xxx ms)".

The generated HelloWorld binary is put under your workspace(e.g /home/test/cces/2.6.0)/HelloWorld /Debug/HelloWorld

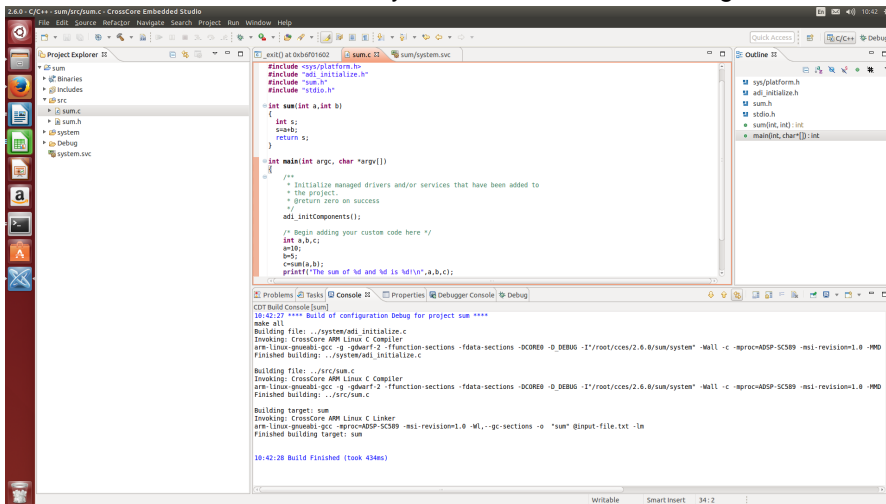


## 8.2 Debug a Linux application from within the CCES IDE

### 1. Create a new Linux application from within the CCES IDE

After you create a new Linux application (e.g sum) from within the CCES IDE by following the previous page:

Create and build a project using the Linux targeting toolchain, we can now start to debug the application if we want to know how the binary executes or root cause bugs.



### 2. Copy your newly built application to your Linux target and set-up debugging

To debug your newly built application, you need to copy the binary from your CCES workspace to your Linux target.

a. scp (Secure Copy) you sum application to your Linux target.

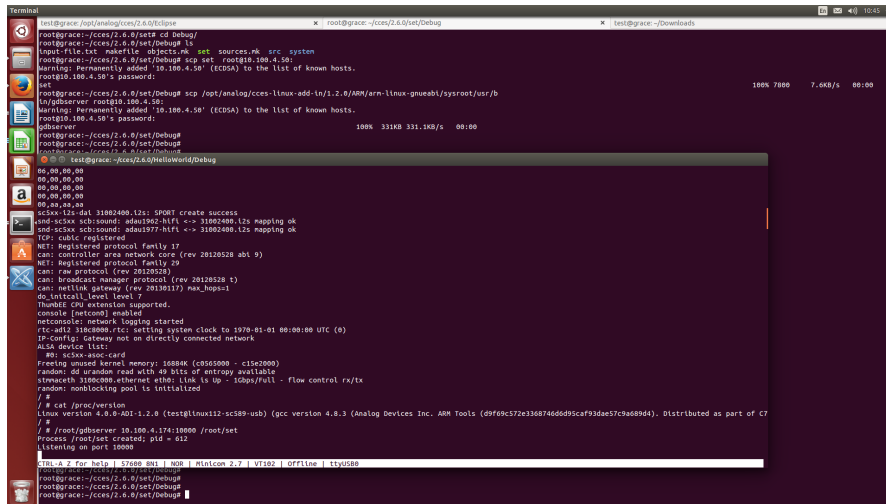
i. `scp ~/cces/2.6.0/sum/Debug/sum root@10.100.4.50:/`

b. If you did not opt to build your Linux Image with gdbserver included, then you need copy it from the Linux Add-in installation directory (`/opt/analog/cces-linux-add-in/1.2.0/ARM/arm-linux-gnueabi/sysroot/usr/bin`) to your Linux target.

i. `scp /opt/analog/cces-linux-add-in/1.2.0/ARM/arm-linux-gnueabi/sysroc`

c. In order to debug using CCES, you should run gdbserver on Linux target and listen on an IP and port.

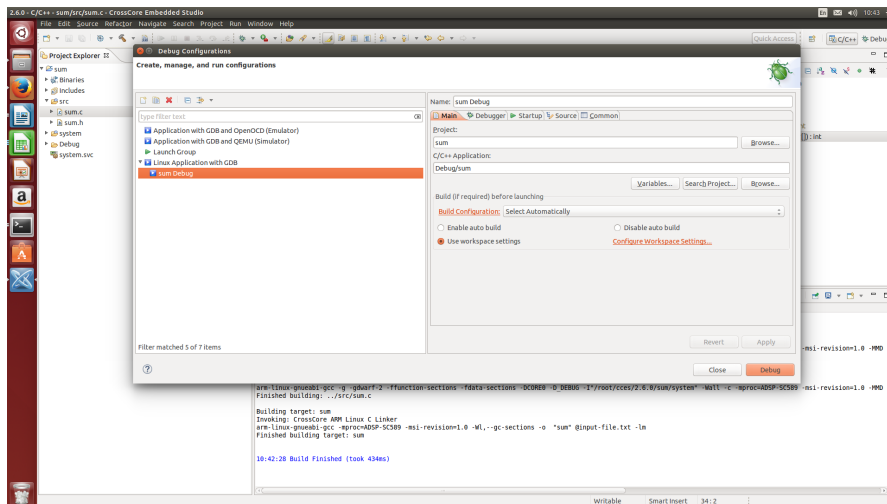
i. `gdbserver 10.100.4.174:10000 sum`



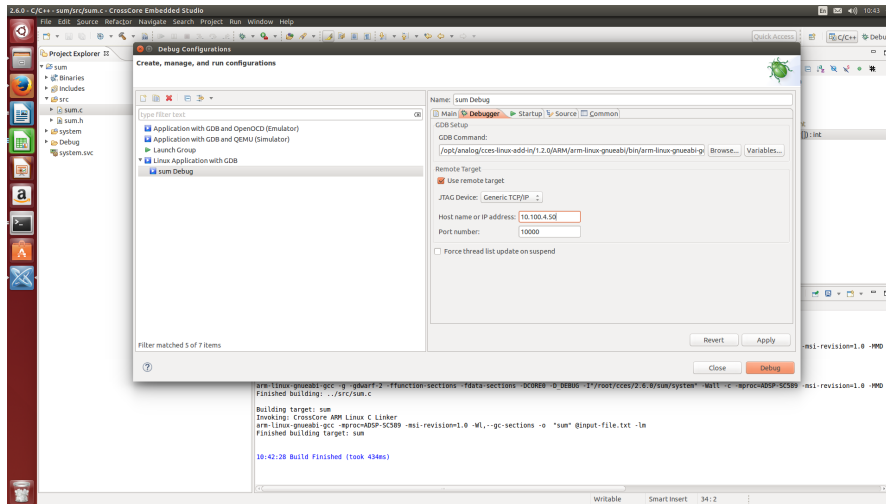
d. You're now set-up to connect to gdbserver via the CCES IDE.

### 3. Launch CCES and create a Debug configuration

- a. Select **Run | Debug configurations...** from the main menu.
- b. Or click the Debug Beetle icon on the toolbar
- c. On the Main tab, select the project and application that we would like to debug. For example, project is sum and the C/C++ application is called Debug/sum.



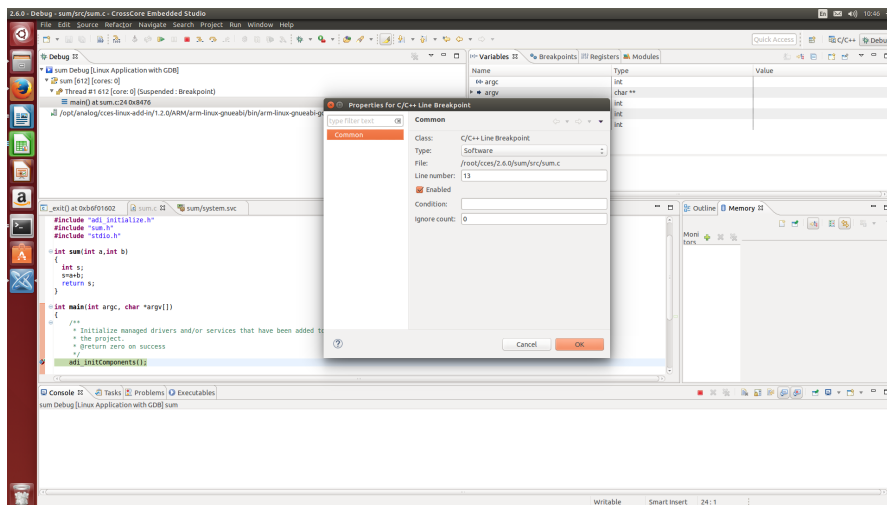
- d. Switch to the Debugger tab
  - i. Enter the IP address for your Linux target. For example, 10.100.4.50.
  - ii. Enter the Port number on which gdbserver is listening. For example, 10000.



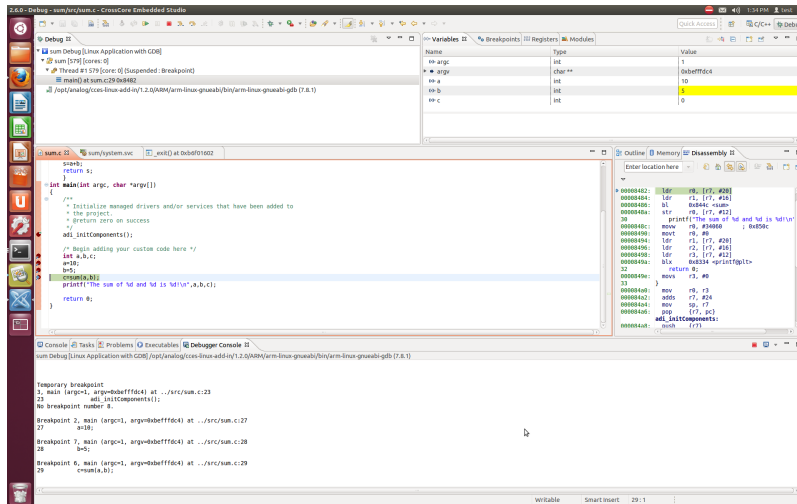
e. Click Apply and then Debug

#### 4. Debugging your application with CCES

- After clicking Debug button in the above pop-up window , CCES will launch GDB and connect to your gdbserver.
- Or right clicking on the vertical bar of the code window, select Add Breakpoint..., you can add breakpoint and stop at the desired code segment.



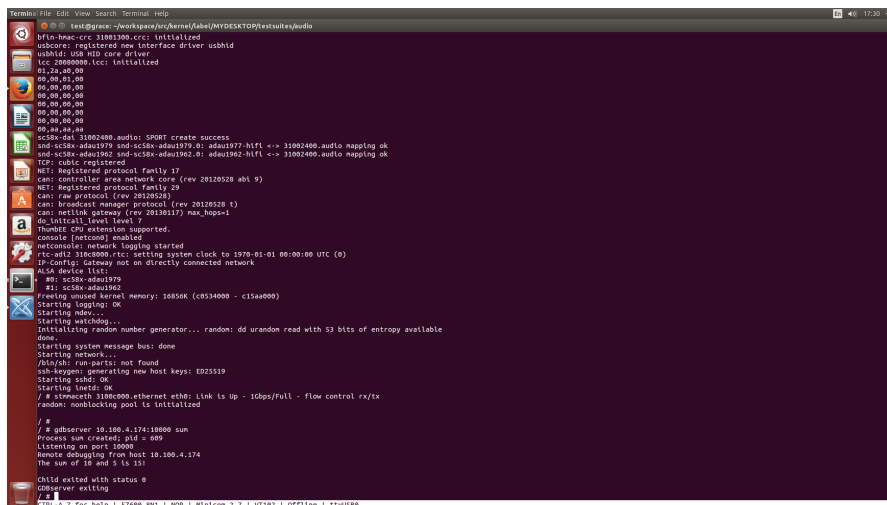
- You are now able to debug your Linux application from within the CCES IDE. In this example, we set up breakpoint at variables assignment, then click "Step into/over" icon continuously on the toolbar:



You can see in the variable window the value of variables(a,b,c) changing as the binary goes on.

The bottom "Debugger console" window shows the output of the running process.





- d. When you terminate your debug session, gdbserver on the Linux target will also terminate.



- e. To relaunch a debug session, you will also need to relaunch gdbserver on the Linux target.

## 9 Appendix A: Hardware Used During Testing

The following hardware was used during validation of the ADSP-SC5xx EZ-Kit by Analog Devices. Analog Devices does not recommend specific hardware to use with the ADSP-SC5xx Ez-Kit, but the following hardware has been proven to work in a testing environment.

	Device	Model	Manufacturer	Picture
Bluetooth ALSA/usb	Bluetooth headset	DR-BT140Q	SONY	
	USB Bluetooth Dongle	CN-512v1 001	SITECOM	
Wireless network driver USB dongle	Wifi dongle	DWL-G122	D-Link	
USB camera test	Camera	PID: LZ115SR	Logitech	
ADV7842 video decoder driver	Set - top box	H7	HIMEDIA	