uClinux CoreCommander BSP User Guide

CoreCommander BSP Version: 2.1.1.1

Document Version: 1.1

Date: January 2009

System Level Solutions, Inc, (USA)

14100 Murphy Avenue,
San Martin, CA 95046
(408) 852 - 0067

System Level Solutions, (India) Pvt, Ltd.

Plot # 32, Zone - D/4, Phase 1,
G.I.D.C. Estate, V.U. Nagar - 388 121
Gujarat, India
91-2692-229280
http://www.slscorp.com

ug_CoreCommanderbsp_1.1

About This Document

This document describes the usage of the uClinux CoreCommander board support package. With the help of the bsp you can develop embedded applications using CoreCommander board and uClinux.

Table below shows the revision history of the document.

Version	Date	Description
1.1	January 2009	- Updated document for CoreCommnader BSP, ver. 2.1.1.1 - Updated section 1.3 Supported devices. - Updated section 4.3 SLS SD Host Controller IP driver - Updated section 4.5 USB 2.0 IP driver - Added Appendix A, B, C.
1.0	September 2008	First release

How to Contact SLS

For the most up-to-date information about SLS products, go to the SLS worldwide website at http://www.slscorp.com. For additional information about SLS products, consult the source shown below.

Information Type	Email
E-mail Product literature services, SLS literature services, Non-technical customer services, Technical	support@slscorp.com

Typographic Conventions

This document uses the typographic conventions shown as below.

Visual Cue	Meaning
Bold type with initial capital letters	All headings, subheadings titles in a document are displayed in bold type with initial capital letters. E.g. **Configuring and Compiling.**
Bold	Project names, Menu commands, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: **Helloworld**, project name, **Exit** menu command, **romfs/bin** directory, **sls_cc_bsp_hw_v10.sof** file.
Courier	Anything that must be typed exactly as it appears is shown in Courier type. For example: `cd /home/uClinux-2.1.1.1/nios2-linux/uClinux-dist`.
1, 2	Numbered steps are used in a list of items, when the sequence of items is important. Such as steps listed in procedure.
•	Bullets are used in a list of items when the sequence of items is not important.
📝	Hand points to special information that requires special attention
⚠ CAUTION	Caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
⚠ WARNING	Warning indicates information that should be read prior to starting or continuing the procedure or processes.
👣	Feet direct you to more information on a particular topic.

Table of Contents

1. Introduction

uClinux CoreCommander Board support package (BSP) provides developers the easiest and fastest way to create embedded applications on the CoreCommander board (target) using uClinux operating system. The hardware, firmware, and software together form a complete package for building, downloading, and testing applications developed on a host machine:

- Install the uClinux board support package (BSP) on the host PC
- Connect JTAG cable between host and target for JTAG communication
- Build custom Linux kernel for the CoreCommander board

When ready to see your application in action, just download it to the CoreCommander board. Now you can test and debug the kernel and application.

1.1 Software

uClinux CoreCommander BSP is the combination of uClinux, IP core drivers and applications developed by SLS Corporation. Current version of the BSP supports Flash, SD Host Controller, Ethernet, and USB 2.0 IP.

uClinux is a port of Linux for MMU less processor. So many different architectures are supported by uClinux. It has almost all necessary power as an embedded operating system. This document is based on kernel version 2.6.28 Visit www.uclinux.org for more information.

1.2 Hardware

The following list summarizes the hardware featuers:

Target Hardware Board

- SLS CoreCommander Board, Cyclone III Edition

Device

- System name: sls_cc_bsp_hw_v10
- Family: Cyclone III
- Device: 3C25F256C8
- Total logic elements (LE) used: 12,965 / 24,624 (53%)
- Total pins used: 119 /157(76%)
- Total memory used: 138,080 / 608,256 (23%)

Processor

- Nios II/f processor core
  - Nominal metrics: 113 DMIPS at 100 MHz, 1,400–1,800 LEs, MMU/MPU option disabled

- 4-KByte data cache, 4-KByte instruction
- JTAG debug module for downloading software, 300–400 LEs

This BSP is targeted for Nios II. Nios II is a soft core FPGA processor. It is 32 bit RISC general purpose processor. Nios II processor system is equivalent to a microcontroller or “Computer on a chip” that includes a processor and a combination of peripherals and memory on a single chip. The term “Nios II processor system” refers to a Nios II processor core, a set of on-chip peripherals, on-chip memory, and interfaces to off-chip memory, all implemented on a Single Altera device. Like a microcontroller family, all Nios II processor systems use a consistent instruction set and programming model.

Memory Interfaces

- Common flash interface (CFI) flash memory
  - 8 Mbytes
- SDRAM Controller
  - 32 Mbytes
- SD Host Controller IP, v2.4

Communication Interfaces

- SLS Ethernet 10/100 MAC v2.0
- JTAG UART with integrated read and write FIFO
- USB 2.0 Device IP Core (USB20HR), v2.3

Display Interfaces

- SLS "1.7" TFT LCD Interface

System Peripherals

- Timers/counters
  - Interval timer
    - 32-bit counter size, 1-ms time-out period
- 6 Keypad PIOs (input only)
- 2 LED PIOs (output only)
- System ID

If you wish to recompile the reference design, download the following IP cores and interfaces:

- USB20HR IP Core
- SD Host Controller IP Core
- Ethernet 10/100 MAC IP Core
- 1.7” TFT LCD Interface

After downloading the IP Cores, get the license for each IP Core from http://www.slscorp.com/pages/ipLicenseinfo.php and follow the instructions.

1.3 Development Platform Requirements

We assume that user is familiar with Altera tools. Following are development platform requirements:

- Any Linux distribution having good hardware configuration with at least 2.5 GB free space.
- Altera Development Tools (v8.1)

This uClinux BSP is tested on Red Hat Fedora Core 7, OpenSUSE 10.3 and Mandriva 2008 but it should work on other Linux Desktop distributions also. You can build the kernel and application on Linux desktop only. Images built on Linux can be downloaded and run on Windows if the Altera Development tools are installed.

This BSP will not work on Cygwin environment.

1.4 Setup

To startup and begin communicating with the board:

1. Establish communication between target (Board) and host (PC) using JTAG cable.
2. Establish a network connection using RJ45 Ethernet cable to debug the application. (Ethernet Add-on board is required)
3. Apply power to the CoreCommander board.

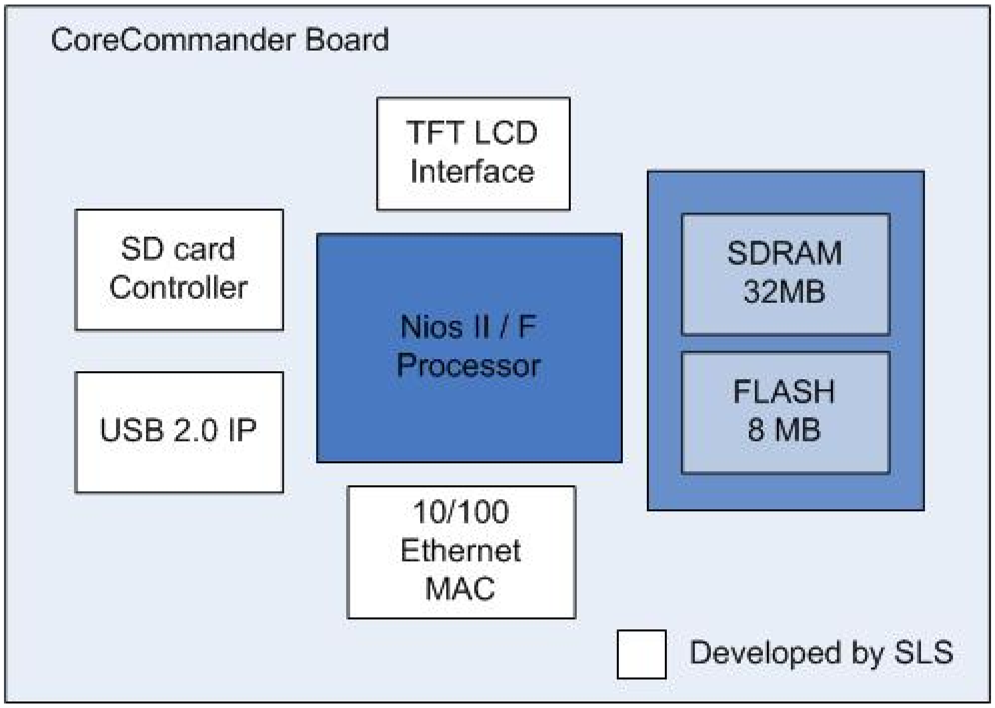
2. uClinux CoreCommander BSP Development Environment

This chapter provides information to help setup the development environment for the CoreCommander board.

2.1 CoreCommander BSP SOPC System

Figure 1 below shows the block diagram of CoreCommander BSP SOPC system.

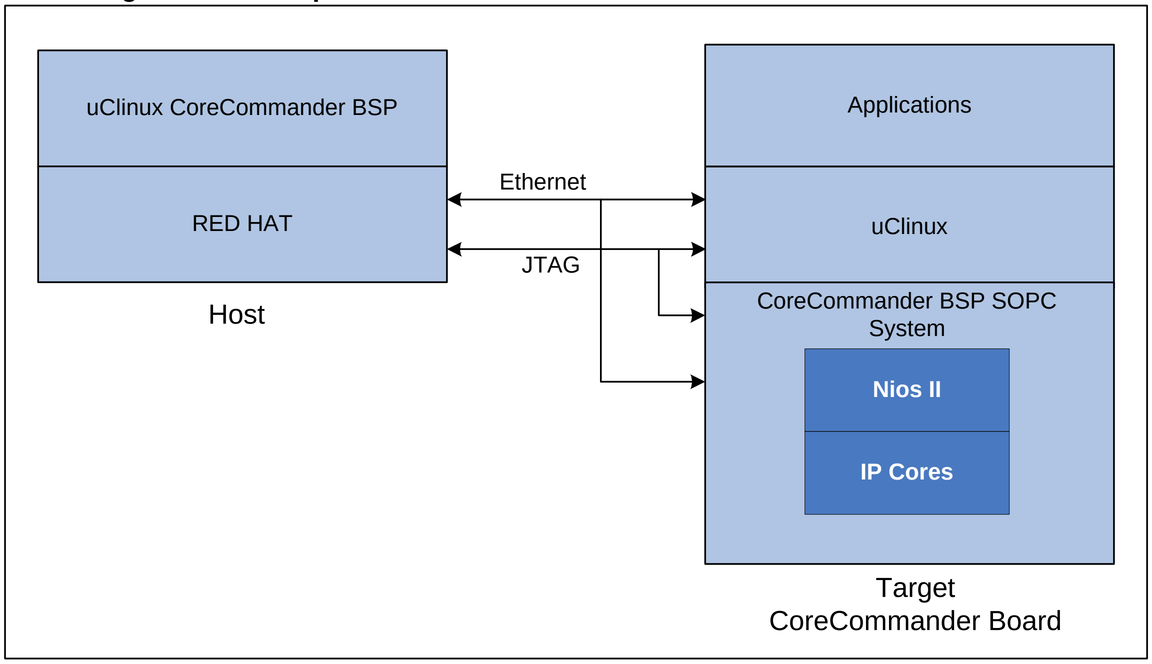
Figure 1 : CoreCommander BSP SOPC System



2.2 Development Environment

Nios II embedded development environment consists of two systems: a host system and a target system. Host system is used for compiling, linking, remote debugging, and associated development activities. Target system, such as the CoreCommander board, is used for application development and testing (Figure 2). Board acts as a target for application development

Figure 2 : Development Environment



2.2.1 CoreCommander BSP Components

Table below lists the components included in the uClinux CoreCommander BSP.

Component	Description
kernel	v2.6.28
gcc	v3.4.6
Ethernet driver	Included
Frame buffer driver	Included
Keypad driver	Included
SDcard driver	Included
FLASH driver	Included
JFFS2 support	Included
VFAT support	Included
NFS support	Included
JPEG support	Included

2.2.2 IP Address Used

Table below lists the IP address and port used throughout this document. It may be different for your system. Please check your network settings before applying any IP address.

IP Address	Description
192.168.0.14	Development Target IP address
192.168.0.26	Development Host and NFS server IP address
192.168.0.205	Gateway IP Address
255.255.255.0	Gateway Mask
9999	TCP port

2.3 Development Host

Refer to the Red Hat documentation for Red Hat Fedora 7.0 core installation and host setup information.

2.3.1 Downloading and Unpacking uClinux CoreCommander BSP

To download the uClinux CoreCommander BSP package visit: http://slscorp.com/pages/ccbspdownload.php.

Now, follow the steps below to unpack the BSP package.

1. Copy the entire package in the home directory.
2. Apply **su** command to login as super user

   ```
   #su ↵
   ```

   It will ask super user password for further process.

3. Issue

   ```
   #cd /home ↵
   ```

   ```
   #tar jxfv uClinux-2.1.1.1.tar.bz2 ↵
   ```

   It starts extracting files and it may take some time.

4. Now go into uClinux directory using the following command

   ```
   #cd /home/uClinux-2.1.1.1 ↵
   ```

   And then

   ```
   #tar jxfv uClinux-dist.tar.bz2 ↵
   ```

   It starts extracting files and it may take some time.

5. Now, issue

   ```
   #cd Bintools ↵
   ```

   ```
   #tar jxfv nios2gcc-20080203.tar.bz2 ↵
   ```

It starts extracting files and it may take some time.

6. Return to uClinux-2.1.1.1

```
#cd .. ↵
```

7. Issue command

```
#ls ↵
```

Here you will see **nios2-linux**, **Bintools**, and **System-files**.

2.3.2 Directory Contents

You have now successfully unpacked the BSP package. You will see following directory structure:

Directory	Description
uClinux-2.1.1.1	Contains nios2-linux, bintools (binary tool chain) and system files. Users, interested only in software side can start working with this directory.
Docs	Contains BSP documents: uClinux CoreCommander BSP user guide, CoreCommander reference manual and boot_message.txt.
Demo	Contains SLS uClinux CoreCommander Demos. In each subdirectory you will find other necessary files.
Ref_design	Contains Quartus archive project of SOPC system used in this BSP.
Quick Reference	Pre-built zImage for ready reference.
ReleaseNotes.pdf	Contains CoreCommander BSP release notes.

2.4 Development Target

2.4.1 Configuring and Compiling uClinux

CoreCommander BSP has two main areas of configurations. Together they define a complete uClinux platform configuration. These two areas include:

- Vendor / Product Settings
- Kernel Settings

In this section, you will learn all the above listed configurations.

Start Configuration Menu

To configure the kernel, go to the directory **/home/uClinux-2.1.1.1/nios2-linux/uClinux-dist** and follow the steps below:

1. Open the **Linux terminal**.

2. On the terminal, change into the **$home/uClinux-2.1.1.1/nios2-linux/uClinux-dist** directory and change the environment path. Use one of the four main kernel configuration methods to start the configuration menu.

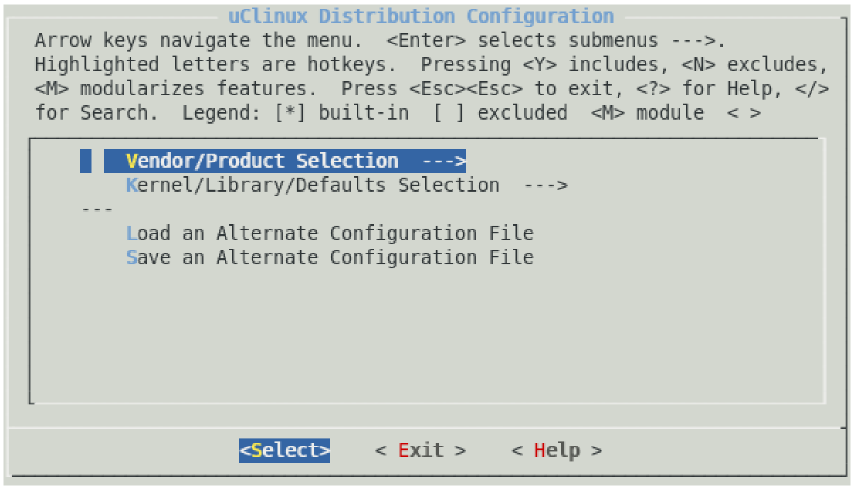
```
#cd /home/uClinux-2.1.1.1/nios2-linux/uClinux-dist ↵

#PATH=$PATH:/home/uClinux-2.1.1.1/Bintools/opt/nios2/
bin ↵
```

4. Configure the kernel.

```
#make menuconfig ↵
```

You will see the kernel configuration main menu as shown in Figure 3.

Figure 3 : Linux Kernel Configuration Main Menu



```
uClinux Distribution Configuration
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      Vendor/Product Selection  --->
      Kernel/Library/Defaults Selection  --->
    ---
      Load an Alternate Configuration File
      Save an Alternate Configuration File

            <Select>    < Exit >    < Help >
```

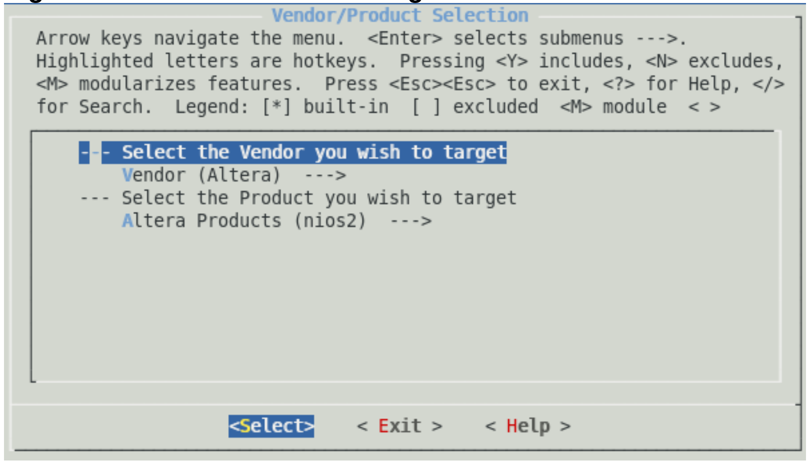
Use one of the two main kernel configuration methods to start the configuration.

Use ↑ or ↓ arrow to select the menu item and then press ↵to go into the submenu. Use ← or → and then press ↵to select the menu command <**Select>, <Exit**> or <**Help**> throughout the configuration.

Configuring Vendor/Product Settings

1. Select the submenu **Vendor/Product Selection --->** from the Figure 3. Press **Enter**. You will see **Vendor/Product Configuration** window. See Figure 4.

Figure 4 : Vendor/Product Configuration Window



```
Vendor/Product Selection
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

    --- Select the Vendor you wish to target
        Vendor (Altera)  --->
    --- Select the Product you wish to target
        Altera Products (nios2)  --->

            <Select>    < Exit >    < Help >
```

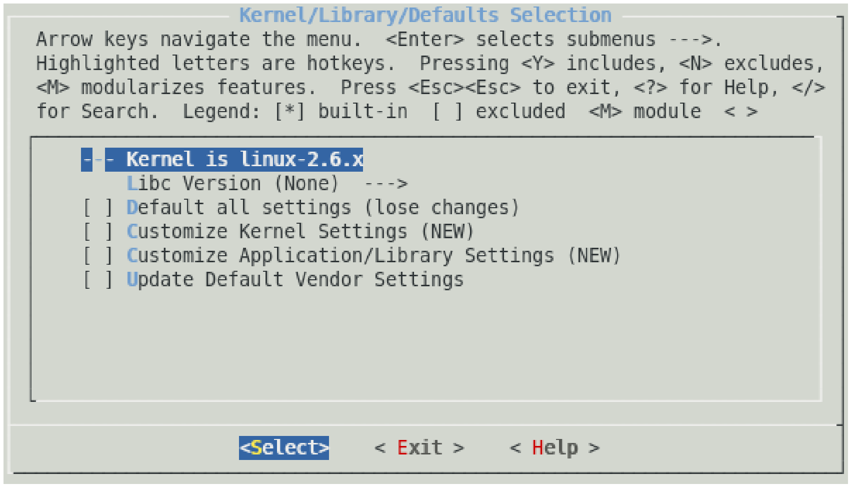
2. Select the following options:
   - Vendor: **Vendor (Altera)**
   - Target Product: **Altera Products (nios2) ---->**

3. Select **<Exit>.** You will return to kernel configuration menu Figure 3.

Configuring Kernel/Library/Defaults Settings

1. Select the option **Kernel/Library/Defaults Selection --->** from the Kernel Configuration Menu Figure 3. You will see the **Kernel/Library/Defaults Selection** submenu as shown in Figure 5.

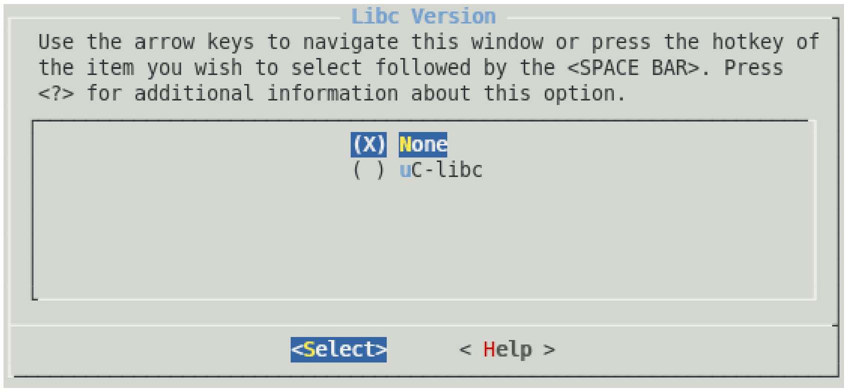
Figure 5 : Kernel/Library/Defaults Selection Window



```
Kernel/Library/Defaults Selection
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

    --- Kernel is linux-2.6.x
        Libc Version (None)  --->
    [ ] Default all settings (lose changes)
    [ ] Customize Kernel Settings (NEW)
    [ ] Customize Application/Library Settings (NEW)
    [ ] Update Default Vendor Settings

            <Select>    < Exit >    < Help >
```

2. Select **Libc Version (None) ---->** .You will see the as shown in Figure 6.

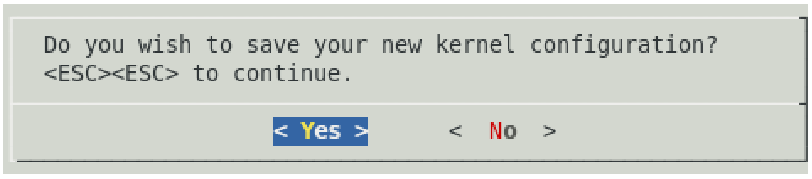
Figure 6 Libc Version Settings



```
Libc Version
Use the arrow keys to navigate this window or press the hotkey of
the item you wish to select followed by the <SPACE BAR>. Press
<?> for additional information about this option.

                    (X) None
                    ( ) uC-libc

              <Select>      < Help >
```

3. Press **Enter** to select **None.**
4. Select <**Exit**>
5. Select <**Exit**>
6. You will be asked to save the kernel configuration. See Figure 7

Figure 7 : Save Option



```
Do you wish to save your new kernel configuration?
<ESC><ESC> to continue.

           < Yes >      <  No  >
```

7. Select <**Yes**> and press **Enter**.

   You have now finished Kernel/Library/Defaults settings.

DO NOT change any other settings until first successful boot.

8. You will be returned to **Linux Terminal**.

2.4.2 Set System.ptf

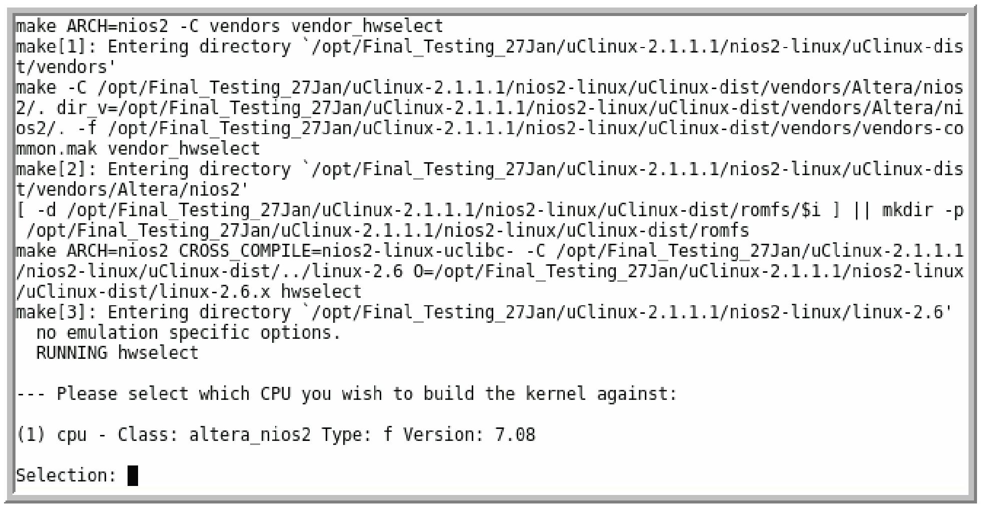
To generate a system header file using **sls_cc_bsp_hw_v10_sopc.ptf** located at **"/home/uClinux-2.1.1.1/System-files**" directory, follow the steps below:

1. Type the following command and press **Enter**.

```
#make vendor_hwselect SYSPTF=/home/uClinux-2.1.1.1/
System-files/sls_cc_bsp_hw_v10_sopc.ptf ↵
```

   You will be asked to select the CPU. See Figure 8

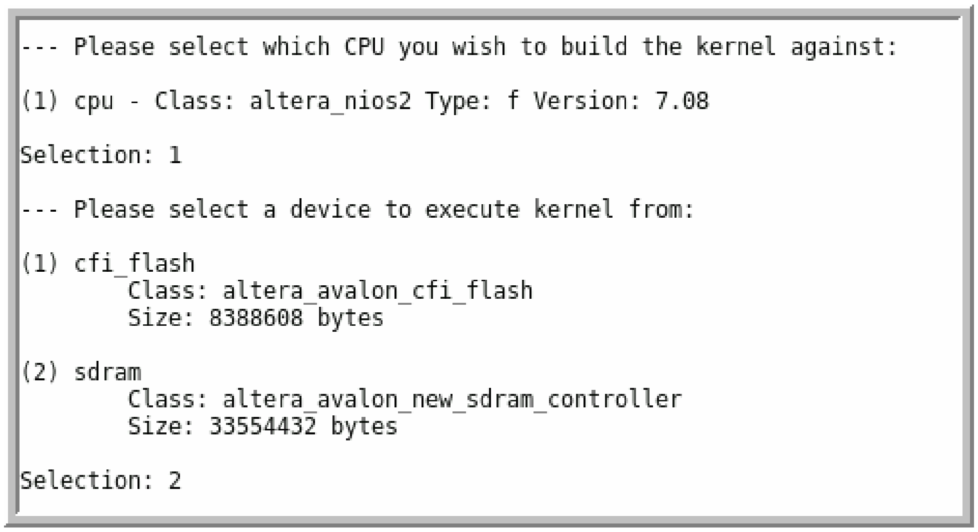
Figure 8 : CPU Selection



```
make ARCH=nios2 -C vendors vendor_hwselect
make[1]: Entering directory `/opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux/uClinux-dis
t/vendors'
make -C /opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux/uClinux-dist/vendors/Altera/nios
2/. dir_v=/opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux/uClinux-dist/vendors/Altera/ni
os2/. -f /opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux/uClinux-dist/vendors/vendors-co
mmon.mak vendor_hwselect
make[2]: Entering directory `/opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux/uClinux-dis
t/vendors/Altera/nios2'
[ -d /opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux/uClinux-dist/romfs/$i ] || mkdir -p
 /opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux/uClinux-dist/romfs
make ARCH=nios2 CROSS_COMPILE=nios2-linux-uclibc- -C /opt/Final_Testing_27Jan/uClinux-2.1.1.1
/nios2-linux/uClinux-dist/../linux-2.6 O=/opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux
/uClinux-dist/linux-2.6.x hwselect
make[3]: Entering directory `/opt/Final_Testing_27Jan/uClinux-2.1.1.1/nios2-linux/linux-2.6'
  no emulation specific options.
  RUNNING hwselect

--- Please select which CPU you wish to build the kernel against:

(1) cpu - Class: altera_nios2 Type: f Version: 7.08

Selection: █
```

2. Enter the choice **(1).** It will ask to select a device to execute kernel from: See Figure 9

Figure 9 : Select a Device to Execute Kernel From



```
--- Please select which CPU you wish to build the kernel against:

(1) cpu - Class: altera_nios2 Type: f Version: 7.08

Selection: 1

--- Please select a device to execute kernel from:

(1) cfi_flash
        Class: altera_avalon_cfi_flash
        Size: 8388608 bytes

(2) sdram
        Class: altera_avalon_new_sdram_controller
        Size: 33554432 bytes

Selection: 2
```

5. To select SDRAM, enter:

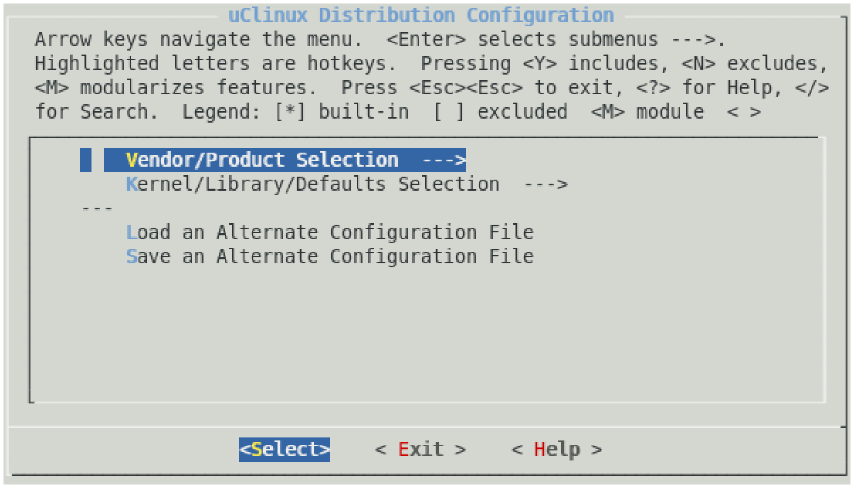
```
Selection:2
```

2.4.3 Customization of Kernel Settings

1. To customize the kernel settings, type on the terminal:

```
#make menuconfig ↵
```

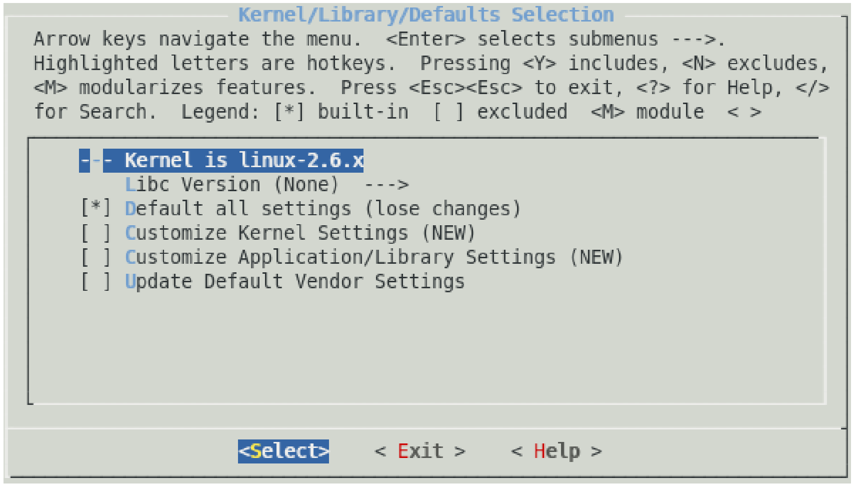
Figure 10 : Linux Kernel Configuration Window



```
uClinux Distribution Configuration
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

        Vendor/Product Selection  --->
        Kernel/Library/Defaults Selection  --->
    ---
        Load an Alternate Configuration File
        Save an Alternate Configuration File

              <Select>    < Exit >    < Help >
```

2. Select **Kernel/Library/Defaults Selection** and press **Enter.** More kernel configuration settings will be displayed. See Figure 11.

Figure 11 : Customize Kernel/Library/Defaults Selection Window



```
Kernel/Library/Defaults Selection
Arrow keys navigate the menu.  <Enter> selects submenus --->.
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
<M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

    --- Kernel is linux-2.6.x
        Libc Version (None)  --->
    [*] Default all settings (lose changes)
    [ ] Customize Kernel Settings (NEW)
    [ ] Customize Application/Library Settings (NEW)
    [ ] Update Default Vendor Settings

              <Select>    < Exit >    < Help >
```

3. Select **Default all settings** by pressing **"Y"**
4. Select <**exit**>.
5. Select <**exit**>
6. Select <**yes**> to save all changed settings.

Console asks you for board selection. Choose **CoreCommander board** option. Further it will ask for including and excluding drivers and applications. Press **N** for all these options.

2.4.4 Building uClinux zImage

Once you have configured the kernel, build uClinux Image by issuing following command.

```
#make ↵
```

It will take some time to build zImage file. After successful build, you will get **zImage** (**.elf** file) *at* **/home/uClinux-2.1.1.1/nios2-linux/uClinux-dist/image.**

3. Downloading and Running zImage

zImage is one type of elf file which contains compressed kernel image and romfs image. Given below are the steps for downloading and running zImage on the CoreCommander board.

3.1 Running zImage on Linux Using JTAG UART Console

To run zImage on Linux, you should have Nios2 EDS installed on your PC with environment variables properly set. We will use utilities from the Nios2 EDS.

1. Configure .sof.

   ```
   #nios2-configure-sof /home/uClinux-2.1.1.1/System-
   files/sls_cc_bsp_hw_v10.sof ↵
   ```

2. Open the terminal to download zImage.

   ```
   #nios2-download –g /home/uClinux-2.1.1.1/nios2-linux/
   uClinux-dist/images/zImage ↵
   ```

3. Run the following command.

   ```
   #nios2-terminal ↵
   ```

Here you can see the message **Linux booting** on the same shell.

4. Configuring Device Drivers and File Systems

If more functions need to be supported on kernel, then before rebuilding the kernel, it requires more kernel configuration. This session describes the procedure to configure the kernel.

To know more about the peripherals and hardware available on the CoreCommander board, refer the *CoreCommander board Getting Started User Guide* located at *<CC BSP Installation Path>*/**Docs**.

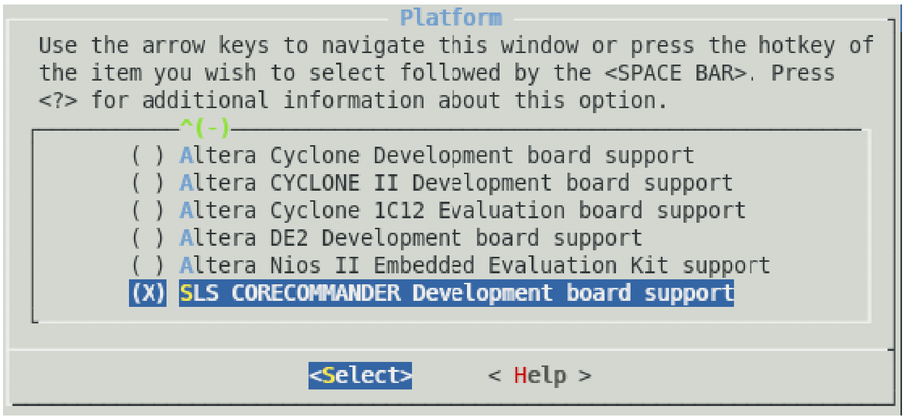
4.1 Board Selection

From the **Customize Kernel Settings**, select the following options.

```
Processor type and features --->
     Platform (Default Board) --->
```

You will see the board selection options as shown in Figure 12. Select the CoreCommander board option and press **Enter**.

Figure 12 : Board Selection



```
Platform
Use the arrow keys to navigate this window or press the hotkey of
the item you wish to select followed by the <SPACE BAR>. Press
<?> for additional information about this option.
^(-)
      ( ) Altera Cyclone Development board support
      ( ) Altera CYCLONE II Development board support
      ( ) Altera Cyclone 1C12 Evaluation board support
      ( ) Altera DE2 Development board support
      ( ) Altera Nios II Embedded Evaluation Kit support
      (X) SLS CORECOMMANDER Development board support

                <Select>        < Help >
```

4.2 Flash Memory (MTD) Driver

To include the Flash Memory (Memory Technology Device) driver in compilation, the following options should be enabled:

```
Device Drivers --->
   [*] Memory Technology Devices (MTD) support --->
      --- Memory Technology Device (MTD) support
      [*] MTD partitioning support
      [*] Direct char device access to MTD devices
      [*] Caching block device access to MTD devices
         RAM/ROM/Flash chip drivers --->
            [*] Detect flash chips by Common Flash
            Interface (CFI) probe
            [*] Support for AMD/Fujitsu flash chips
         Mapping drivers for chip access --->
            [*] CFI Flash device in physical memory map
            (0x8000000) Physical start address of flash
            mapping (NEW)
            (0) Physical length of flash mapping (NEW)
            (2) Bank width in octets (NEW)
```

4.2.1 JFFS2 File System Configuration

JFFS2 is a log-structured file system designed for use on flash devices in embedded systems. Rather than using a kind of translation layer on flash device to emulate a normal hard drive, as is the case with older flash solutions, it places the file system directly on the flash chips.

If you want to use JFFS2 file system on Flash Memory then you have to configure both the Flash Memory driver and JFFS2 file system. First do the Flash Memory Driver configuration as described above. And then follow the procedure below to configure the JFFS2 File system.

```
File systems --->
   Miscellaneous File systems --->
      [*] Journaling Flash File System v2 (JFFS2)
      support
      (0) JFFS2 debugging verbosity (0 = quiet, 2 =
      noisy)
      [*] JFFS2 write-buffering support
```

4.3 SLS SD Host Controller Driver

To include the SD Host Controller Driver in compilation, following options should be enabled:

```
Device Drivers --->
[*] MMC/SD/SDIO card support --->
         --- MMC/SD/SDIO card support
              *** MMC/SD/SDIO Card Drivers ***
        [*]   MMC block device driver
        [*]     Use bounce buffer for simple hosts
        [*]   SLS SD Host Controller Driver
```

To know more about using SD Host Controller driver, refer Appendix B.

4.3.1 VFAT

Virtual File Allocation Table (VFAT) is the part of the Windows 95 and later operating system that handles long file names, which otherwise could not be handled by the original file allocation table (FAT) programming.

If you want to use VFAT file system on SD Card then you have to configure both the SD Host Controller driver and VFAT file system. First do the SD Host Controller Driver configuration as described above and then follow the procedure below to configure the VFAT File system.

```
File systems --->

   DOS/FAT/NT File systems --->

      [*] MSDOS fs support

      [*] VFAT (Windows-95) fs support

      (437) Default codepage for FAT

      (iso8859-1) Default iocharset for FAT

      Native Language Support
```

```
    --- native language support
        (iso8859-1) Default NLS Option
        [*] Codepage 437 (United States, Canada)
        [*] NLS ISO 8859-1 (Latin 1; Western European
        Languages)
```

4.4 SLS Ethernet IP Driver

To include the SLS Ethernet IP Driver in compilation, following options should be enabled:

```
Networking --->
    --- Networking support
        Networking options --->
            [*] Packet socket
            [*] Unix domain sockets
            [*] TCP/IP networking
Device Drivers --->
    [*] Network device support --->
        [*] Ethernet (10 or 100Mbit) --->
            [*] SLS MAC support
```

To know more about using Ethernet driver, refer Appendix A.

4.4.1 NFS

NFS is a network file system protocol originally developed by Sun Microsystems in 1984, allowing a user on a client computer to access files over a network as easily as if the network devices were attached to its local disks.

If you want to use NFS file system on Ethernet then you have to configure both the Ethernet IP driver (as described above) and NFS file system. Follow the procedure below to configure the NFS File system.

```
File systems -->
   [*] Network File Systems -->
      --- Network File Systems
      [*] NFS client support
      [*] NFS client support for NFS version 3
```

4.5 SLS USB 2.0 Driver

To include SLS USB 2.0 ULPI support in compilation, follow the steps below:

```
Device Drivers --->
    Character devices --->
        [*] SLS USB20 ULPI support
```

To know more about using USB 2.0 driver, refer Appendix C.

4.6 SLS Keypad Driver

To include SLS Keypad driver in compilation, follow the steps below:

```
Device Drivers --->
    Character devices --->
        [*] SLS Keypad Driver Support
```

4.7 SLS LCD Frame buffer Driver

To include the SLS LCD Frame buffer Driver in compilation, following options should be enabled:

```
Device Drivers --->
   Graphics support --->
      [*] Support for frame buffer devices --->
         [*] SLS FrameBuffer Driver Support
```

4.8 JTAG UART Driver

To include the JTAG UART Driver in compilation, make sure that following options are enabled:

```
Device Drivers --->
  Character devices --->
    Serial drivers --->
      [*] Altera JTAG UART support
      [*] Altera JTAG UART console support
```

If you want to bypass the JTAG, then select the following options.

```
Device Drivers --->
  Character devices --->
    Serial drivers --->
      [*] Altera JTAG UART support
      [*] Altera JTAG UART console support
      [*] Bypass output when no connection
```

If you are using the bypass JTAG option then application should not have any printf statements.

5. User Applications

Using the User Applications you can access the drivers, file systems and peripherals available on the board. This chapter describes the following user applications.

5.1 Flash and JFFS2 Application

To access Flash and JFFS2 application, include its driver and file system as described in the section: Flash Memory (MTD) Driver and JFFS2 File System Configuration.

5.1.1 Flash Tools

Select **Customize Application/Library Settings>Flash tools** to select the Flash tools. Select the following Flash Tools.

```
Flash Tools --->
   --- MTD utils
   [*] mtd-utils
   [*] erase
   [*] eraseall
   [*] lock
   [*] unlock
   [*] mkfs.jff2
```

Before compiling the Flash Tools, make sure that the **lzo-devel** library is installed on your linux PC.

Build the kernel as explained in the section Development Target.

5.1.2 Accessing Flash and JFFS2 Applications

To access the Flash and JFFS2 in your application use the required commands as described in the table below:

Command	Description
`cat /proc/mtd`	Displays flash partitions.
`cat /proc/mounts`	Displays already mounted files.
`ls -l /dev/mtd*`	Lists mtdblocks and other details.
`unlock /dev/mtdx`	Unlocks all the sectors of mtd device. e.g.: unlock /dev/mtd0, this will unlock all sectors in mtd device0.
`flash_erase device offset number_of_blocks`	Erases number of blocks of a device starting from the given address. e.g. : flash_erase /dev/mtd0 0x00000 5 .This command would erase 5 blocks of mtdblock0 starting from the offset address 0x00000.
`flash_eraseall /dev/mtdx`	Erases all the contents of mtd device. e.g. : flash_eraseall /dev/mtd0. This would erase all the content from mtd device 0. ***Note:*** here mtd0 means mtd device0 and not mtdblock0. If you are not able to erase the content and get the message "**read only file system**" then use the unlock command mentioned above and after that use eraseall.
`mount -t jffs2 dev/mtdblockx /mnt`	Mounts mtdblockx partition on the /mnt directory. You can also use other directory than /mnt e.g. : mount –t jffs2 /dev/mtdblock2 /mnt This would mount mtdblock2 on the mnt directory. ***Note:*** Erase entire flash before mounting JFFS2 file system.
`mkdir new`	Creates a new directory
`cp /new /dev/mtdx`	Copies a new directory to mtdx. e.g. : cp /new /dev/mtd0
`lock /dev/mtdx`	Locks all sectors of the mtd device. e.g. : lock /dev/mtd0 This would lock all sectors of mtd device0.
Note: (1). Where x=0, 1, 2.. mtd device or mtdblocks in the system. (2). By default sectors of flash device are locked. So you can not write anything before unlocking the flash device. Use unlock command to write on flash device.	

5.1.3 Configuring Flash Partition

To configure the flash partition, either you can create your own mapping driver or modify the partition with **config.c** file. Here **config.c** is the mapping driver for mtd device. It is located at <*path_to_dist*>**/linux-2.6/arch/nios2/kernel**.

5.2 LCD and SD Card Application

To access the LCD and SD Card applications, select necessary drivers and file systems for SD Card and LCD as mentioned in the SLS SD Host Controller Driver, VFAT and SLS LCD Frame buffer Driver

5.2.1 Using LCD Frame Buffer Driver API

Using LCD Frame buffer driver API you can draw a drawing on the CoreCommander LCD. SLS has provided the sample drawing. To view the drawing select following:

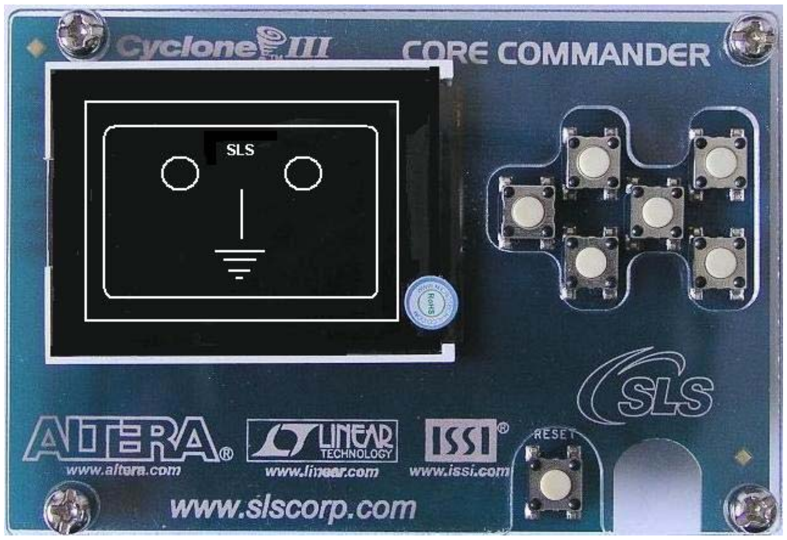
```
Miscellaneous Applications --->
   [*] sls_gui
```

Aftter making above selection apply **make** command and download the zImage, Apply following command on the nios terminal.

```
/>sls_gui ↵
```

You will see following drawing on the CoreCommander board.

Figure 13 : Drawing using LCD Frame Buffer API



5.2.2 Viewing JPEG Images from SD Card

Before going further, copy **.jpg** images from your PC in to the SD Card and make following selection.

```
Miscellaneous Applications --->
   ---video tools
   [*] jpegview
```

1. To mount the SD Card on **mnt** directory, issue the following command on the target when the system is up and running.

   ```
   />mount -t vfat /dev/mmcblk0 /mnt ↵
   ```

   Sometimes, when the user uses this command on the console, it is failed to mount and returns with the following error message.

   ```
   FAT: bogus number of reserved sectors
   VFS: Can't find a valid FAT file system on dev SLS_SD
   ```

   This is due to the problem in the file system formatted on the SD Card, which is supported in windows but is not supported in uClinux. To solve this problem, please download Quick Reference.

   Prebuilt_zImage directory contains following files:

   - zImage
   - sls_cc_bsp_hw_v10.sof
   - Readme.txt

   Download .sof and zImgae respectively. You will get console on JTAG UART. When zImage is up and running, plug the SD card into the CoreCommander socket. Issue the following command to format the SD card.

   ```
   />mkdosfs –I –F 16 /dev/mmcblk0 ↵
   ```

   After formatting the SD card, you have to copy the JPG file from your PC through card reader in the CoreCommander board. Once again mount the SD card with VFAT file system.

2. Go to the **mnt** directory and see the contents of the SD card by issuing the following command.

   ```
   />cd mnt ↵
   /mnt>ls ↵
   ```

   You will see on the console, the list of images contained in the SD card.

   ```
   1.jpg
   ```

```
2.jpg

3.jpg
```

3. View the images contained in the SD card on the CoreCommander LCD by issuing the following command.

```
/mnt>jpegview -s1 –f 1.jpg 2.jpg 3.jpg ↵
```

5.3 Adding New User Application

This section explains you about adding a user application named **hello** in uClinux. Follow the steps below to add a new user application.

1. Create **hello** directory in the **/home/uClinux-2.1.1.1/nios2-linux/uClinux-dist/user** directory.

2. Copy source file (.C file) to the hello directory

3. Add the configuration variable **hello** to the **user/Kconfig** file:

```
config USER_HELLO
bool "hello"
help
help_words_here.....
```

   This adds the **hello** menu option to the userland configuration menu.

4. Add following lines to the **user/Makefile** to add the hello directory in compilation.

```
...

dir_$(CONFIG_USER_HELLO) += hello
...
```

5. Create the **Makefile** under **user/hello** directory as mentioned below to compile the hello application.

```
EXEC = hello

OBJS = hello.o

all: $(EXEC)

   $(EXEC): $(OBJS)

   $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS)

romfs:

   $(ROMFSINST) /bin/$(EXEC)

clean:

   -rm -f $(EXEC) *.elf *.gdb *.o
```

6. Build the kernel with this application and issue the following command to view the output of this application.

   ```
   />hello ↵
   ```

   Here, you will see the output of your application.

5.4 Build New User Application Using SLS IP Drivers

To build a new user application using SLS IP drivers refer Appendix A, Appendix B, Appendix C.

5.5 Shell Commands

Using shell commands you can perform operations on uClinux running on the CoreCommander board. Shell is the basic application on the Linux system. Default shell provided in BSP is “sh”. “sh” uses the current directory as the prompt string. Commands can be executed under shell. (It works the same way as Linux PC). Help command will display internal commands provided by shell.

Table below lists the useful shell commands:

Command	Definition	Description
cat	cat filename	Shows file on screen
cd	cd [directory]	Changes current directory
chgrp	chgrp GROUP FILE…	Changes the group membership of each FILE to GROUP
chmod	chmod mode file/dir	Changes file/directory mode
chown	chown group:user file/dir	Changes file/directory own
cmp	cmp file1 file2	Compares two files
date	date [MMDDhhmm[YYYY]]	Get/set date
cp	cp file1 file2	Copy source to destination
df	df [device]	Shows information about the filesystem on which each FILE resides, or all filesystems by default
echo	echo arguments [>filename]	Outputs the ARGs or redirect to file
exec	exec file	Exec FILE, replacing this shell with the specified program
exit	exit [N]	Exits the shell with a status of N. If N is omitted, the exit status is that of the last command executed
free	free	Shows memory status
help	help	Shows help message

hexdump	hexdump file	Hex dump file
hostname	hostname	Shows host name
Kill	kill [-s sigspec | -n signum | -sigspec] [pid | job]…\n or kill -| [sigspec]	Sends signal to process
In	ln –s file1 file2	Creates a link to the specified TARGET
Is	ls [options]	List information about the FILES
mkdir	mkdir dirname	Creates the DIRECTORY
mknod	mknod type major minor	Creates device file
more	more filename	File perusal filter
mount	mount –t type device dir	Mount file system
mv	mv source dest	Rename SOURCE to DEST, or move SOURCE(S) to DIRECTORY
printenv	printenv	Print environment variables
pid	pid	Shows current process
ps	ps	Shows process information
pwd	pwd	Shows current directory
quit	quit	Quits current process
rm	rm file	Removes file
rmdir	rmdir dir	Removes dir
sleep	sleep number	sleeps several seconds
setenv	setenv var value	Sets environment variable
source	source file	Runs command in file
sync	system sync	sync
touch	touch [option] file	Update the access and modification times of each FILE to the current time
umask	umask octal number	user file-creation mask is set to MODE
umount	umount dir	Umount file system

6. Configuring Network utilities and NFS (Client)

This chapter introduces you about the various network utilities like ftp, dhcpcd, telnet, boa, and inetd. Before going further, follow all the steps for Ethernet driver as described in the section SLS Ethernet IP Driver.

6.1 Configuring DHCP Client

1. Select **Customize Application/Library Settings.** Do the following settings for Network Application and Busybox.

```
Network Applications ---->
   [*] dhcpcd-new (2.0/2.4)
   BusyBox ---->
   [*] Busybox
      Networking Utilities ---->
         [*] ifconfig
         [*]   Enable status reporting output (+7k)
```

2. Configure the IP address of the board dynamically (automatically) by issuing the following command on the target.

```
/>dhcpcd ↵
```

3. Issue following command on the target to know the IP address received from the above command and Ethernet port configuration.

```
/>ifconfig ↵
```

   You will find IP assigned to your board.

4. Ping your board from the host with the IP address (xx.xx.xx.xx) that you received from step 3.

```
#ping xx.xx.xx.xx ↵
```

   Make sure that DHCP server is available in your network and there is enough space for new IP allocation.

6.2 Static IP Allocation

To configure the static IP, make the following settings:

```
BusyBox ---->
[*] Busybox
   Networking Utilities ---->
      [*] ifconfig
      [*]   Enable status reporting output (+7k)
```

To allocate static IP to the board, issue the following command on the target.

```
/>ifconfig eth0 192.168.0.14 ↵
```

6.3 Mounting NFS on CoreCommander

Before mounting NFS on CoreCommander, select the options as mentioned in SLS Ethernet IP Driver and NFS. Then go to the user application and select the following.

```
BusyBox --->
   [*] BusyBox
      Linux System Utilities --->
         [*] mount
         [*] Support mounting NFS file systems
      Networking Utilities --->
         [*] ifconfig
         [*] Enable status reporting output (+7k)
```

Now build the kernel.

6.3.1 NFS Server (Host) Set Up

To set up the NFS server, follow the steps below:

1. Login as a **super user** on the host pc in your network.
2. Create a directory called **nfs** in the **/home** directory.

   ```
   #mkdir /home/nfs ↵
   ```

3. Edit the file named **exports** under **/etc** directory and add the following line:

   /home/nfs 192.168.0.0/255.255.255.0(sync,no_root_squash,rw)

   This setting may differ as per your network

4. Restart NFS server.

   ```
   #service nfs restart ↵
   ```

5. Verify it by issuing the following command.

   ```
   #showmount –e ↵
   ```

6.3.2 NFS Client (CoreCommander) Setup

When the **zImage** is up and running, issue the following command:

```
/>mount -t nfs -o nolock 192.168.0.26:/home/nfs /mnt ↵
```

After successful mounting, you can access the **/home/nfs** directory on the NFS server (host PC) using **/mnt** directory on the NFS client (CoreCommander board).

6.4 Configuring inetd, telnetd, ftpd Server

To configure inetd, telnetd, and ftpd follow the steps below:

1. First of all follow all steps for Ethernet driver as mentioned in SLS Ethernet IP Driver.

   ```
   Network Applications ---->
   [*] ftpd
   [*] inetd
   [*] telnetd
   ```

2. When the kernel is up and running, issue the following command on console:

   ```
   />inetd & ↵
   ```

   This would run inetd server in the background which in turn will run the telnet and ftp server.

3. Now connect FTP client on Host PC to the server running on the board. Here board IP is configured as **192.168.0.14** so issue following command on the Host PC.

```
#ftp 192.168.0.14 ↵
```

You will see the following on the host terminal. Enter the ftp user name and password.

```
Connected to 192.168.0.14

220- Welcome to the uClinux ftpd!

220 uClinux FTP server (GNU inetutils 1.4.1)ready

User (192.168.0.14 :( none)): ftp

331 Guest login ok, type your name as password.

Password: ↵

230 Guest login ok, access restrictions apply.

ftp>
```

4. Now connect Telnet client on the Host PC to the server running on the board. Here board IP is configured as 192.168.0.14 so issue following command on the Host PC.

```
#telnet 192.168.0.14 ↵
```

You will see the following on the host terminal.

```
uClinux login: root
Welcome to
          ____ _  _
         /  __| ||_|
    _   _| |  | | _ ____  _   _  _  _
   | | | | |  | || |  _ \| | | |\ \/ /
   | |_| | |__| || | | | | |_| |/    \
   |  ___\____|_||_|_| |_|\____|\_/\_/
   | |
   |_|

For further information check:
http://www.uclinux.org/


Sash command shell (version 1.1.1)
/>
```

6.5 Configuring boa Server

To configure the boa server on the board, follow the steps below:

1. Make the following selection.

```
[*] Customize Vendor/User Settings (NEW) ---->
     Network Applications ---->
        [*] boa
```

2. Issue the following command on the target.

   ```
   />boa &  ↵
   ```

3. On the Host PC, go to your web browser and type **board_ip** in the address bar. uClinux default webpage opens.

7. Debugging Kernel and User Application

To debug the user application you will require following software and hardware:

- Cross/Straight Network cable
- JTAG cable
- Eclipse IDE for C/C++ developers. You can download Eclipse IDE from: http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/20071103/eclipse-cpp-europa-fall2-linux-gtk.tar.gz

7.1 Debugging uClinux kernel using nios2-elf-insight over JTAG

To debug the uClinux kernel using nios2-elf-insight over JTAG, first build the zImage by using the following steps:

Before debugging, configure the kernel as explained below:

```
#make menuconfig ↵
```

Menu configuration window opens:

```
Kernel hacking ---->
   [*] Kernel debugging
   [*] Compile the kernel with debug info
```

Save, exit and build the zImage.

Following are the steps to debug the uClinux kernel using JTAG.

1. Download **.sof** file to your board using command:

   ```
   #nios2-configure-sof /home/uClinux-2.1.1.1/System-
   files/sls_cc_bsp_hw_v10.sof ↵
   ```

2. Go to **/home/uClinux-2.1.1.1/nios2-linux/uClinux-dist/linux 2.6.x** .You will find **vmlinux**. Copy and paste it in same directory with the name **vmlinux.elf**.

3. Now there will be a debug script called **nios2-debug** in NIOS II IDE. The script is located at <*altera_tools_installation_path*>**/nios2eds/bin**.

4. Modify the script to make it compatible with uClinux kernel. Save a copy of this script before modifying it and change line no 70 in nios2-debug from :

```
break *main
```

to

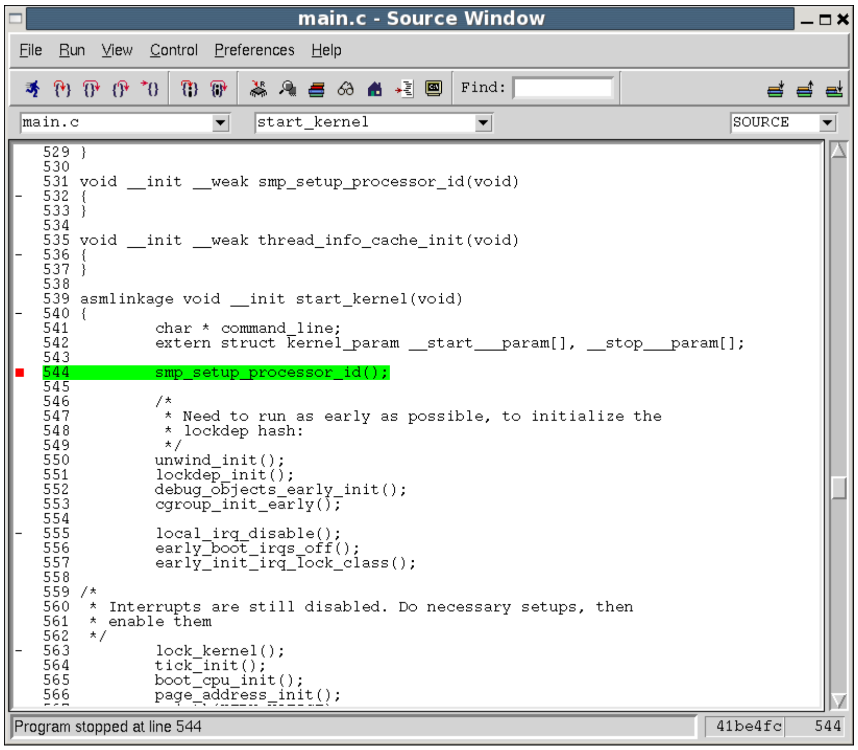
```
break *start_kernel
```

5. Issue the command:

```
#nios2-debug home/uClinux-2.1.1.1/nios2-linux/uClinux-
dist/linux2.6.x/vmlinux.elf ↵
```

Wait for few seconds and following window will be opened.

Figure 14 : main.C



```
}

void __init __weak smp_setup_processor_id(void)
{
}

void __init __weak thread_info_cache_init(void)
{
}

asmlinkage void __init start_kernel(void)
{
        char * command_line;
        extern struct kernel_param __start___param[], __stop___param[];

        smp_setup_processor_id();

        /*
         * Need to run as early as possible, to initialize the
         * lockdep hash:
         */
        unwind_init();
        lockdep_init();
        debug_objects_early_init();
        cgroup_init_early();

        local_irq_disable();
        early_boot_irqs_off();
        early_init_irq_lock_class();

/*
 * Interrupts are still disabled. Do necessary setups, then
 * enable them
 */
        lock_kernel();
        tick_init();
        boot_cpu_init();
        page_address_init();
```

Program stopped at line 544 | 41be4fc | 544

Now you can debug easily with the aid of GUI. You can set the break point, check the content of registers and also see the assembly code.

When you want to perform kernel debugging then only use the modified script otherwise use the default script.

7.2 Debugging User Application using gdbserver over Ethernet

Insight is the GUI of gdb debugger. Often it’s not easy to remember the debug Command. Therefore insight provides the graphical interface so user can debug easily. Aim of the document is to explain how to debug user space application. Steps are mentioned below:

For debugging user application over ethernet you should have proper zImage .To build zImage, refer the section SLS Ethernet IP Driver.

Here is the example of simple **helloworld.c** program for debugging:

1. Select **gdbserver (old)** in menuconfig under user application → miscellaneous application.

2. Select the **ifconfig** utility in your busybox.

3. On your Host PC, set environmental variable “PATH” to toolchain with the following command:

   ```
   #PATH=$PATH:/home/uClinux-2.1.1.1/Bintools/opt/
   nios2/bin ↵
   ```

   ```
   #export PATH ↵
   ```

4. Compile the application with debug symbols and no optimization on your host PC, Here is **helloworld.c** example. Your current working directory should have this **.c** file.

   ```
   #nios2-linux-uclibc-gcc -g helloworld.c -o helloworld
   -elf2flt ↵
   ```

   This will generate **Helloworld** and **Helloworld.gdb** files in your current working directory. Copy the generated Helloworld to **romfs/bin** in **uClinux-dist** directory.

5. Build the image, download it to the target and configure the Ethernet IP, 192.168.0.14 through the following command

   ```
   />ifconfig eth0 192.168.0.14 ↵
   ```

   Then start the gdbserver listening on an unused port eg. 9999, with command:

6. Apply the command given below:

   ```
   /> gdbserver localhost:9999 /bin/Helloworld ↵
   ```

7. It will display,

   ```
   Process /bin/Helloworld created; pid = 20
   ```

   ```
   Listening on port 9999
   ```

8. Next, on your Linux PC, in the hello source directory, run insight gdb,

   ```
   #nios2-linux-uclibc-insight Helloworld.gdb ↵
   ```

9. A source window will open and display the source **Helloworld.c**.

10. Open a gdb console with **View>Console**, enter gdb command in this window.

    ```
    gdb>target remote 192.168.0.14:9999 ↵  (board_ip:9999)
    ```

11. It will report the target address of the program. Now you can set break points and debug. On the target you can see the message: **Remote debugging from host IP**, here you will see your IP (Host IP) address.

    ```
    gdb> b main ↵  (insert break point at main)
    gdb> c ↵        (continue)
    ```

12. Now you can debug via command or through GUI of insight. Insight GUI is quite easy to understand. If you want to use command then you can use:

    ```
    gdb> s ↵       (for single stepping)
    gdb> r ↵       (to run program )
    ```

7.3 Debugging User application using Eclipse IDE

To debug the user application, follow the steps below.

1. Follow all steps mentioned in Ethernet driver selection, from Configuring Device Drivers and File Systems.
2. Build the uClinux kernel by selecting

    ```
    User application ---->
       miscellaneous application ---->
          [*] gdbserver(old).
    ```

3. Open the terminal. Change directory to the Eclipse IDE folder
4. Write the command on the terminal to open the Eclipse IDE.

    ```
    #./eclipse ↵
    ```

5. Type **workspace** to choose your workspace.
6. Select **File>New>C project** to Create a new project.
7. Enter **Helloworld** as your Project name.
8. Project types, executable, Hello world ANSI C Project, (or empty project and add your source).Finish.

9. Now, setup for nios2 tool chain in Eclipse IDE on the host by following the steps below:

   a) Select the project **Helloworld** under **Project Explorer**.
   b) Right click on **Helloworld** and select **Properties**. Project properties dialog box opens.
      - Select **Settings>GCC C Compiler** under **C/C++ build**. Make following settings:
        - Command: **nios2-linux-uclibc-gcc**
        - All options: **-O0 -g -Wall -c -fmessage-length=0**
      - Click **Apply**.
      - Select **Settings>GCC C Linker** under **C/C++ build** and make following settings
        - Command: **nios2-linux-uclibc-gcc**
        - All Options: **-elf2flt**
      - Select **Miscellaneous** under **GCC C Linker**
        - Linker flags: **-elf2flt**
      - Click **Apply**

   c) Select **Run/Debug Settings** under Project properties. Run/Debug settings dialog box opens
      - Click on **New** button. Select Configuration Type dialog box opens
        - Select C/C++ Local application.
      - Click **OK**. Properties for new Configuration dialog box opens.
      - Under **Main** tab do the following settings:
        - Browse the **Helloworld** project under project.
        - Type **Debug/Helloworld.gdb**
      - Select Debugger tab. Make following settings
        - Debugger: **gdbserver Debugger**
      - Under **Main** tab of Debugger options make following settings
        - GDB Debugger: **nios2-linux-uclibc-gdb**
      - Under **Connection** tab
        - Type: **TCP**
        - Host name or IP address: **192.168.0.14** (**IP address of CORECOMMANDER board)**
        - Port Number: **9999** (you can give any number)
      - Click **Apply** and click **OK**. You have finished setting of run/debug settings.
      - Click **Apply** and click **OK**. This finishes project properties settings.
   d) Select **Project>Build Configuration>Build>All** to build your project.
      - Select **workspace>Helloworld>Debug>Helloworld** and copy the file into */uclinux-dist/romfs/bin*.

e) Build the zImage as explained in early section.

f) Click **OK**.

10. Now, select **Project-->Build project** to compile your project.

11. Copy from your **workspace/Helloworld/Debug/Helloworld** to **romfs/bin** in uClinux-dist and build the zImage as explained earlier.

12. Configure the **SOF** on your board and download the **zImage**.

13. Kernel starts booting. After successful booting, configure the IP address of the board (same as entered in the Eclipse IDE). Here IP is: 192.168.0.14

```
/>ifconfig eth0 192.168.0.14 ↵
```

14. Now issue the command:

```
/>gdbserver localhost: 9999 /bin/Helloworld ↵
```

15. It will display:

```
Process /bin/Helloworld created; PID = 20
Listening on port 9999
```

16. Now move to the IDE on host.

    a) Select **Run>Open Debug Dialog**
        - Select **Helloworld** as Debug application
            - C/C++ Application: **Debug/Helloworld.gdb**
        - Click **Apply.**
        - Click **Debug**.

    b) If there is no error then debug perspective opens and it will establish a connection with the board. You will see the following message on the target.

    ```
    remote debugging from: 192.168.0.26 (your host ip)
    ```

17. Now you can debug easily with the help of step into/step over feature of IDE.

8. Demonstrations & Quick Reference

Download **Demonstrations and Quick Reference** from http://slscorp.com/pages/ccbspdownload.php.

8.1 Demonstrations

This demo package contains following applications:

- Picview
- ChatServer
- Portinterface

Following section explains how to run each application on the CoreCommander board.

8.1.1 Picview

Picview is a Video and SD Host Controller based application for CoreCommander board to display JPEG images and play MPEG files on LCD.

Picview directory contains following folders and files:

- SDCard_Contents
- CIII_zImage_Picview
- CIII_zImage_Picview.bat
- CIII_zImage_Picview_sh
- sls_cc_bsp_hw_v10.sof
- Readme.txt

Before going further, please refer **Readme.txt**.

Running Picview Application

Before running Picview application, make sure that SD Card contains JPEG and MPEG files at **\CC_Applications\JPEG_MPEG_Viewer** directory.

To run the Picview application, follow the steps below:

4. Plug the SD card into the socket (CoreCommander board).
5. Run **CIII_zImage_Picview.bat** file. You will see booting uClinux on JTAG UART console. When the zImage is up and running, you will see the Home page as shown in Figure 15 on the CoreCommander board.

Figure 15 : Picview Application Running on CoreCommander Board

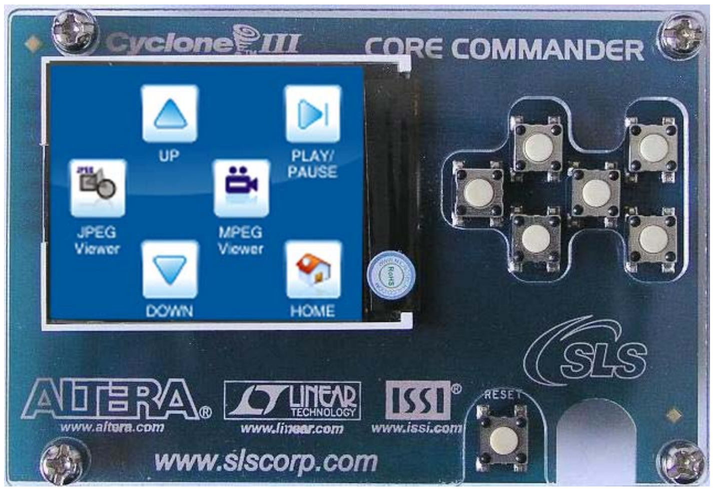
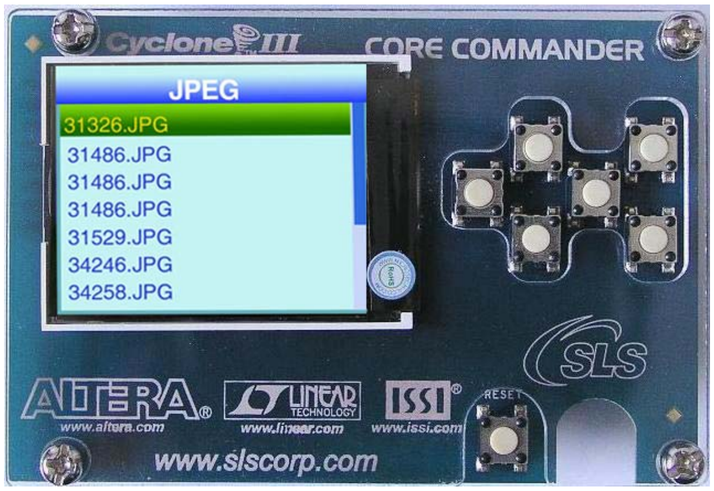


Table below lists the keys and its description:

Key	Description
JPEG Viewer	Pressing this key displays list of JPEG files in the SD Card on the CoreCommander LCD.
MPEG Viewer	Pressing this key displays list of MPEG files in the SD Card on the CoreCommander LCD.
Up	Moves up in the list of JPEG/ MPEG files
Down	Moves down in the list of JPEG/MPEG files
Play/Pause	Plays or pauses JPEG / MPEG files.
Home	Returns back to home page or stops playing MPEG file.

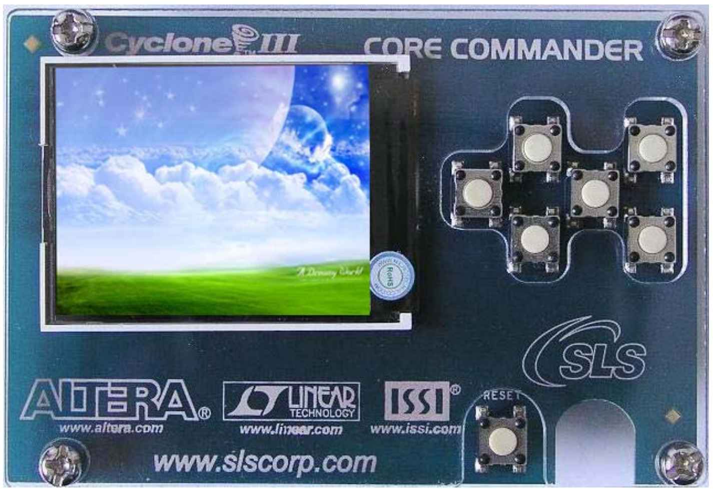
6. Press **JPEG/MPEG** viewer key.
7. You will see the list of **JPEG/MPEG** files on LCD. See Figure 16.

Figure 16 : JPEG File List



8. Press **Enter** key. It will display the selected JPEG/MPEG file on LCD. See Figure 17.

Figure 17 : Displaying JPEG Files



9. Now press **Up** and **Down** key to view previous and next JPEG/MPEG files.
10. Press **Home** key to go home page.

8.1.2 ChatServer

A chat server application allows users to transfer real time text between two or more client computers connected in the network.

ChatServer directory contains following files.

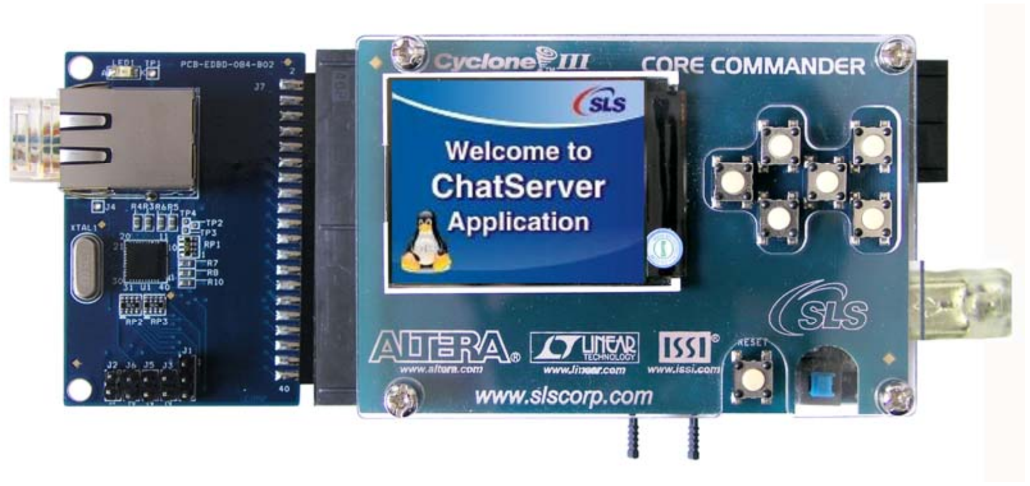
- CIII_zImage_eth0
- CIII_zImage_eth0.bat
- CIII_zImage_eth0_sh
- Readme.txt
- sls_cc_bsp_hw_v10.sof

Before going further, please refer **Readme.txt**.

Running ChatServer Application

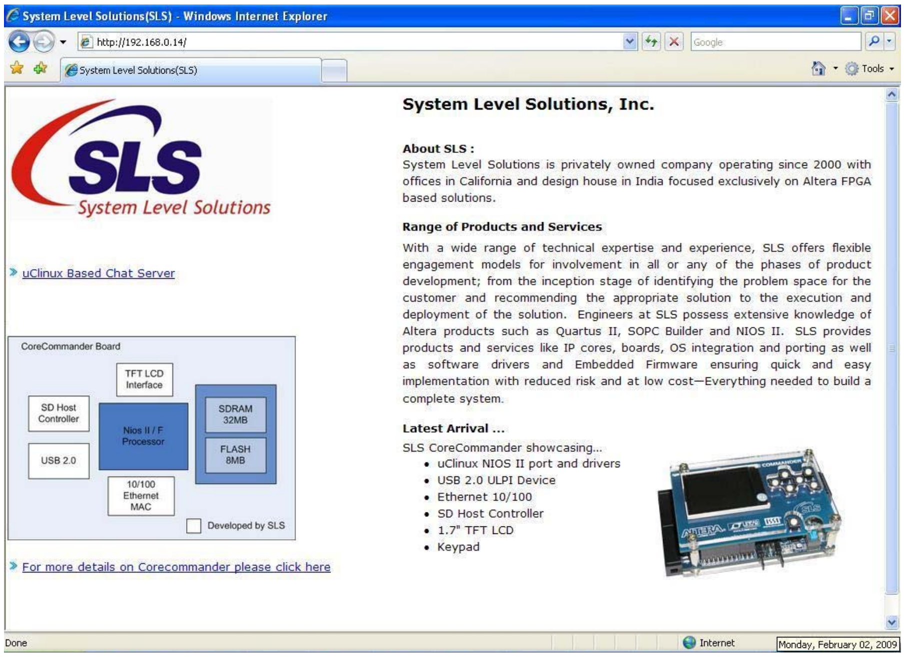
1. Mount the GPIO Connector (J7) of the Ethernet Add On board to the GPIO header (J4) of the CoreCommander board. See Figure 18.
2. Plug the Ethernet cable to the Ethernet connector on the Ethernet Add on board.
3. Run **CIII_zImage_eth0.bat** file, you will see booting uClinux on JTAG UART console. When the zImage is up and running, the board is configured as a chat server.
4. Now any computer in the network can connect with chat server and communicate with each other.
5. CoreCommander LCD displays startup screen as shown in Figure 18.

Figure 18 : CoreCommander Board Startup Screen Running ChatServer Appl.



6. Open any browser on client machine connected in the same network where the CoreCommander board is connected.
7. Type **IP address** of chat server in the URL bar and press **Enter**.
8. You will see the chat sever main page in the browser window. See Figure 19.

Figure 19 : Chat Server Application Running on Client Machine



System Level Solutions, Inc.

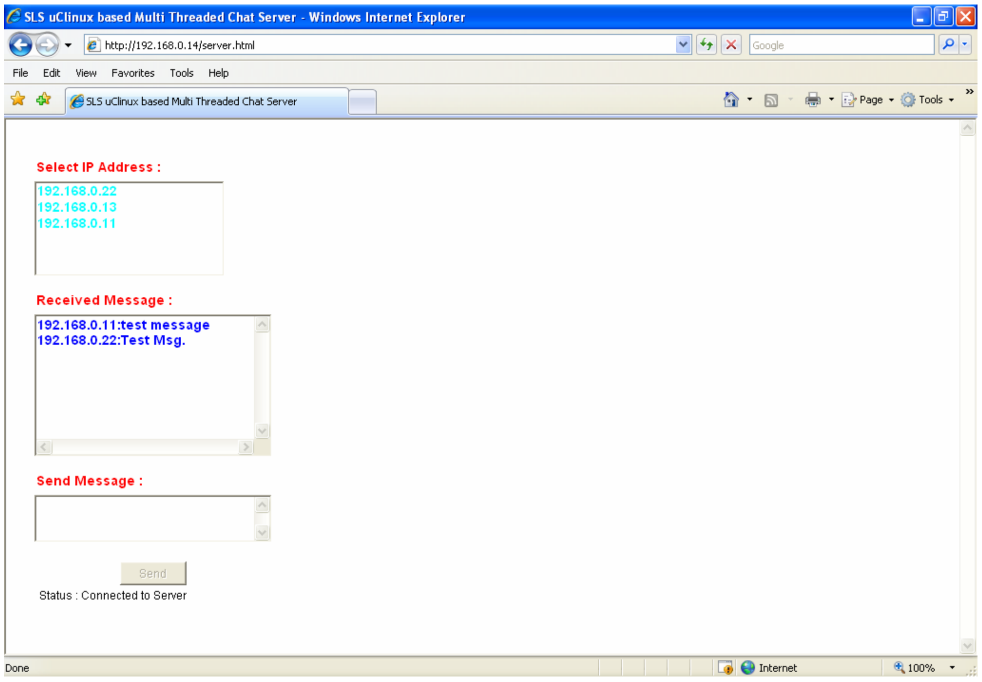
About SLS :
System Level Solutions is privately owned company operating since 2000 with offices in California and design house in India focused exclusively on Altera FPGA based solutions.

Range of Products and Services
With a wide range of technical expertise and experience, SLS offers flexible engagement models for involvement in all or any of the phases of product development; from the inception stage of identifying the problem space for the customer and recommending the appropriate solution to the execution and deployment of the solution. Engineers at SLS possess extensive knowledge of Altera products such as Quartus II, SOPC Builder and NIOS II. SLS provides products and services like IP cores, boards, OS integration and porting as well as software drivers and Embedded Firmware ensuring quick and easy implementation with reduced risk and at low cost—Everything needed to build a complete system.

Latest Arrival ...
SLS CoreCommander showcasing...
- uClinux NIOS II port and drivers
- USB 2.0 ULPI Device
- Ethernet 10/100
- SD Host Controller
- 1.7" TFT LCD
- Keypad

9. Now click on **uClinux based chat server**.
10. Chat application window opens in browser. See Figure 20
11. You will see the IP addresses of all client machines connected to the chat server See Figure 20
12. To chat with the client machine, select **IP address** of the client machine.
13. Type the message you want to send to the selected client machine in the Send Message Text box.
14. Received message area displays all messages.

Figure 20 : Running Chat Server


SLS uClinux based Multi Threaded Chat Server - Windows Internet Explorer
http://192.168.0.14/server.html
File Edit View Favorites Tools Help
SLS uClinux based Multi Threaded Chat Server
Page Tools
Done
Internet
100%

Select IP Address :

192.168.0.22
192.168.0.13
192.168.0.11

Received Message :

192.168.0.11:test message
192.168.0.22:Test Msg.

Send Message :

Send

Status : Connected to Server

8.1.3 PortInterface

PortInterface application is designed for reading, writing and verifying data on USB 2.0 device.

Figure 21 displays directory structure of PortInterface application.

Figure 21: PortInterface Application Directory Structure

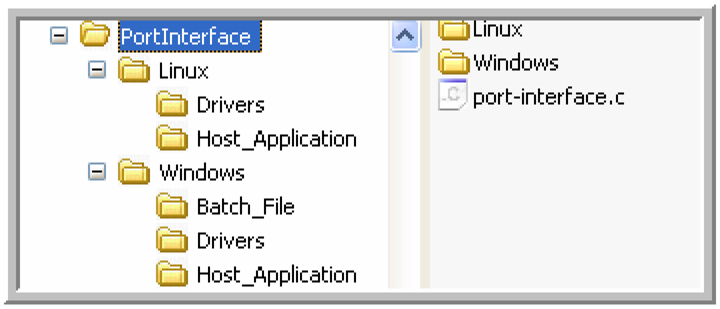


Table below gives description of directory contents.

Directory	Description
Linux	Contains Port Interface application files for Linux
Drivers	Contains USB 2.0 driver
Host_Application	Contains Port Interface application for Host PC
CIII_zImage_USB20	zImage file
Readme.txt	Readme file for running zImage and .sof file

sls_cc_bsp_hw_v10.sof	.sof file
Windows	Contains Port Interface application files for Windows
Drivers	Contains USB 2.0 driver
Host_Application	Contains Port Interface application for Host PC
Batch_File	Batch file to run zImage
port-interface.c	Portinterface application source code for CoreCommander board

Reading/Writing/Verifying Data From/to the USB 2.0 Device on Linux PC

Installing USB2.0 device driver on Linux PC

To install USB2.0 device driver, follow the steps below:

1. Copy the file **SLSUSB.ko** to **/lib/modules/kernel version/** directory.
2. Open the terminal and run **depmod** command to register your driver into **modules.dep** file.

To confirm the USB2.0 device driver installation, follow the steps below:

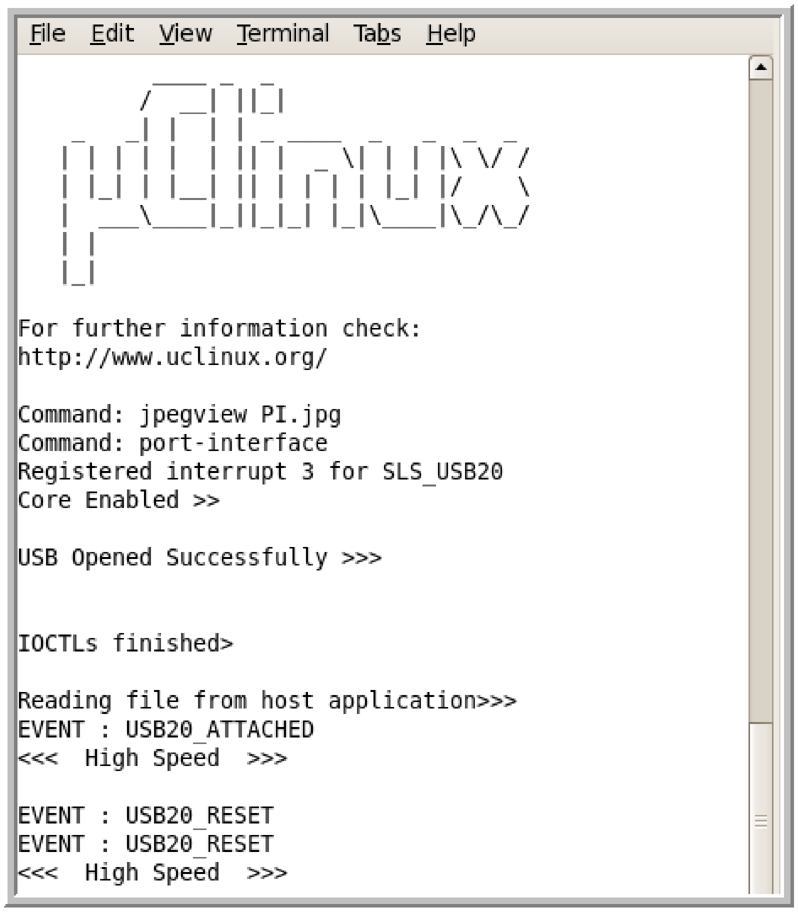
1. Open the **modules.dep** file. You will find the path of **SLSUSB.ko** file there.
2. Attach a device supported by this driver and check the **/dev** directory. It should be there with the name of **SLSUSB0**.

Downloading and Running zImage

1. Open the **Linux** terminal.
2. Download and run the **CIII_zImage_USB20** on the CoreCommander board. You will see the Linux terminal window as shown in Figure 22. Keep this terminal open.

To know how to communicate with the device using uClinux USB driver, refer the **port_interface.c** file located at /**PortInterface** directory

Figure 22 : Linux Terminal running zImage



```
        ____ _  _
       /  __| ||_|
  _   _| |  | | _ ____  _   _  _  _
 | | | | |  | || |  _ \| | | |\ \/ /
 | |_| | |__| || | | | | |_| |/    \
 |  ___\____|_||_|_| |_|\____|\_/\_/
 | |
 |_|

For further information check:
http://www.uclinux.org/

Command: jpegview PI.jpg
Command: port-interface
Registered interrupt 3 for SLS_USB20
Core Enabled >>

USB Opened Successfully >>>


IOCTLs finished>

Reading file from host application>>>
EVENT : USB20_ATTACHED
<<<  High Speed  >>>

EVENT : USB20_RESET
EVENT : USB20_RESET
<<<  High Speed  >>>
```

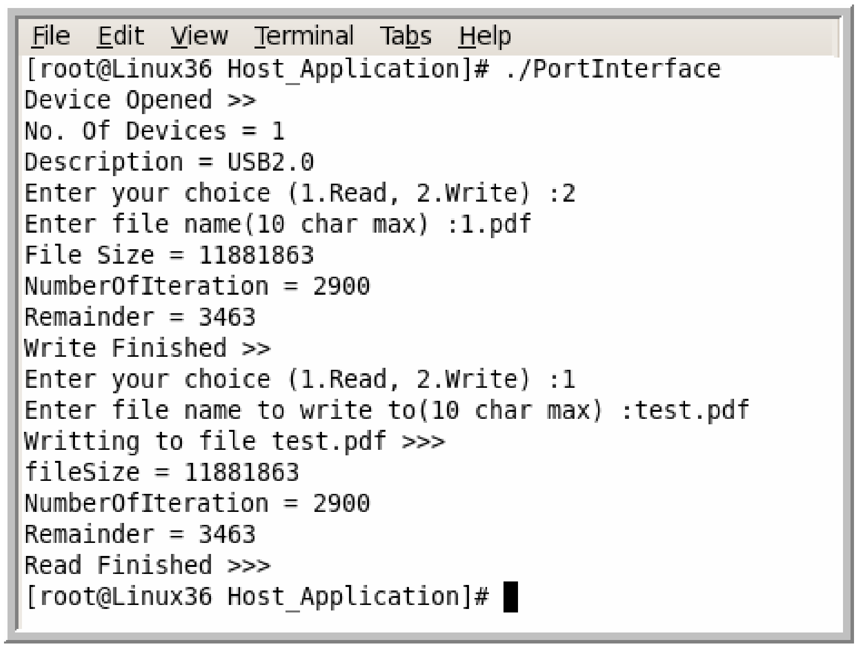
Running PortInterface Host Application

To run the PortInterface Host application, follow the steps below:

1. Open another Linux terminal.
2. Browse to the directory **PortInterface/Linux/Host_Application/.** You will find PortInterface file.
3. Issue the following command:

   ```
   ./PortInterface ↵
   ```
4. You will see the device description. See Figure 23.

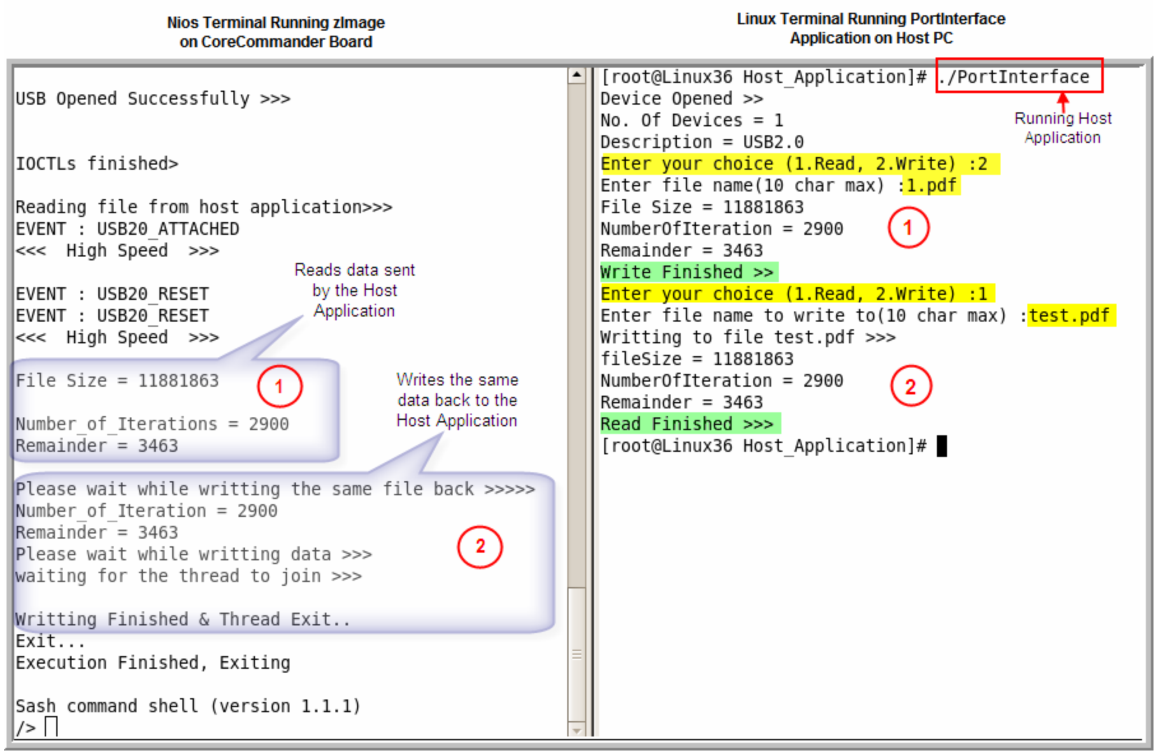
Figure 23 : Linux Terminal Running PortInterface Host Application



```
[root@Linux36 Host_Application]# ./PortInterface
Device Opened >>
No. Of Devices = 1
Description = USB2.0
Enter your choice (1.Read, 2.Write) :2
Enter file name(10 char max) :1.pdf
File Size = 11881863
NumberOfIteration = 2900
Remainder = 3463
Write Finished >>
Enter your choice (1.Read, 2.Write) :1
Enter file name to write to(10 char max) :test.pdf
Writting to file test.pdf >>>
fileSize = 11881863
NumberOfIteration = 2900
Remainder = 3463
Read Finished >>>
[root@Linux36 Host_Application]# 
```

5. It prompts for **choice (1. Read, 2. Write)**.
6. Type **2** to write data on USB device.
7. It again prompts for the **filename** which you want to write. Make sure that the file should be present in the host application directory.
8. It starts writing data from the file to the device. You will see the result on Linux terminal as shown in Figure 24
9. After writing data, again it prompts for choice **(1. Read, 2. Write)**.
10. Now type **1** to read data from the device.
11. Type the **file name** on which you want to write the read data from the device.
12. New file will be created with the given name at the same directory.
13. Verify both files.
14. Final view of both the terminals will be as shown in Figure 24.

Figure 24 : Final View of Nios Terminal and Linux Terminal


Nios Terminal Running zImage on CoreCommander Board
Linux Terminal Running PortInterface Application on Host PC
Reads data sent by the Host Application
Writes the same data back to the Host Application
Running Host Application
1
2

```
USB Opened Successfully >>>


IOCTLs finished>

Reading file from host application>>>
EVENT : USB20_ATTACHED
<<<  High Speed  >>>

EVENT : USB20_RESET
EVENT : USB20_RESET
<<<  High Speed  >>>

File Size = 11881863

Number_of_Iterations = 2900
Remainder = 3463

Please wait while writting the same file back >>>>>
Number_of_Iteration = 2900
Remainder = 3463
Please wait while writting data >>>
waiting for the thread to join >>>

Writting Finished & Thread Exit..
Exit...
Execution Finished, Exiting

Sash command shell (version 1.1.1)
/>
```

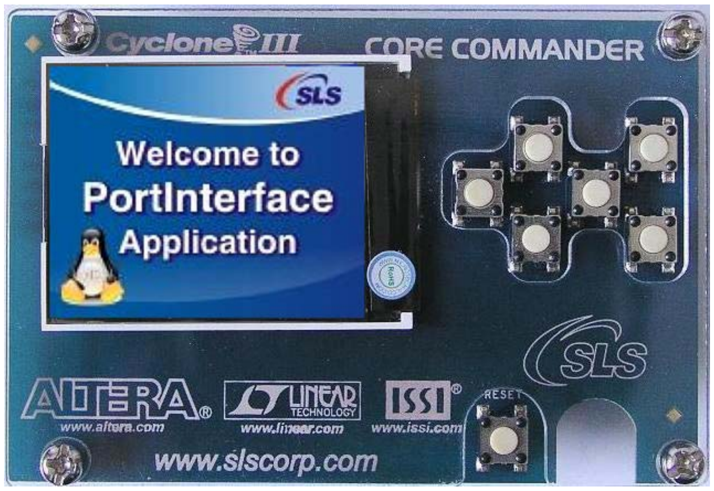
```
[root@Linux36 Host_Application]# ./PortInterface
Device Opened >>
No. Of Devices = 1
Description = USB2.0
Enter your choice (1.Read, 2.Write) :2
Enter file name(10 char max) :1.pdf
File Size = 11881863
NumberOfIteration = 2900
Remainder = 3463
Write Finished >>
Enter your choice (1.Read, 2.Write) :1
Enter file name to write to(10 char max) :test.pdf
Writting to file test.pdf >>>
fileSize = 11881863
NumberOfIteration = 2900
Remainder = 3463
Read Finished >>>
[root@Linux36 Host_Application]#
```

Reading/Writing/Verifying Data from/to the USB 2.0 Device on Windows PC

Downloading and Running zImage for Windows

Run **CIII_zImage_USB20.bat** file. After successful downloading of zImage you will see the Startup screen as shown in Figure 25.

Figure 25 : CoreCommander board Startup Screen Running PortInterface Appl.



If the USB 2.0 driver is not installed on your PC then New Hardware Found Wizard will be displayed.

To install the USB 2.0 driver follow the steps below:

1. Follow the onscreen installation instructions; choose default options.
2. While asking to select the device driver, browse to the directory **\PortInterface\Windows\Drivers**.
3. Click **Finish** to finish the installation.

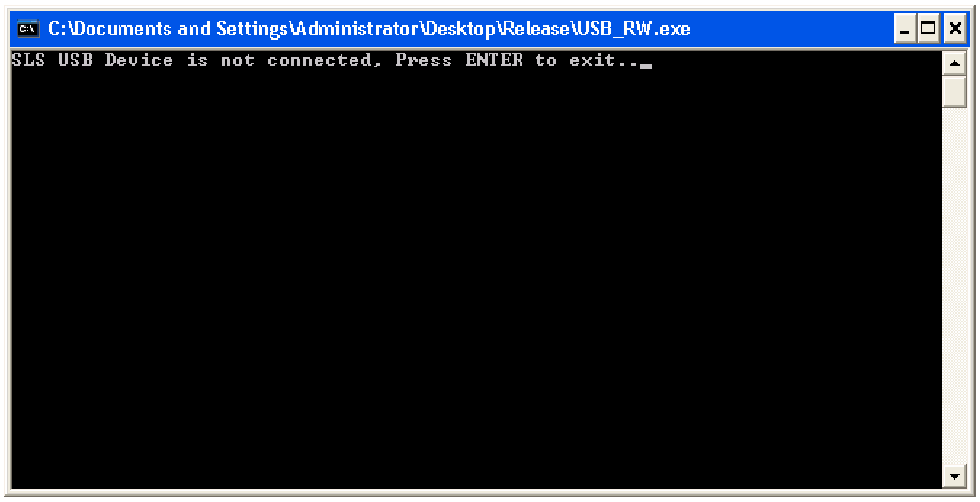
To know how to communicate with the device using uClinux USB driver, refer the **port-interface.c** file located at **\PortInterface** directory.

Running Host Application on Windows PC

1. If the Microsoft Visual Studio 6.0 is not installed on your PC then run **USB_RW.exe** located at **\PortInterface\Windows\Release**. If the Microsoft Visual Studio 6.0 is installed on your PC, then double-click on **USB_RW.dsw** located at **\PortInterface\Windows\Host_Application**. It opens the host application project. Run the host application from Visual Studio.

   If the device is not properly connected, you will see the following message in the window. See Figure 26.

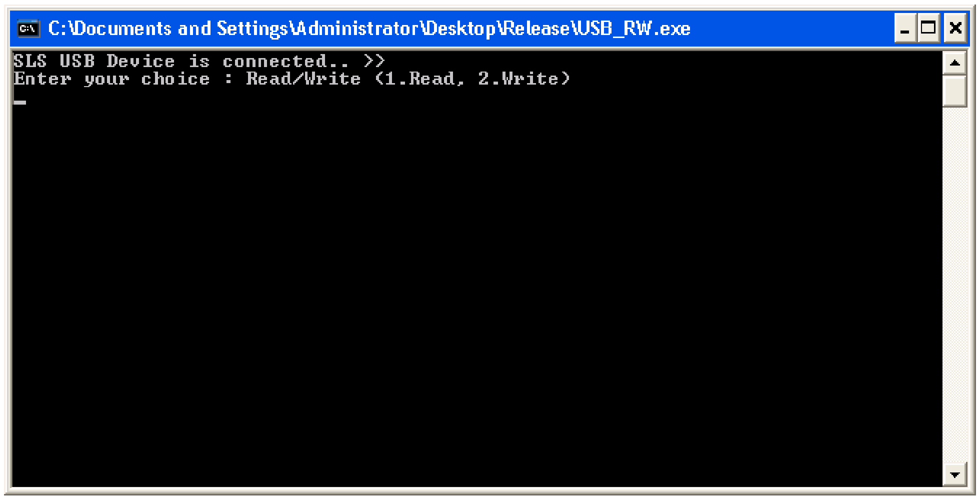
   Figure 26 : Running Host Application when USB Device is not connected


C:\Documents and Settings\Administrator\Desktop\Release\USB_RW.exe

```
SLS USB Device is not connected, Press ENTER to exit..
```

   If the device is properly connected you will see the following message in the window.

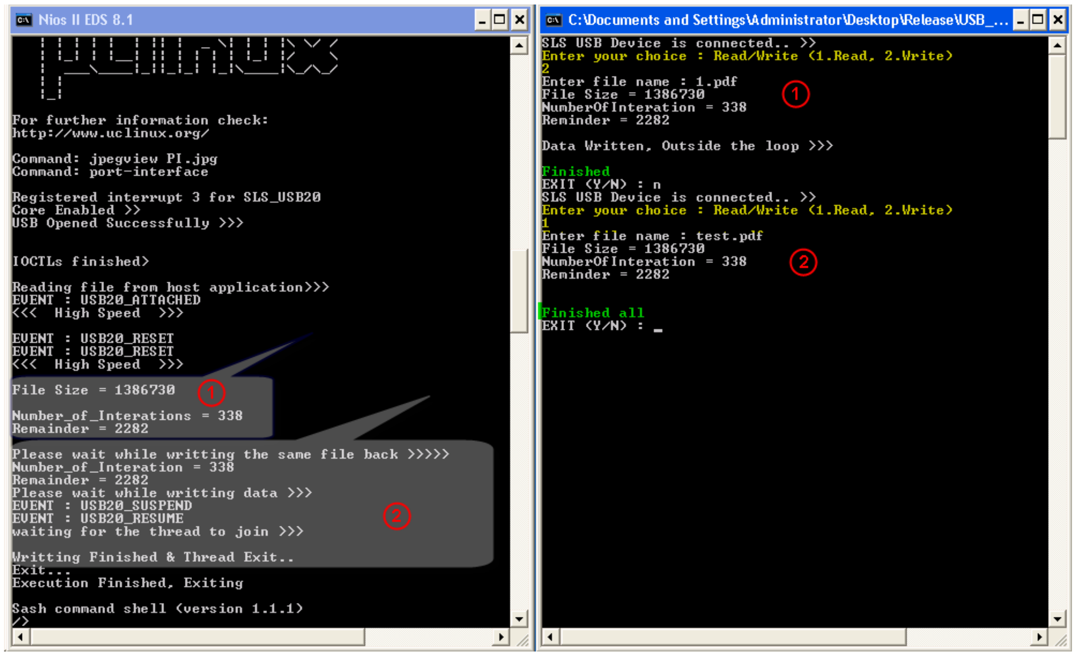
Figure 27 : Running Host Application when Device is connected


C:\Documents and Settings\Administrator\Desktop\Release\USB_RW.exe

```
SLS USB Device is connected.. >>
Enter your choice : Read/Write (1.Read, 2.Write)
_
```

2. It prompts for **choice (1. Read, 2. Write)**.
3. Type **2** to write data on USB device.
4. It again prompts for the **filename** which you want to write. Make sure that the file should be present in the host application directory.
5. It starts writing data from the file to the device. You will see the result on Nios terminal. See Figure 28.
6. After writing data, again it prompts to exit from the application. Select **N.**
7. Again, it prompts for choice **(1. Read, 2. Write)**.
8. Type **1** to read data from the device.
9. Type the **file name** on which you want to write the read data from the device.
10. New file will be created with the given name at the same directory.
11. Verify both files.
12. Final view of both the terminals will be as shown in Figure 28 below:

Figure 28 : Final View of Nios Terminal and Host Application Running on Windows


Nios II EDS 8.1
C:\Documents and Settings\Administrator\Desktop\Release\USB_...
1
2

```
For further information check:
http://www.uclinux.org/

Command: jpegview PI.jpg
Command: port-interface

Registered interrupt 3 for SLS_USB20
Core Enabled >>
USB Opened Successfully >>>


IOCTLs finished>

Reading file from host application>>>
EVENT : USB20_ATTACHED
<<<  High Speed  >>>

EVENT : USB20_RESET
EVENT : USB20_RESET
<<<  High Speed  >>>

File Size = 1386730

Number_of_Interations = 338
Remainder = 2282

Please wait while writting the same file back >>>>>
Number_of_Interation = 338
Remainder = 2282
Please wait while writting data >>>
EVENT : USB20_SUSPEND
EVENT : USB20_RESUME
waiting for the thread to join >>>

Writting Finished & Thread Exit..
Exit...
Execution Finished, Exiting

Sash command shell (version 1.1.1)
/>
```

```
SLS USB Device is connected.. >>
Enter your choice : Read/Write (1.Read, 2.Write)
2
Enter file name : 1.pdf
File Size = 1386730
NumberOfInteration = 338
Reminder = 2282

Data Written, Outside the loop >>>

Finished
EXIT (Y/N) : n
SLS USB Device is connected.. >>
Enter your choice : Read/Write (1.Read, 2.Write)
1
Enter file name : test.pdf
File Size = 1386730
NumberOfInteration = 338
Reminder = 2282


Finished all
EXIT (Y/N) : _
```

If you want to make your own zImage using given *.c* files in the PortInterface directory, you need to modify the **sls_cc_bsp_hw_v10_sopc.ptf** file and recompile uClinux by referring Set System.ptf.

Follow the steps below to modify the **.ptf** file:

1. Modify the following code at line no 2629

   ```
   Address_Span = "33554432";
   ```

   With

   ```
   Address_Span = "16777216";
   ```

2. Modify the following code at line no 2638

   ```
   Address_Width = "24";
   ```

   With

   ```
   Address_Width = "23";
   ```

8.2 Quick Reference

Quick Reference gives you quick overview of the applications explained in chapter 5 and 6. When you extract Quick Reference, you will find following directory:

- Prebuilt_zImage

8.2.1 Prebuilt_zImage

Prebuilt_zImage directory contains following files:

- zImage
- sls_cc_bsp_hw_v10.sof
- readme.txt

Before going further, please refer the readme.txt.

SD Card Formatting

Download **.sof** and **zImage**. You will get console on JTAG UART. When the zImage is up and running, plug the SD card into the socket (CoreCommander board). Issue the following command to format the SD card.

```
/>mkdosfs –I –F 16 /dev/mmcblk0 ↵
```

Other Applications

zImage given here is a ready reference to test all applications explained in chapter 5 and 6. See User Applications and Configuring Network utilities and NFS (Client) for more details. Using this zImage you can only explore some of the capabilities of BSP.

Appendix A. Ethernet Driver

As kernel can only access Ethernet driver, there is not any specific API to access Ethernet driver. There are different ways to allocate an Ethernet interface to the driver.

To allocate Ethernet interface to the driver, you have to configure the interface. Once the interface is configured, user can send/receive packets into the network.

To allocate an Ethernet interface to the driver, use **ifconfig** utility.

Example:

```
/>ifconfig eth0 192.168.0.14 multicast

ETH0: Registered SLS MAC interrupt 5
```

Above example uses Ethernet driver interface **eth0** with IP number **192.168.0.14**. To verify interface settings, use **ifconfig** utility.

Example:

```
/>ifconfig

eth0

    Link encap: Ethernet HWaddr 12:12:12:12:12:12

    inet addr:192.168.0.14  Bcast:192.168.0.255
    Mask:255.255.255.0

    UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

    RX packets:76 errors:4 dropped:0 overruns:0
    frame:8

    TX packets:0 errors:0 dropped:0 overruns:0
    carrier:0

    collisions:0 txqueuelen:1000

    RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

    Base address: 0x1000
```

To ping any PC from Network, connect CoreCommander board to the Network with RJ-45 cable but if you are connecting a PC to the CoreCommander then use RJ-45 cross cable.

Example:

```
    />ping -c 2 192.168.0.13

    PING 192.168.0.13 (192.168.0.13): 56 data bytes
```

```
64 bytes from 192.168.0.13: icmp_seq=0 ttl=128
time=0.0 ms

64 bytes from 192.168.0.13: icmp_seq=1 ttl=128
time=0.0 ms

--- 192.168.0.13 ping statistics ---

2 packets transmitted, 2 packets received, 0%
packet loss

round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Appendix B. SD Host Controller Driver

SD Host Controller Block driver can’t be accessed directly from the user space. Kernel can manage read/write block. i.e. there is not any specific API to access SD card Blocks.

To access SD Card blocks, mount SD Card with proper file system using mount utility. Before mounting SD Card, insert the SD card into the CoreCommander board.

Example:

```
>mount -t vfat /dev/mmcblk0 /mnt ↵
```

Example below shows the vfat file system is mounted on the SD card. Now, user can access Blocks as a file.

Sometimes, when user uses this command on console, mounting fails and returns with the following message.

Example:

```
/>mount -t vfat /dev/mmcblk0 /mnt ↵

FAT: bogus number of reserved sectors

VFS: Can't find a valid FAT file system on dev
SLS_SD.
```

Error occurs due to the file system mounted on the SD Card is supported by windows only and not by uClinux. To solve this problem format the SD Card with **dosfs** in Dos and in uClinux, select **mkdosfs** utility from user application in uClinux-dist.

Example

```
Filesystem Applications --->

     --- MSDOS

     [*] mkdosfs

     [ ] dosfsck
```

When you build uClinux, you will get **mkdosfs** utility in your romfs. Give following command on the console window to format the SD card.

Example

```
/>mkdosfs -I -F 16 /dev/mmcblk0 ↵

mkdosfs 2.10 (22 Sep 2003)
```

Appendix C. USB 2.0 Driver

Use SLS USB 2.0 driver API to open, write and read data to/from the USB device.

Example below shows how to open the driver.

Example:

```
usb_fd = open("/dev/usb_20", O_RDONLY |O_NOCTTY);
if(usb_fd < 0)
 printf("Sorry, No USB found >>>\n");
else
 printf("USB Opened Successfully >>\n\n\n");
```

Before reading or writing data to/from the USB device, configure the driver. To configure the driver, first set all the variables defined in the following structures and pass the address of these structures to ioctl () with appropriate macro arguments.

Structure Definition:

```
typedef union
{
  unsigned int temp;
  struct
  {
    unsigned int MAX_PL_SIZE : 11;
    unsigned int TRANSFER_FRAME : 2;
    char         RESERVED3 : 3;
    unsigned int SMALL_OK : 1;
    unsigned int LARGE_OK : 1;
    unsigned int EP_NO : 4;
    unsigned int EP_STAT : 2;
    unsigned int EP_TRNASFER_TYPE : 2;
    unsigned int EP_TYPE : 2;
    unsigned int DATA_PID : 2;
    unsigned int BUF_SEL : 2;
  }bitmap;
}EP_CSR;
```

Table 1 : EP_CSR Structure Description

Field	Description
MAX_PL_SIZE	Maximum Payload Size High Speed = 512 bytes Full Speed = 64 bytes
TRANSFER_FRAME	Number of transaction per microframe.
RESERVED3	Reserved

SMALL_OK	1 - Accept data packets of less than MAX_PL_SZ (RX only) 0 - Ignore data packets of less than MAX_PL_SZ (RX only)
LARGE_OK	1 - Accept data packets of less than MAX_PL_SZ (RX only) 0 - Ignore data packets of less than MAX_PL_SZ (RX only)
EP_NO	Endpoint Number
EP_STAT	Endpoint Status
EP_TRNASFER_TYPE	Transfer Type 00 : Interrupt 01 : Isochronous 10 : Bulk 11 : Reserved
EP_TYPE	Endpoint Type 00 : Control Endpoint 01 :IN EndPoint 10 : OUT Endpoint 11 : Reserved
DATA_PID	These two bits are used by the USB core to keep track of the data PID for high speed endpoints and for DATA0/DATA1 toggling.
BUF_SEL	Buffer Select 00 : Buffer 0 01 : Buffer 1 1x :Reserved

Example:

```
ioctl(usb_fd, CONFIG_ENDPOINT_CSR, &<Structure
Name> );
```

`Where Structure Name is variable of type` **`EP_CSR`**`.`

Structure Definition:

```
typedef struct
{
 unsigned int No;
 unsigned int call_type;          //polling or irq
 unsigned int fn_ptr;
 void *ptr;
}Config_Driver;
```

Table 2 : Config_Driver Structure Description

Field	Description
No	Endpoint Number
call_type	polling based = 1 & irq based = 2
fn_ptr	NULL if call_type = 1 User function pointer if call_type = 2
ptr	NULL if call_type = 1 Buffer Pointer if call_type = 2

Example:

```
ioctl(usb_fd, USB20_CONFIG_DRIVER, &<Structure
Name>);Where Structure Name is variable of type
Config_Driver.
```

Structure Definition:

```
typedef struct
{
 volatile unsigned int b_size_hs;
 volatile unsigned int b_size_fs;
 volatile char ep_no;
 unsigned char *p_buff;
}Config_Buffer;
```

Table 3 : Config_Buffer Structure Description

Field	Description
b_size_hs	Buffer Size in High Speed Mode
b_size_fs	Buffer Size in Full Speed Mode
ep_no	Endpoint Number.
p_buff	Buffer Pointer

Example:

```
ioctl(usb_fd, USB20_CONFIG_BUFFERSIZE, &<Structure
Name>);
Where Structure Name is variable of type
Config_Buffer.
```

After configuring the driver, If you want to define your own error handler function then you can define and pass the address of the same function to the ioctl(), otherwise you can use the default error handling function given in the *port-interface.c* example file located at */PortInterface*.

Example:

```
ioctl(usb_fd, USB20_CONFIG_ERROR_FUNCTION, &<Error
Handling function Name>);
```

If you don’t want to use the error handlilng mechanism then you can omit it.

Now, you can write/read data to/from device using Bulk-in/out operation by defining appropriate values to all the variables inside **Transfer_details** structure.

Structure Definition:

```
typedef struct
{
  unsigned char ep_no;
  unsigned char Transfer_direction;
  unsigned char *d_Buffer;
  unsigned int size;
  unsigned int d_completed;
  unsigned char use_dma;
}Transfer_details;
```

Table 4 : Transfer_details Structure Description

Field	Description
ep_no	Endpoint Number
Transfer_direction	Bulk out = 1 Bulk in = 0
d_Buffer	Data Buffer Pointer
size	Size of Data
d_completed	No. of Data bytes transfered by driver. Default value is 0.
use_dma	DMA Use status use_dma=1 (uses DMA) use_dma=0 (driver will not use DMA)

Example:

```
Transfer_details t_details;
Char Read_Buff[4];

t_details.ep_no = 2;
t_details.Transfer_direction = 1;
t_details.d_Buffer = Read_Buff;
t_details.size = 4;
t_details.d_completed = 0;
t_details.use_dma = 1;
ioctl(usb_fd, DATA_TRANSFER_REQUEST,&t_details );
```

This code will request the host to bulk out 4 bytes in *Read_Buff* buffer on endpoint 2 using DMA.

To get more information about USB 2.0 driver API, refere **port-interface.c** example file located at *\Demo\PortInterface*.