Running on uClinux on DE2 Board

EE8205: Lab Project Level-II

1. Introduction

The goal of the lab-project is to build an embedded ucLinux on the Altera Cyclone 2 DE2 development board. We used Altera Quartus 6.1 tools and SOPC builder to build a NIOS II CPU and various controller interfaces to interact with the peripherals on the DE2 board. The embedded system will include uClinux kernel 2.6.11 with networking support, and some applications can be implemented at the top of ucLinux.

2. Procedures

This section will outline the steps in a tutorial manner for anyone wishing to reproduce this project.

2.1 Altera DE2

The project will be implemented on the DE2 development board. The board has a Cyclone 2 FPGA and various peripherals. We will make use of the Audio and Ethernet controllers. The DE2 board has a DM9000A Ethernet controller for network connection and a WM8731 Wolfson Audio Codec for audio playback. We will also make use of the 8MB of SDRAM and 4MB of Flash memory available on the DE2 board for storing and execution of our uClinux kernel and applications.

The NIOS II CPU along with all the hardware controllers, such as SDRAM controller, Common Flash interface controller, Ethernet Controller, and Audio controller will be implemented in the Cyclone II FPGA on the DE2 board. We will use Quartus 6.1 and SOPC Builder to accomplish this.

The CD that came with the DE2 board provides demonstration projects. Any of the demonstration projects can be used as a base to start your own project. For example, we will use the project in the folder: "\DE2_demonstrations\DE2_SD_Card_Audio\". The step to do this is as follow:

1. If you don't have the DE2 board CD-ROM that came with the board, you can download it at: ftp://ftp.altera.com/up/pub/de2/DE2_System_v1.2.zip

2. Extract the zip file to anywhere you like. Start Quartus 6.1. This can be done by using the command "quartus6" in a terminal. Quartus 6.1 is no more available in the lab try to higher level available Quartus-II and explore this project.

3. In quartus, go to "File->Open Project…" and navigate to "..\DE2_demonstrations\DE2_SD_Card_Audio\" and select the project file called "DE2_SD_Card_Audio". Once this file is opened, you can start SOPC builder under "Tools->SOPC Builder"

4. When SOPC Builder is loaded, you should see that it already have all the necessary component of the DE2 board selected. This is shown in figure 1.

5. You can choose to unselect the components you do not need or you can choose to leave it as it is. If you decide to unselect some components, you have to also edit the top verilog file in quartus for this project "DE2_SD_Card_Audio.v", and remove all the declarations of those components you have unselected. I suggest keeping all the components selected until you are more familiar with all of this.

6. Once you are satisfied with the components, make sure to go (in SOPC builder) to "System->Auto-assign base addresses" and "System->Auto-assign IRQs".

7. Note: make sure to rename the Ethernet Controller from DM9000A to dm9000, otherwise you will get compilation errors later on with uClinux. You have to also change DM9000A to dm9000 in the "DE2_SD_Card_Audio.v" file in "..\DE2_demonstrations\DE2_SD_Card_Audio\".

8. Build it by clicking on "Generate". This should create a file called "system_0.ptf" in the "DE_SD_Card_Audio" folder. This file will be needed later when we want to compile uClinux for our system.

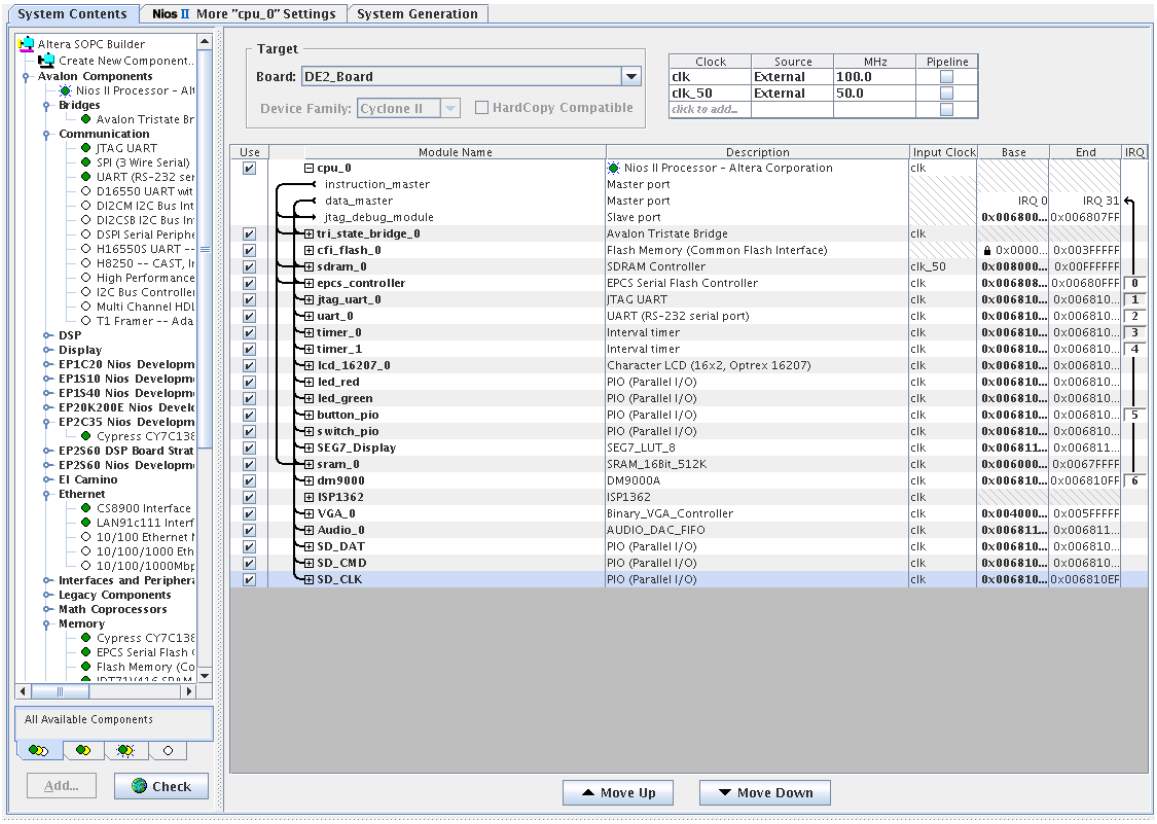

Figure 1 – SOPC with NIOSII and controllers.

9. Now go back to the Quartus window, go to "Process->Start compilation". Once compilation is completed without errors, you can program the image onto the DE2 board.

10. Go to "Tools->programmer". Choose the "USB-Blaster" for hardware setup, JTAG for mode, select the "DE_SD_Card_Audio.sof" and check "Program/Configure", then hit start to begin programming. Once the programming is done, the FPGA on the DE2 board now have the NIOS II CPU with all the other components you have selected. The next step now is to compile uClinux with the necessary drivers.

2.2 uClinux

The embedded system will have uClinux as the operating system. The reason why we need an operating system is because uClinux already have internet capabilities. If we didn't use uClinux, we would have to code our own drivers to handle the TCP/IP protocol. We use uClinux 2..6.11 which have support for our Ethernet Controller (DM9000A). To compile our uClinux image, we did the following:

1. Open a terminal and set up the environment with the following commands:
   "Source /usr/local/nios2-linux-tools/nios2-linux-tools.sh"
   "export ARCH=nios2nommu"
   "export CROSS_COMPILE=nios2-linux-uclibc- "
   "export CROSS_COMPILER_PREFIX=nios2-linux-uclibc-"

2. If your account has 300mb of space free, you can create a folder in your home directory called "uclinux" by typing "mkdir uclinux". If you don't have enough space, you can use the "tmp" folder. The data in this folder are erased when you log off, but it provides you with a lot of space to work it. Go to the "tmp" folder by typing "cd /tmp/". Then make a folder called "uclinux" by typing "mkdir uclinux". Then go into the "uclinux" folder by typing "cd uclinux".

3. Now we need to copy the uClinux kernel source code into our "uclinux" folder. To do this, type "cp -r /usr/local/nios2-linux-tools/uclinux-2.6.11 ./"

4. You now need to copy the "system_0.ptf" we have created earlier from SOPC builder and put into "uclinux/uclinux-2.6.11/" folder. Once this is done, have your terminal be in "uclinux/uclinux-2.6.11/" and type "make hwselect SYSPTF=system_0.ptf".

5. The above command will start to configure the uClinux to work with our embedded system we have already created using Quartus and SOPC Builder. It will ask you to choose the CPU, FLASH, and SDRAM. Choose cpu_0 for cpu, choose cfi_flash_0 for where the kernel will be uploaded to, choose sdram_0 as where to execute the kernel from.

6. Now we can select more options for the kernel by typing “make menuconfig”. It is important to unselect any device drivers that are not needed and are not in your list of components (from the SOPC Builder); otherwise there will be compilation errors.

7. Since we want to have driver support for our Ethernet chip. We have to enable these options under device drivers.
   Device Drivers -->Network device support —>
   Networking options --->
   <*> Packet socket
   <*> Unix domain sockets
   [*] TCP/IP networking
   [*] Network device support
   [*] Ethernet (10 or 100Mbit)
   [*] DM9000 support

8. We also want to enable
   Device Drivers -->Character Devices->Serial Drivers
   [*]Nios serial Support
   <*>Altera JTAG UART support
   [*]Support for console on Altera JTAG UART

9. Exit and save. Before we compile the uClinux kernel, we have to modify the network driver. Open “uclinux/uclinux-2.6.11/drivers/net/dm9000x.c” with any text editor. Find all occurrences of (2 occurrences usually at the beginning of the file)
   “#define DM9000_STD_DELAY_uSecs 0” and replace it with “#define DM9000_STD_DELAY_uSecs 20”. This sets the delay to 20usec between read/write commands for the network interface and it is required to get the network interface to work.

10. Now we compile uClinux with our options by typing “make”. If there are compilation errors, it is probably due to options selected in the “make menuconfig” stage. For example, if you build a system without a DM9000A Ethernet Controller in your SOPC Builder, and then choose to build uClinux with DM9000 Ethernet support, the compiler will give you an error such as “na_dm9000 undeclared…” To fix the error, unselect the device drivers that are causing the error by going through “make menuconfig” again.

11. Once compilation is done without errors, an executable file called “vmlinux” is created. We need to convert this file into a binary file by typing “nios2-linux-uclibc-objcopy -O binary vmlinux linux.bin”. We will need the “linux.bin” file for later, so copy it to your “uclinux” folder.

2.3 File-system

We need to have a file-system for the uClinux operating system. The file-system will consists of drives and tools and commands, etc. We will create a file-system by doing the following:

1. Go to your "uclinux" folder and create a new folder called "apps" by typing "mkdir apps".

2. Go into your "apps" folder and type "cp -r /usr/local/nios2-linux-tools/uclinux-apps/* ./". This will copy the files we need.

3. We need to prepare the environment with the command "export LINUX_NE_ROOT=/xxx/uclinux/apps". IMPORTANT: replace "/xxx/uclinux/apps" with the full path to your "apps" folder.

4. Go to the "apps" folder, type "make menuconfig". Here you can also choose what tools and utilities you want in your file-system. You need to disable busybox to save space. Save and exit.

5. Type "make" to compile.

6. Once compilation is done without error, type "make romfs". This will create a directory called "target" in the "apps" folder. The "target" folder is your actual file-system.

7. You need to edit the files "rc" and "inittab" in the "../apps/target/etc/" folder. The "rc" file contains commands to mount directories, and the "inittab" file contains the command to open a shell terminal when uClinux starts up. You can add your own start up commands in the "rc" file too. Example: setting up the ip address of eth0 by adding "ifconfig eth0 192.168.1.2" into the "rc" file. Check Appendix B for the "rc" and "inittab" file.

8. At this stage, if you wish to include the mpg123 mp3 player program into the file system, jump to section 2.4. Do all the steps in section 2.4, then resume with the next step.

9. Now type "make image" to turn the "target" folder into "romdisk.img". Copy the "romdisk.img" into your "uclinux" folder where "linux.bin" is at.

2.4 Linux Application

The application should be compiled targeting our uClinux kernel. Assuming that we use mpg123 mp3 player, which is an open source mp3 player. The mpg123 MP3 player is small and is able to play Shoutcast streams. A dummy soundcard driver will need to be coded to send data to the audio controller hardware. Here are the steps in compiling the application (e.g. mpg123) source code to work with uClinux:

1. Go to http://www.mpg123.de/ and download the source code.
2. Extract it to "/uclinux/mpg123/".
3. Open a terminal and set up the environment with the following commands:
   "Source /usr/local/nios2-linux-tools/nios2-linux-tools.sh"
   "export ARCH=nios2nommu"
   "export CROSS_COMPILE=nios2-linux-uclibc- "
   "export CROSS_COMPILER_PREFIX=nios2-linux-uclibc-"
4. Navigate to "/uclinux/mpg123/". We need to configure it before we compile. To do so, type:

   "./configure -host=nios2-linux-uclibc -target=nios2-linux-uclibc -with-cpu=generic_nofpu --with-audio=dummy CC="nios2-linux-uclibc-gcc -D__KERNEL__ -elf2flt" --with-optimization=4 --enable-gapless=no --prefix=/home/grad/m4hoang --enable-shared=no".

   Note: replace "--prefix=/home/grad/m4hoang" with your home directory. This is where the finally executable file will appear. If you can't execute the "./configure" command, make sure you have set the permission for the "configure" file in the mpg123 directory to allow execution. E.g. using "chmod" command.
5. Before we compile, we have to edit the dummy.c audio driver in "uclinux/mpg123/src/output/dummy.c". The code is shown in Appendix A. Make sure to edit the audio controller base address in the code to match your own. The audio base address can be found in the SOPC window. The dummy driver will handle the process of sending audio data to our sound controller.
6. Now in the "uclinux/mpg123/" directory, type "make" to start compiling. Once that is successful, type "make install". This will create the mpg123 executable file in your home directory under the bin folder. E.g. "/home/grad/m4hoang/bin/".
7. Copy the executable file (rename the file to "mpg123"), and put it into the "../uclinux/apps/target/home/" directory.
8. Resume with step 9 in section 2.3.

2.5 Programming the DE2 board and accessing uClinux terminal

Now we have everything we need to program the board. You should have linux.bin and romdisk.img in your "/uclinux/" folder.

1. Start the Nios II sdk shell from SOPC Builder. Once inside the shell, navigate to your "uclinux" folder where the linux.bin and romdisk.img reside.

2. We need to convert the binary image into flash. Type:
   "bin2flash --input=linux.bin --output=linux.flash --location=0x0"
   "bin2flash --input=romdisk.img --output=romdisk.flash --location=0x200000"
   The location parameter indicate the starting address of where the image will be stored in the flash. So the uClinux operating system will occupy the space

starting at 0x0, and the file-system will occupy the space starting at 0x200000. Both should not exceed 2MB each since our flash memory has only 4MB of space.

3. We now upload the flash image onto the DE2 board by typing:
   "nios2-flash-programmer --cable="Cable Name" --base=0x00000000 linux.flash"
   "nios2-flash-programmer --cable="Cable Name" --base=0x00000000 romdisk.flash"

   Note: the base parameter should be replaced with the address of the cfi_flash_0 shown in your SOPC Builder. The cable name is the same as the cable used in the programmer tool in quartus. e.g. "USB-Blaster".

4. Once the images are uploaded, you can communicate with the board by typing "nios2-terminal" in the NIOS2 SDK shell.

5. Press the KEY0 on the DE2 board to reset uClinux. You should be able to see uClinux booting up.

6. When ask for a username and password use "nios" as the user and "uClinux" as the password.

7. Now you can type commands like you would in a normal unix environment. For example, you can check your ip address by typing "ifconfig".

8. The uClinux terminal should now start in "/home/" folder where your mp3 player executable file is. If not, you can navigate to it by typing "cd /home/".

9. Set up a computer or laptop as the Shoutcast server with ip address of 192.168.1.1. Connect the laptop to the DE2 board using Ethernet cable (crossover cable or straight through cable if using a hub). To test if the connection is working, from the uClinux terminal, type "ping –c 5 192.168.1.1". This will ping the laptop computer 5 times (make sure firewall is off). If the packet is received, then the connection is working properly. You can also ping from the laptop by opening up a command prompt and typing "ping –c 5 192.168.1.2".

10. Start the Shoutcast server on the laptop and begin playing music in winamp. Connect winamp to the server to start streaming. (Check Shoutcast.com for instruction).

11. If your Shoutcast server is working properly, you can connect to it from the uClinux terminal by typing: "mpg123 -@http://192.168.1.1:8000". This will take a few minutes to start. Once it starts you should be able to see the title of the song that is playing and hear sound coming out of the audio jack. That's it!

3. System Architecture

Figure 2 shows the block diagram of the system architecture level starting from the application layer, down to the uClinux kernel layer, then to the CPU and hardware controllers layer, and then to the peripherals layer.

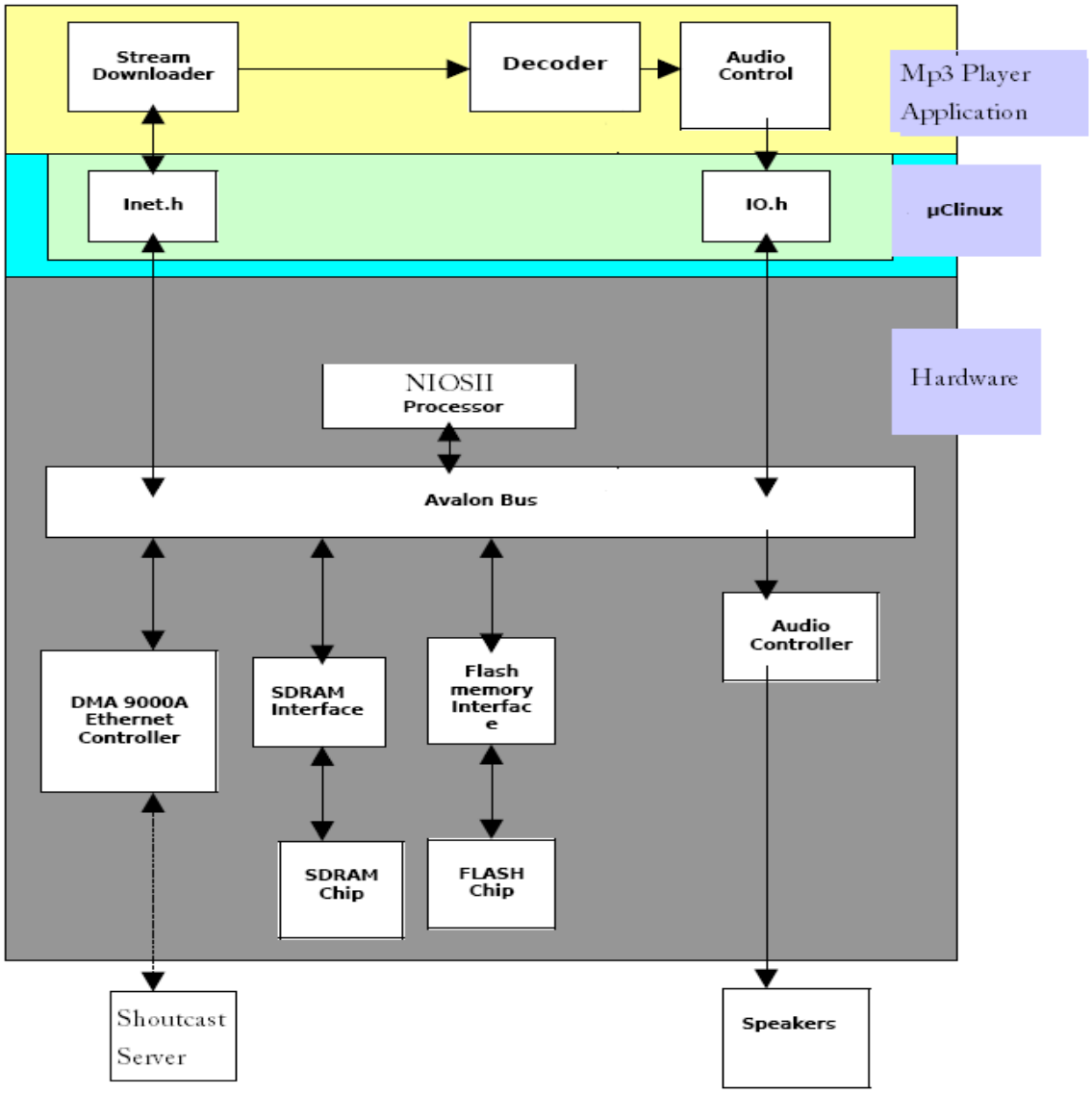

Figure 2 – System Architecture of Embedded Shoutcast client.

When the mp3 player starts playing the Shoutcast stream, it first converts the TCP/IP stream into mp3 frames. This is achieved using mpg123 mp3 player and uClinux. The mp3 frame is then decoded by the mpg123 mp3 player. The decoded sound is then passed onto the audio controller by a custom sound driver. The audio controller outputs the sound to the speakers. The uClinux kernel, file-system, and mp3 application are stored on the flash memory. Executions of the kernel and mp3 application are done on the SDRAM.

4. Results

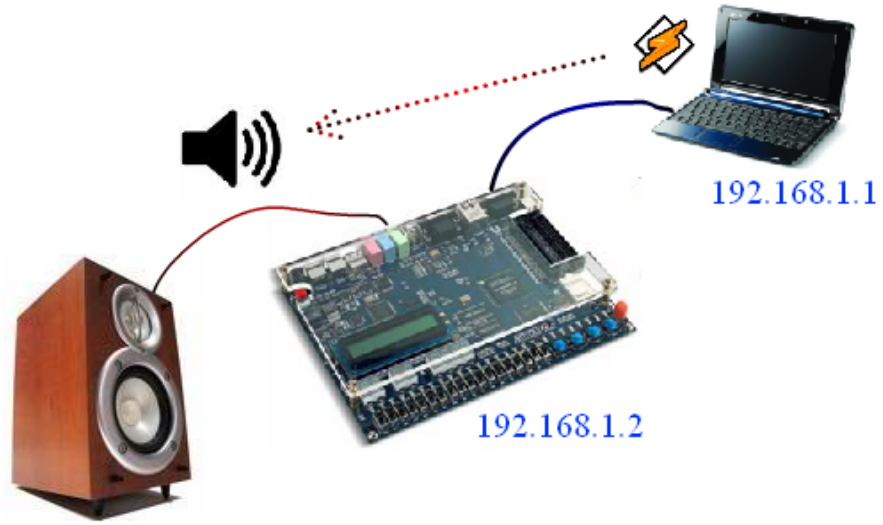

Figure 3 - Experimental Setup.

You will be able to compile the uClinux kernel with network interface support. You will also be able to get the TCP/IP networking protocol to work, which allowed you to stream Shoutcast internet radio. You will be able to to compile an application (e.g. mpg123 mp3 player) to work under uClinux.

5. Conclusions

You will be accomplishing a lot in this project. You are able to get uClinux with networking support to work on the DE2 board. You are also able to compile an application (e.g. mpg123 mp3 player) to work with uClinux.

APPENDIX A.
dummy.c

```
/*
	dummy: dummy audio output
	copyright ?-2006 by the mpg123 project - free software under the terms of the LGPL 2.1
	see COPYING and AUTHORS files in distribution or http://mpg123.org
*/
#include "mpg123app.h"
#include "config.h"
#include "debug.h"
#include <asm/io.h>
#include <asm/ioctl.h>
#include <stdio.h>
#include <stdlib.h>

static int open_dummy(audio_output_t *ao)
{
	debug("open_dummy()");
	return 0;
}
static int get_formats_dummy(audio_output_t *ao)
{
	debug("get_formats_dummy()");
	return MPG123_ENC_SIGNED_16;
}
static int write_dummy(audio_output_t *ao,unsigned char *buf,int len)
{
	unsigned short int i=0,Tmp1=0;
	for(i = 0; i < len; i+=2)
	{
	  if (!readw(0x80681104)) { //0x80681104 = audio controller base address
	    Tmp1=(buf[i+1]<<8)|buf[i];
	    writew(Tmp1,0x80681104);
	  }
	}
	return len;
}
static void flush_dummy(audio_output_t *ao)
{
	debug("flush_dummy()");
}
static int close_dummy(audio_output_t *ao)
{
	debug("close_dummy()");
	return 0;
}
static int deinit_dummy(audio_output_t *ao)
{
	debug("deinit_dummy()");
	return 0;
}
static int init_dummy(audio_output_t* ao)
{
	if (ao==NULL) return -1;
	debug("init_dummy()");
```

```
	/* Set callbacks */
	ao->open = open_dummy;
	ao->flush = flush_dummy;
	ao->write = write_dummy;
	ao->get_formats = get_formats_dummy;
	ao->close = close_dummy;
	ao->deinit = deinit_dummy;

	/* Success */
	return 0;
}

/*
	Module information data structure
*/
mpg123_module_t mpg123_output_module_info = {
	/* api_version */  MPG123_MODULE_API_VERSION,
	/* name */                              "dummy",
	/* description */   "Dummy audio output - does not output audio.",
	/* revision */                  "$Rev:$",
	/* handle */                    NULL,

	/* init_output */   init_dummy,
};
```

APPENDIX B.

“rc” file

```
echo "/etc/rc processing ..."
/bin/hostname Nios2
echo "Making Filesystems ..."
/bin/expand /etc/ramfs.img /dev/ram1
echo "Mounting Filesystems ..."
/bin/mount -t proc proc /proc -n
/bin/mount -t sysfs sysfs /sys -n
/bin/mount -t tmpfs tmpfs /tmp -n
/bin/mount -t ext2 /dev/ram1 /var -n
/bin/ifconfig lo 127.0.0.1
/bin/ifconfig eth0 down
/bin/ifconfig eth0 192.168.1.2 netmask 255.255.255.0
/bin/route add default gw 192.168.1.1
/bin/ifconfig eth0 up
echo "rc done ..."
echo "Welcome to Minh's Mp3 player. Login:nios password:uClinux"
```

“innitab” file

```
ttyJ0:vt100:/bin/agetty 115200 ttyJ0
```