Using U-Boot for LDR Boot Streams on the ADSP-SC589 Processor

Contributed by Yi, Gabby

Rev 1 – November 29, 2018

Introduction

U-Boot is a secondary boot loader that is typically used to boot Linux images. However, this EE-Note describes how to use U-Boot to boot binary LDR boot stream files. The LDR boot streams can be single-core boot streams that boot onto the ARM core or multi-core boot streams that boot onto both the ARM core and the SHARC+® DSP cores of the ADSP-SC589 processor. U-Boot features include support for Ethernet, mobile storage interface, FAT file system, and GPIO. These features facilitate a path toward this objective.

This EE-Note demonstrates:

- The steps used to modify U-Boot
- How to selectively boot one of multiple LDR files stored on an SD card
- How to boot an LDR file over Ethernet.

In the two booting examples, from SD card and over Ethernet, the LDR files are transferred from the source into external DDR memory and then the LDR boot stream is parsed and booted.

Modifying U-Boot

To modify the bootloader, complete the following steps:

1. Obtain the U-Boot Source Code.
2. Create Command to Parse LDR Boot Stream.
3. Use GPIOs to Select Boot Stream.
4. Update GPIO Command.

Obtain the U-Boot Source Code

The U-Boot source code is available as part of the Linux Add-in for the ADSP-SC5xx processor. The Linux Add-in can be downloaded from the Analog Devices website.

Follow the instructions outlined in the *Linux Add-in for CrossCore® Embedded Studio*[1] to set up the tools and sources so that U-Boot code can be compiled. The instructions are found in the section *Configure and build from source code*. The `makefile` configuration depends on the development board being used.

Create Command to Parse LDR Boot Stream

Once the source code is configured to be updated, a boot kernel can be developed that can process the boot stream stored in memory. The LDR boot stream is basically a series of headers and payload data. Following the description and details found in the **Boot ROM and Booting the Processor** section of the *ADSP-SC58x/ADSP-2158x SHARC+ Processor Hardware Reference*[2], a command, `bootldr`, was created to parse the boot stream. The command takes in one input: the memory address where the boot stream is stored.

The source code is found in the file `cmd_bootldr.c` that is included in the zip file associated with this EE-Note [3].

The boot stream parser is simple and does not process all the possible flags and commands that the boot stream format supports. Since this is only a proof of concept demonstration, enhancements are left up to the reader.

Information on creating custom commands can be found in *U-Boot programming: A tutorial – Part II*[4].

This parser assumes a binary LDR file.

Debugging bootldr Command

If the `bootldr` command must be debugged, uncomment the macro `COMPILED_IN_LDR_DEBUG` and comment out `READ_LDR_FROM_MEMORY`. The boot stream stored in `bs_arm.c` will be compiled into the U-Boot code with the command. It contains a buffer which is byte data of a boot stream of a program that blinks the LEDs on the SHARC+ Audio Module development board.

Use GPIO Pins to Select Boot Stream

On the SHARC+ Audio Module development board, there are two push buttons. Their corresponding GPIO pins are `PF_00` and `PF_01` which translate to GPIO numbers 80 and 81, respectively.

For this demonstration, the push buttons determine which one of the boot streams stored on an SD card is loaded.

Update GPIO Command

In U-Boot, there is a `gpio` command utility to set, clear and read the state of a given GPIO pin. When used to read the state of a GPIO pin, the status is displayed on the console. Unfortunately, the console output cannot be piped or redirected. So, to use the status of a GPIO pin, modifications were made to the existing `cmd_gpio.c` file to set the status of a GPIO pin in an environment variable.

```
…
do{
   char pinenvvar[64];
   sprintf(pinenvvar, "pin%d", gpio);

   setenv_ulong(pinenvvar, value);
}while(0);
```

Listing 1: Setting GPIO Status into Environment Variable

Listing 1 shows the snippet of code that sets an environment variable in the U-Boot code with the status of the pin. It declares a buffer `pinenvvar` to hold the string of an environment variable name. It is then post-fixed with the GPIO number. Finally, the value of the GPIO is set into the environment variable using `setenv_ulong()`. With the status of the GPIO/push button in an environment variable, it can be used to direct which LDR file is chosen to be booted.

Booting from SD Card

U-Boot also supports the FAT file system. Once U-Boot is compiled and programmed into flash memory, the user no longer depends on a Linux host machine.

Prepare the SD Card

The user can use CrossCore Embedded Studio on a Windows host machine to develop applications and generate LDR files and then copy the files directly to a FAT-formatted SD card.

U-Boot Scripting

After the state of the push button is saved to an environment variable, additional environment variables are executed using the `run` command.

```
bootfromsd=run selectbootfile; fatload mmc 0:1 ${loadaddr} ${bf}; bootldr
${loadaddr}
```

Listing 2: Environment variable to boot from SD card

Listing 2 shows the environment variable `bootfromsd` that boots a file from the SD card. First, it runs another environment variable, `selectbootfile`. This environment variable determines which LDR boot stream file is chosen based on how the push buttons are pushed. The resulting filename is stored in the environment variable `bf` (bootstream file). The command `fatload` is used to read a file from a FAT file system on the SD card and store it at the address specified in the `loadaddr` environment variable. Once the file is transferred to memory, the `bootldr` command (the simple boot loader) is executed, taking in the address of where the boot stream file is stored in memory.

```
selectbootfile=run pb_pressed; if test ${pin80} = 0 -a ${pin81} = 0; then set bf
${f1}; elif test ${pin80} = 1 -a ${pin81} = 1; then set bf ${f4}; elif test
${pin80} = 1; then set bf ${f2};elif test ${pin81} = 1; then set bf ${f3}; fi
pb_pressed=gpio input 80; gpio input 81

f1=blinksam_core0.ldr
f2=turnonledssam_core0.ldr
f3=blinksamall_core0.ldr
f4=sam_baremetal_framework_core2.ldr
```

Listing 3: Environment variables to select boot stream file

In Listing 3, the environment variables are used to select the boot stream file. Assuming the filenames are known, they can be saved into environment variables. In this example, three different blinking routines are stored in environment variables `f1`, `f2` and `f3`, while `f4` holds the filename of an audio talk-through program boot stream.

The `selectbootfile` environment variable first determines which push buttons were pressed. This operation is done by running another environment variable, `pb_pressed`, which uses the `gpio` command. On the SHARC+ Audio Module as well as the ADSP-SC589 EZ-Board®, the two push buttons are mapped to GPIO 80 and 81. Since the `gpio` command had been modified, the status is stored in the environment variables `pin80` and `pin81`. If the push button for GPIO 80 is pressed down during the call to the `gpio` command, then variable `pin80` is set to `1`; otherwise, it is set to `0`. The same applies to GPIO 81.

After the status of the push buttons is obtained and stored in environment variables, a test can be applied using the `if-elif-else-then` syntax understood by U-Boot to determine which file is selected and to set the filename into `bf`. In this case, `bf` is set to the first file, `f1`, if both `pin80` and `pin81` are both `0`. If they are both `1`, `bf` is set to the filename held in `f4`. Or, `bf` is set to the second file, `f2`, if `pin80` is `1`, and `pin81` is set to `0`. Finally, if `pin80` is set to `0` and `pin81` is set to `1`, `bf` is set to the filename stored in `f4`.

More information on U-Boot tests and scripting can be found on the U-Boot scripts wiki page[4].

The environment variables can be set at the U-Boot command line and saved to flash memory. Or, the user can program the variables into the U-Boot source code and compile them. In this demonstration, the environment variables were manually set at the U-Boot command line.

Booting From Host

U-Boot provides Ethernet support. Not only can a boot stream be selected and booted from an SD Card, the same can be applied to an LDR boot stream file sitting on a host PC. Following the instructions from the Linux Add-in guide, an Ethernet connection can be set up to communicate between the board and a host computer, and a TFTP server can be set up on a host computer.

```
bootfromhost=tftpboot ${loadaddr} ${serverip}:${bf}; bootldr ${loadaddr}
```

Listing 4: Environment variable to boot LDR file from host machine

Listing 4 shows the environment variable that can be used to boot an LDR file from the host machine. The command `tftpboot` is used to transfer the file whose filename is stored in the `bf` environment variable from the host using TFTP into a memory location specified in the `loadaddr` environment variable. In this example, the IP addresses were set up so that the board uses IP address 192.168.1.9 while the host computer used IP address 192.168.1.10, which is stored in `serverip`. Once in memory, the `bootldr` command is called to boot the boot stream data from the memory location.

This configuration allows remote debugging and testing of different versions of an application. The user just updates the LDR boot stream file on the host computer. U-Boot can be configured to automatically fetch the file from the host via TFTP and boot it.

Other Use Cases

Booting Different LDR Boot Streams in Flash

Due to hardware restrictions, the primary boot kernel is unable to selectively boot from a flash location other than the flash base address. This option allows various boot streams to be stored in different flash memory locations with the capability of selectively booting any of them.

Memory Usage and Constraints

There are a few things to consider when implementing the second stage loader. Both U-Boot and the boot stream reside in memory during the second stage boot.

By default, when configuring and building U-Boot for the SHARC+ Audio Module (target `sc589-mini_defconfig`), it is mapped to the higher end of the second DDR memory interfaced to the ADSP-SC589 processor at address **0xC2200000.** The `loadaddr` used in the example is set to **0xC2000000**.

The application to be booted should avoid this upper range of DDR1 memory so that the U-Boot or boot stream do not get overwritten while booting. Also, the DDR initialization occurs as part of the U-Boot booting procedure. Therefore, do not include an `init` file to initialize the DDR in the generation of the LDR file in CCES. Note that the user can still change how U-Boot is linked and where the boot stream is stored, as needed.

Associated ZIP Files

The accompanying files provide the modifications to U-Boot and the example projects with the LDR boot stream files. The example presented here can be recreated by programming the flash with one of the U-Boot LDR files, depending on which board is used. The example application LDR files can be copied over to a FAT-formatted SD card. At a U-Boot command prompt, set the environment variables `f1` to `f4`, to the file names of the LDR files on the SD card. A directory listing can be shown by using the command `fatls mmc 0:1`. Finally, execute the `saveenv` command to save the updated environment variables to flash memory. The next time U-Boot starts, execute the `run bootfromsd` command while pressing down on the push buttons (or not). From the combination of the state of the push buttons, one of the four LDR files on the SD card will be booted.

To make further modifications to U-Boot, download the full source code attached to the Linux Add-in and apply the patch in the ZIP file to the sources. Refer to the `readme` file in the associated ZIP file for more information.

Conclusion

This EE-Note demonstrates how U-Boot can be modified and used as a second stage loader to boot different LDR boot stream files from multiple sources (SD card and remote host). Using this approach, U-Boot can also boot LDR boot streams from different memory locations in external flash storage.

U-Boot was modified to add a command to process an LDR boot stream from a memory location. U-Boot was also modified so the `gpio` command saves the status output to environment variables. Finally, scripts were written to choose a file based on the status of push buttons, transfer the file into memory and boot it

References

[1] *Linux Add-in for CrossCore® Embedded Studio.*Version 1.2.0, August 2017. Analog Devices, Inc.

[2] *ADSP-SC58x/ADSP-2158x SHARC+® Processor Hardware Reference.* Rev 1.0, September 2017. Analog Devices, Inc.

[3] *Associated ZIP File for Using U-Boot for LDR Boot Streams on the ADSP-SC589 Processor.* Rev 1, November 2018. Analog Devices, Inc.

[4] *U-Boot programming: A tutorial – Part II (http://xillybus.com/tutorials/uboot-hacking-howto-2)* Xillybus.

[5] *U-Boot Scripts (http://www.compulab.co.il/ utilite-computer/wiki/index.php/U-Boot_Scripts).*

Readings

[6] *The DENX U-Boot and Linux Guide (DULG) for canyonlands (https://www.denx.de/wiki/DULG/Manual)*

Document History

Revision	**Description**
Rev 1 – November 16, 2018 by G Yi	Initial Draft