Repurposing 2009 KwikByte FRC Driver Station

Carlos Gross Jones
c.gross.jones@gmail.com
Jet Propulsion Laboratory, Pasadena, CA
FRC 418 Alumnus
FRC 696 Mentor

March 2, 2017

1 Introduction

For the 2009 FRC game Lunacy™, an entirely new control system was introduced consisting primarily of a National Instruments cRIO-FRC as the robot controller and a Kwikbyte DS9260 as the driver station. While the cRIO proved reliable and versatile, and continued to be used until replaced by the cRIO-FRC II and later RoboRIO, the DS9260 was not as successful. As a result, many FRC teams had one or more DS9260 driver stations which were completely useless for later competitions. This lead to occasional interest in repurposing the DS9260, basically a Linux single-board computer with an array of peripherals, for general use.

Unfortunately, this proved not to be straightforward. Not only is neither the root password nor source code for the DS9260 released to the FRC community, but KwikByte is unwilling to provide any information or guidance other than to state that the DS9260 firmware uses a Das U-Boot bootloader and Linux kernel. Therefore, in order to unlock the DS9260 for other applications, many aspects of the hardware and software must be reversed-engineered.

2 Background

2.1 Hardware

The following bullet points are pulled from what remains of KwikByte's DS9260 site:

1. RoHS Compliant
2. Atmel AT91SAM9260 processor
3. 200 MHz, ARM926EJ-S core with Java acceleration and DSP instruction extensions
4. Independent 8KB instruction and 8KB data caches
5. 64 MB SDRAM
6. 8 MB boot/kernel Flash
7. 1 GB NAND Flash - not installed
8. 2 x 10/100 Ethernet ports
9. 4 x USB 2.0 full speed host ports

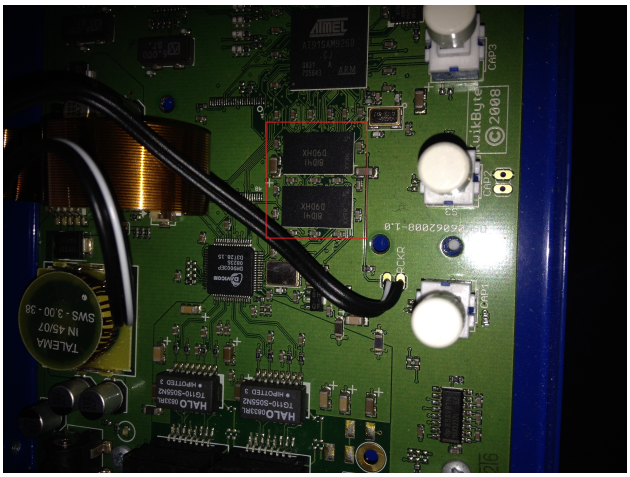


Figure 1: Dual MT48LC16M16A2BG SDRAM

10. 1 x competition port
11. 8 x digital input channels (+5VDC in user header)
12. 8 x digital output channels (+5VDC in user header)
13. 4 x analog input channels (+5VDC range in user header)
14. 3 x user buttons
15. 1 x mode toggle switch
16. 1 x LCD monochrome graphic (128x64)
17. Sturdy metal case

However, examination of the PCB shows that several of the above items appear to be incorrect.

1. Rather than the 64 MB SDRAM in item 5 above, the DS9260 instead has 512 MB of SDRAM, implemented with two Micron MT48LC16M16A2BG ICs (Fig. 1);
2. Rather than the 8 MB referenced in item 6, the DS9260 has an AT45DB642D 64 MB DataFlash IC (Fig. 2).

2.2 Related Hardware

Based on the processor, specifications, and name, it is reasonable to assume that the DS9260 is a modification of KwikByte's KB9260 general-purpose single-board computer. The upshot of this is that slightly more documentation exists for the KB9260 than the DS9260. Specifically, KwikByte offers sample GPIO and ADC interface code for the KB9260 (GPIO, ADC) which are also mirrored on carlosgj.org (GPIO, ADC). These samples should, when compiled properly, run on the DS9260 as well (after modifying the `#defines` to reflect the DS9260's pin mapping). This is investigated in §6.

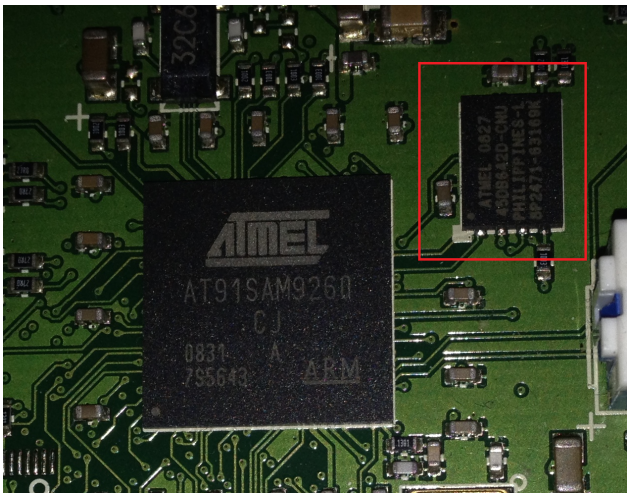


Figure 2: AT91SAM9260 processor and AT45DB642D DataFlash

2.3 Connection & Communication

Communication with the DS9260 takes place primarily over the DE9 "competition port". Despite appearances, this is not a plain serial port. While many of the pin functions are not yet know, basic RS232 serial capability is present on the normal pins: 2 (RX), 3 (TX), and 5 (ground). Therefore, a custom null-modem adapter was made which connects only those three pins. (The "USB Adapter Clip" from KwikByte was likely a similar connector, plus an FT232R chip for USB operation.)

Opening the serial port (115200, 8, N, 1) presents a banner and login prompt:

```
 --------------------
-                    -
- KwikByte DS9260    -
-                    -
- www.kwikbyte.com   -
-                    -
--------------------
DS9260 login:
```

While the root account is password protected, trial and error revealed a user account called "default" that did not require a password. This allowed exploration of much of the filesystem. It appears to be a fairly standard embedded Linux system with BusyBox. The application software binaries are in /ds60x/bin, consisting of nine files: lcd, mgr_joysx, recv_pc, send_fms, send_robot, manager, recv_fms, recv_robot, and send_pc.

For firmware updates and other pre-boot functionality, the normal boot process can be interrupted and a "utility loader" sent by serial as described in App. A. This has basic facilities for memory read and write, as well as interaction with the SPI Flash memory. This small utility can be downloaded from KwikByte or carlosgj.org.

3 Dissection of Factory Image

The first software item investigated was the image "raw_otb.bin" provided by KwikByte (also mirrored on carlosgj.org for posterity).

This file is not a first-level bootloader, but is instead called from the kernel loader. As such, it is expected to contain at least the Linux kernel and an initrd. In fact, the most recognizable thing in the file is the Linux boot parameters at offset 0x58:

```
console=ttyS0,115200 root=/dev/ram rw initrd=0x22400000,851719 mem=64M
```

The purpose of the preceding data (including the string "KwikByte") and following data (mostly null bytes) is unknown. However, an educated guess would be that a compressed Linux kernel (zimage) exists in the file. Knowing that the zimage magic number, 0x016F2818, should be found at 0x24 from the beginning of the zimage, and finding this value at 0x443 in raw_otb.bin, the zimage is expected to start at 0x420. Furthermore, based on the start and end addresses at 0x28 and 0x2C from the start of the zimage, respectively, the zimage is expected to end at 0x160494 of raw_otb.bin. In order to examine the kernel image, the actual compressed kernel would have to be extracted (i.e., separated from the self-extraction code). Assuming the zimage uses gzip, the gzip "magic number", 0x1F8B, plus the expected compression method, 0x08, should be found (see the gzip standard for details). A search for 0x1F8B08 showed the first occurrence at offset 0x44F8. Metadata present in the gzip header reveals that the data was zipped on Fri, 03 Oct 2008 at 23:56:34 GMT on a Unix system. A partial copy of the raw_otb.bin file from 0x44F8 to 0x160493 can indeed be unzipped, resulting in what appears to be a Linux kernel based on literal strings in the binary. In fact, one such string offers potentially valuable information:

```
Linux version 2.6.23DS60v1.0 (root@kbdev-laptop13) (gcc version 3.4.2) #7
Fri Oct 3 16:56:31 MST 2008
```

Just past the end of the kernel zimage, another occurrence of 0x1F8B08 is found, indicating the start of another archive. The metadata shows that it was zipped on Mon, 03 Nov 2008 at 20:23:39 GMT, with maximal compression, on a Unix system, and additionally that the original filename was "initrd_img". Copying 0x160494 onwards results in another valid gzip, which successfully uncompresses into a filesystem.

4 Gaining Root Access

As a first step towards repurposing the DS9260, root access was needed to more thoroughly explore the system. First, on the host computer (running Debian Jessie), the `gunzip`ped initrd was mounted. Then, the root entry in /etc/shadow was edited to remove the password, resulting in:

```
root::10933:0:99999:7:::
bin:*:10933:0:99999:7:::
daemon:*:10933:0:99999:7:::
adm:*:10933:0:99999:7:::
lp:*:10933:0:99999:7:::
sync:*:10933:0:99999:7:::
shutdown:*:10933:0:99999:7:::
halt:*:10933:0:99999:7:::
uucp:*:10933:0:99999:7:::
operator:*:10933:0:99999:7:::
nobody:*:10933:0:99999:7:::
default::10933:0:99999:7:::
```

This "rooted" filesystem image was then `gzip`ped, making sure that the gzip header and parameters matched the original. Knowing that the compressed initrd begins at 0x160494 of raw_otb.bin, and that raw_otb.bin is written to Flash at 0x42000 (per App. A), the rooted gzip was written to SPI at 0x1A2494 (0x42000+0x160494) using the utilLoader. This resulted in a working system with a root account which does not require a password. Surprisingly, although the rooted initrd was slightly larger than the original, the "initrd=0x22400000,851719" boot argument did not have to be changed.

This allowed much more in-depth experimentation with the DS9260. For example, to easily move files on and off the device, a flash drive could be used. Any USB mass storage device plugged into the DS9260 is assigned `/dev/sda` (multiple mass storage devices are apparently not supported). Then, the drive could be accessed by:

```
mkdir temp
mount /dev/sda1 temp/
```

Additionally, other devices (LCD framebuffer, joysticks, etc.) could be read from and written to. (See later sections for details on interacting with peripherals.)

5 Memory Organization

The following output during boot describe the major sections of memory in the SPI Flash:

```
[    2.750000] 0x00000000-0x00001080 : "BOOT1"
[    2.760000] 0x00001080-0x000413a0 : "BOOT2"
[    2.770000] 0x000413a0-0x00042000 : "PARAM"
[    2.770000] 0x00042000-0x00840000 : "RAW"
```

These partitions are mounted to `/dev/mtd0` through `/dev/mtd3` at boot. After gaining root access to the kernel, these sections were copied to a USB flash drive for examination, with `dd if=/dev/mtd0 of=temp/mtd0.bin bs=1` etc. The RAW partition begins at 0x42000, which, as described in App. A, is where the raw_otb.bin image is written. This is, therefore, where the kernel and filesystem are stored. In addition, in accordance with App. B, the "altloader" is programmed into Flash starting at 0x7FE000. Finally, the 1024-byte block starting at 0x83FBE0 contains the splash image shown on the LCD at boot, in monochrome bitmap format. The PARAM partition is apparently unused, as it is blank (null bytes) except for ASCII string "test" at the beginning. As described in App. B, the "kernel loader" (possibly a version of Das U-Boot) is programmed at 0x1080, indicating that BOOT2 is the second-stage bootloader. BOOT1, therefore, must the first-stage bootloader, responsible for initializing the processor on power-up and launching U-Boot. It is called "CopyLoader64 DS60x1.0" according to a string in its contents.

6 Compilation of Application Software

In order to use the DS9260 for anything other than its original application, it is necessary to compile new software for it. Luckily, as described in §2.2, KwikByte provides example code for the related KB9260. As a proof-of-concept, the GPIO example was compiled to run on the DS9260.

First of all, it was necessary to elucidate which AT91SAM9260 pins (e.g., PIO[A, B, C][0-31]) routed to which user-accessible interface before attempting to interact with the GPIO. As mentioned in App. A, "...you should not set a system-defined input pin as an output as this could damage the Driver Station." The AT91SAM9260 pins for the auto/teleop, up, down, and select buttons were found to be PIOB20, PIOB21, PIOB25, and PIOB27, respectively, using the "pin" command in the utilLoader.

In the name of safety, the gpio/main.c file was modified to remove all of the digital output, leaving just the digital input polling. In addition, the input pin was changed to PIOB20. After generating a basic buildroot configuration (just cross-compilation toolchain, no kernel, rootfs, or bootloader) and modifying gpio/Makefile to use the buildroot toolchain (App. D), the gpio software was successfully compiled. After copying it to the DS9260 via flash drive, it did indeed print the state of the auto/teleop switch to the console.

7 Digital And Analog Peripherals

As described above, the auto/teleop, up, down, and select buttons map to PIOB20, PIOB21, PIOB25, and PIOB27, respectively. The assignments of the 8 digital input pins were found in a similar fashion. Oddly, they do not map to a contiguous range (Table 1). The purpose of PIOA18 is unknown; it seems to always read as 1.

The ADC example, available from KwikByte's website along with the GPIO example, showed that analog inputs 1-4 simply map to the AT91SAM9260's ADC0-3, respectively.

Digital Input	PIO Assignment
1	A12
2	A13
3	A14
4	A15
5	A16
6	A17
7	A19
8	A20

Table 1: Digital input pin assignments

8 Writing To The LCD

The DS9260 supports a 128x64 pixel monochrome LCD display. This can be accessed in two different ways from the OS. First of all, for graphics or other custom patterns, up to 1024 bytes of binary data can simply be written to the `/dev/fb0` framebuffer device. The data is indexed row first: for example, echoing 16 0xAA bytes to `/dev/fb0` results in a toggling pattern across the first row of pixels on the LCD.

In addition, the OS has built-in support to use the LCD as a character device. Sending text to the `/dev/tty0` device results in the text appearing on the LCD. In fact, it appears that the OS is configured to start a login shell on the LCD, and that one of the ds60x applications (run on startup) kills this shell and takes over the LCD for the driver station application. After killing all of the DS60x processes, `getty 38400 /dev/tty0` results in a working login shell on the LCD. Furthermore, booting a filesystem without driver station applications in init.d (see §9) automatically launches a console on the LCD at boot. Since it is intended to be used as a console, each line of text written appears near the bottom of the LCD, with previous strings shifted up.

The default font used by the console device is quite small and difficult to read. However, a built-in command in the OS, `loadfont`, loads a font (in PSF format) from standard input.

9 TL;DR: Quickstart Guide

This section explains how to get up and running with DS9260 development, assuming a starting point of a working, unmodified driver station.

The two binary files needed are utilLoader.bin, described in §2.3, and the modified filesystem: carlosgj.org/FRC/DS9260/rooted/initrd_img.gz

This filesystem image has been modified in two ways. First of all, the root password has been removed as described in §4. Second, the file `/etc/init.d/rcS` has been modified to remove references to executables in /ds60x. Thus, while the driver station application binaries are still present in the filesystem, they will not be started on boot.

In addition, a serial communication utility is needed on the host computer, with xmodem functionality. The work for this whitepaper was performed using TeraTerm on Windows.

After setting up the serial connection (§2.3), power up the DS9260 and send the utilLoader as described in App. A. Next, send the initrd_img.gz file via xmodem to a free location in memory (0x21020000, per App. A, should still work fine). Finally, write the modified filesystem to SPI Flash location 0x1A2494 using the `spi_write` command. After waiting for the write to finish and rebooting the DS9260, the OS should allow a root login with no password through the serial console.

For compilation of new code, a very basic configuration of buildroot, as described in §6, can be used to generate a cross-compilation toolchain. If necessary, the configuration file used for this whitepaper can be found at carlosgj.org/FRC/DS9260/buildroot/.config. Note that, while the current configuration only builds a toolchain, future work may leverage buildroot to generate custom kernels, filesystems, and bootloaders.

To load test applications onto the DS9260, a USB flash drive was used. However, since the OS has utilities for FTP and TFTP transfers, network file transfer would make the development process more efficient. In addition, the utilLoader includes TFTP functionality, which allows sending large files (e.g., the filesystem image) much faster than via serial. Note that, during the boot process, the filesystem image is downloaded from SPI Flash into RAM and then mounted; therefore, the root filesystem is writable from the OS, but any changes made are not written to Flash, and will be lost on reboot.

A Utility Loader & Firmware Update Instructions

FIRST DRIVER STATION UTILITY LOADER RE-IMAGE INSTRUCTIONS

1 Introduction

This document describes steps to load the Driver Station (DS) v1.0 Utility Loader (UL). The UL can be used to re-image the DS, perform low-level operations on the board, or load alternate operating systems or user programs.

*1.1 *** Warnings ****

The boot loader(s) are write-protected. Do not unlock these sections. Doing so may render the unit unusable.

Do not overwrite the released image(s) or otherwise modify these sections with your own code. Doing so may render you ineligible to compete.

Be careful when using some of the UL commands. For example, you should not set a system-defined input pin as an output as this could damage the Driver Station.

The steps listed here do not violate these warnings. To be safe, just follow the instructions.

1.2 Equipment

You need:

1) A means of communicating with the DS at a low-level. This document uses the DS USB Adapter Clip with supplied USB extension cable.
2) A PC. The host OS can be Windows® or Linux. Other OS may work, but have not been tested.

1.3 Software

You need:

1) A host PC terminal emulator program like HyperTerm, minicom, Kermit, etc.
2) The DS UL binary image.
   ftp://kwikbyte.com/pub/DS/binary/utilLoader.bin
   or
   http://www.kwikbyte.com/driverstation/binary/utilLoader.bin

3) The original factory firmware.
   ftp://kwikbyte.com/pub/DS/binary/raw_otb.bin
   or
   http://www.kwikbyte.com/driverstation/binary/raw_otb.bin

1.4 Support

Please read the instructions carefully. If you have questions or helpful comments, please send them to driverstation@kwikbyte.com.

2 Re-image Instructions

2.1 Load the UL

The DS firmware provides a means of updating the system before the OS is loaded. During boot, the DS waits for two characters (must be caps) from the host: K B.

1) Insert the USB extension cable into a free USB port on the PC.

2) Insert the DS USB Adapter Clip on the end of the USB extension cable. Notice the little green LEDs flash on the Adapter Clip.

3) When using Windows®, a driver may be required to use the clip. In that case, see http://www.ftdichip.com/Drivers/VCP.htm. The device used is FT232R. Select your OS, understand the comments listed on the web page, download and install the driver.

4) The OS detects the Adapter Clip and reports that it is ready for use.

5) Start the terminal emulator: e.g., HyperTerm (Start -> All Programs -> Accessories -> Communications -> Hyperterm).

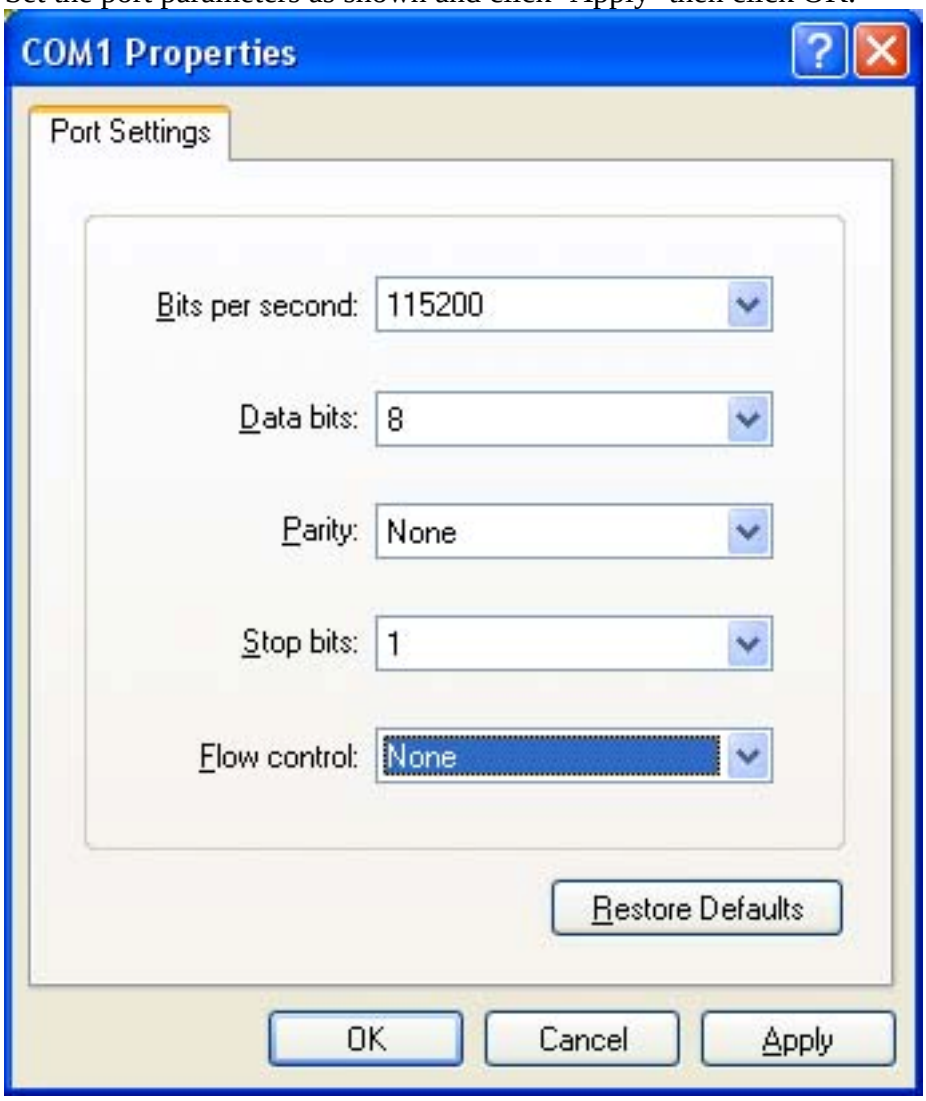
6) If asked, you do not have to make HyperTerm the default telnet program.

7) Provide a meaningful name in the Connection Description and click OK.

8) Depending on your computer configuration, the Adapter Clip may be recognized as a different serial port than the one shown. Trial and error will tell which port is correct. On many PCs, the last serial channel listed (e.g., COM4) is correct.

Select the port and click OK.

9) Set the port parameters as shown and click ‘Apply’ then click OK.

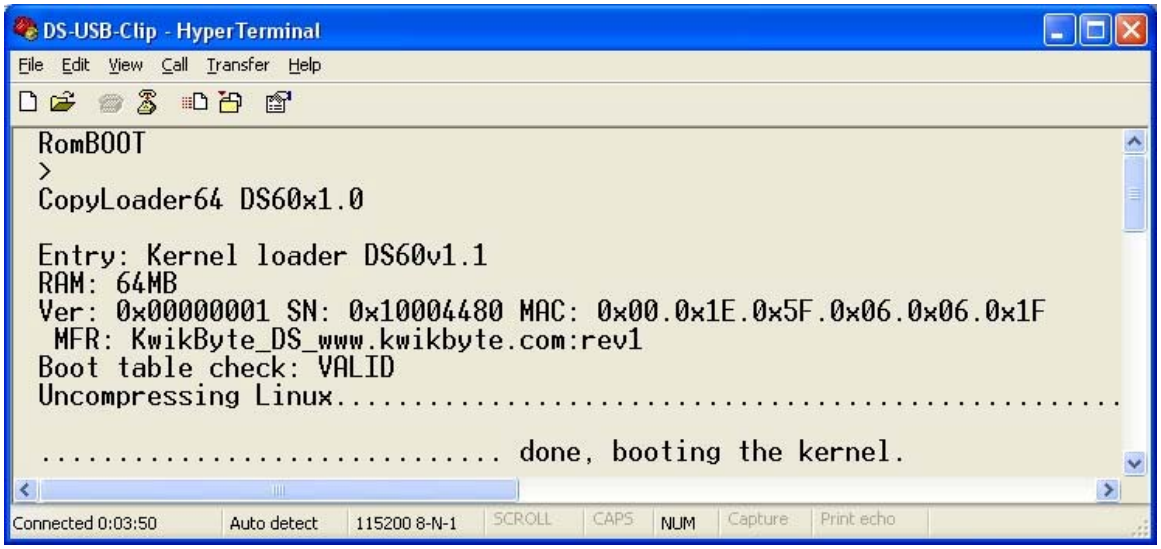


10) Now, you see an empty window.

11) Plug the Adapter Clip into the Driver Station Competition Port.

12) Apply power to the Driver Station by plugging-in the Driver Station power cord.

13) Now, you will see the output from the Driver Station. If not, you chose the wrong serial port. Close HyperTerm and go back to Step #5.

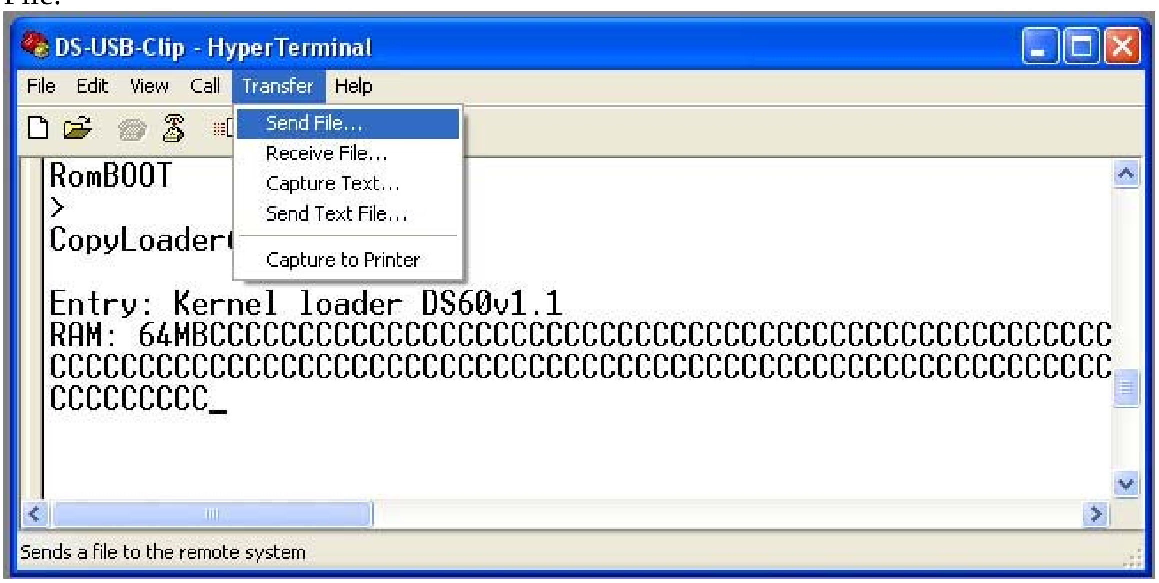


14) Remove power from the Driver Station by unplugging the power cord. Look at the first few lines of output. This is showing the Driver Station loading the Linux kernel and beginning to boot the system! In the next step we will trigger a secret hook in the "Kernel loader" by pressing some keys at the right moment. Timing is important because the kernel loader only looks for these characters for a short period before booting the normal system.

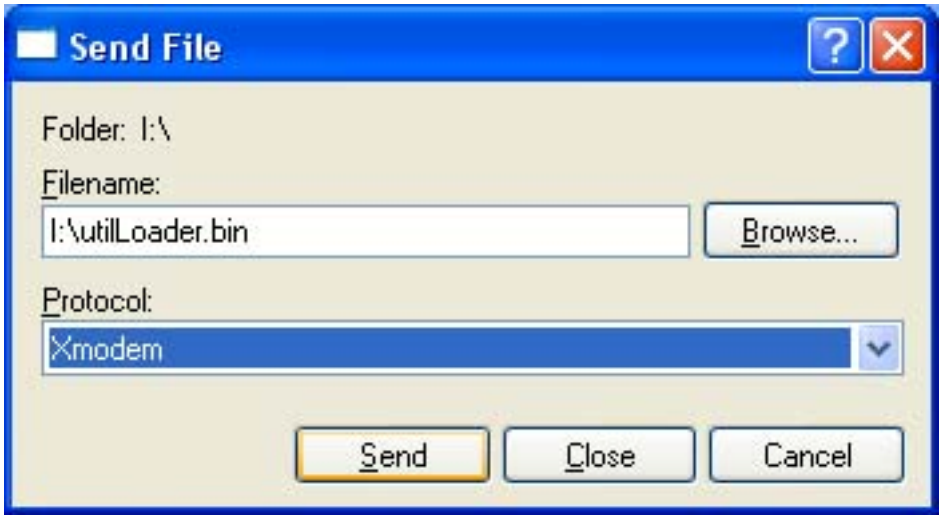
15) Get ready to type the two, capital case characters K B right after you see the "RAM: 64MB" output on the screen. On the Driver Station, hold down all three white pushbuttons (up, down, and select) – and keep them held down during this step. Apply power to the Driver Station by plugging-in the power cord. As soon as the "RAM:" line is output, type the two keys.

16) If you executed the previous step correctly, you now see the Driver Station sending 'C' characters at about one per second. These characters continue until you send the UL image to the Driver Station.

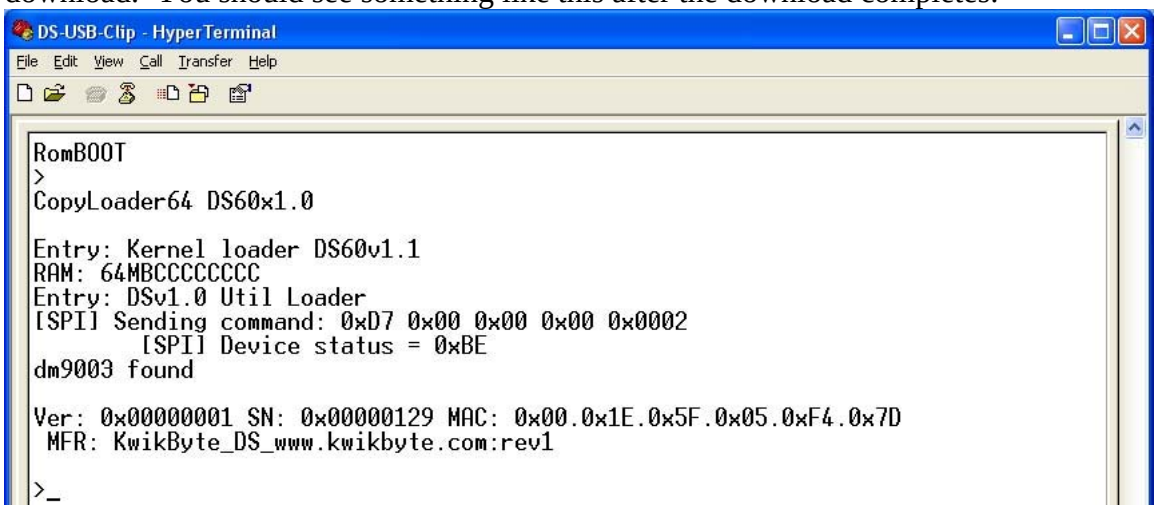
17) Send the utilLoader.bin file to the Driver Station by selecting Transfer -> Send File.



18) Select the file utilLoader.bin and set the transfer as Xmodem – this is **not** the same as 1K Xmodem. Then, click Send.



19) Watch the transfer progress. The first version of the UL takes about 12 seconds to download. You should see something like this after the download completes:



NOTE: The UL was not written specifically for network capability. Some Driver Stations correctly identify the dm9003 net chip. The network function will be used in a different document – when we boot a different Linux kernel and mount a larger file system on a USB stick. For now, we will use serial and just understand that Ethernet is a lot faster than serial!

The LCD display may also change when running the UL. This is normal.

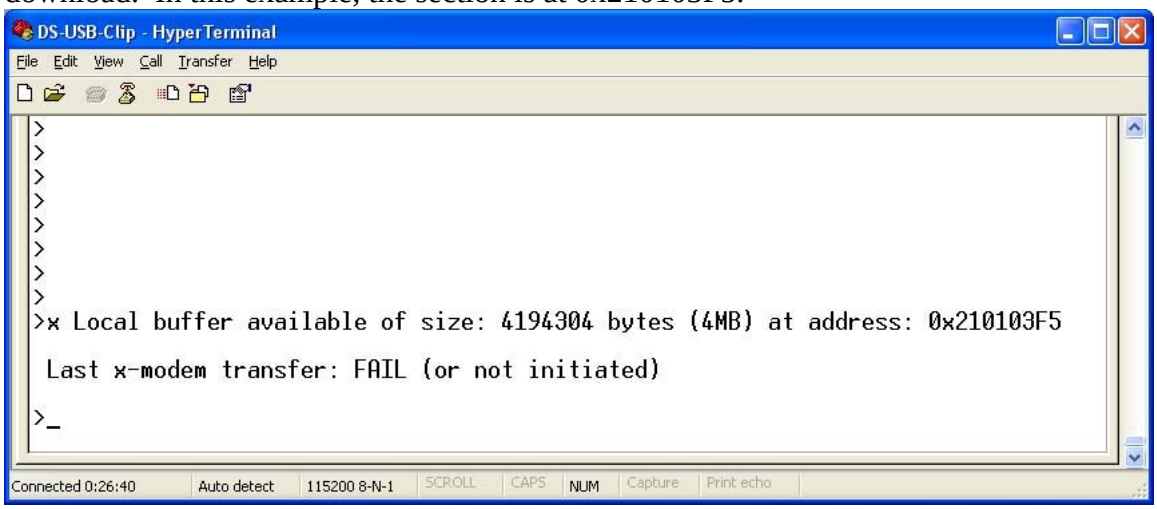
20) Some *very* brief help is available with ? Enter.

NOTE: The UL will repeat the last command if you type only Enter without entering a new command.

Be careful! Some of these commands can damage the Driver Station if used

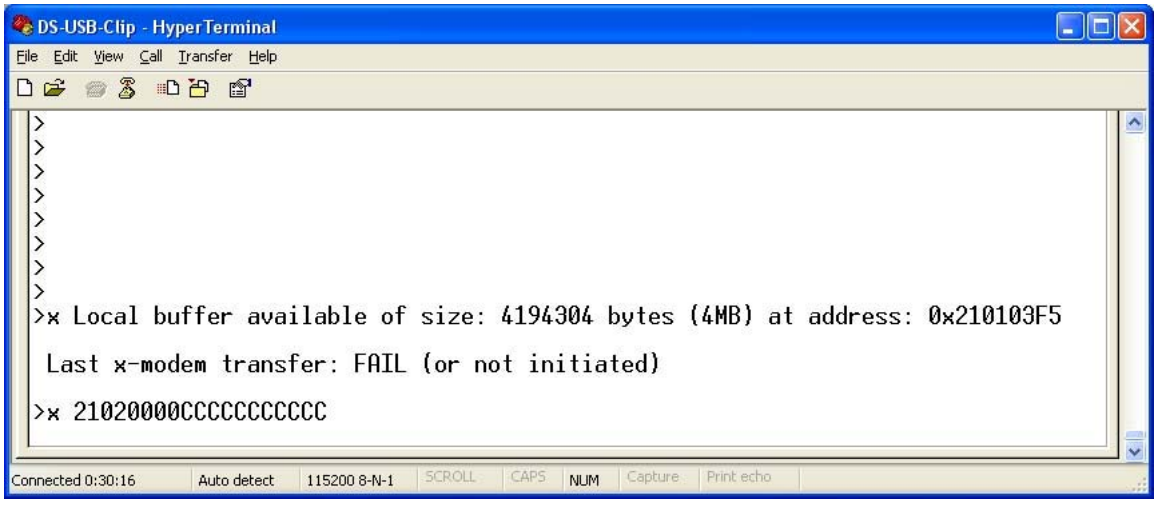
incorrectly.

21) Type x Enter. This reports the location of a free memory section we can use for download. In this example, the section is at 0x210103F5.



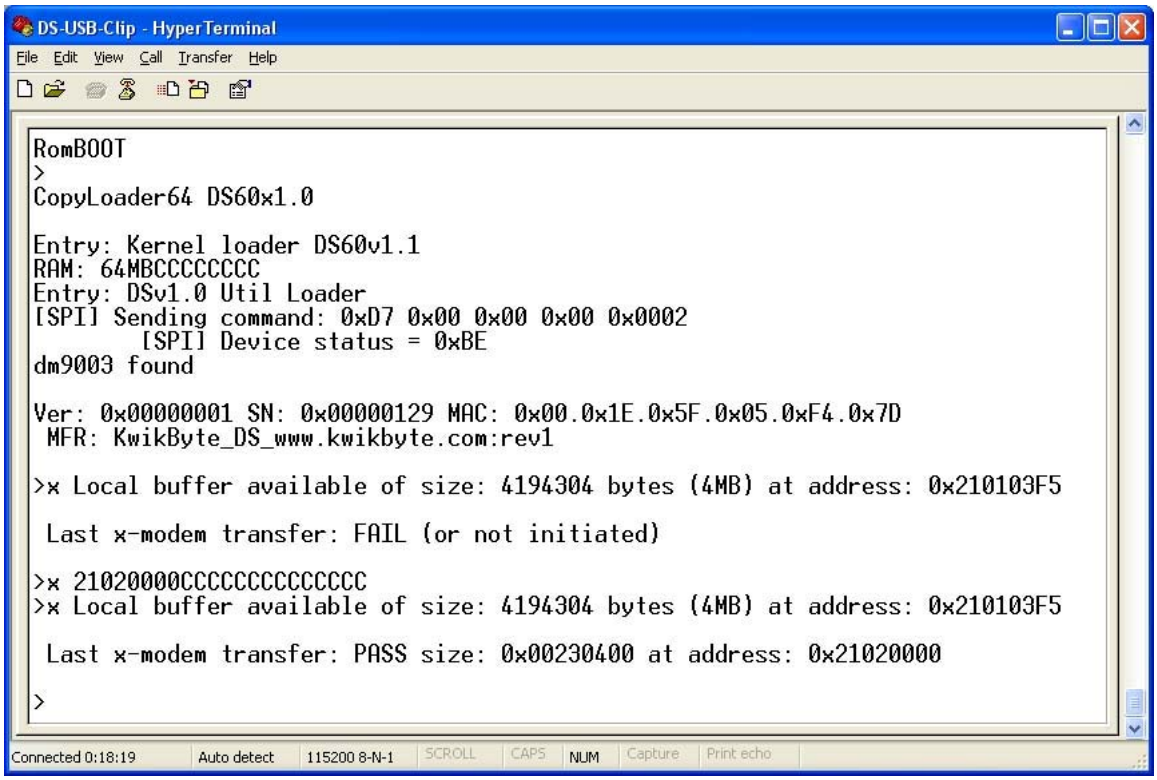
22) Let's initiate a download to receive the factory Driver Station Firmware. We will round-up the address provided by the buffer to a nice even number – 0x21020000. This works fine as long as the image to be received is less than the space allocated (4MB). Type x 21020000 Enter.

23) Now you see more 'C' characters indicating that the Driver Station is waiting for another Xmodem transfer.



24) As before, execute a transfer from the PC using Transfer -> Send File. Select the original raw_otb.bin Driver Station firmware for Xmodem download – just like we did earlier. Click Send and watch the progress. This will take a long time depending on the image size.

25) The Driver Station now has the firmware image in RAM. We need to program it to non-volatile memory (NVM). The NVM used on the Driver Station is serial flash. Type x Enter to get the size of the last transfer from the Driver Station's view.



26) From this example, the transfer size is 0x230400. Let's program flash with a command of the format "spi_write <flash destination> <source address> <size>". In this case,

spi_write 42000 21020000 230400 Enter

You should see lots of "[SPI]" messages scrolling during the programming operation. This command also takes a *long* time – depending on image size. Of course, there are faster ways to do this but let's start with this method. When the program operation is complete, the Driver Station reports "Buffer Written".

The re-image is complete. Reset the Driver Station by entering the command reset Enter.

3 Revisions

22DEC2008 Creation

B Kernel Loader Update Instructions

FIRST DRIVER STATION KERNEL LOADER UPDATE INSTRUCTIONS

1 Introduction

This document describes steps to update the Driver Station (DS) v1.0 Kernel Loader (KL). The KL is used to boot the operating system (Linux) on the DS or load alternate operating systems or user programs.

The purpose is to open the Driver Station for use as an embedded software development/learning tool.

*1.1 *** Warnings ****

You must follow the instructions exactly!
Potential risk: you could lock-up the Driver Station – making it unusable.
If you are not comfortable with the risk, do not perform this procedure.

The loader does not impact the applications running on the Driver Station during normal operation.

1.2 Documentation

In addition to this document, you need a copy of the Utility Loader document:
http://www.kwikbyte.com/driverstation/doc/DS_utility_loader.pdf

1.3 Equipment

You need:

1) A means of communicating with the DS at a low-level. This document uses the DS USB Adapter Clip with supplied USB extension cable.
2) A PC. The host OS can be Windows® or Linux. Other OS may work, but have not been tested.

1.4 Software

You need:

1) A host PC terminal emulator program like HyperTerm, minicom, Kermit, etc.
2) The DS UL binary image.
   ftp://kwikbyte.com/pub/DS/binary/utilLoader.bin
   or
   http://www.kwikbyte.com/driverstation/binary/utilLoader.bin

3) The new kernel loader (current version is v1.2).
   http://www.kwikbyte.com/driverstation/binary/kernelLoaderv12.bin

4) The altLoader program.
   http://www.kwikbyte.com/driverstation/binary/altLoader.bin

1.5 Support

Please read the instructions carefully. If you have questions or helpful comments, please send them to driverstation@kwikbyte.com.

2 Kernel Loader Update Instructions

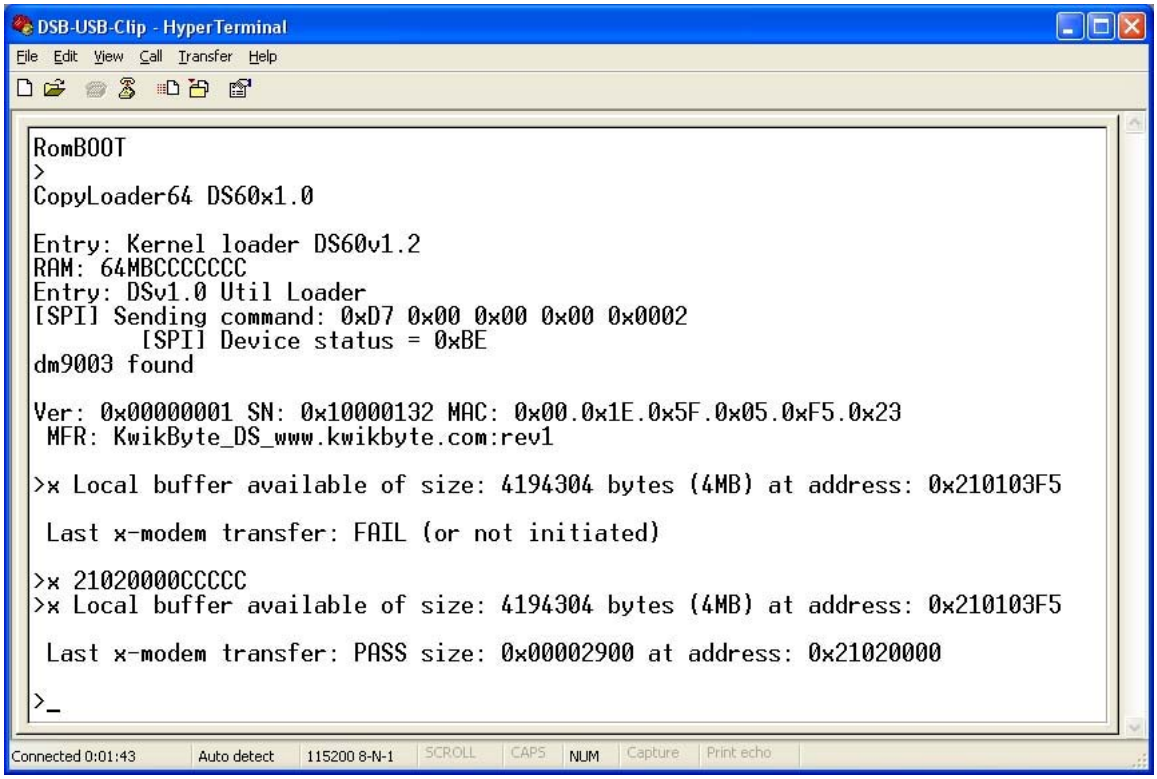
2.1 Load the UL

Follow the instructions from the UL document exactly (http://www.kwikbyte.com/driverstation/binary/utilLoader.bin) through step #23. Finish step #23. Stop. Do not perform step #24. You can now close that document because we won't use it any more in this process.

2.2 Update the KL

1) As performed previously, execute a transfer from the PC using Transfer -> Send File. Select the kernelLoaderv12.bin file for Xmodem download – just like we did earlier. Click Send and watch the progress. This only takes a few seconds because the file size is small.

2) The Driver Station now has the kernelLoaderv12.bin image in RAM. We need to program it to non-volatile memory (NVM). The NVM used on the Driver Station is serial flash. Type x Enter to get the size of the last transfer from the Driver Station's view.



3) This step is critical!!! Pay very close attention to the commands and make sure you type them exactly as listed.

   From this example, the transfer size is 0x2900. Unlock the boot loader with the following two commands:

   spi_erpx Enter
   spi_wrpx 0 Enter

   Let's program flash with a command of the format "spi_write <flash destination> <source address> <size>". In this case,

   spi_write 1080 21020000 2900 Enter

   Restore the lock on the boot loader with the following commands:

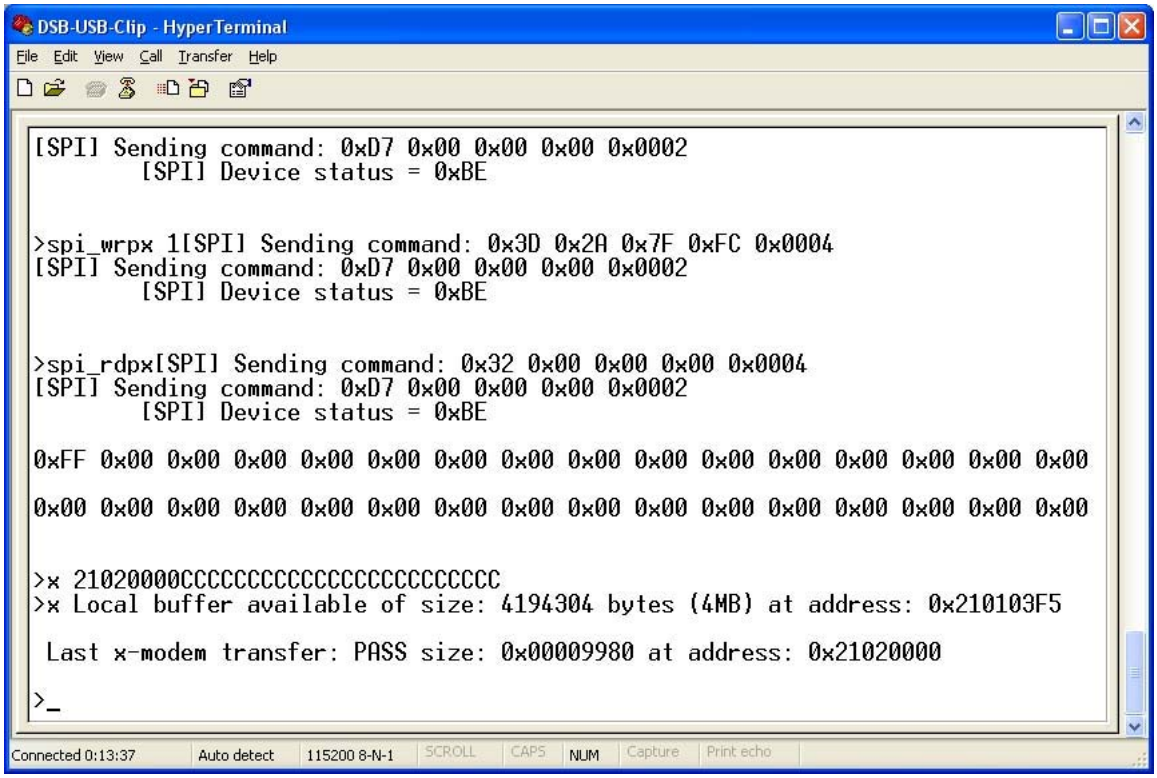
   spi_erpx Enter
   spi_wrpx 1 Enter

   You should see lots of "[SPI]" messages scrolling during these operations.

4) The kernel loader is now updated. Let's download the alternate OS loader while we are here. The rest of the steps are not as important as step #3 because they are considered recoverable. Initiate a transfer, again, to the same buffer address:

x 21020000 Enter

5) Notice the 'C' characters again and send the altLoader.bin file for Xmodem transfer. This takes about 6 seconds to transfer. Verify the transfer size by typing x Enter.



The transfer size is shown as 0x9980.

6) Now, let's program this image at the beginning of the last 256-page block of flash:

spi_write 7FE000 21020000 9980 Enter

7) Again, many [SPI] type message scroll by during the programming operation. After about 20 seconds, it completes with "Buffer written" message.

We're done with this process! Verify everything runs as normal by entering the command reset Enter.

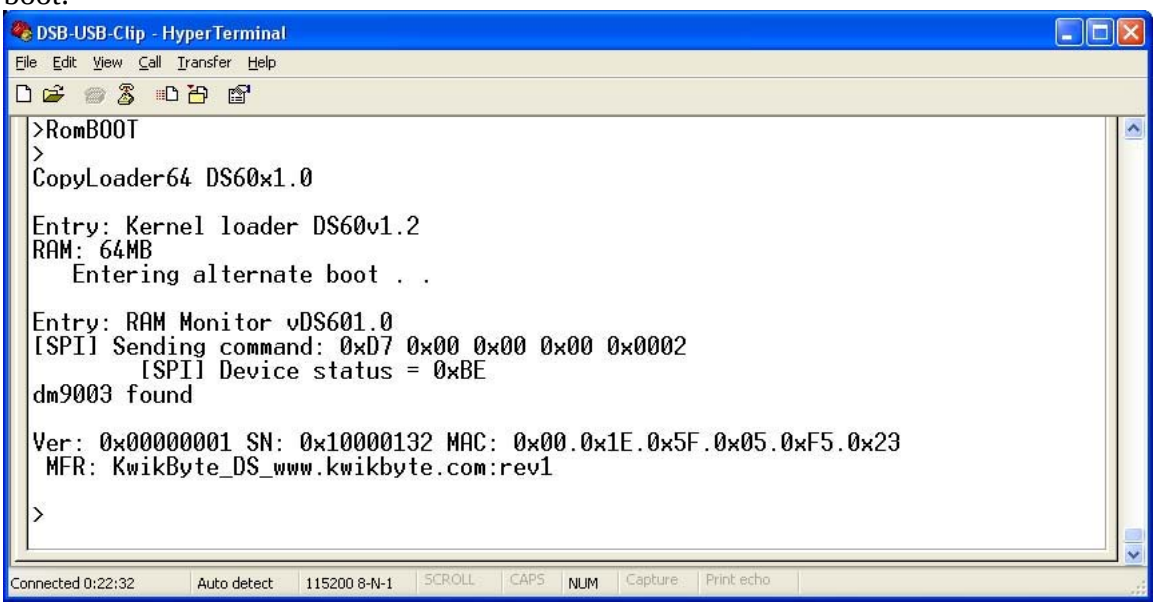
The only observable differences are

1) The boot-up logo is really strange! We will fix that (easily) in the next installment by loading your custom image.
2) The version reported by the kernel loader is now 1.2.

2.3 Results

The new kernel loader obtains the boot-up logo information from a separate flash location. This makes it very easy to modify the logo without changing the extra-important boot sections.

The new kernel loader also accepts a new "special key" sequence. If you apply power to the DS while holding the up-arrow and SELECT buttons, the DS will perform an alternate boot:



This feature will be used to boot another version of Linux.

3 Revisions

16JAN2009 Creation

C Logo Update Instructions

FIRST DRIVER STATION KERNEL LOADER UPDATE INSTRUCTIONS

1 Introduction

This document describes steps to change the boot-up logo on the Driver Station (DS) v1.0. The updated kernel loader (KL) is required in order to perform this procedure. The KL must be version 1.2 or later.

By this time, you should be familiar with how-to connect to the DS using the low-level serial clip. This document assumes you already have the terminal window up and running with a live connection to the DS.

1.1 Documentation

It is assumed the steps in the KL document have been completed already:
http://www.kwikbyte.com/driverstation/doc/DS_kernelLoaderv12.pdf

1.2 Equipment

You need:

1) A means of communicating with the DS at a low-level. This document uses the DS USB Adapter Clip with supplied USB extension cable.

2) A PC. The host OS can be Windows® or Linux. Other OS may work, but have not been tested.

1.3 Software

You need:

1) A host PC terminal emulator program like HyperTerm, minicom, Kermit, etc.

2) The bmpConversion utility program:
http://www.kwikbyte.com/driverstation/binary/bmpToLogo.bin

3) A Windows® formatted bitmap image for the new logo. This must be monochrome image of size 128x64. We will call this image newlogo.bmp.

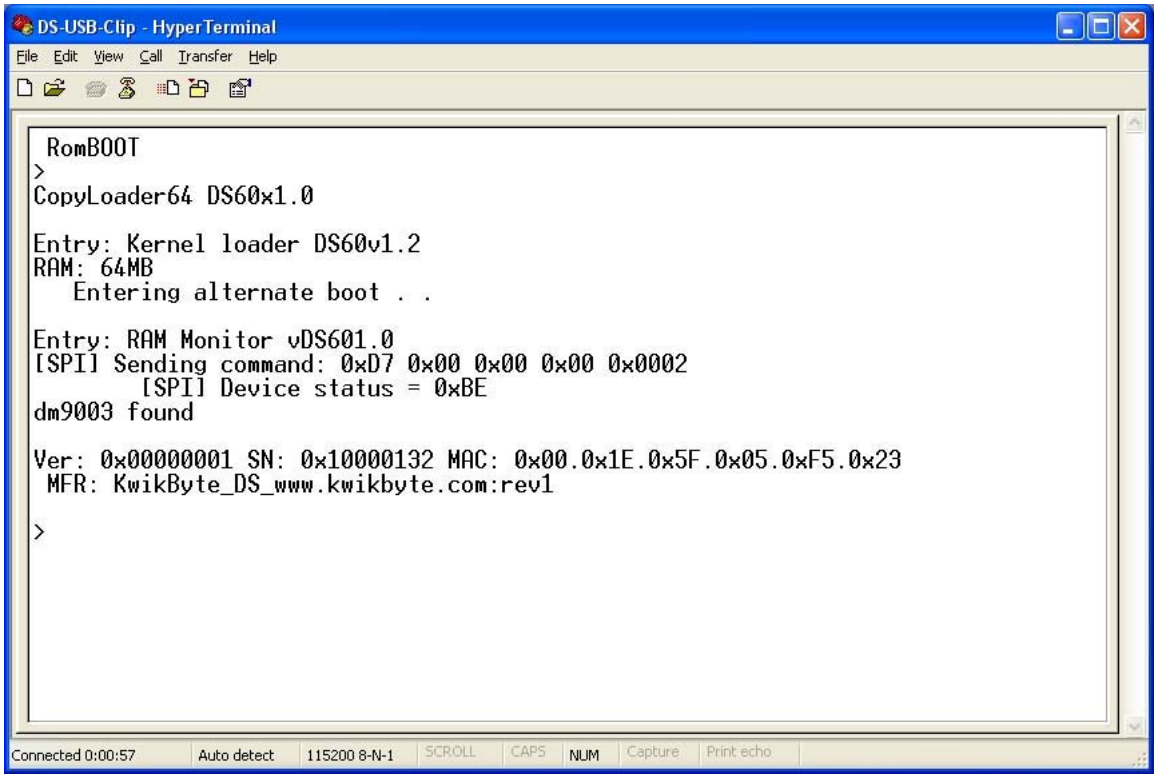
1.4 Support

Please read the instructions carefully. If you have questions or helpful comments, please send them to driverstation@kwikbyte.com.

2 Logo Update Instructions

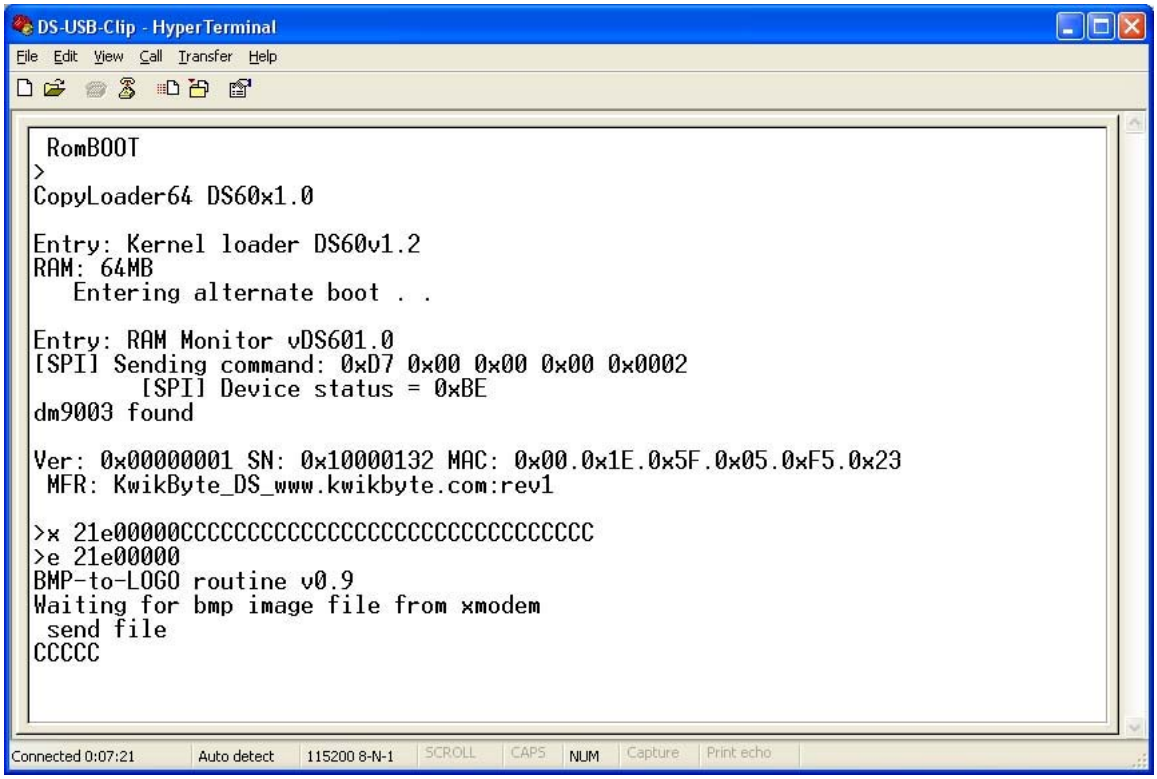
2.1 Boot the Alternate Loader

1) Hold down the 'UP Arrow' and 'SELECT' buttons while applying power (cold-boot) to the DS. You should see this screen:

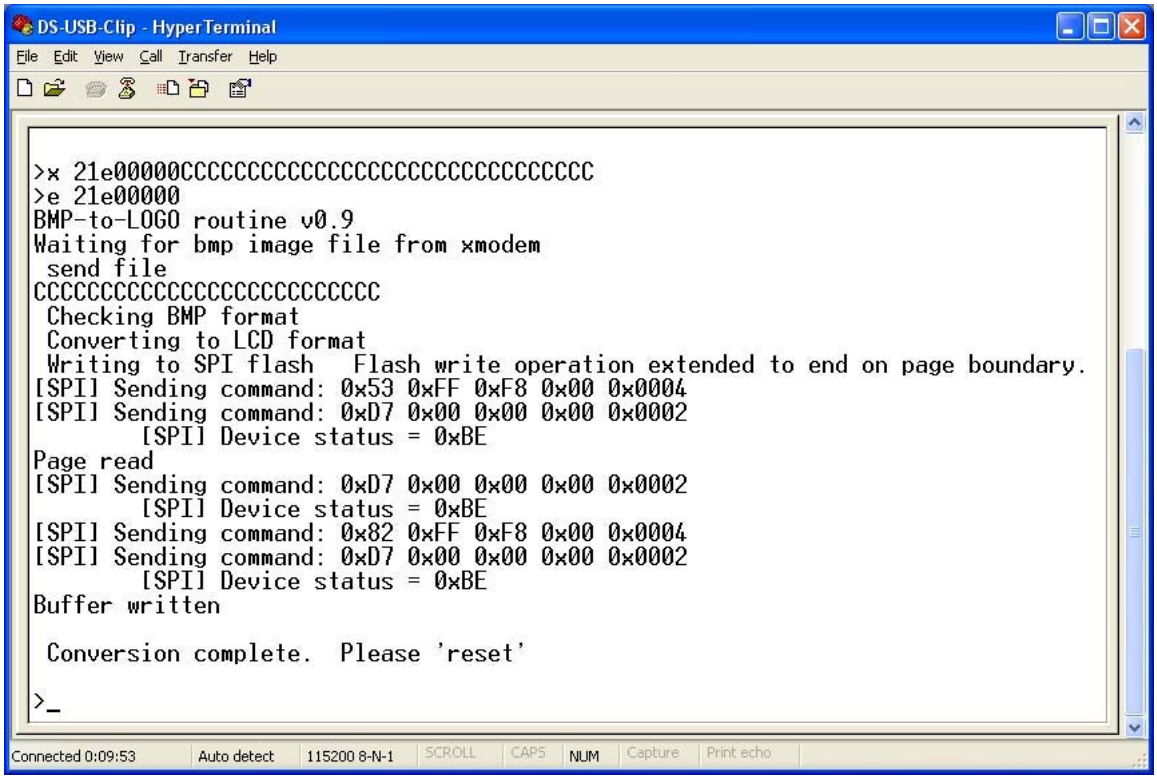


2) Start a transfer to address 0x21e00000 by typing x 21e00000 Enter. Notice the 'C' characters indicating the DS is waiting for a Xmodem transfer.

3) Send the bmpToLogo.bin file to the DS using Transfer->Send File with Xmodem protocol. This takes about 4 seconds once the transfer is started.
4) Now, execute the utility by typing e 21e00000 Enter.



5) Send the newlogo.bmp file to the DS using Transfer->Send File with Xmodem protocol (just like before). If the file is in the correct format and size, you will see this screen reporting 'Conversion complete':



6) Now type reset Enter and watch the DS boot with your new logo. You can repeat this process to update the logo again. You can change the polarity of the image with your favorite image editing software program.

3 Revisions

19JAN2009 Creation

D Working GPIO Test

D.1 main.c

```
/*
*
* Sample application to toggle a GPIO pin and read another
*  on the KB9260 and related products.
*
* Copyright (C) 2009 KwikByte <www.kwikbyte.com>
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
* or visit http://www.gnu.org/copyleft/gpl.html
*
* 22MAY2009  Initial creation (KwikByte)
*/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/shm.h>

#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <unistd.h>
#include <fcntl.h>
#include <strings.h>
#include <string.h>
#include <time.h>

#include "AT91SAM9260.h"

/* ********************** LOCAL DEFINES ****************************** */

/* We will use signals in header SV3 for this demo
 *
```

```
 * Reference: SV3 pin 1  = +3.3V
 *            SV3 pin 9  = GND
 *            SV3 pin 10 = GND
 *
 * Use PC14 (SV3 pin 2) as output pin
 * Use PC5  (SV3 pin 4) as input pin
 *
 * see AT91SAM9260 datasheet for more information
 */
//#define OUTPUT_PIN AT91C_PIO_PC14
#define INPUT_PIN AT91C_PIO_PB20

static int memMapFile;
static AT91PS_PIOMAP at91PioCtrlr;

/* ********************** UTILITY FUNCTIONS ****************************** */


/*
 * Read the input pin
 */
static unsigned GetInput(void)
{
if ((at91PioCtrlr->PIOB_PDSR) & (INPUT_PIN))
{
return (1);
}

return (0);
}


/* *************** HARDWARE SUPPORT FUNCTIONS ************************ */


/*
 * Access the hardware
 */
static int OpenSystemController(void)
{
if ((memMapFile = open("/dev/mem", O_RDWR | O_SYNC)) < 0)
{
printf("ERROR: Unable to open /dev/mem\n");
return (errno);
}
at91PioCtrlr = mmap(NULL, 4096, PROT_READ | PROT_WRITE,
MAP_SHARED, memMapFile, (unsigned)AT91C_BASE_AIC);
if (at91PioCtrlr == MAP_FAILED)
{
printf("ERROR: Unable to mmap the system controller\n");
close (memMapFile);
```

```
memMapFile = -1;
return (errno);
}
/* set digital input ports */
at91PioCtrlr->PIOB_ODR = INPUT_PIN;
at91PioCtrlr->PIOB_IFER = INPUT_PIN;
//at91PioCtrlr->PIOB_PPUDR = INPUT_PIN;
at91PioCtrlr->PIOB_PER = INPUT_PIN;

/* set digital output ports init value = 0 */
//at91PioCtrlr->PIOC_CODR = OUTPUT_PIN;
//at91PioCtrlr->PIOC_PPUDR = OUTPUT_PIN;
//at91PioCtrlr->PIOC_PER = OUTPUT_PIN;
//at91PioCtrlr->PIOC_OER = OUTPUT_PIN;

/* sync memfile */
return (0);
}


/* ********************** MAIN LOOPS ****************************** */


int main(int argc, char **argv)
{
printf("Starting...\n");
if (OpenSystemController())
{
printf("Unable to map hardware resources\n");
return (-1);
}

while (1)
{
/* loop forever toggling output at 1 second interval */

/* set high */
//SetOutput(1);
//printf ("GPIO output is now high\n");

/* get input */
printf ("GPIO input is %d\n", GetInput());

/* wait for 0.5 seconds */
usleep(500000);

/* set low */
//SetOutput(0);
//printf ("GPIO output is now low\n");

/* get input */
```

```
printf ("GPIO input is %d\n", GetInput());

/* wait for 0.5 seconds */
usleep(500000);
}

return (0); /* never get here */
}
```

D.2 Makefile

```
#
# Sample GPIO Program for KB9260 and related products
#
# Copyright (C) 2009 KwikByte <www.kwikbyte.com>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
# or visit http://www.gnu.org/copyleft/gpl.html

# 22MAY2009  Initial creation (KwikByte)

BASE_TOOLS=../buildroot/output/host/usr/bin/arm-buildroot-linux-uclibcgnueabi-
CC=${BASE_TOOLS}gcc
LD=${BASE_TOOLS}ld

LDFLAGS= -g -Wall -Wl,--dynamic-linker=/lib/ld-uClibc.so.0

.PHONY: all

all:: gpio_app

gpio_app: main.o
${CC} ${LDFLAGS} $? -o $@ ${LIBS} ${EXTRALIBS}

.c.o:
${CC} ${CFLAGS} -c $<

cleandir::
$(RM) *.o gpio_app
```

```
clean:: cleandir

distclean:: cleandir
```