ABSTRACT

This application note describes how to communicate with the Hercules UART boot loader. The UART boot loader is a small piece of code that can be programmed at the beginning of flash to act as an application loader as well as an update mechanism for applications running on a Hercules Cortex-R4 based microcontroller (RM4x and TMS570LSx).

Table of Contents

1. Introduction

An important requirement for most flash-memory-based systems is the ability to update firmware when installed in the end product. This ability is referred to as in-application programming (IAP). The UART bootloader provides a means of writing, reading, and erasing a predefined section of the program flash memory that typically holds the user application code.

This document describes how to work with and customize the Hercules UART boot loader application. The boot loader is provided as source code which allows any part of the boot loader to be completely customized. The bootloader has been built and validated using CCSv5 on the RM48 Hercules Development HDK.

The following is an overview of the organization of the source code provided with the boot loader.

Table 1, List of Source Code Files Used in SPI Boot Loader

sys_startup.c	The start-up code used when TI's Code Composer Studio (CCS) compiler is being used to build the boot loader.
bl_check.c	The code to check if a firmware update is required, or if a firmware update is being requested by the user.
bl_check.h	Prototypes for the update check code.
bl_commands.h	The list of commands and return messages supported by the boot loader.
bl_config.h	Boot loader configuration file. This contains all of the possible configuration values.
bl_flash.c	The functions for erasing, programming the flash, and functions for erase/program check
bl_flash.h	Prototypes for flash operations
bl_link.cmd	The linker script used when the CCS compiler is being used to build the boot loader.
bl_main.c	The main control loop of the boot loader.
bl_packet.c	The functions for handling the packet processing of commands and responses.
bl_packet.h	Prototypes for the packet handling functions.
bl_uart.h	Prototypes for the UART0 transfer functions.
bl_uart.c	The functions for transferring data via the COM0 port.
bl_vimram.c	VIM RAM table definition and initialization
hw_gio.c	Low level GIO driver
hw_gio.h	Prototypes for low level gio driver
hw_het.c	Low level NHET driver
hw_het.h	Prototypes for low level NHET driver
hw_interrupt_handler.c	Define the INT handlers
hw_pinmux.c	Function for define the pinmux

hw_pinmux.h	Prototypes for pinmux functions
hw_sci.c	Low level SCI driver
hw_sci.h	Prototypes for low level SCI driver
hw_system.c	Initialize system registers and PLL
bl_ymodem.c	Function for define the ymodem protocol
bl_ymodem.h	Prototype define the variables and functions
startup_eabi.c	Global variables initialization
sys_intvecs.asm	Interrupt vectors
sys_svc.asm	SW INT routines

2. Software Coding and Compilation

- The bootloader code is implemented in C, ARM Cortex-R4F assembly coding is used only when absolutely necessary. The IDE is TI CCS5.2.1.
- The bootloader is compiled in the 32-BIT ARM mode.
- The bootloader is compiled and linked with the TI TMS470 code generation tools V 4.9.5.
- The maximum size of the bootloader executable is less than 32KB (Size of the 1st sector flash on RM48 and TMS570LS31x devices).

3. On Reset

On reset, the MCU enters in supervisor mode and starts executing the bootloader. The interrupt vectors are setup as follows:

Table 2, Vector Table in CAN Boot Loader

Offset	Vector	Action
0x00	Reset Vector	Branch to entry point of bootloader (c_int00)
0x04	Undefined Instruction Interrupt	Branch to application vector table
0x08	Software Interrupt	Branch to application vector table
0x0C	Abort (Prefetch) Interrupt	Branch to application vector table
0x10	Abort (Data) Interrupt	Branch to application vector table
0x14	Reserved	Endless loop (branch to itself)
0x18	IRQ Interrupt	Branch to VIM
0x1C	FIQ Interrupt	Branch to VIM

4. During Bootloader Execution

During bootloader execution:

- MCU operates in supervisor mode
- MCU Clock is reconfigured and is maintained throughout the bootloader execution.
    - Clock Source: OSCIN = 16MHz
    - System clock: HCLK = 160Mhz
    - Peripheral clock: VCLK = 80 Mhz
- No interrupts are used.
- Fix point is used throughout the bootloader execution.
- F021 API V1.5 executes in RAM
- The SCI is configured as 115200, 8-N-1

5. MCU Initialization during Bootloader Execution

- Operating Mode: supervisor mode
- HCLK Frequency: OSCIN x 120 / 12
- VCLK Frequency: VCLK = HCLK / 2
- Peripheral Control: Peripherals enabled
- SCI setup: The SCI/LIN is setup for SCI communication. The setup is 115200 8-N-1
- MCU do self-test, pbist, parity check at startup.

6. The Protocol Used in UART Bootloader

The bootloader is based on Ymodem protocol. The Ymodem protocol sends data in 1024-byte blocks. Error detection is applied to data blocks transmitted to the RM48 internal RAM. This is done through a comparison between the transmitted and received data. Blocks received unsuccessfully are acknowledged with a NAK (Negative Acknowledgement). For more details about the Ymodem protocol, please refer to the existing literature.

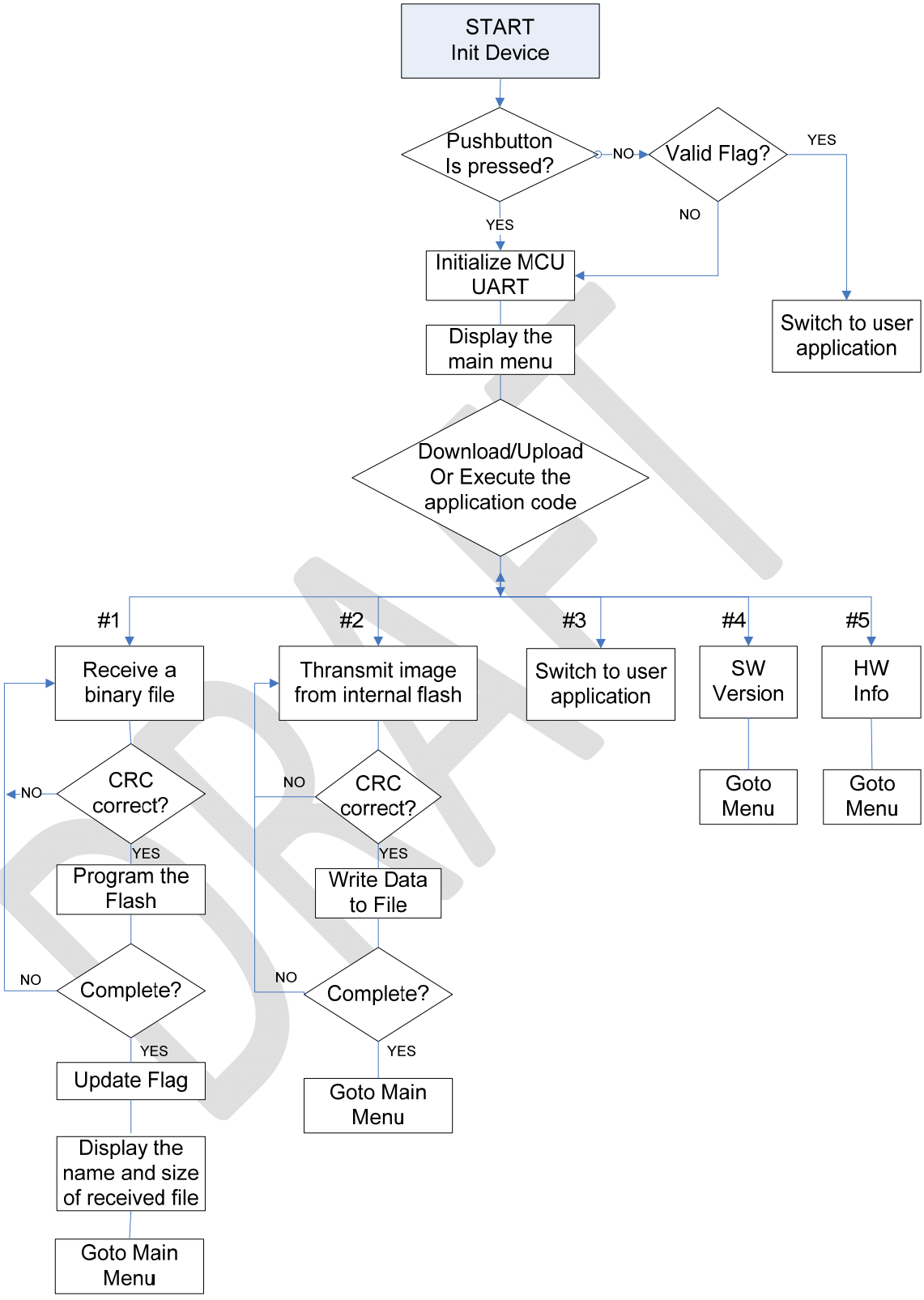

Figure 1, The FlowChart of UART Bootloader

7. UART Bootloader Operation

The UART bootloader is built with CCSv5.x and loaded through the JTAG port into the lower part of the program memory at 0x0000.

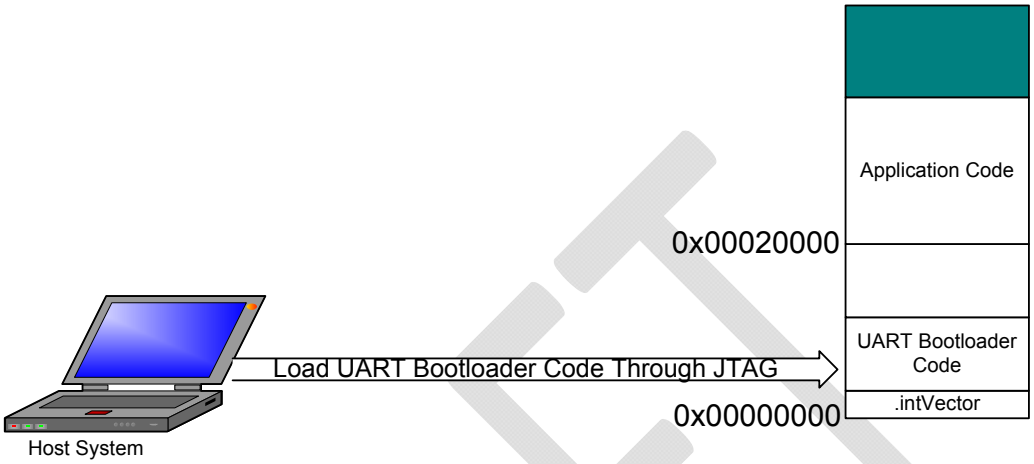

Figure 2, The UART Bootloader Is Loaded Through The JTAG port

After HDK reset, the start-up code copies the Flash API of boot loader from flash to SRAM, and execute the boot loader in Flash.

First, it will checks to see if the GPIO_A7 pin is pulled. If GPIO-A7 is pulled LOW, the application code is forced to be updated. The GPIO pin check can be enabled with ENABLE_UPDATE_CHECK in the bl_config.h header file. On RM48 HDK, the push button S1can toggle GPIO-A7.

Then, it will check the flag at 0x0007FF0. If the flag is a valid number (0x5A5A5A5A), the bootloader will jump to the application code at 0x00020000. If the flag is not the valid number, it will configure the UART, then start to update the user application code by calling UpdaterUart(). After all the application code is programmed successfully, the flag is also updated.

The updating results in the following menu displayed in the HyperTerminal window.

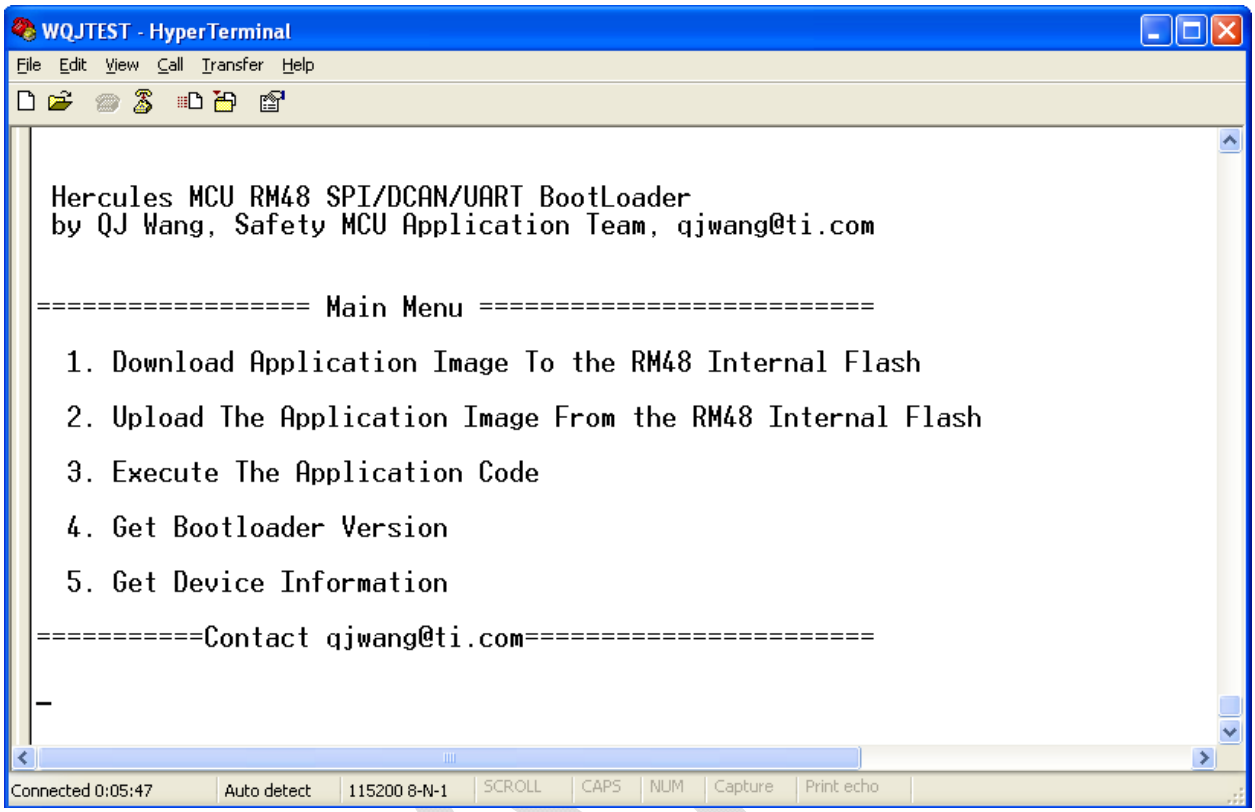

Figure 3, UART Bootloader Main Menu

1. **Download the user application code to RM48 flash.**

   To download a binary file via HyperTerminal to the RM48 internal flash, follow the procedure below:

   ● Press “1” on the keyboard to choose the menu “Download image to internal Flash”

   Then, in the **Transfer** menu, select “**Send file…**”

   ● In the **Filename** field, type the name and the path of the binary file to be sent.

   ● In the **Protocol** list, choose the **Ymodem** protocol,

   ● Click the “**Send**” button.

   Following these steps, the bootloader loads the binary file into the RM48 internal Flash. The bootloader will display the file name, and file size in the Hyperterminal window.

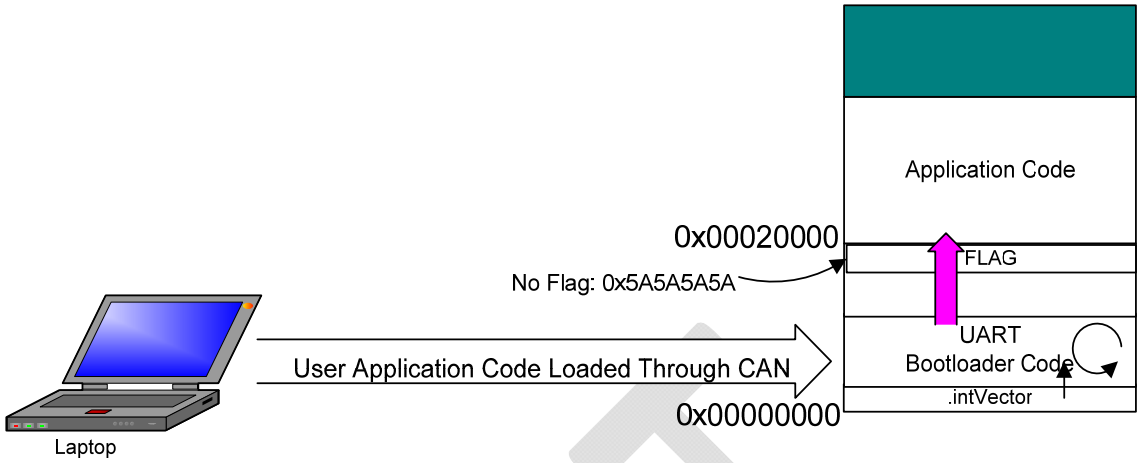

Figure 4, The User Application Code Is Loaded Through The UART Bootloader.

2. Upload the application code from RM48 flash

To upload a copy of the internal Flash memory started from the user application address, do as follows:

- Press 2 on the keyboard to select Upload image from the RM48 internal Flash
- Select **Receive File** in the **Transfer** menu.
- Choose the **Directory** of the binary file you want to create.
- From the **Protocol** list, select the **Ymodem** protocol.
- Click on the **Receive** button.

The file *`UploadedApplicationImage.bin`* is uploaded to the directory you defined in step 2.

3. Execute the application

Once the new application is downloaded successfully, press 3 on the keyboard to select the Execute.

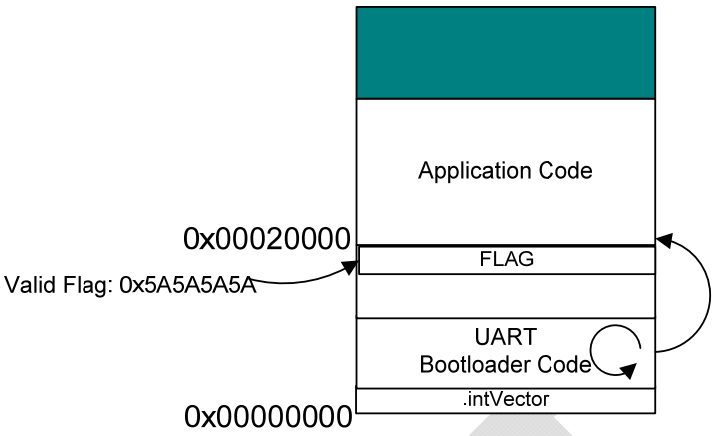

Figure 5, The UART Bootloader Jumps to Application Code

4. **Get Bootloader SW Version**

   To get the SW version, press 4 on the keyboard to retrieve the SW version from RM48 UAR bootloader.

5. **Get the RM48 Device Information**

   To get the device information, press 5 on the keyboard to retrieve the device information from RM48 UAR bootloader.

8. HyperTerminal Configuration

The bootloader requires a PC running HyperTerminal with the following settings:

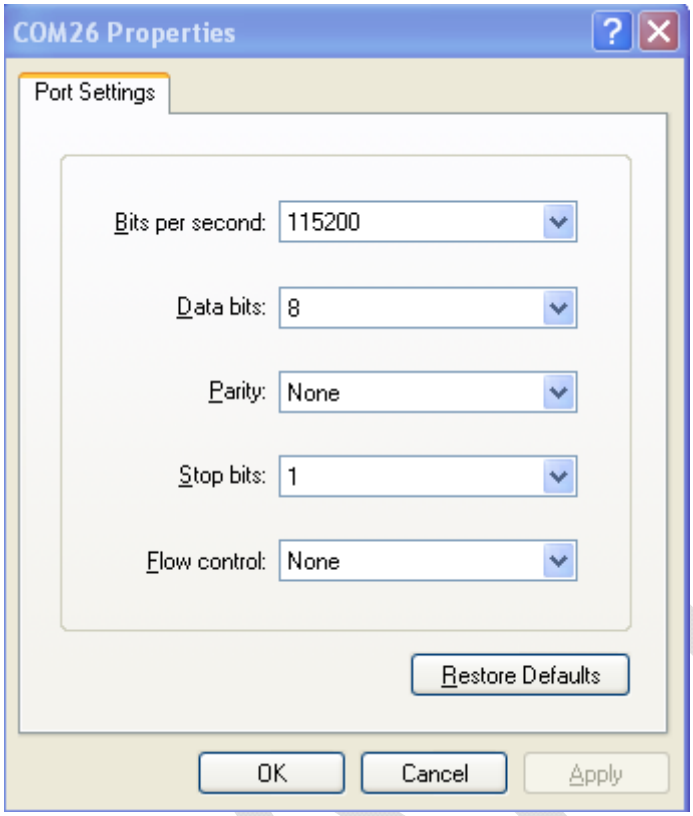

Figure 6, COM Port Properties

9. References

- Datasheet of RM48Lx50 16/32-Bit RISC Flash Microcontroller (SPNS174)
- F021 Flash API (V1.5, SPNU501A)