Source Code for Implementing a Firmware Downloader in Your Host Microcontroller System

BL600, BL652, BT900, RM1XX, RM1XX_PE

Application Note *v1.0*

Introduction

New firmware is provided by Laird when there are new features or bug fixes available. It is delivered in a file with the extension ".uwf,"the contents of which are formatted in a customized way which is neither pure binary nor in hex. This format ensures that if the firmware is partially loaded, there is no risk of the bootloader being confused and launching the partial firmware. Laird offers a Windows utility that uploads the contents of the ".uwf" file to that module.

Many customers have requested the ability to download the new firmware from a host microcontroller which is physically connected to the UART pins of the module, a scenario where connecting a Windows PC is difficult. In some cases customers have solved this by writing bridging code between an external facing UART and the UART that is connected to the module.

Many Laird modules are capable of having firmware downloaded over the module's UART interface. This download process is expedited with the help of a bootloader embedded in the module. Laird now offers C++ source code that you may embed into your host microcontroller. This code acts like a black box so that a firmware upgrade is as simple as calling a single blocking function called UwFlashUpgradeFirmware();. The only requirement is that the host microcontroller has access to the firmware .uwf file (for example, from a serial flash, SDIO card, Wi-Fi or Ethernet link).

Technical Overview

The blocking black box engine does not spawn any threads and blocks for around 50 to 120 seconds.

To avoid interfering with your normal functionality, it polls a function called MiscPollBackgroundTarget() where you can put code that can be polled; alternately, you may launch the function UwFlashUpgradeFirmware() in a thread if you have an RTOS. If you decide to run the engine in a separate thread, you must ensure multi-thread access to serial buffers are appropriately syncronised.

Obviously, Laird cannot fully account for the specifics of the UART interface that exists in your microcontroller. Therefore, you must provide code in functions that are called from within the black box to enable interaction with your UART. In addition, you must provide code for file open/read/close functions that get the content of the .uwf file. Similarly, as the firmware upgrade progresses there are functions that convey the progress. The upgrade progress function calls can remain as stubs if you prefer, but you may find that monitoring the state value will convey what is happening in real time.

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610

FILES TO INCLUDE IN YOUR PROJECT

- (*) All .cpp and .c files in folder *SourceLibrary*
- (*) TargetEmbed.h and the appropriate TBootLdr<PRODUCT>.cpp from folder *WirelessModule*. Select the TBootLdr file that contains your product's name in the filename. You are encouraged to contact a Laird representative with any questions.
- (*) Copy all files from folder 'HostTemplate' into a folder of your choice and then update them.

Note: Look for the text "TODO" in all these files. This is where you add your code that provides UART and file read access.

CONFIGURATION

There are five files in the subfolder *HostTemplate* that you must modify. Laird recommends that you duplicate that folder and modify the duplicates. Load them in any text editor and search for the text "TODO". These files are commented in great detail to explain the functionality you need to code.

For a completed example, see the subfolder *WindowsExample* where the black box has been used to prove the upgrade process in a Windows PC.

The file responsible for implementing your custom UART access code is *TSerialPortTarget.cpp*. One working strategy is to provide a low level receive buffer into which a thread or an interrupt routine stores the incoming data and then leave it to the engine to read from its thread (or background). This happens without any further intervention.

The function SerialRxDataCount() is polled by the engine, and when you provide a count that is non-zero, the function TSerialPortTarget::GetBlock() is eventually called to read the newly-arrived data. You must ensure that when the function reads the data there are no multithreading issues. Once TSerialPortTarget::GetBlock() reads the data, it is assumed that it will be removed from the underlying low level buffer.

Firmware upgrade time can vary from 50 to 120 seconds for the same file and same module. This timing is dependent on the value of #define UWF_FIRMWAREFILE_MAX_LINE_LEN which is set by default to 1024. If that value is reduced (for example, to 256) the engine works with smaller chunks of information from the .uwf file, which has the impact of increasing the time. It is left to you to determine how much memory you have spare, which is taken from the heap.

If you do decide to alter the value of UWF_FIRMWAREFILE_MAX_LINE_LEN or any other #define in the file WirelessModule\TargetEmbed.h, please do NOT alter that file directly. Instead, **modify Target.h** as shown in the following example:

- #undef UWF_FIRMWAREFILE_MAX_LINE_LEN
- #define UWF_FIRMWAREFILE_MAX_LINE_LEN (some_new_value)

TESTING

The code has been tested in the harness in the folder *WindowsExample* where Borland Builder v6 has been used along with a serial port VCL component called Asyncpro.

Both TARGET_BL65X and TARGET_BL6XX has been tested.

REVISION HISTORY

Version	Date	Notes	Approver
1.0	25 Jan 2017	Initial Release	Jonathan Kaye

Americas: +1-800-492-2320
Europe: +44-1628-858-940
Hong Kong: +852 2923 0610