Reed MFi - Firmware

The CRD42L42-MFi is an accessory that interfaces with an Apple device as described in the Accessory Interface Specification document that is available to members of the Apple Developer program. This document provides detailed information to describe the firmware in the CFD42L42-MFi. The DS1133RD3 should be reviewed before using this document. This document provides details to perform the following tasks:

1. Obtain the tools used to generate the firmware.
2. Understand the bootloader implementation
3. Modify the customer-specific LAM configuration fields.
4. Use the example iOS app.

Table of Contents

1. STM8 Software Development Toolchain

Install the tools provided by ST Microelectronics in preparation for making changes to the project. Release notes provided by ST Microelectronics list the requirements for the host PC running Windows operating system. The following tools are used:

1. ST Visual Develop (STVD)
2. ST Visual Programmer
3. Cosmic C compiler for STM8

Follow the instructions to select the Cosmic compiler in the Project Settings. The part number of the MCU used on both the CRD42L42-MFi and the CDB42L42-MFi is STM8L101F3U6ATR.
Cirrus Logic provides demonstration source code in a workspace named CRD42L42-MFi-FW.stw. The workspace is opened within the ST Visual Develop tool.

The workspace includes two essential projects.

1. stm8l10x_LAM_init_test_Programming.stp
2. stm8l10x_FW_Programming.stp

After the workspace is open, the active project can be set.

2. iOS Development Tools

To develop the iOS app that performs the firmware update you need a Mac computer (OS X 10.10 or later) running the latest version of Xcode. Xcode includes all the features you need to design, develop, and debug an app. Xcode also contains the iOS SDK, which extends Xcode to include the tools, compilers, and frameworks you need specifically for iOS development.

Download the latest version of Xcode on your Mac free from the App Store.

3. Bootloader

As detailed in the Accessory Interface Specification (Section 4.13.6 – Firmware Update), the accessory's firmware can be updated by an end user from an Apple product running iOS.

The MCU on both the CRD42L42-MFi and the CDB42L42-MFi is delivered pre-programmed with default firmware that implements a custom bootloader. This bootloader (BL) is permanently write-protected in the user boot code (UBC) area. The memory map for this project is shown in the following table.

Starting Address	Starting Block	Total Bytes	Name	Protected?
0x8000	1	128	Interrupt Vector Table (IVT)	Yes
0x8080	3	2112	Bootloader (BL)	Yes
0x8980	35	128	Firmware Code (FC) Interrupt Vector Table (IVT)	No
0x8A00	37	5760	Firmware Code (FC)	No
0x9FC0	128	64	EEPROM	No

The BL manages memory to ensure that under all conditions the accessory's firmware can be updated via the Lightning interface.The BL code is always executed immediately after the MCU resets. Under normal conditions, the BL will then transfer control to the Firmware Code (FC). In the event that the integrity of the FC cannot be assured, the system will enter a mode that is waiting for an update from an app on the iOS device.

To determine the integrity of the FC, the BL reads several locations in the FC IVT (See the memory map table) to verify that the FC has been

properly loaded. The IVT is always the last location programmed when new FC is loaded.

After the code has successfully loaded during a firmware update, the host and the BL agree that the FC has been properly loaded and then the BL transfers control to the FC.

4. Lightning Audio Module (LAM) Configuration

As detailed in the Accessory Interface Specification (Section 4.13.1 – Connectivity), the accessory must be visible in the Setting app and the accessory must identify with the correct information strings. Configuration strings are stored in the LAM by sending configuration packets as described in the Accessory Interface Specification (Section 4.12.3 – Configuration Packets),

These strings can be modified in the firmware by making the appropriate changes in the file: version.h .

#define INFO_NAME	"CRD42L42-MFi"
#define INFO_MANF	"Cirrus Logic"
#define INFO_MODEL	"CS42L42"
#define INFO_PREF_APP	"com.cirrus.CRD42L42-MFi"
#define INFO_EA_NAME	"com.cirrus.CRD42L42-MFi"

Figure 1: Example information strings

For firmware purposes, each of the unique configuration packets is assigned the packet type value (as listed in the in the Accessory Interface Specification) in the file packet.h.

#define accConfigurationInformation	0x03
#define accNameInformation	0x04
#define accManufacturerInformation	0x05
#define accModelInformation	0x06
#define accSerialNumberInformation	0x07
#define accPreferredAppBundleIdentifierInformation	0x08
#define accExternalAccessoryProtocolNameInformation	0x09
#define accHIDComponentInformation	0x0A
#define accAudioTerminalInformation	0x0B

Figure 2: Example values assigned to configuration packet names

4.1. Model Specific Configuration

As detailed in the Accessory Interface Specification (Section 4.12.3.9), the accessory must inform the LAM of it's audio input/output terminal configuration in the accAudioTerminalInformation packet.

```
case accAudioTerminalInformation:

        tx_payload[0] = 0x02;          // audio output terminal type (jack)
        tx_payload[0] = 0x01;          // audio output terminal type (headset)
```

Figure 3: Lightning Audio Module Audio Output Terminal Type

```
case accAudioTerminalInformation:

        tx_payload[6] = 0x00;          // audio input terminal type (no microphone)
        tx_payload[6] = 0x01;          // audio input terminal type (headset microphone)
        tx_payload[6] = 0x02;          // audio input terminal type (headset jack)
```

Figure 4: Lightning Audio Module Audio Input Terminal Type

5. Example LAM Configuration Project

This example will change the accManufacturerInformation that is stored in the LAM. A special firmware will be generated to perform LAM configuration. This project does not use the bootloader. After the LAM configuration has executed, the MCU will be restored to the default factory state.

5.1. Generate new firmware.

1. Start the ST Visual Develop tool.
2. Open the workspace: CRD42L42-MFi-FW.stw
3. Set the Active Project to: stm8l10x_LAM_init_test
4. Open the file: version.h
5. Change line 12:
    a. from: #define INFO_MANF "Cirrus Logic"
    b. to: #define INFO_MANF "Acme Company"
6. Build the modified project.

5.1.1. Modified example for changing terminal type

The procedure for changing the audio output terminal type to headset and the audio input terminal type to no microphone is similar.

In step 4, open the file packet.c

In step 5, under case accAudioTerminalInformation: assign tx_payload[0]=0x01 and assign tx_payload[6]=0x00

5.2. Load the new firmware.

1. Start the ST Visual Programmer tool.
2. Open the project: stm8l10x_LAM_init_test_Programming.stp
3. Connect the headset to the 3.5mm jack.
4. Connect ST-LINK/V2 to J3.
5. Connect CDB Lightning connector into the Lightning Receptacle of the iOS device.
6. Program (all tabs) in the MCU.
7. Remove Lightning connector from iOS device.
8. Disconnect ST-LINK/V2 from the CDB (J3).

5.3. Run the new firmware to perform LAM initialization.

1. Plug Lightning connector into the iOS device
2. Wait ten seconds. First observe the “orange LED” and then wait until the LED is dark.
3. Disconnect Lightning connector from the iOS device. (This is necessary to reset.)
4. Connect Lightning connector into the iOS device
5. Make the iOS device play music
6. Confirm that the new manufacturer name is visible in the Setting app

5.4. Restore the firmware with bootloader

1. In the ST Visual Programmer tool open the Project: stm8l10x_FW_Programming.stp
2. Connect ST-LINK/V2 to J3.

3. Connect CDB Lightning connector into the Lightning Receptacle of the iPhone
4. Program (all tabs) in the MCU.
5. Disconnect Lightning connector from iPhone.
6. Remove ST-LINK/V2 from the CDB (J3).
7. Plug Lightning connector into iPhone
8. Wait for “green LED” and then verify music is playing in headset.
9. Use the Setting app on the iOS device to confirm that the new manufacturer name is visible in the Setting app.

6. Using the iOS app

Cirrus Logic has created an app that implements a firmware update. This app may be modified as required by opening the Xcode project. The app can be built and tested on your iOS device. After the app has been installed, pressing the MODE button on the CDB or CRD will cause the iOS device to prompt for allowing the app to communicate. The app may be used to perform a firmware update on the CDB or CRD. After the Xcode project has been built (including the new .s19 file) the Update feature will load the new firmware into the CDB or CRD.

7. Example

The following steps are required to generate and load a firmware update.

1. Prepare the ST environment to create the build environment for the STM8L101F3U6ATR.
2. Modify the firmware.
3. Build a new image.
4. Prepare a firmware update app for the iOS device.
5. Use the Apple Developer tools to make the update app available on the iOS device.
6. Update the accessory firmware using the app.

8. CRD42L42 - iOS Communication Protocol

8.1. General Description

This protocol governs the data exchange between the iOS device and the CRD42L42/CDB42L42 MCU. The MCU runs in one of two modes, either executing bootloader firmware or CS42L42 codec control firmware. In both modes the MCU supports an exchange of a set of data packets with the connected iOS device running the CL MFi application. In bootloader mode the MCU waits for data packets containing blocks of the codec control FW. Once received, these blocks are flashed into persistent memory. In the codec control mode the MCU supports data packets with the codec register read/write operations. All instances of the data exchange are initiated by the iOS application by sending a data packet with the request to the MCU. There are 7 requests that the MCU firmware must support:

- Status Request
- Update FW
- Flash FW Block Request
- Boot FW Request
- Cancel Request
- Register Write Request
- Register Read Request

Not all requests are supported in all modes. For each of the seven, the MCU shall send back a data packet with the response. If the MCU does not respond within 3 seconds, the iOS application resends the request and repeats that process until a response is received or until a 30 second timeout has expired. In the latter case, the CL MFi application stops sending requests and reports to the user that the connected device is not responding. The requests Boot FW and Cancel may not send a response when they cause a mode switch. Once either of these two requests are sent, the iOS application waits for approximately two seconds and sends “Status Request”.

The format of the CRD42L42 data packet on the iOS application side is shown in Figure 5:

Field	Bytes	Description
Packet Tag	1	The type of the packet
Data	0~n	The payload

Figure 5: iOS Data Packet Format

The total size of the packet cannot exceed 128 bytes and the iOS application assumes that the data packet is always sent and received in a single write/read transaction. The packet size is not included in the packet itself and is determined using the byte count returned by the write or read operation. On the MCU side, the CRD42L42 packet is a payload of the Lightning Audio Module (LAM) serial protocol packet (see Lightning Audio Module topic in the Apple Accessory Interface Specification document).

8.2. Bootloader Mode Requests, Packets and Responses

8.2.1. Status Request (PacketTag 0x01)

No payload.

This packet is sent by the iOS application to determine the current MCU mode.

Status Response (PacketTag 0x01)

Field	Bytes	Description
Status	1	Possible values: 1 – codec control mode 2 – bootloader mode 3 - failed
Status details	0~3	Depends on the value of the Status field: - Status 1 (codec control mode) Three bytes with the version of the codec control FW returned - Status 2 (bootloader mode) No status detail bytes returned - Status 3 (failed) 1 byte of failure code returned

Figure 6:Status Response Packet

This packet is sent by the MCU in three cases:

- as the response to the status request from the iOS application (in both Codec Control and Bootloader modes)
- as the response to the flash FW block request (in Bootloader Mode)
- when receiving request packet fails or packet is not supported in the current mode

In response to a status request when the MCU is running codec control, the FW returns the status value 1 and three bytes with the firmware version.
In response to a status request or a flash FW block request that succeeds, the MCU running bootloader FW returns just the status value 2
In response to the request to flash a FW block that has failed, it returns status value 3 and the byte with the failure code:

1 – flashing failed (flashed data does not match received one)

When the MCU fails to receive a valid request packet it returns status code 3 and one of the following failure codes:

2 – invalid checksum (received data is corrupted)
3 – invalid data length (received data is corrupted or invalid)
4 – timeout while receiving packet (data is invalid or lost in transmission)
5 – invalid request (data is invalid or request is not supported in the current mode)

8.2.2. Flash FW Block Request (PacketTag 0x03)

Field	Bytes	Description
FW block number	1	The MCU physical flash memory block number

FW block data	64	Block of FW code

Figure 7: Flash FW Block Request Packet

This packet is sent by the iOS application in the process of updating codec control FW on the MCU. It holds one block of FW code along with the flash memory block number where it should be written. The first 35 blocks of the MCU flash memory are occupied by bootloader FW code and write protected. Any attempt to flash FW block data in the range 0~34 will fail. Also, the bootloader expects codec control code interrupt table in the blocks 35 and 36. The rest of the codec control FW code can reside in any flash memory blocks between 37 and 127 inclusive. To prevent the possibility of running incomplete FW code, the bootloader erases blocks 35 and 36 on the first request to flash FW block, and expects FW blocks to be sent starting from the last and ending with the first (which is always block #35). This ensures that if the MCU device is disconnected before all FW blocks are successfully flashed, the bootloader will see a missing interrupt table block(s) and will keep MCU in the bootloader mode until codec control FW is fully flashed.

In response to the Flash FW Block Request the MCU sends a Status Response packet (PacketTag 0x01) with the results of the flash operation (see above).

8.2.3. Boot FW Request (PacketTag 0x04)

No payload

This packet is sent by the iOS application after it has successfully sent all codec control FW blocks for flashing. On receiving this request, the MCU validates and starts executing the newly flashed FW. As the new FW code starts its state (memory) initialization it is not aware of any previous requests and does not send any responses. However, if validation of the newly flashed FW fails, the MCU sends a Status Response (PacketTag 0x01) with the status field value 2, i.e. that it is still in the bootloader mode and waits for FW blocks to flash.

8.2.4. Cancel Request (PacketTag 0x05)

No payload

The iOS application can send this packet to abort flashing new FW. If that request is received by the MCU before any flash FW block requests, it will start executing the old codec control FW and no response will be sent back. If new FW flashing has already started the MCU sends Status Response packet (PacketTag 0x01) with the status value 2, i.e. that it is still in the bootloader mode and waits for FW blocks to flash.

8.3. Codec Control Mode Requests, Packets and Responses

Status Request (PacketTag 0x01)

No payload.

This packet is sent by the iOS application to determine the current MCU mode.

The MCU running codec control FW responds with the Status Response (PacketTag 0x01) and status value 1 (codec control mode). The response packet also contains FW version (3 bytes) in the status details field.

8.3.1. Update FW Request (PacketTag 0x02)

No payload

The iOS application sends this request packet when the MCU is running codec control FW and the user has requested a FW update. On receiving this request the MCU switches into bootloader mode by starting execution of the bootloader FW. The bootloader FW detects that mode switch and responds with the Status Response (PacketTag 0x01) and status value 3 (bootloader mode)

8.3.2. Read Register Request (PacketTag 0xEB)

Field	Bytes	Description
Page	1	Codec register page
Address	1	Register offset from start of page
Length	1	Number of sequential registers to read

Figure 8: Read Register Request Packet

This request can be used to read codec control register(s) from the iOS application. The MCU reads the requested registers and sends their values back in the *Read Register Response (PacketTag 0xEB)*

8.3.3. Read Register Response (PacketTag 0xEB)

Field	Bytes	Description
Page	1	Codec register page
Address	1	Register offset from start of page
Length	1	Number of sequential registers read
Values	1~124	Register values

Figure 9: Read Register Response Packet

8.3.4. Write Register Request (PacketTag 0xEA)

Field	Bytes	Description
Page	1	Codec register page
Address	1	Register offset from start of page
Length	1	Number of sequential registers to write
Values	1~124	Register values to write

Figure 10: Write Register Request Packet

This request can be used to write codec control registers from the iOS application. The MCU writes received register values into the target registers and then reads their values back. It then responds with the *Read Register Response (PacketTag 0xEB)* (see above).

9. Revision History

Revision	Date	Changes
RC1	Mar '16	Initial Release
RC2	Apr '16	Updated text in Section 6

Contacting Cirrus Logic Support

For all product questions and inquiries, contact a Cirrus Logic Sales Representative. To find the one nearest you, go to www.cirrus.com.