Application Note

BRT_AN_063

FT9xx Programming, Debugging and Troubleshooting

Version 1.2

Issue Date: 10-06-2025

This document shows FT9xx programming and debugging methods and troubleshooting when developing with FT9xx MCU.

Bridgetek Pte Ltd (BRTChip)
1 Tai Seng Avenue, Tower A #03-05, Singapore 536464
Tel: +65 6547 4827
Web Site: http://www.brtchip.com

Table of Contents

1 Introduction 4
1.1 What's New in this version 4
1.1.1 Improved Utilities GUI 4
1.1.2 Removed Components 4
1.2 Known Issues and Limitations 4
2 Programming Methods 5
2.1 Using the FT9xx Programming Utility 5
2.1.1 Programming via One-Wire 6
2.1.2 Programming via DFU 8
2.1.3 Programming via UART 12
2.1.4 Restore the device to Factory Settings 13
2.2 Using FT9xxProg.exe Command Line Utility 14
2.2.1 Command Line 14
2.2.2 Eclipse 17
3 Debugging Methods 19
4 Programming Errors 21
4.1 The MCU is Not Powered/Connected 21
4.1.1 Programming Utility 21
4.1.2 Command Line Utility 21
4.2 The UMFTPD2A is Not Connected to the PC 22
4.2.1 Programming Utility 23
4.2.2 Command Line Utility 23
4.3 Active Debug Session Open 23
4.3.1 Programming Utility 24
4.3.2 Command Line Utility 24
4.4 FT90x vs FT93x 24
4.5 Incorrect Board State 24
5 Debugging Errors 25
5.1 No MCU Connected 25
5.2 No UMFTPD2A Connected 25

5.3 Release Build....26
5.4 Project Build Error....27
5.5 FT90x vs FT93x Debug....28
5.6 Build Configuration Name....29
6 USB DFU Programming Errors....30
6.1 No DFU Device....30
6.2 Binary File Preprocessing....30
7 Other Tips and Tricks....31
7.1 MicroMatch Connection....31
7.2 UART connection....32
7.3 UMFTPD2A Drivers....32
7.4 Restore the Bootloader....32
7.5 Reinstall the FT9xx Toolchain....33
7.6 FT9xx Debugging....33
7.7 Data logger....33
7.8 D2XX utilities....35
7.8.1 Scan and Select the Programming Board....35
7.8.2 Read the D2XX Configuration....35
7.8.3 Write the configuration....36
7.8.4 Restore the default configuration....36
7.8.5 Create the configuration file for building project....36
8 Conclusion....37
9 Contact Information....38
Appendix A– References....39
9.1 Document References....39
9.2 Acronyms and Abbreviations....39
Appendix B – List of Tables & Figures....40
List of Tables....40
List of Figures....40
Appendix C– Revision History....42

1 Introduction

Bridgetek's MCU family consists of FT90x and FT93x MCUs.

Based upon Bridgetek's proprietary FT32 high-performance, 32-bit RISC core, the FT9xx series provides a plethora of connectivity options, making it the ideal choice for advanced technology bridging solutions. By executing instructions from program memory in RAM, rather than Flash Memory, the FT9xx can operate at true Zero Wait States (0WS) up to 100MHz with 310 DMIPS performance.

The FT9xx Toolchain provides developers with debug and programming abilities which are used in the development and production stages of a project.

This document shows programming and debugging methods and details common problems when developing with FT9xx MCUs and how to overcome them.

This version of the document specifically deals with programming the FT9xx on Microsoft Windows systems. Please contact Bridgetek for details of alternative provisions for Linux-based systems.

What's New in this version

1.1.1 Improved Utilities GUI

The FT9xxProg.exe and "FT9xx Programming Utility" have been updated with enhanced programming speed, refreshed icons, and an improved appearance.

Sections for the D2XX and DLOG Utility Guides have been added.

1.1.2 Removed Components

Eclipse IDE, Eclipse Plugin for FT9xx is now removed from installers. Eclipse Plugin for FT9xx can be downloaded via FT9xx Eclipse Plugin repos. Please refer to AN_325 for more information.

The FT9xxProg.py script and the associated Python environment have been removed.

Known Issues and Limitations

NA

2 Programming Methods

There are three ways to program the FT9xx MCU as described in this section.

A programming board such as the UMFTPD2A is mandatory for Sections 2.1 and 2.2. Section 2.3 does not require the programming board.

Using the FT9xx Programming Utility

Bridgetek has provided a GUI utility to program the FT9xx MCU. This is provided with the FT9xx Toolchain and can be found on the desktop after installation. Figure 1 shows the initial launch window.

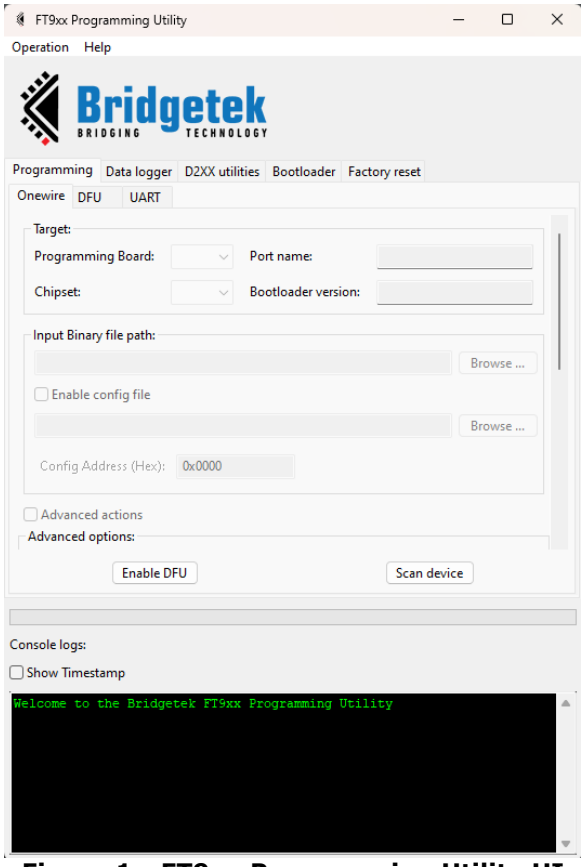

Figure 1 - FT9xx Programming Utility UI

The operation of the utility is selected by the "Operations" page. The operations available are shown in Figure 2.

Figure 2 - FT9xx Programming Operation Selection

2.1.1 Programming via One-Wire

The first operation is "One-Wire Programming". This will program a binary image (produced by the compiler with the "`.bin`" extension) to a device.

First click on the "Scan for Devices" button. This will perform a scan for programming devices and any FT9xx device connected to them.

Note: Once a Scan for Devices operation has been performed the device will be reset by the programmer if another operation is selected.

After the scan is complete the first device found will be selected. An example is shown in Figure 3. Click on the button beside the "Binary File" box to browse for a file to program to the device. The type of device is automatically filled in and cannot be changed.

The option for Flash Memory or Program Memory can be selected to change where the binary file is programmed. If it is in Program Memory, then it will be run from there after programming is complete. For Flash Memory the device will be reset to allow the program to be run.

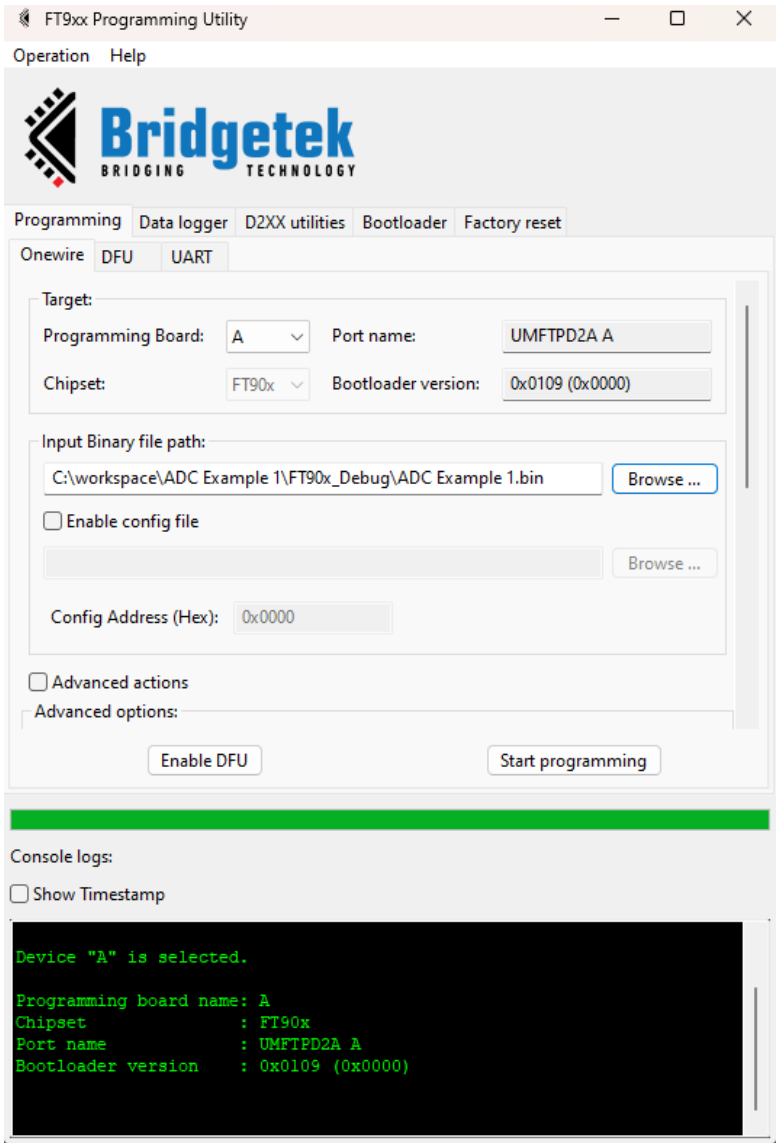

Figure 3 - First Device Selected After Scan for Device

A bootloader option is included. This can be used for advanced operations on the device.

The 'Keep Existing' option prevents the program from overwriting any existing bootloader on the device, which may have been intentionally modified.

The 'Default' option overwrites the current bootloader with the default value specified in the 'Customize Bootloader' field.

The Exclude option must only be used when the device does not need a bootloader and will not require debugging. It is not recommended to use this option.

The Custom option allows modification of the default bootloader before it is programmed to the device. When the bootloader presents a DFU interface it will use the VID, PID and BCD from these settings. The bootloader timeout is a time that it waits after reset for a UART connection from the PC system before running the program stored in Flash Memory. Options are shown in Figure 4.

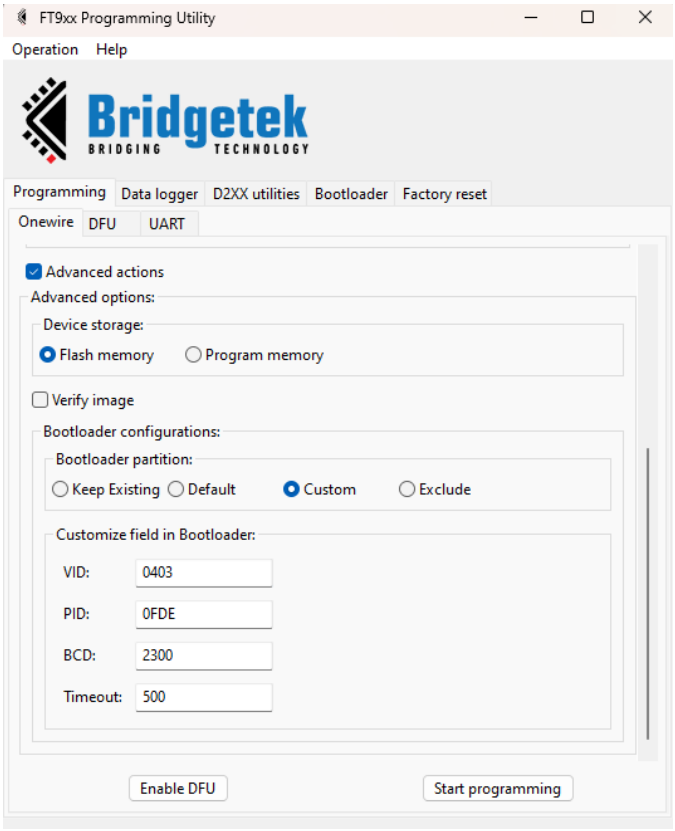

Figure 4 - Bootloader Custom Options

The Enable Config File option is used when another binary file is to be placed in Program Memory or Flash Memory at a set location. For example, a program may use a config file for additional information which could be device specific. The D2XX library for FT9xx will use a config file for setting up information for the D2XX ports created. Refer to Section 7.8.2 for instructions to create the Config File.

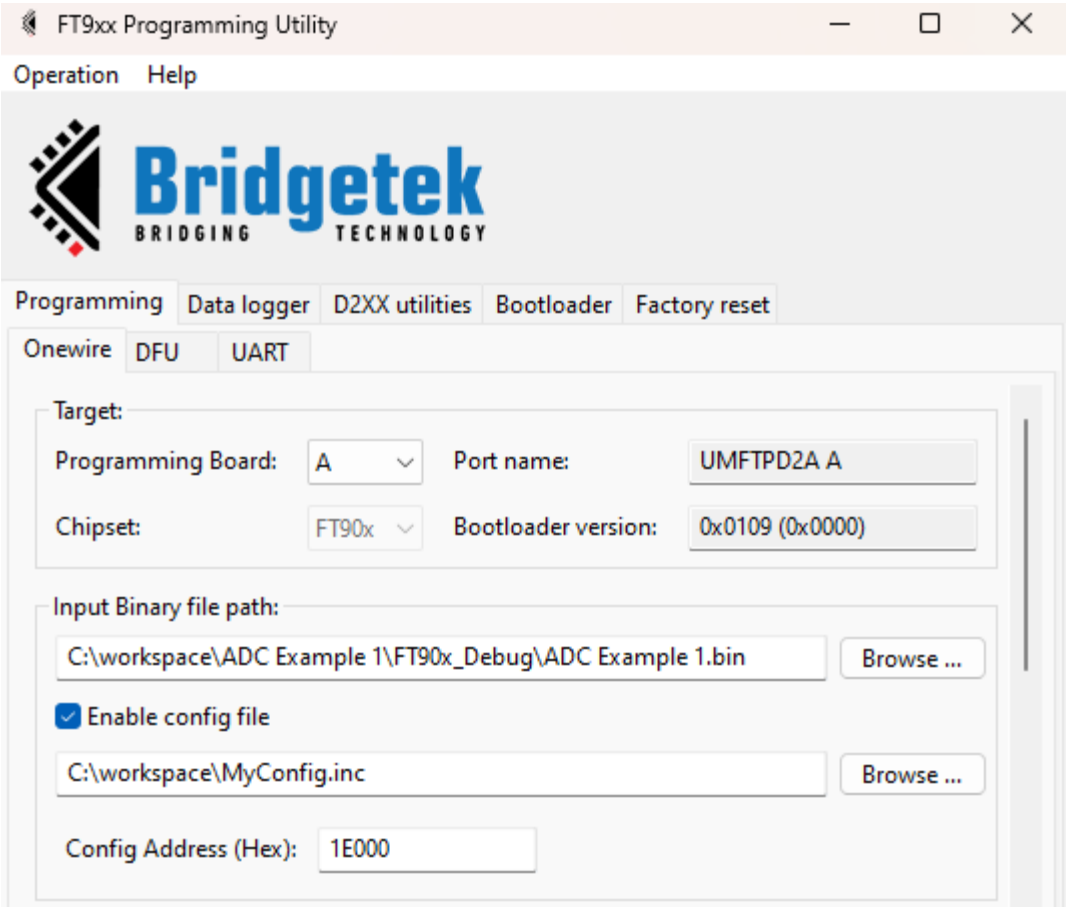

Figure 5 - Setting a Config File

2.1.2 Programming via DFU

A USB DFU interface is pre-programmed into production ICs and modules. It will only become active if the bootloader detects that there is no valid firmware present in the flash memory. This may occur due to a "Factory Restore" operation or because the programmed firmware fails the checksum test at power-on. The latter could result from an unsuccessful programming attempt or a user application writing to flash memory without updating the checksum accordingly.

An optional DFU code module can be added to user programs to allow the same features under program control.

DFU code allows users to program via the USB device interface rather than using the UMFTPD2A programming module. DFU is generally not suited for development purposes as the DFU interface can be easily removed by programming firmware devoid of the USB DFU capability. The DFU interface cannot be used for debugging.

To use this feature, we need to enter DFU mode via two ways below:

Note: This feature is only visible on Bootloader version 1.09 or newer. If you use the code to enter DFU mode just skip section Enable DFU mode using One-wire and Enable DFU mode using UART.

2.1.2.1 Enable DFU mode using One-wire

Go to the **Programming** page, then navigate to the **Onewire** page. Locate and press the **Enable DFU** button. This process requires the UMTFPDA2A board. This action will activate DFU mode; refer to *Section* [2.1.2.3](#) to continue with the next steps.

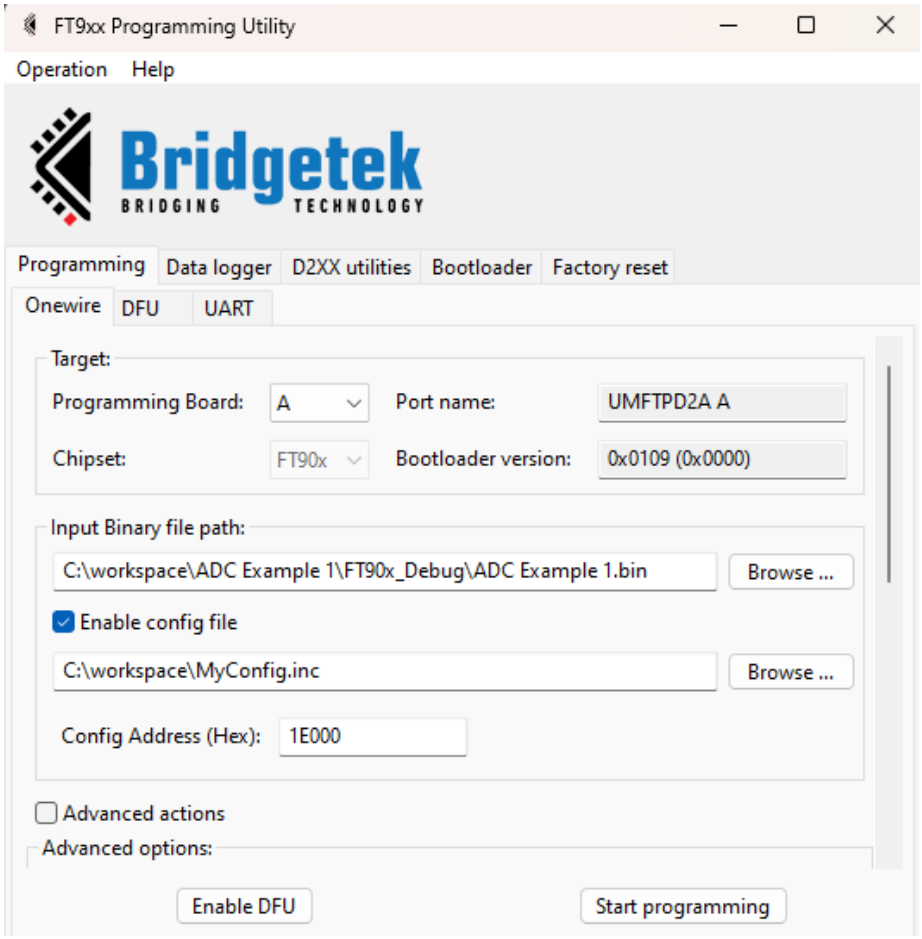

Figure 6 - Enable DFU mode via Onewire

2.1.2.2 Enable DFU mode using UART

Go to the **Programming** page, then navigate to the **Onewire** page do Scan and select the correct port locate and press the **Enable DFU** button. This process does not require the UMTFPD2A board; instead, any USB to UART converter module can be used. Refer to the UART connection section for details. By using a USB to UART converter board in place of the UMTFPD2A, you can proceed. Don't forget to select the COM port and Chipset.

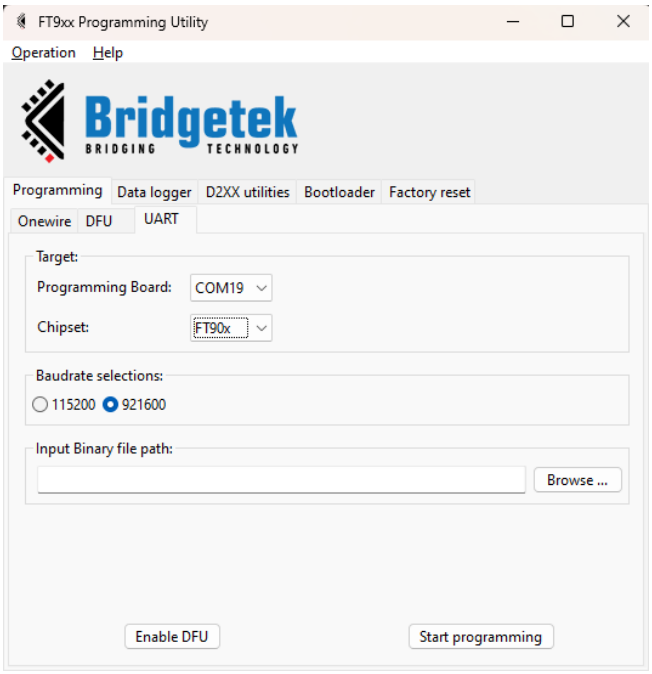

Figure 7 - Enable DFU mode using UART

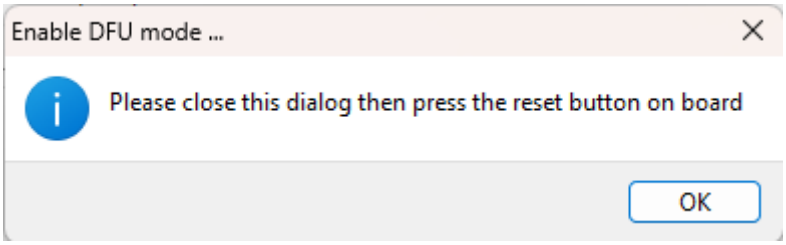

Figure 8 - UART dialog

The dialog above will show up. Just close it then press the reset button on board to switching to DFU mode. This log below will show up on the console if you successfully switch to DFU mode.

```
FT9xxProg.exe --dfumode --uart --device 0 -s COM19
Successfully activate the DFU mode
```

This action will activate DFU mode; refer to *Section* 2.1.2.3 to continue with the next steps.

2.1.2.3 Check the DFU port is activated

You can verify DFU mode activation by accessing the **Device Manager** and looking for the "**FT90x DFU Device**" as shown in Figure 9.

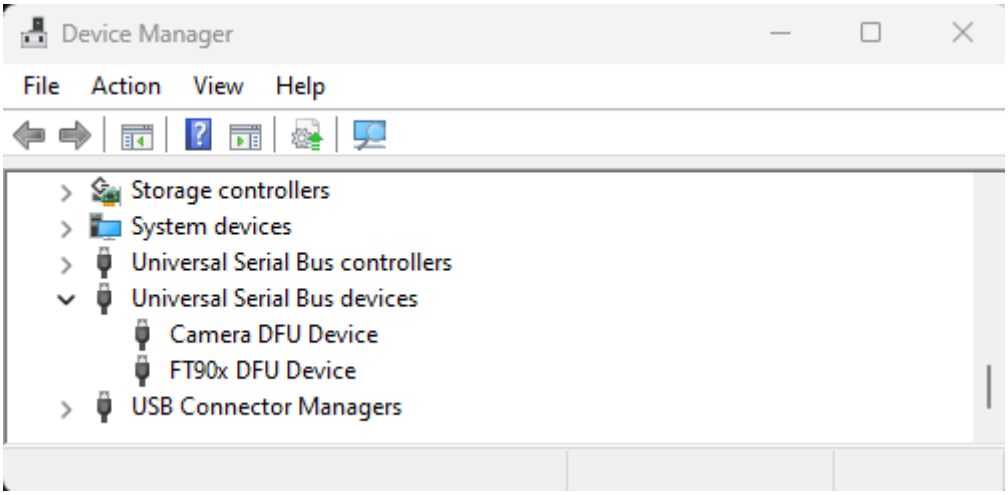

Figure 9 – FT90x DFU Device

2.1.2.4 Start programming via DFU

Go to the **Programming** page, then select the **DFU** page. Press the **Scan** button, and the **BRTCHIP** DFU port should appear (it may take a moment). Once it appears, select it. Configure your settings as needed, then press the **Start Programming** button.

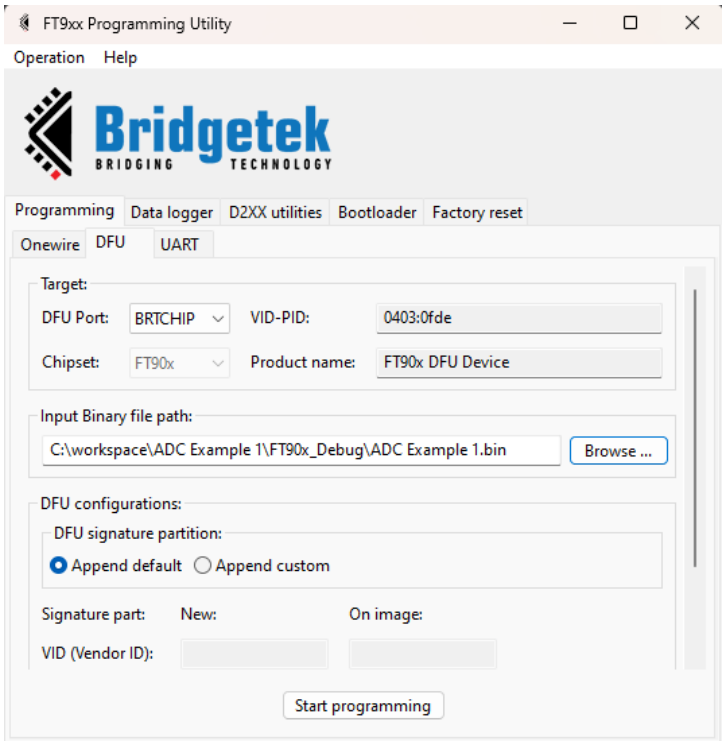

Figure 10 - USB DFU Programming

2.1.3 Programming via UART

Before using this method, some wiring setup is needed by referring to the UART connection section. Any UART-to-USB converter board can be used; the UMFTPD2A is not required.

As with other programming methods, start by selecting the Programming page, then go to the UART tab as shown in Figure 11.

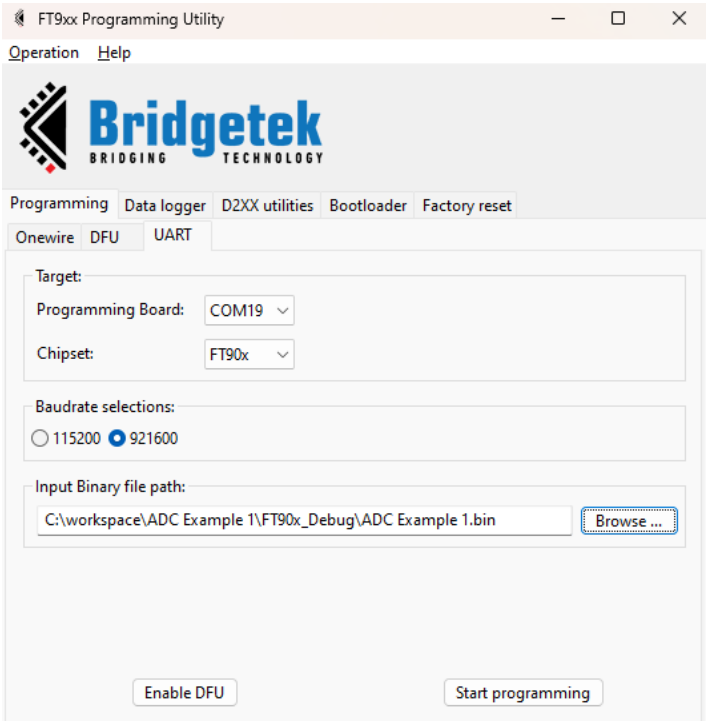

Figure 11 - UART programming page

Select the appropriate port name and chipset. Next, choose the baud rate (921600 is recommended for faster programming). Then, select the path to your binary file and press the **Start Programming** button. The dialog shown below will appear.

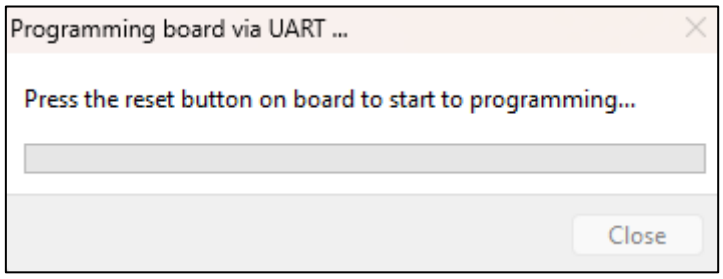

Figure 12 - UART programming dialog

Simply press the reset button on the board and allow a moment for the process to complete.

2.1.4 Restore the device to Factory Settings

A Factory Restore will not program any image to the device, just erase the Flash Memory and restore the default bootloader to the device.

Step 1: Do scan

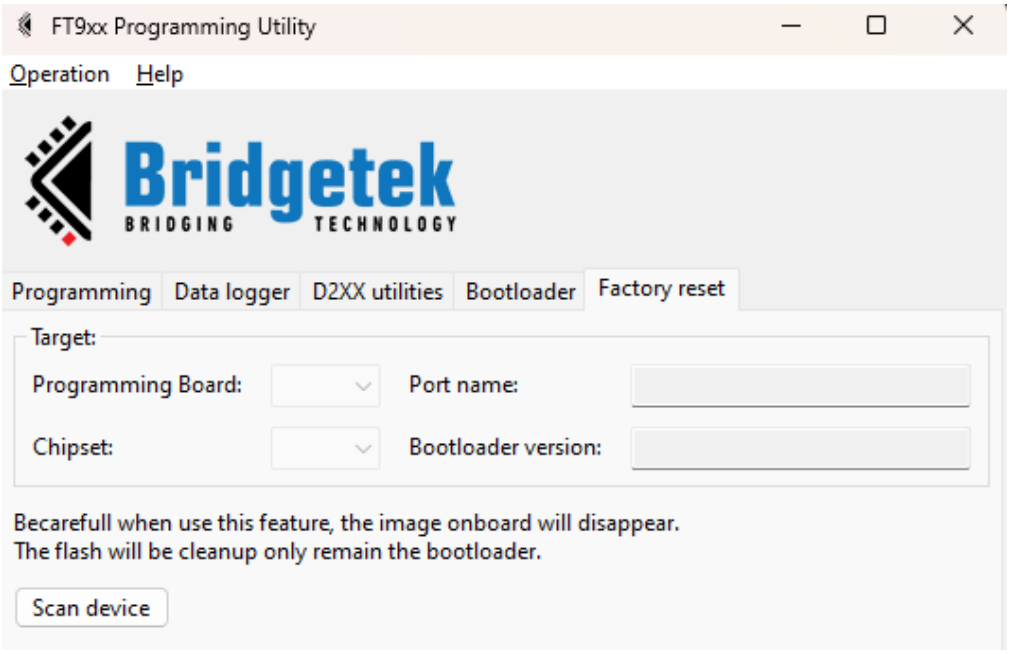

Figure 13 - Factory reset scan

Step 2: After that select the programming board then press the "Factory reset" button.

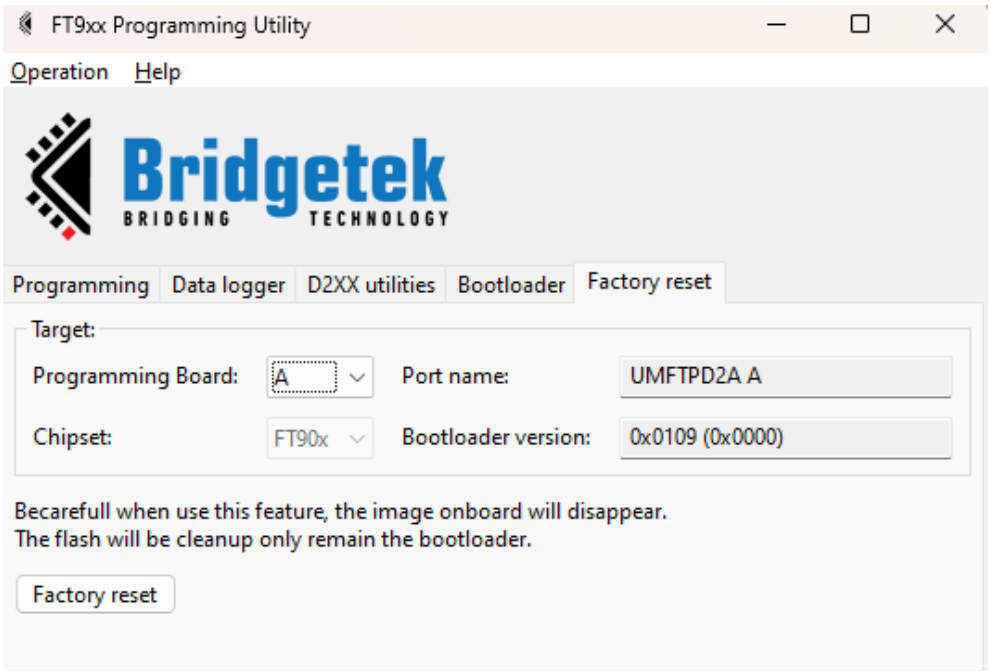

Figure 14 - Start Factory reset

Using FT9xxProg.exe Command Line Utility

Bridgetek has also provided a command line utility (included within the FT9xx Toolchain). This can be run from a command prompt, script or another utility. It can also be run from within Eclipse for FT9xx.

2.1.5 Command Line

From a command prompt or PowerShell on Windows, the program can be run.

```
C:\workspace> FT9xxProg.exe –loadflash ‘.\FT90x_Debug\USBH Example HID to UART.bin’ –
onewire
Initializing…
Device = FT900
Erasing…
Still erasing…
Flash programming progress:
[========================] 100%
```

The `--help` parameter passed to the utility will print the latest command line arguments and their meaning to the screen.

```
C:\workspace> FT9xxProg.exe --help
```

The command line arguments are split into “Operations”, “Interfaces” and “Options” sections.

Operations:

No operation	List available devices for programming.
-h, --help	Show help message and exit.
-f <file>, --loadflash <file>	Load the binary <file> into the flash memory.
-p <file>, --loadpm <file>	Load the binary <file> into the program memory.
-b [settings], --loadbl [settings]	Configure the bootloader to occupy the upper 4 kB of flash memory. You can customize settings by referring to the section on Settings.
-e, --info	Display the chip ID and the CRC16 checksum of the flash. When a serial number is not specified it will also list all available programming devices.
-C <file>, --checksuffix <file>	Check whether binary <file> has a DFU-suffix attached, if so, get the VID, PID and BCD from the dfu-suffix of the binary file.
-E, --editBL	Edit the bootloader settings and write back the edited bootloader back to predefined location.
-g <file> [settings], --dfufix <file> [settings]	Ensure the binary file is compatible with DFU programming. You can customize settings to specify various options and remember to designate an output file using the --outfile/-o flag.
-y, --restore	Restore the chip to the factory settings.
-j <file>, --prepareImage <file>	Prepare the flash image from binary file provided.
-r, --chipReset	Reset the chip.
-v, --version	The version of this utility.

Options:

-s <serial number>, --serialNo <serial number>	Serial number to select a specific programming device. If no serial number is specified then the first available device is used. Available devices can be listed with the --info/-e operation.
-v, --verify	Verify the flash memory contents after flash programming. Only used for one-wire programming.
-n, --noReset	Prevent the device from resetting after the image is downloaded.
-x, --nobl	Prevent the bootloader from being programmed to the last 4 kB of the flash.
-k, --keepoldbl	Keep the existing bootloader at the top 4 kB of the flash.
-c <file>, --conf <file>	Embed a second binary file to the main binary file specified by the --loadflash/-f flag. Embed from the address specified by the --confaddr/-a flag below.
-a <addr> --confaddr <addr>	Specify the address to embed a second binary file with the --conf/-c flag above. Hexadecimal.
-o <file>, --outfile <file>	Specify the output file for operations that require one.
-a <file>, --infile <file>	Specify the input file for operations that require one.
-L, --Verbose	Verbose output for extra debugging info.
-D <0,1>, --device <0,1>	Specify device type to be operated on. • FT900 -- 0, • FT930 -- 1. The default is FT900. This is optional when using operations that target the one-wire interface. It is mandatory when using the UART interface or file operations.

Settings:

-V <value>, --vid <value>	A 4-digit hex value which specifies the idVendor.
-P <value>, --pid <value>	A 4-digit hex value which specifies the idProduct.
-B <value>, --bcd <value>	A 4-digit hex value which specifies the bcdDevice.
-T <value>, --timeout <value>	When modifying the bootloader settings this is the timeout (in MS, decimal) before the bootloader starts verifying the firmware. For the UART bootloader this is the timeout to detect the bootloader presence. This is signalled by a chip reset or at power on.

Attention:

Please ensure the programmer is the only device connected when programming or ensure the serial number flag is set correctly.

2.1.5.1 List All Available Devices

The command with no parameters will list all available devices using the one wire interface:

```
C:\workspace>FT9xxProg.exe
Number of devices: 4
Device 0:Serial Number: FT4UHELNA,      Description: UMFTPD2A A.
Device 1:Serial Number: FT4UHELNB,      Description: UMFTPD2A B.
Device 2:Serial Number: FT4UHELNC,      Description: UMFTPD2A C.
Device 3:Serial Number: FT4UHELND,      Description: UMFTPD2A D.
```

The information there can be used to select a device for programming.

A UMFTPD2A board consists of 4 interfaces. The first interface is the "A" interface and provides the One-Wire programming interface. The third interface is the "C" interface and that is taken to the UART pin headers on the UMFTPD2A.

To select the device required with the "`--serial/-s`" argument only the serial number of the "A" interface is used. So, for example, the above programmer boards would be selected with "`--serial FT4UHELNA`".

2.1.5.2 Read the Chip ID

Read the chip ID and the flash checksum of the first device using the one wire interface:

```
C:\workspace>FT9xxProg.exe --info
Initializing...
Chip ID:     0x09000002
Flash CRC16: 0xF7D1
```

The first device is selected in this example. To choose a different device use the "`--serial/-s`" parameter to the program.

Reading the Chip ID in this way will halt execution of the program on the FT9xx and it will require to be reset afterwards if the program in flash is to be re-run. The program memory is altered when this command is run.

2.1.5.3 Program a Binary File

Program `helloworld.bin` into the flash via the one-wire interface and verify the image:

```
C:\workspace>FT9xxProg.exe -f "helloworld.bin" -O -V
Initializing...
Device = FT900
Erasing...
```

```
Still erasing...
```

```
Flash programming progress:
[========================] 100%
```

The same operation using long options:

```
C:\workspace>FT9xxProg.exe --loadflash "helloworld.bin" –onewire --verify
```

2.1.5.4 Generate a Binary Image File with a Bootloader

The program can be used when there is no device or programmer connected. Operations such as the `` `--prepareImage/-j` `` operation which compiles a binary file and a bootloader into a whole flash image may need to have the device type specified explicitly. There is no default device type so it must be set with the "`--device/-D`" option.

This example will form a whole flash image for an FT93x using the file `helloworld.bin` and attaching a bootloader. The output will go to `image.bin`.

```
C:\workspace>FT9xxProg.exe -j "helloworld.bin" --outfile image.bin --device 1
Initializing...
FT930 Flash image written to file image.bin
```

2.1.5.5 Write a Bootloader to the Device

Rewrite the bootloader with the default settings to the top 4 kB of the flash memory via the one-wire interface:

```
C:\workspace> FT9xxProg.exe -b -O
```

2.1.5.6 Modify the Bootloader on the Device

Write a customized bootloader with a timeout of 3 seconds to the top 4 kB of the flash memory via the one-wire interface:

```
C:\workspace> FT9xxProg.exe -b -O -T 3000
```

The timeout parameter alters the length of time that the bootloader will wait for a connection from a PC system to power up or reset. This allows a program running on a PC system connected to the UART0 of the FT9xx to program the device. Refer to Section 2.3 for instructions on using this interface.

2.1.5.7 Restore the Device to Factory Settings

```
C:\workspace> FT9xxProg.exe --restore -O
```

This command will erase the Flash Memory and restore the default bootloader to the device.

2.1.5.8 Add a DFU Suffix to an Image

To use the DFU programming method in Section 2.1.2 an informational suffix must be added to the image.

```
C:\workspace>FT9xxProg.exe -g "helloworld.bin" --outfile image.bin --device 1 -V 0403
```

The options for the DFU suffix are "`--vid/-V`", "`--pid/-P`" and "`--bcd/-B`" can be used in this command. These options tell the DFU utility what device type the image is targeted for.

2.1.6 Eclipse

The command line programming utility is used by a Debug Configuration within Eclipse when debugging. Refer to Section 3 for more information. It can also be used as a Run Configuration or as an External Tool.

To use the command line utility as an External Tool can be added as a new External Tool Configuration as shown in Figure 15. When it runs, it will output to a console window as shown in Figure 16.

The path to the utility is clearly shown in the figures using the environment variable **`FT9XX_TOOLCHAIN`** to locate the installation directory of the toolchain and programmer. The installation will add these directories to the `PATH` environment variable, so this is not necessary.

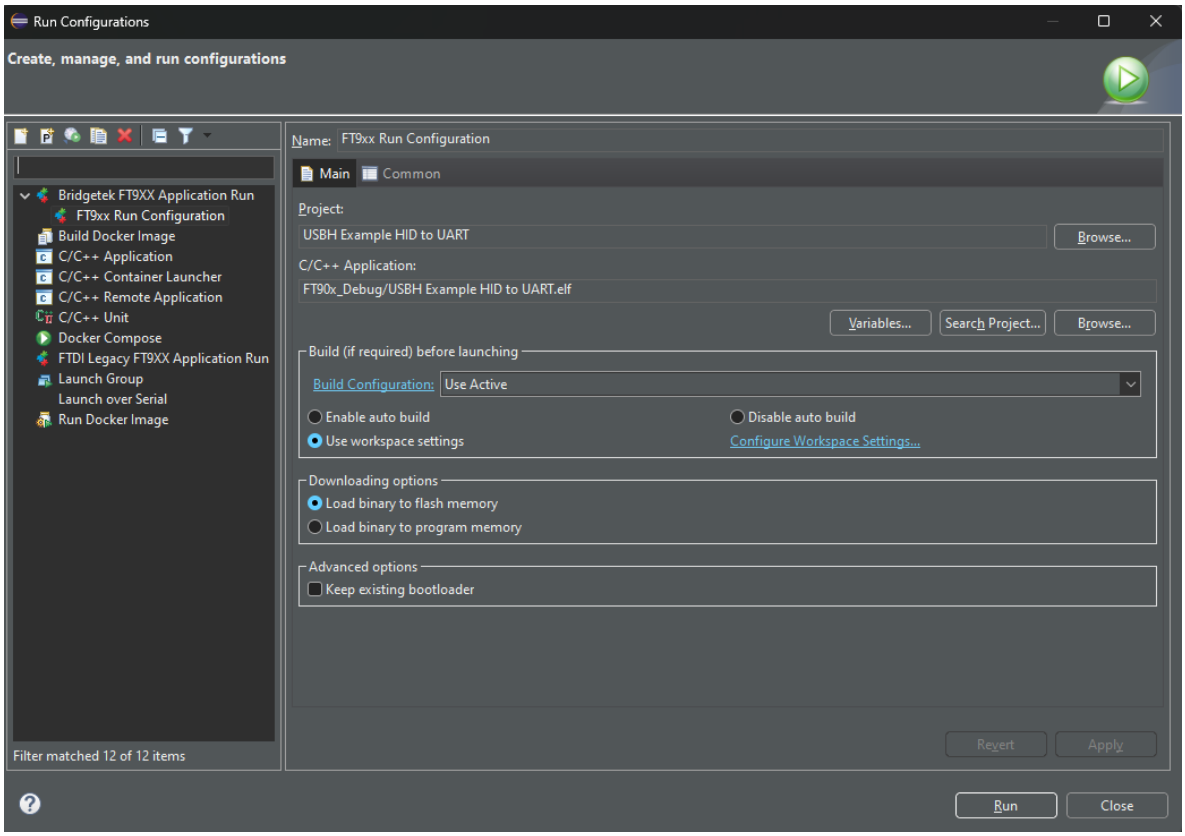

Figure 15 - FT9xxProg in Eclipse

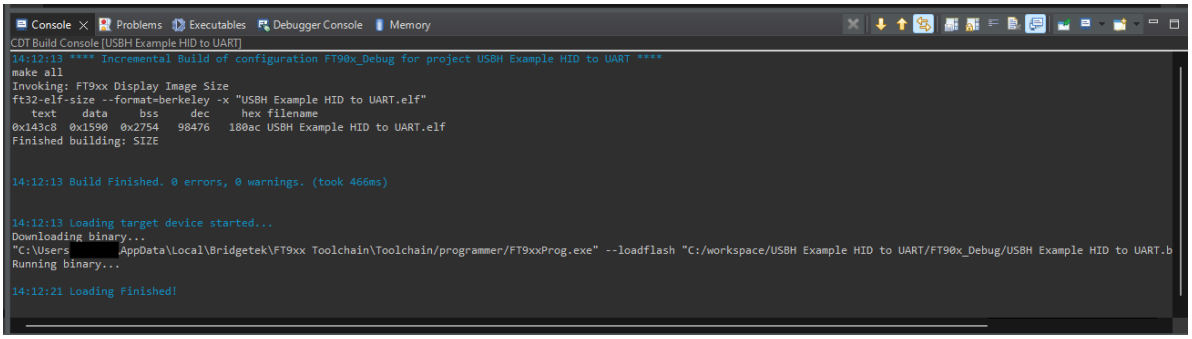

Figure 16 - FT9xxProg in Eclipse Console

3 Debugging Methods

The FT9xx Toolchain includes a seamless debugging feature. When you click on FT9XX DEBUG within the IDE, it programs the device and launches the debugging environment automatically.

This can be accessed from the **Run → Debug Configurations** menu as shown in Figure 17.

The UMFTPD2A is required for debugging as it is only supported over the one-wire interface.

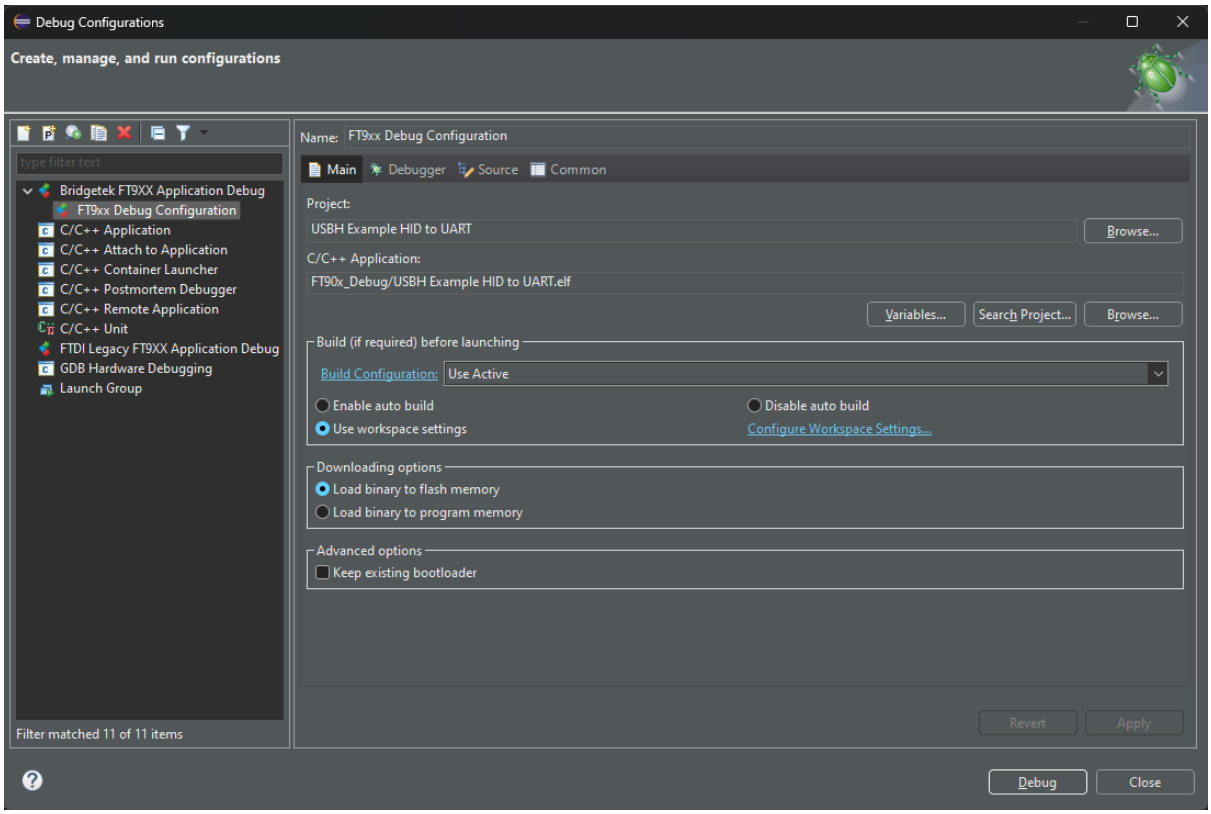

Figure 17 - Debug Configurations

Note that when this has been run once, it can be found via the debug icon as shown in Figure 18.

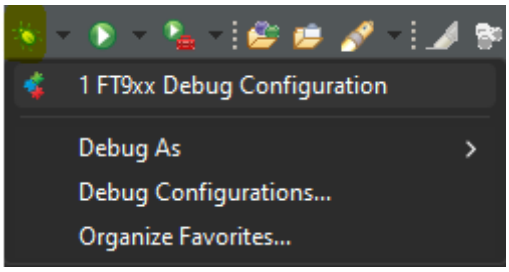

Figure 18 - Debug Icon

The Debug environment is shown in Figure 19 with key features highlighted.

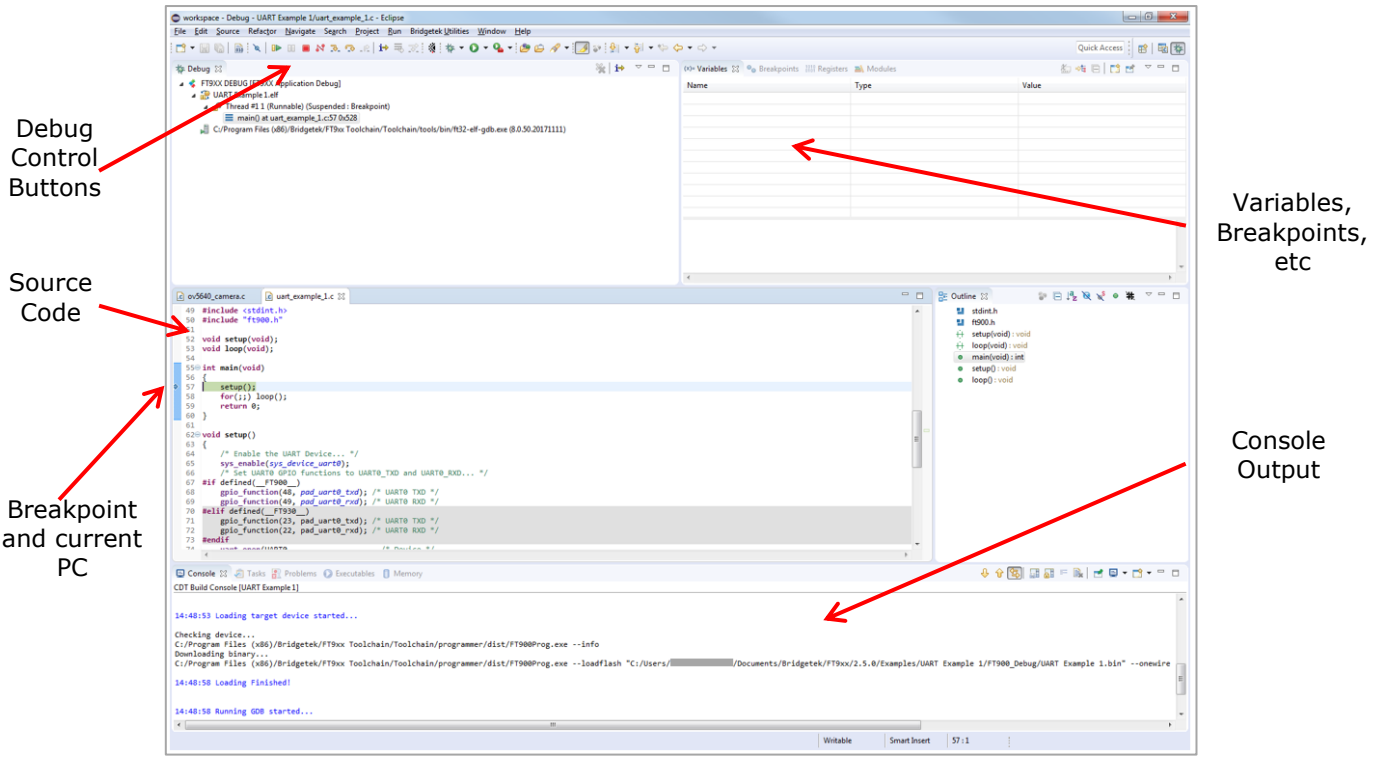

Figure 19 - Debug Environment

4 Programming Errors

This section describes the common programming errors.

The MCU is Not Powered/Connected

The FT9xx Development Module or custom hardware must be powered along with the UMFTPD2A Debug/Programming module as shown in Figure 20.

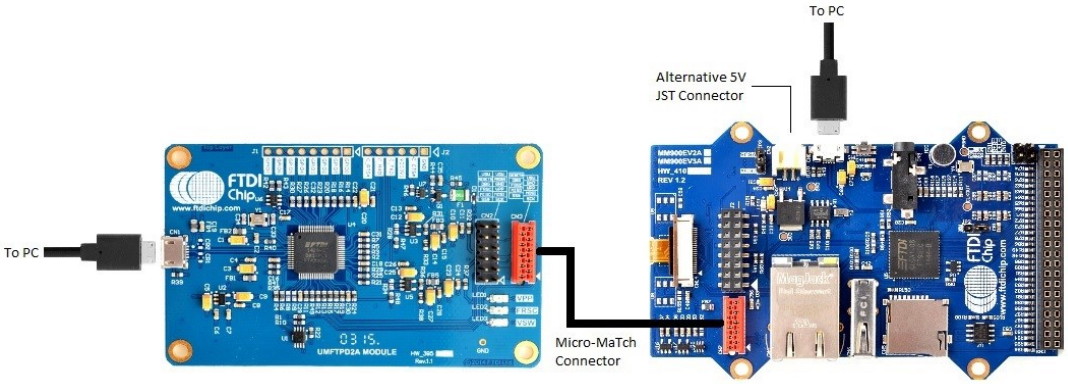

Figure 20 - Hardware Connection

If it is not powered, then error messages will be displayed as shown in Figure 21 or Figure 22 depending on which programming method is being used.

The same error is encountered when there is no physical connection between the two boards via the MicroMatch connector.

4.1.1 Programming Utility

The FT9xx Programming Utility will see the UMFTPD2A programmer module but the FT9xx device will not be found, so the user cannot continue to the next window as shown in Figure 21.

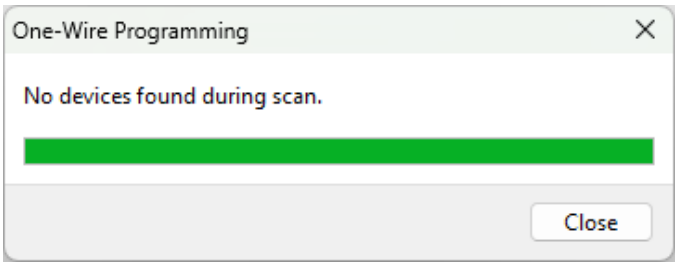

Figure 21 - Programming Utility No MCU

4.1.2 Command Line Utility

The command line utility will report an unknown device as shown in Figure 22.

```
C:\workspace\USBH Example HID to UART> FT9xxProg.exe -e
FT9xxProg: error: Failed to open device.
FT9xxProg: error: UMFTPD2A Module not found. Please check the connection and close all active debug connections.
```

Figure 22 - Command Line Utility No MCU

To resolve this error, ensure that the FT9xx MCU is powered and connected to the UMFTPD2A via the MicroMatch connector. In some cases, the FT9xx needs to be power cycled to resolve this issue.

The UMFTPD2A is Not Connected to the PC

If the UMFTPD2A is not connected to the PC, the FT9xx tools cannot find the D2XX interface used to program them. The same error as in Section 4.1 will be shown.

This is also the case when the UMFTPD2A USB drivers have not been installed correctly. Refer to Section 7.3 for more information.

If the UMFTPD2A is not connected to the host, then the green LED on the board will not be lit.

If the FT9xx to be programmed currently has firmware with D2xx USB device enabled, and if it is connected via USB to the same PC as the UMFTPD2A used for programming, the Programming Utility may try to open the FT9xx D2xx ports instead of the UMFTPD2A port. In this case you may see the error messages as shown in Figure 23. One solution is to power the FT9xx board via the power-in connector instead or via a USB cable which provides power only.

Note: This error will only occur on programming utility versions before 2.8.0 as the latest version has been updated to avoid this.

To resolve these errors, ensure that UMFTPD2A is connected to the PC and to the FT9xx MCU via the MicroMatch connector.

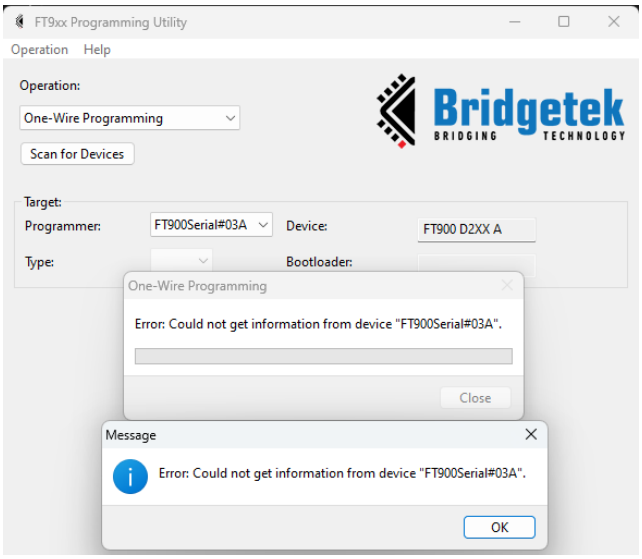

Figure 23 - Firmware image includes the D2XX device firmware

4.1.3 Programming Utility

The Programming Utility will fail to see the UMFTPD2A Programmer and report the same error as shown in Figure 21.

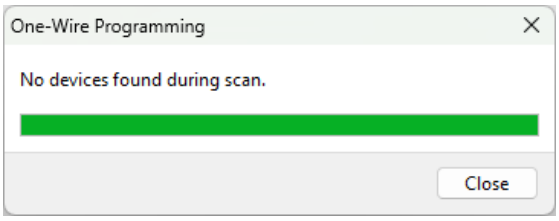

Figure 21 - Programming Utility No MCU

4.1.4 Command Line Utility

```
C:\workspace\USBH Example HID to UART> FT9xxProg.exe -e
FT9xxProg: error: Failed to open device.
FT9xxProg: error: UMFTPD2A Module not found. Please check the connection and close all
active debug connections.
```

The command line utility will report an unknown device as shown in Figure 22.

Active Debug Session Open

When there is an active debug session open, or there is another instance of the Programming Utility or command line Utility running, the D2XX interface which is used for programming the FT9xx is claimed and cannot be used. The same error as in Section 4.1 will be shown.

If Eclipse is debugging with the UMFTPD2A programming board, terminate the active debug session as shown in Figure 24.

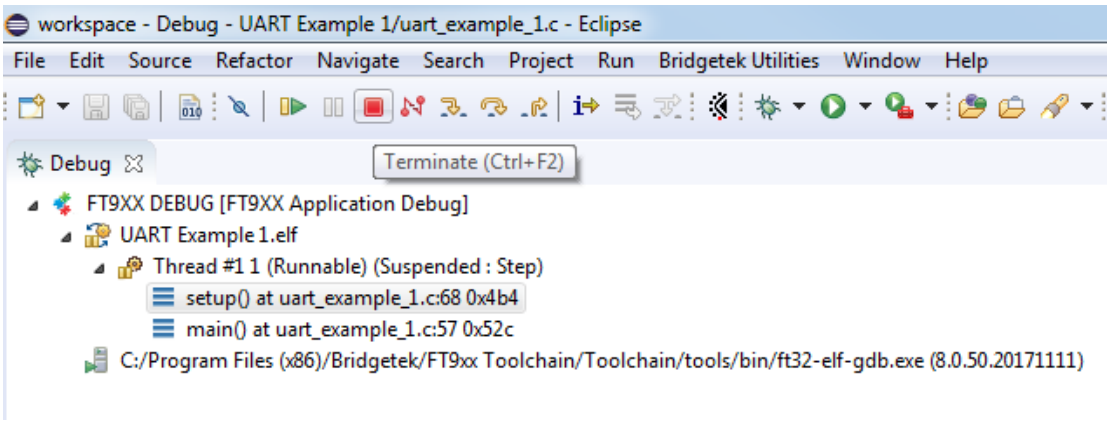

Figure 24 - Terminate Active Debug Session

Otherwise, check that there are no more instances of the programmer running on the system. Open "Task Manager", press the Windows "Start" button then type "Task Manager", or press Win-X (Windows key + the "X" key together) then the "T" key for Task Manager.

Once it is open then click on the "Details" tab, search for and select "FT9xxProg.exe" as shown in Figure 25. Click on the "End Task" button to terminate the task and allow other connections to the device.

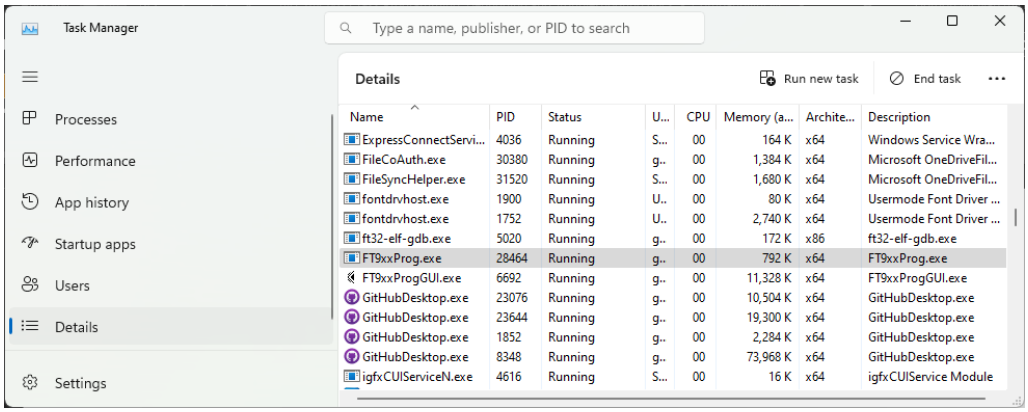

Figure 25 - Terminate Programmer in Task Manager

4.1.5 Programming Utility

The Programming Utility will fail to see the UMFTPD2A Programmer as shown in Figure 21.

4.1.6 Command Line Utility

The command line utility will report UMFTPD2A Module not found as shown in Figure 22.

FT90x vs FT93x

The FT9xx Toolchain allows the user to build a project for the FT90x or FT93x MCU as shown in Figure 26.

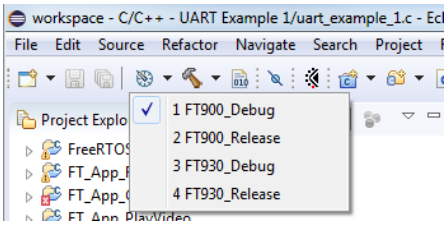

Figure 26 - FT9ox and FT93x Build

Care must be taken to build for the correct target MCU. The programmer will not show any errors in this case but if the wrong build is programmed, the MCU will not work.

Eclipse will not allow a program compiled for FT90x to be programmed to an FT93x and vice versa. If, however, the names of the Build Configuration within the Eclipse project are changed then it is possible to write or debug the wrong type of device.

Incorrect Board State

If the UMFTPD2A board or the FT9xx device gets into an incorrect state, then the message in Figure 27.

```
C:\workspace\USBH Example HID to UART> FT9xxProg.exe --loadflash '.\FT90x_Debug\USBH
Example HID to UART.bin' –onewire
Can't get the lock from MCU!
FT9xxProg: error: Chip ID incorrect or unknown.
```

Figure 27 - Incorrect Board State

To rectify this, re-run the programmer after disconnecting and reconnecting the board and programmer board to ensure they are in a consistent state.

5 Debugging Errors

This section describes some common debug errors and how to overcome them.

No MCU Connected

The error shown in Figure 28 can occur when the MCU is not powered or there is no physical connection between the FT9xx and UMFTPD2A via the MicroMatch connector.

```
**** Incremental Build of configuration FT900_Debug for project UART Example 1 ****


make all
'Invoking: FT90x Display Image Size'
ft32-elf-size --format=berkeley -x "UART Example 1.elf"
   text    data     bss     dec     hex filename
 0x10c4   0x1b4     0x8    4736    1280 UART Example 1.elf
'Finished building: SIZE'
' '
Build Finished (took 967ms)
Loading target device started...
Checking device...
C:/Program Files (x86)/Bridgetek/FT9xx Toolchain/Toolchain/programmer/FT9xxProg.exe --info
Error encountered while checking device. No MCU connected.
Loading Finished!
```

Figure 28 - Debug No MCU Connected

To resolve this error, ensure that the FT9xx MCU is powered and connected to the UMFTPD2A via the MicroMatch connector.

No UMFTPD2A Connected

If the UMFTPD2A is not connected or not enumerated with the PC, the error shown in Figure 29 is displayed.

```
**** Incremental Build of configuration FT900_Debug for project UART Example 1 ****
make all
'Invoking: FT90x Display Image Size'
ft32-elf-size --format=berkeley -x "UART Example 1.elf"
   text    data     bss     dec     hex filename
 0x10c4   0x1b4     0x8    4736    1280 UART Example 1.elf
'Finished building: SIZE'
' '
Build Finished (took 281ms)
Loading target device started...
Checking device...
C:/Program Files (x86)/Bridgetek/FT9xx Toolchain/Toolchain/programmer/FTxxProg.exe --info
```

```
Error encountered while checking device. Could not detect UMFTPD2A or VII programmer
module.
Loading Finished!
```

Figure 29 - Debug No UMFTPD2A Connected

To resolve this error, ensure that the FT9xx MCU is powered and connected to the UMFTPD2A via the MicroMatch connector.

Release Build

If the debug session is started when the release build is selected as shown in Figure 30, the debug session will launch but the release build contains no debug symbols.

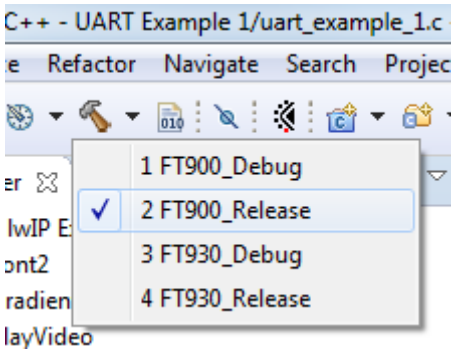

Figure 30 - Release Build

The source code won't be able to be visible in the debug session, as shown in Figure 31.

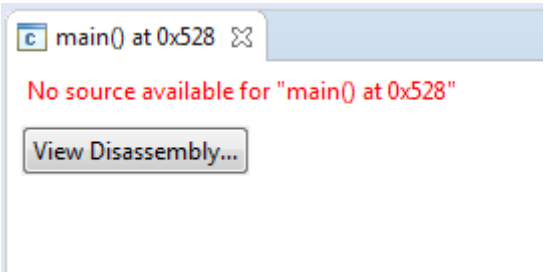

Figure 31 - Release Build No Source

To resolve this error, ensure that the Debug build is selected within Eclipse.

Project Build Error

If the project contains code errors, the debug cannot proceed because a .bin file is required to be programmed before the debug session can start. A typical build error will be displayed as shown in Figure 32.

```
**** Incremental Build of configuration FT930_Debug for project UART Example 1 ****
make all
'Building file: ../uart_example_1.c'
'Invoking: FT90x GCC Compiler'
ft32-elf-gcc -D__FT930__ -D__RAMSIZE=32K -D__PMSIZE=128K -I"C:/Program Files
(x86)/Bridgetek/FT9xx Toolchain/Toolchain/hardware/include" -O0 -Og -g1 -fvar-tracking -
fvar-tracking-assignments -Wall -c -fmessage-length=0 -ffunction-sections -mft32b -
mcompress -MMD -MP -MF"uart_example_1.d" -MT"uart_example_1.o" -o "uart_example_1.o"
"../uart_example_1.c"
cc1.exe: warning: variable tracking requested, but useless unless producing debug info
../uart_example_1.c:50:10: fatal error: fd900.h: No such file or directory
 #include "fd900.h"
          ^~~~~~~~
compilation terminated.
make: *** [uart_example_1.o] Error 1
12:53:13 Build Finished (took 257ms)
```

Figure 32 - Build Error

FT90x vs FT93x Debug

The FT9xx Toolchain allows the user to build a project for the FT90x or FT93x MCU as shown in Figure 33.

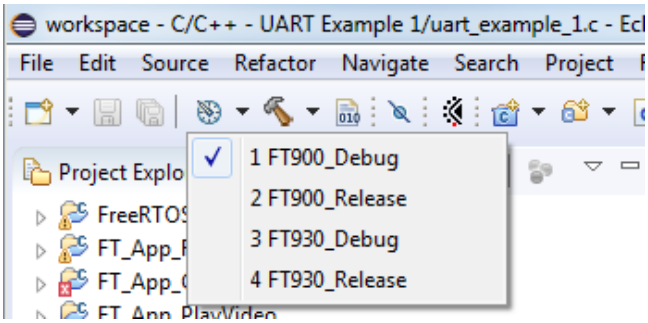

Figure 33 - FT90x and FT93x Build

Care must be taken to select the correct target MCU before launching the debug session or the error shown in Figure 34 will be appear.

```
**** Incremental Build of configuration FT930_Debug for project UART Example 1 ****
make all
'Invoking: FT90x Display Image Size'
ft32-elf-size --format=berkeley -x "UART Example 1.elf"
   text    data     bss     dec     hex filename
 0x11dc   0x1ac     0xc    5012    1394 UART Example 1.elf
'Finished building: SIZE'
' '
Build Finished (took 286ms)
Loading target device started...
Checking device...
C:/Program Files (x86)/Bridgetek/FT9xx Toolchain/Toolchain/programmer/FT9xxProg.exe --info
Error encountered while checking device. Device and binary do not match! Device is FT900.
Loading Finished!
```

Figure 34 - FT90x and FT93x Error

To resolve this error, ensure that the correct target build is selected within Eclipse.

Build Configuration Name

Eclipse gives 'device and binary mismatch error' if the build configuration name is other than 'FT9xx_Debug' or 'FT9xx_Release'.

Check the build configuration names by right-clicking on the project, selection **Properties → C/C++ Build → Settings**, then click on Manage Configurations.

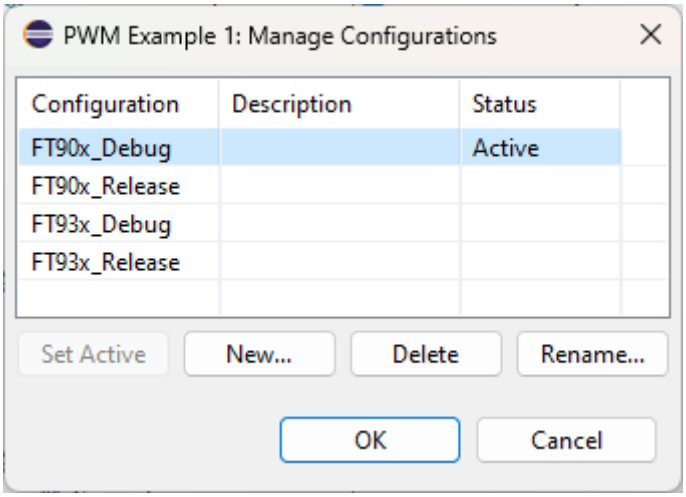

Figure 35 - Manage Configurations

6 USB DFU Programming Errors

Common USB DFU errors are detailed in the forthcoming sections.

No DFU Device

If there is no USB DFU interface available, the error is shown in Figure 36.

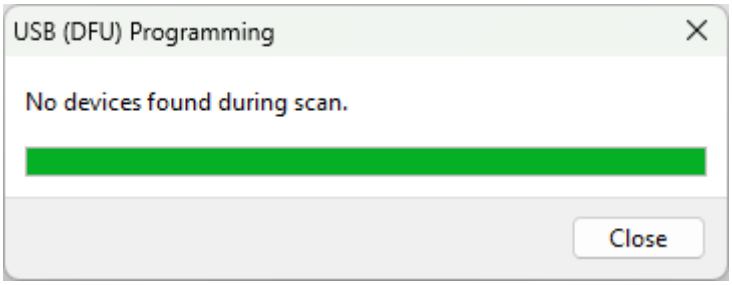

Figure 36 - No DFU Device

This can happen when the IC is not in factory programmed state, or there is no DFU interface available in the application firmware. It can also happen when the FT9xx device is not connected. To resolve this issue, there are a couple of options:

- Restore the bootloader. Refer to Section 0 for more information.
- Include a USB DFU interface in your application firmware. More information can be found on the USB DFU interface in AN_365 FT9xx API Programmers Manual.

Binary File Preprocessing

To program via USB DFU, the .bin file must be padded, and a valid DFU-suffix added. If this is not done, the error shown in Figure 37 will be appear.

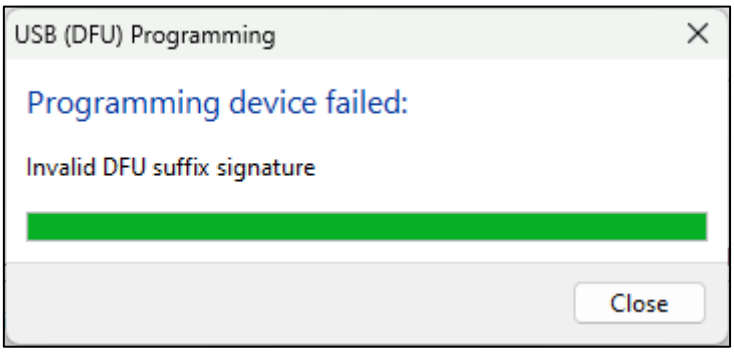

Figure 37 - DFU Caution

To resolve this error, ensure that pre-processing has been done on the .bin file. This can be done using the FT9xx Programming Utility as shown in Section 2.1.2. This is a one-time step on any binary file.

7 Other Tips and Tricks

This section details some other useful tips and tricks to help overcome any programming and debugging issues.

MicroMatch Connection

Ensure the MicroMatch connector between the UMFTPD2A and the FT9xx is the correct way round. There is an arrow which signifies pin 1 and the cable itself has a red side which helps with the connection as shown in Figure 38.

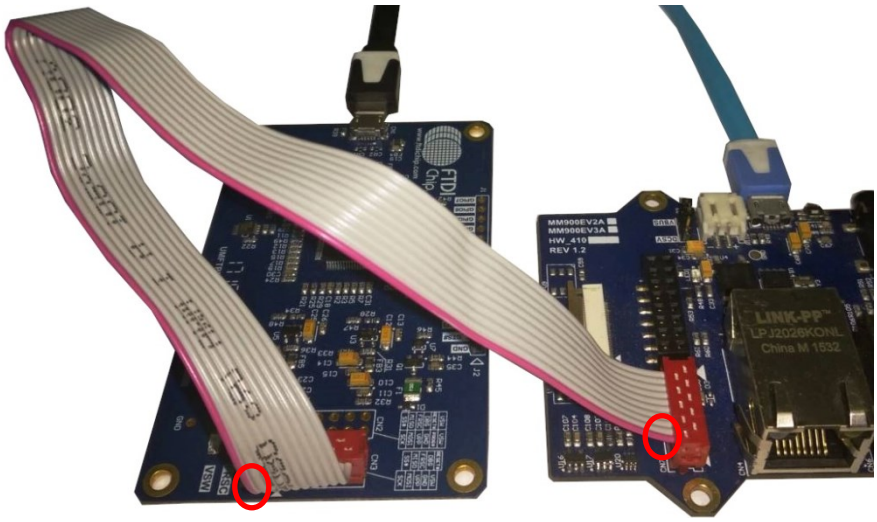

Figure 38 – MicroMatch Connection

UART connection

The UMFTPD2A board also provides the UART port to monitor/debug. Ensure that the wires are connected properly as shown in Figure 39. You can use any COM port application to interact with the 3rd "USB Serial port" as shown in Figure 40.

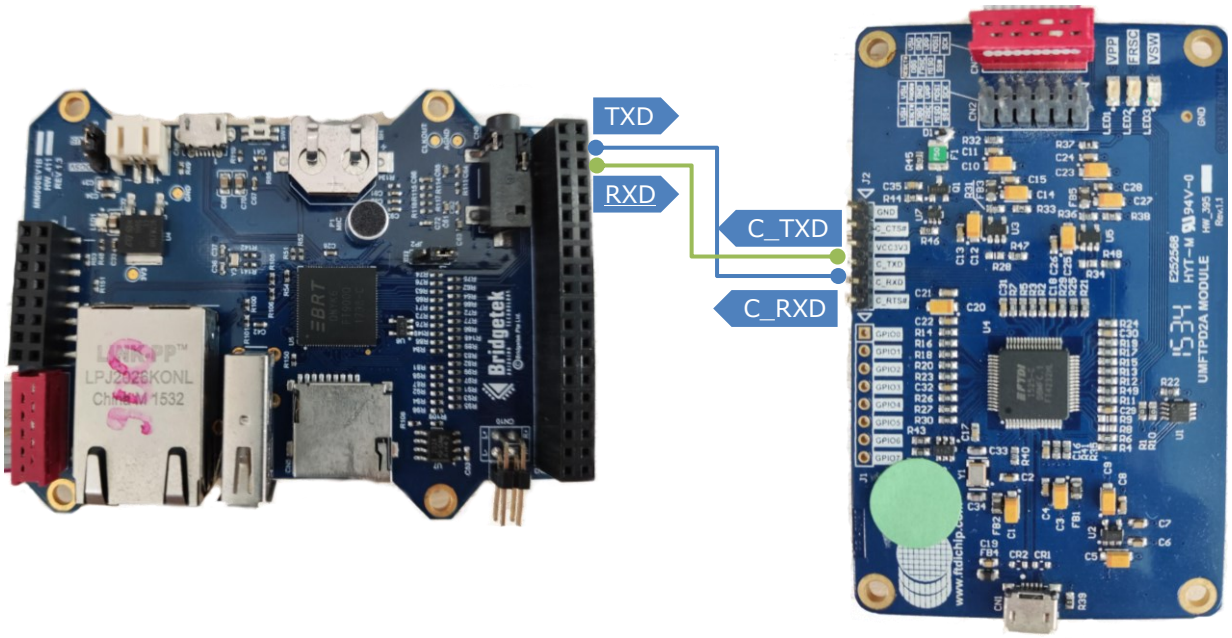

Figure 39 - UART Connection

Note: If the boards are powered by different sources, make sure to connect their GND pins to ensure the UART communication works correctly.

UMFTPD2A Drivers

Check that the UMFTPD2A drivers have been installed correctly in Windows Device Manager as shown in Figure 40. COM10 to COM13 is the UMFTPD2A in this screenshot. The UMFTPD2A contains an FT4232H IC which is a four port USB device.

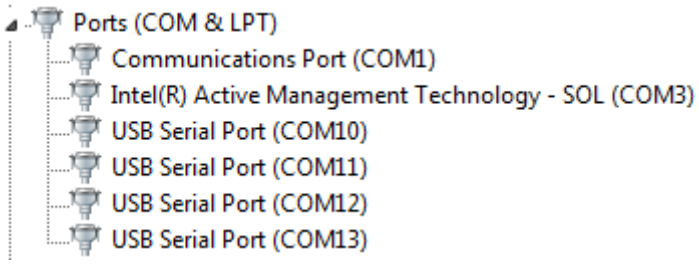

Figure 40 - Device Manager

The drivers can be easily uninstalled using CDM Uninstaller. When the device is plugged back into the PC, the drivers should install automatically via Windows Update, otherwise refer to our Installation Guides.

Restore the Bootloader

Restoring the bootloader of the FT9xx MCU can be done using the FT9xx Programming Utility and UMFTPD2A only as shown in Figure 41.

This programs the MCU into a default factory state.

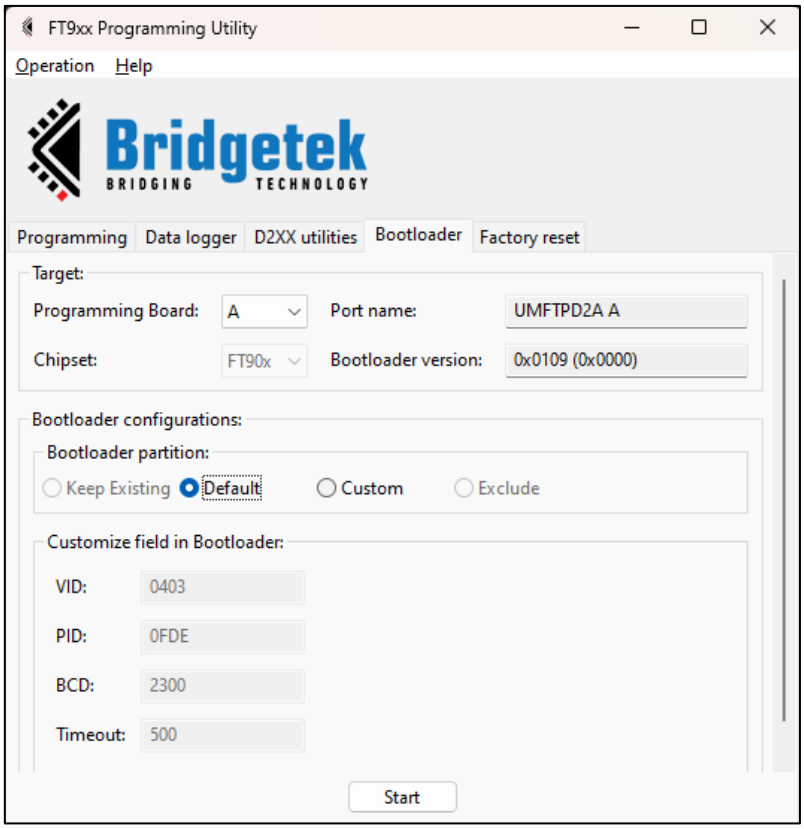

Figure 41 - Restore Bootloader

Reinstall the FT9xx Toolchain

Reinstalling the FT9xx Toolchain should always be the last resort.

FT9xx Debugging

FT9xxProg.exe is run automatically when initiating the FT9XX DEBUG session via the Eclipse IDE. It provides a "gdbserver" interface for the "gdb" utility which Eclipse uses to perform the debugging.

Therefore, the debugging is affected by the same issues as the command line programmer utility.

Data logger

The FT9xx Toolchain also offers a logging method that stores user data directly in flash memory within a small 4KB section. The location of this section depends on the specific image you are building, and the tool automatically detects it. If you want to read this section, follow the steps below. Note that this process requires the UMFTPD2A board, as it communicates via the One-wire protocol, and ensure that your firmware includes the DLOG library.

So we need to navigate to the "Data logger" page then perform a Scan.

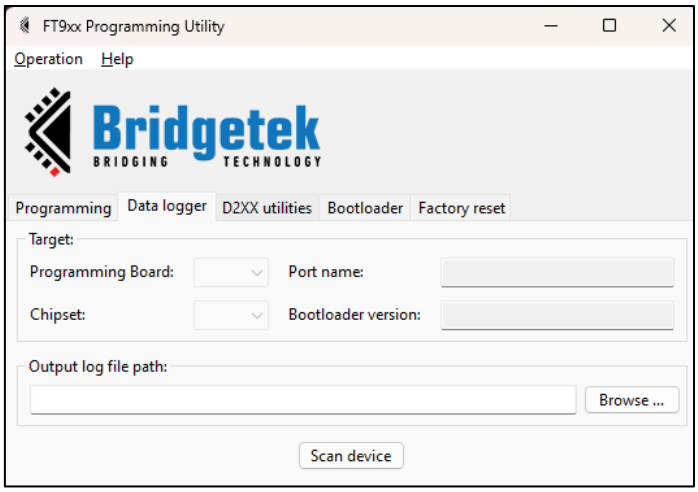

Figure 42 - Data logger scan

After that select the Programming board, log output file path then presses the button "Save log file".

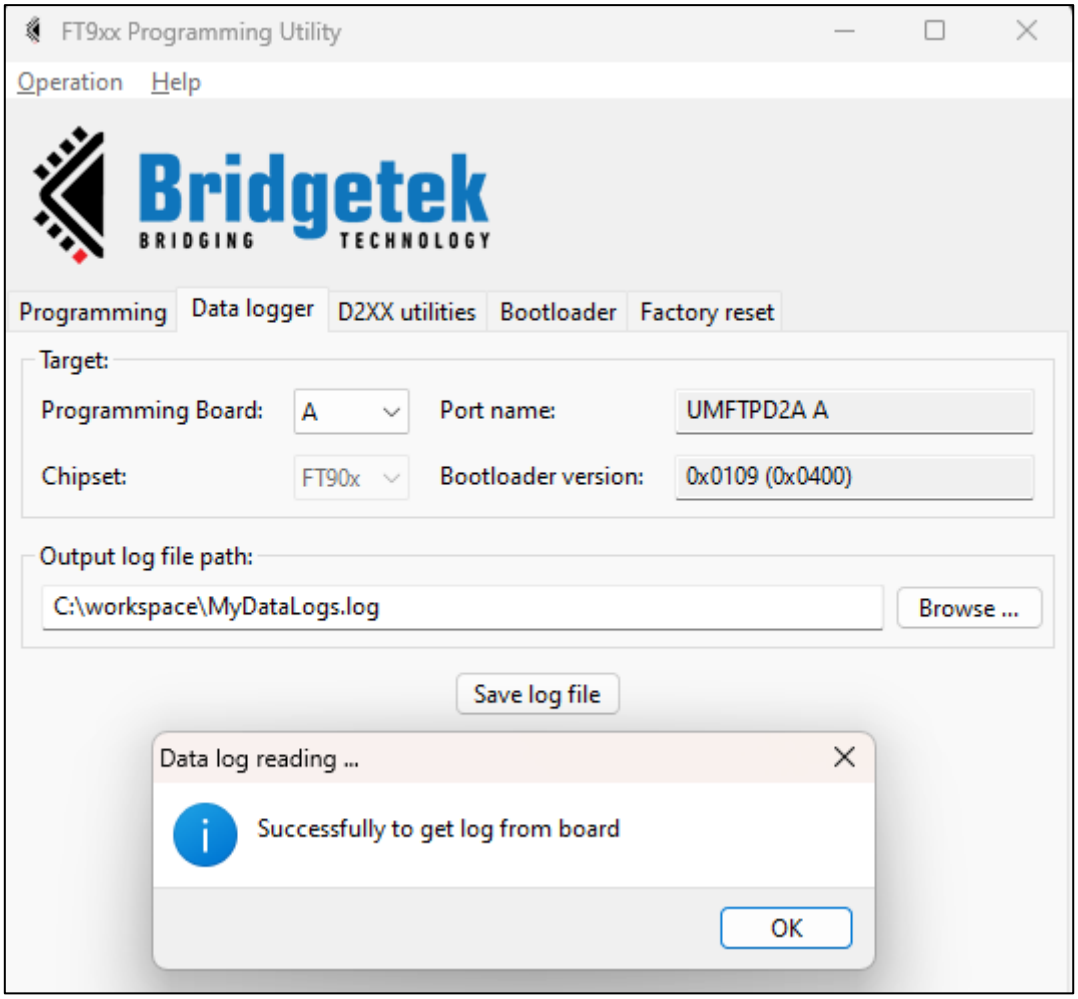

Figure 43 - Successfully read log from board

D2XX utilities

The FT9xx Toolchain also includes the D2XX library, which enables you to configure the USB port for your application. In some cases, you may need to adjust the configuration without reprogramming the board or perhaps recover from a corrupted configuration. This feature provides a straightforward way to achieve that. Below are the supported scenarios for using this feature.

Note: Make sure the image on the board contains the D2XX library.

7.1.1 Scan and Select the Programming Board

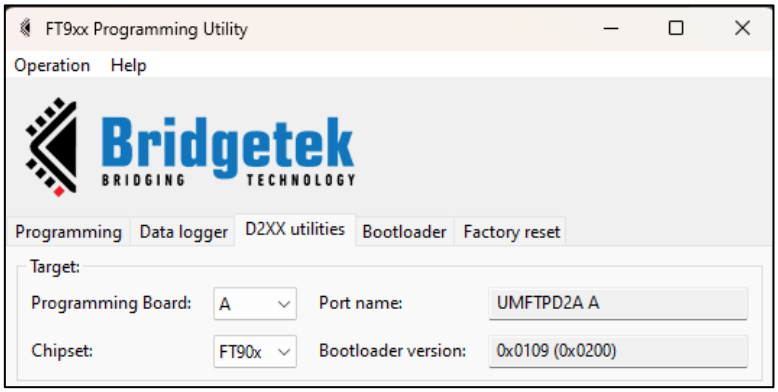

Figure 44 - D2XX Scan and Select Device

Perform a scan, then select the programming board. It will automatically detect the chipset for you.

7.1.2 Read the D2XX Configuration

Scan and select programming board. Next, press the **Read** button. All configurations on the board will be displayed on the panel.

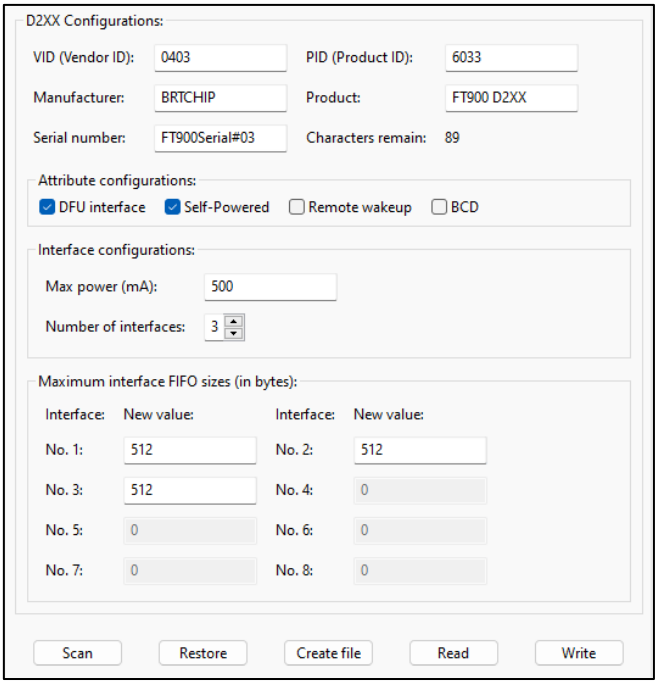

Figure 45 - Read the D2XX Configuration

7.1.3 Write the configuration

Ensure you Scan and select programming board. After making your modifications, press the **Write** button to save the changes to the board.

7.1.4 Restore the default configuration

We have the following use cases:

1. **Restore Configuration**: Scan and select programming board, press **Restore**, and then press **Write** to restore the configuration.
2. **Recover Configuration**: Scan and select programming board, then press **Read** to retrieve the current configuration. After making the necessary modifications, press **Write** to save the updated configuration.

7.1.5 Create the configuration file for building project

This action can be used for various purposes:

- **Modify the Existing Configuration**: To update the configuration on the board, go to *Read Configuration*, make the desired changes, and then select *Write Configuration* to apply it directly to the board.

- **Create a New Configuration**: For a new project configuration, select the chipset and press *Restore* to create a default setup. You can then modify it as needed, press "*Create File"*, and be sure to specify the *Output File Path*.

- **Migrate Configuration to Another Project**: Easily transfer an existing configuration to a different project. Read the configurations; after making the new configurations, press "Create file".

8 Conclusion

This document shows programming and debugging methods, common debug and programming errors when developing with FT9xx MCU and how to overcome them. The tools provided by Bridgetek are comprehensive and are powerful tools to help developers create their own custom applications.

9 Contact Information

Refer to https://brtchip.com/contact-us/ for contact information.

Distributor and Sales Representatives

Please visit the Sales / Distribution Network page for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A– References

Document References

FT90x and FT93x Product Page

FT9xx Toolchain

FT9xx Development Modules

CDM Uninstaller

Installation Guides

AN_365 FT9xx API Programmers Manual

Acronyms and Abbreviations

Terms	Description
DFU	Device Firmware Update
DMIPS	Dhrystone MIPS (Million Instructions per Second)
GDB	GNU Project debugger
GUI	Graphical User Interface
IC	Integrated Circuit
MCU	Micro-Controller Unit
PC	Personal Computer
RAM	Random Access Memory
USB	Universal Serial Bus

Appendix B – List of Tables & Figures

List of Tables

NA

List of Figures

Figure 1 - FT9xx Programming Utility UI 5
Figure 2 - FT9xx Programming Operation Selection 5
Figure 3 - First Device Selected After Scan for Device 6
Figure 4 - Bootloader Custom Options 7
Figure 5 - Setting a Config File 8
Figure 6 - Enable DFU mode via Onewire 9
Figure 7 - Enable DFU mode using UART 10
Figure 8 - UART dialog 10
Figure 9 – FT90x DFU Device 11
Figure 10 - USB DFU Programming 11
Figure 11 - UART programming page 12
Figure 12 - UART programming dialog 12
Figure 13 - Factory reset scan 13
Figure 14 - Start Factory reset 13
Figure 15 - FT9xxProg in Eclipse 18
Figure 16 - FT9xxProg in Eclipse Console 18
Figure 17 - Debug Configurations 19
Figure 18 - Debug Icon 19
Figure 19 - Debug Environment 20
Figure 20 - Hardware Connection 21
Figure 21 - Programming Utility No MCU 21
Figure 22 - Command Line Utility No MCU 21
Figure 23 - Firmware image includes the D2XX device firmware 22
Figure 24 - Terminate Active Debug Session 23
Figure 25 - Terminate Programmer in Task Manager 24
Figure 26 - FT9ox and FT93x Build 24
Figure 27 - Incorrect Board State 24
Figure 28 - Debug No MCU Connected 25
Figure 29 - Debug No UMFTPD2A Connected 26
Figure 30 - Release Build 26
Figure 31 - Release Build No Source 26
Figure 32 - Build Error 27
Figure 33 - FT90x and FT93x Build 28

Figure 34 - FT90x and FT93x Error 28
Figure 35 - Manage Configurations 29
Figure 36 - No DFU Device 30
Figure 37 - DFU Caution 30
Figure 38 – MicroMatch Connection 31
Figure 39 - UART Connection 32
Figure 40 - Device Manager 32
Figure 41 - Restore Bootloader 33
Figure 42 - Data logger scan 34
Figure 43 - Successfully read log from board 34
Figure 44 - D2XX Scan and Select Device 35
Figure 45 - Read the D2XX Configuration 35

Appendix C– Revision History

Document Title: FT9xx Programming, Debugging and Troubleshooting

Document Reference No.: BRT_000285

Clearance No.: BRT#200

Product Page: https://brtchip.com/ft9xx-toolchain/

Document Feedback: Send Feedback

Revision	Changes	Date
1.0	Initial Release	29-08-2023
1.1	Corrected procedure for DFU programming	29-07-2024
1.2	Updated FT9xx Programming Utilities images to reflect the new UI. Added a "D2xx Utilities" section. Added a "Data Logger" section. Removed FT9xxProg.py and its related sections.	10-06-2025