Application Note

BRT_AN_063

FT9xx Programming, Debugging and Troubleshooting

Version 1.1

Issue Date: 29-07-2024

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

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.

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.

2.1 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.

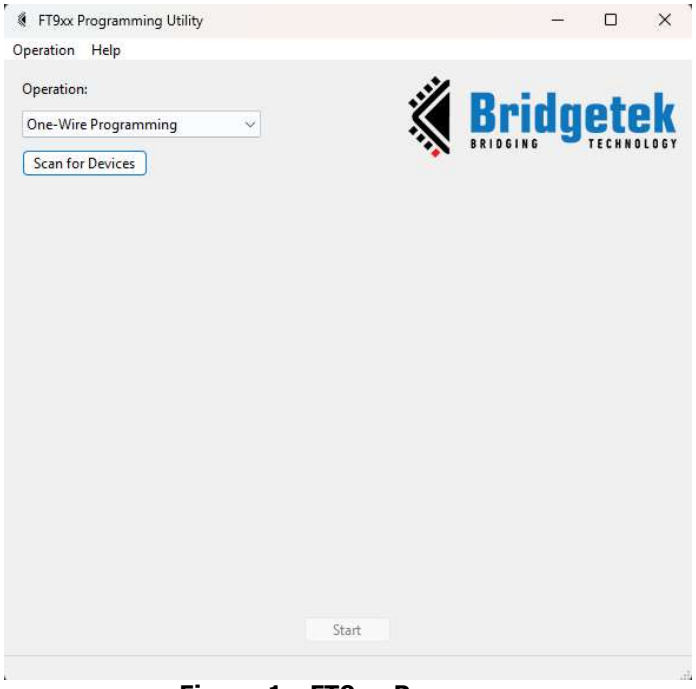

Figure 1 - FT9xx Programmer

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

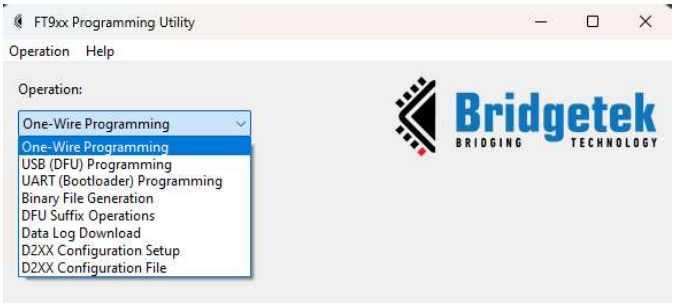

Figure 2 - FT9xx Programming Operation Selection

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.

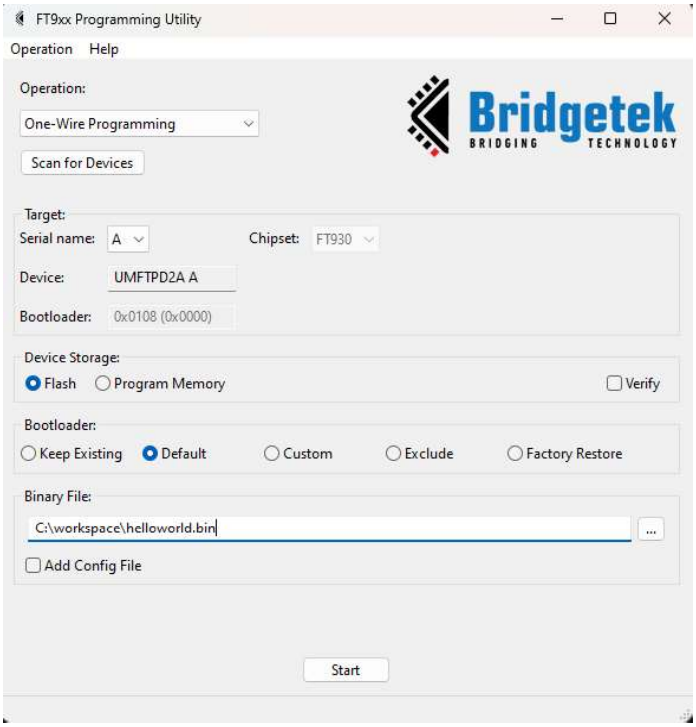

Figure 3 – First Device Selected After Scan for Device

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

The Keep Existing option will prevent the program from overwriting any bootloader currently on the device, this may be because it has been modified previously.

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.

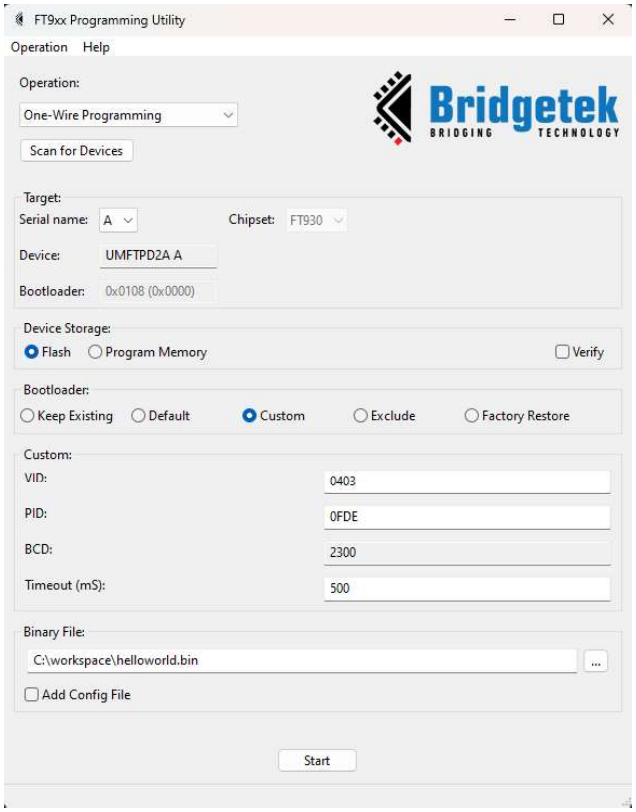

Figure 4 – Bootloader Custom Options

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

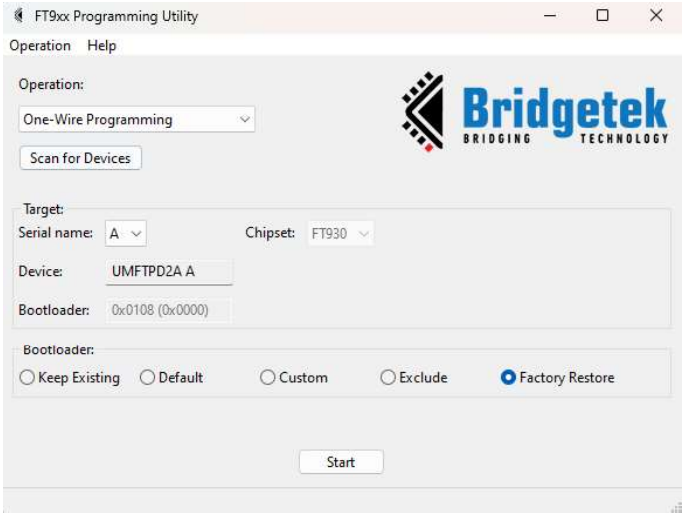

Figure 5 – Factory Restore

The Add 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 setup information for the D2XX ports created.

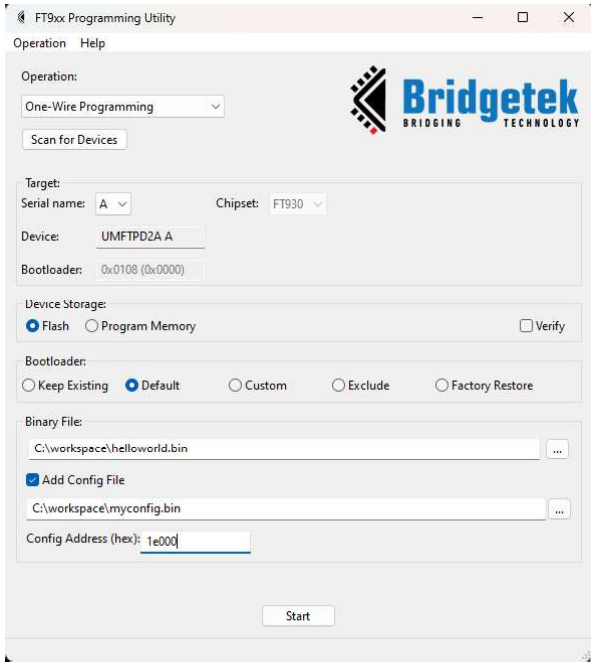

Figure 6 – Setting a Config File

2.1.2 Generating a Binary Image File

If an image file needs to be generated for use, then the "Binary File Generation" option is used. This works similarly to the other methods except that the output is an image file rather than programming the device.

The image is the size of the entire Flash Memory and includes the bootloader.

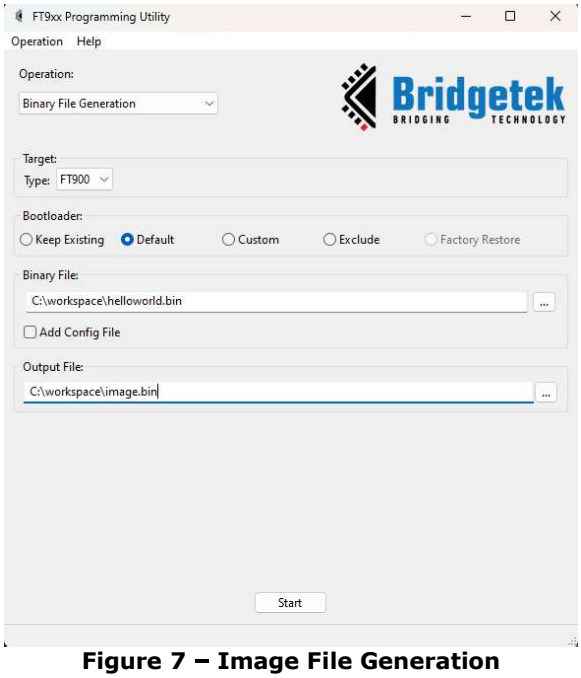

Figure 7 – Image File Generation

Figure 7 shows a file called "`image.bin`" being created from one called "`helloworld.bin`".

2.1.3 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 be because a "Factory Restore" operation has been performed or the programmed firmware fails a checksum test at power-on. The later may be because of a failed programming operation or a user program writing to flash memory without action to correct the checksum.

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.

The programmable binary file (`.bin` file) generated from the Eclipse IDE requires a DFU-suffix added and file padding before it can be programmed over the USB DFU interface.

The first step to making a file for either DFU is to use the method in section 2.1.2 Generating a Binary Image File to make a basic file which contains the required firmware, padding and checksum information. This step will make a file of either 252kB or 256kB for FT90x, or 124kB or 128kB for FT93x. The size difference is explained by the setting of the "Exclude" or "Keep Existing" for the bootloader.

The final step is to add a DFU suffix to the file generated above. This is accomplished in the "DFU Suffix Operations" section of the programming utility. Figure 8 shows the programming utility in the "DFU Suffix Operations" section.

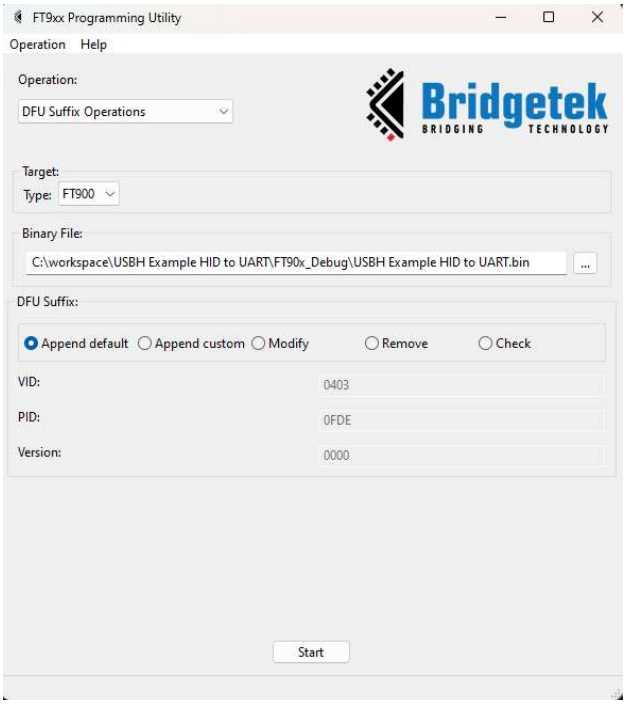

Figure 8 – Add DFU-Suffix

The default suffix will be suitable for use with the built-in DFU firmware on production ICs. DFU Firmware can be modified to accept different settings to prevent incorrect images being loaded. This is achieved by adding a "custom" suffix as shown in Figure 9.

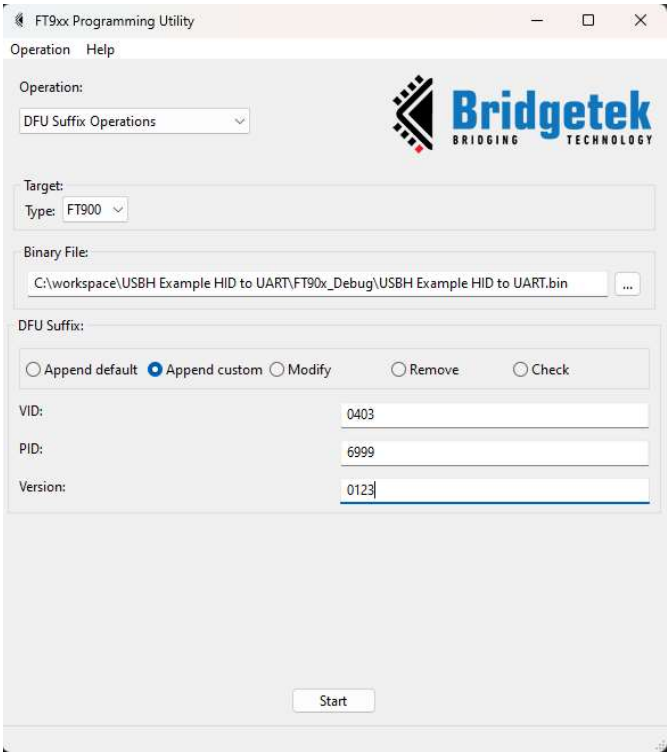

Figure 9 – Add Custom DFU-Suffix Options

Once this one-time step is complete, the FT9xx Programming Utility can be used to program via the USB DFU interface as shown in Figure 10 and Figure 11.

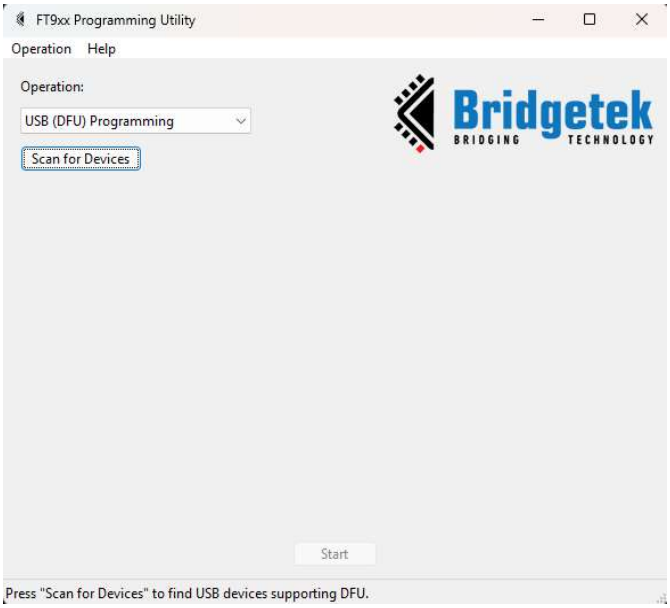

Figure 10 – Program via USB DFU

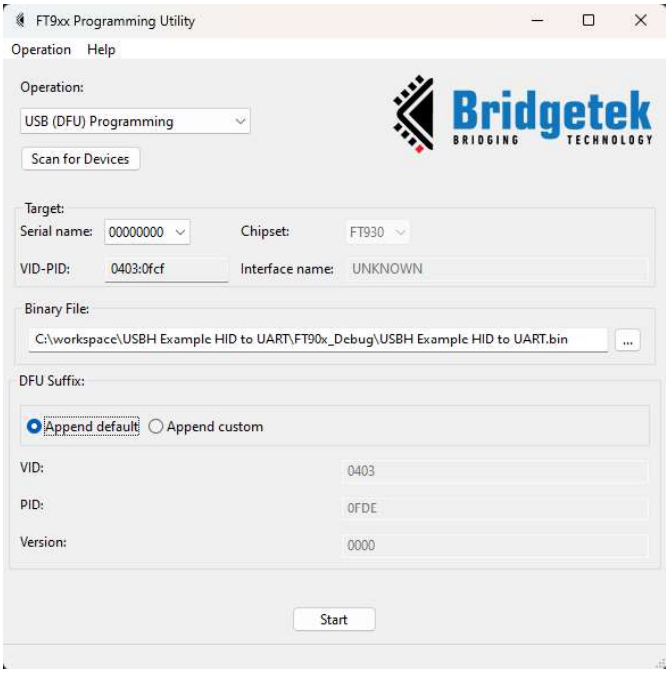

Figure 11 – USB DFU Programming

2.2 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.

2.2.1 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.
-i <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 make sure the programmer is the only device connected when programming or ensure the serial number flag is set correctly.

2.2.1.1 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 the 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`".

2.2.1.2 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.

2.2.1.3 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
```

2.2.1.4 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
```

2.2.1.5 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
```

2.2.1.6 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. See Section 2.3 for instructions for using this interface.

2.2.1.7 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.

2.2.1.8 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.

2.2.2 Eclipse

The command line programming utility is used by a Debug Configuration within Eclipse when debugging, see Section 3. 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 12. When it runs, it will output to a console window as in Figure 13.

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.

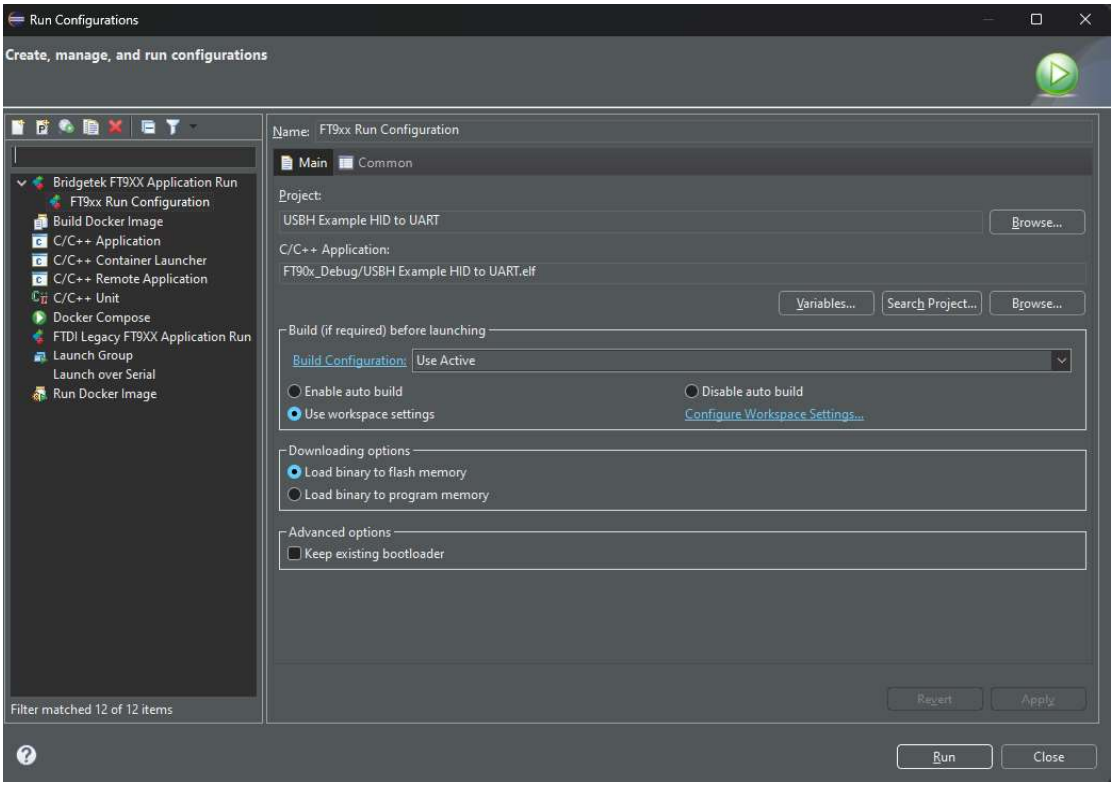

Figure 12 - FT9xxProg in Eclipse

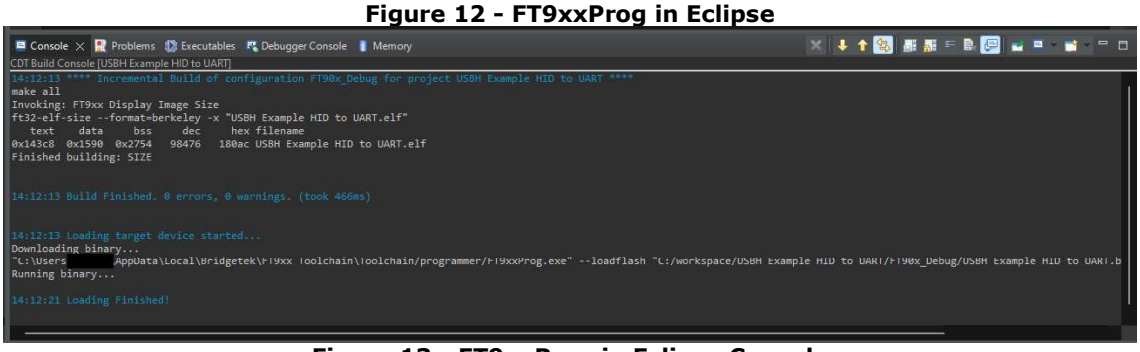

Figure 13 - FT9xxProg in Eclipse Console

2.3 Using the Python Script Utility

There is an alternative method of programming the FT9xx using only a UART on the PC system. This is in the form of a python script. It will wait until it recognises a reset or power-on event from the FT9xx bootloader and commence a programming sequence. The default time it will wait is 200

milliseconds. This can be changed with the Timeout value for a custom bootloader. An example of changing it is in Section 2.2.1.6 for the command line programmer utility.

A specially created binary file, which covers the whole of the Flash Memory, must be used for this method. The file always includes a replacement bootloader. An example of how to generate this file is in Section 2.2.1.4.

The UART0 interface on the FT9xx must be connected to the PC system and configured as a VCP (Virtual COM Port). This does not need flow control and the utility will configure the VCP to the required settings.

```
C:\workspace> FT9xxProg.py --loadflash 'image.bin' --device 0 –com COM5
------------------------------------------------------------------------
|              !!!Welcome to FT9xx Programming Utility!!!               |
|                   Copyright   Bridgetek Pte Ltd                      |
------------------------------------------------------------------------
Device = FT900
Initializing...
Reset the device to be programmed.
Waiting for bootloader...
```

Figure 14 - FT9xxProg.py Windows Command Line Console

At this point the program will wait until the bootloader is activated on the FT9xx. This can be done by resetting the device or power-cycling the device.

Once connected the programming will proceed as follows.

```
[========================] 100%
```

Figure 15 - FT9xxProg.py Windows Command Line 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 16.

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

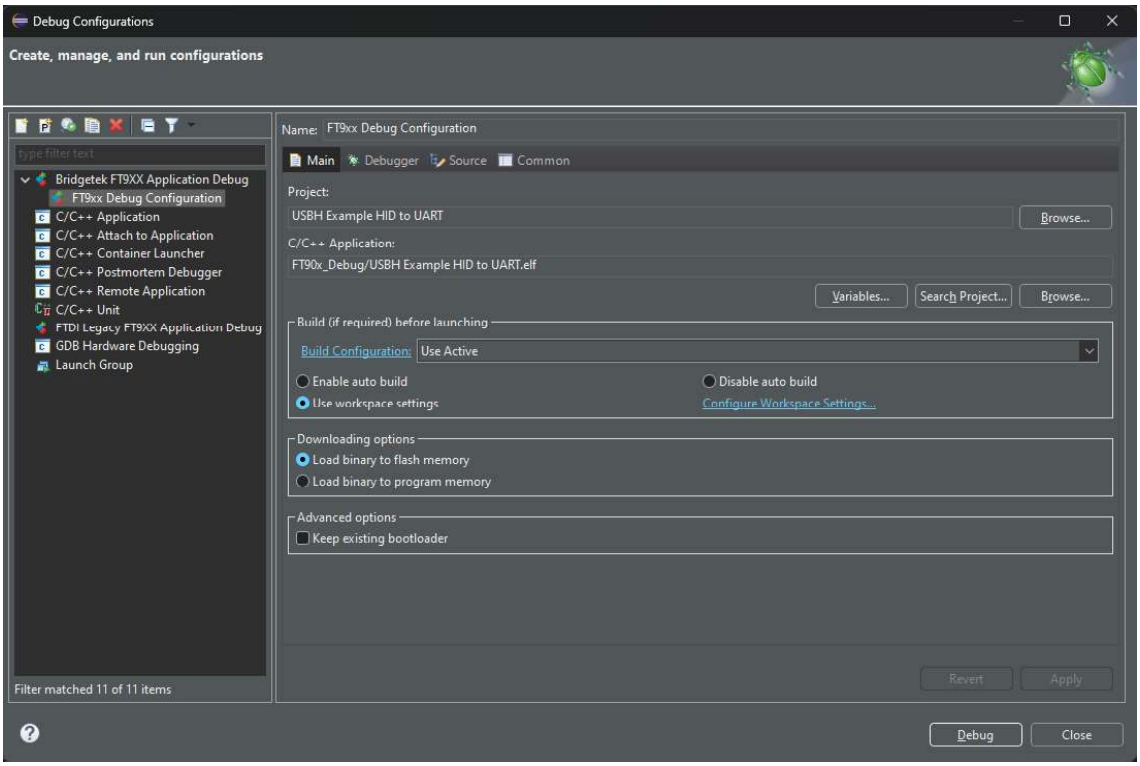

Figure 16 - Debug Configurations

Note that when this has been run once, it can be found via the debug icon as shown in Figure 17.

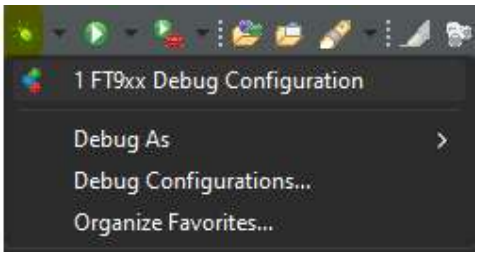

Figure 17 – Debug Icon

The Debug environment is shown in Figure 18 with key features highlighted.

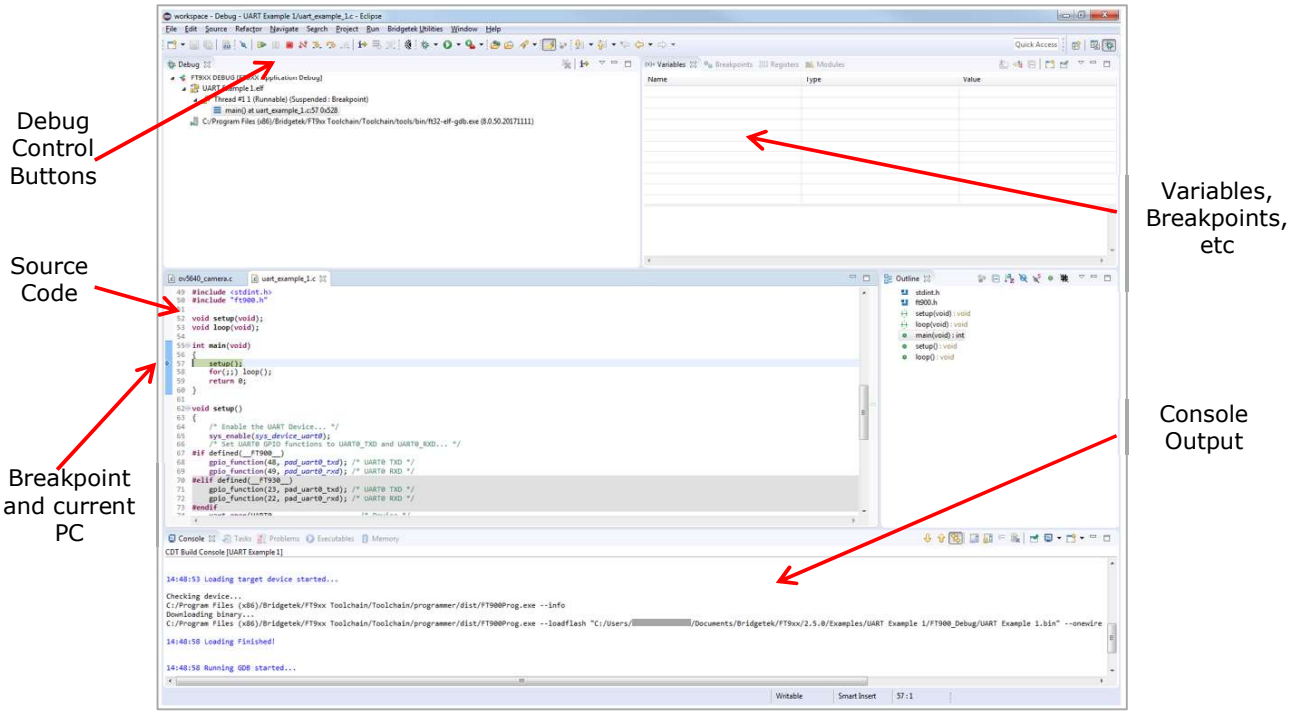

Figure 18 – Debug Environment

4 Programming Errors

This section describes the common programming errors.

4.1 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 19.

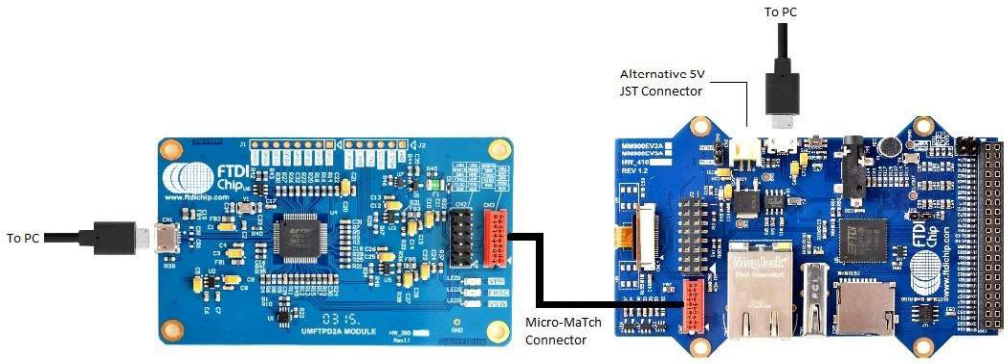

Figure 19 – Hardware Connection

If it is not powered, then errors will be seen as shown in Figure 20 or Figure 21 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 20.

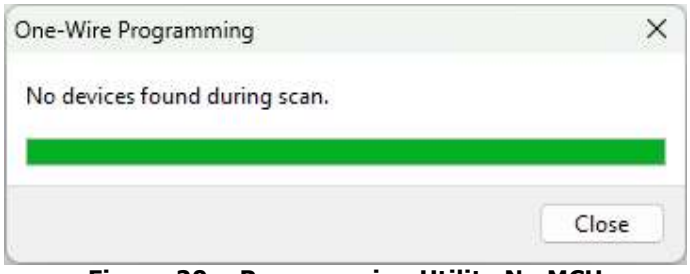

Figure 20 – Programming Utility No MCU

4.1.2 Command Line Utility

The command line utility will report an unknown device as shown in Figure 21.

```
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 21 – 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.

4.2 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. See 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 in **Figure 22 - Firmware image includes the D2XX device firmware**. 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.

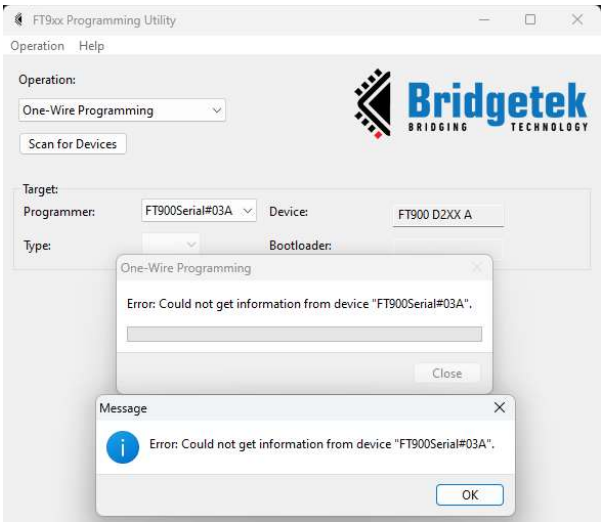

Figure 22 - Firmware image includes the D2XX device firmware.

4.2.1 Programming Utility

The Programming Utility will fail to see the UMFTPD2A Programmer and report the same error as in Figure 20.

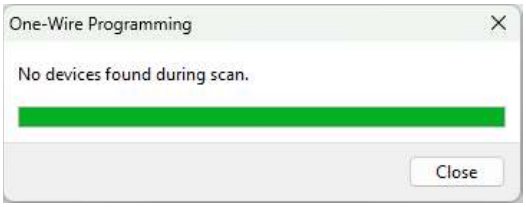

Figure 20 – Programming Utility No MCU

4.2.2 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 21 – Command Line Utility No MCU

4.3 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 23.

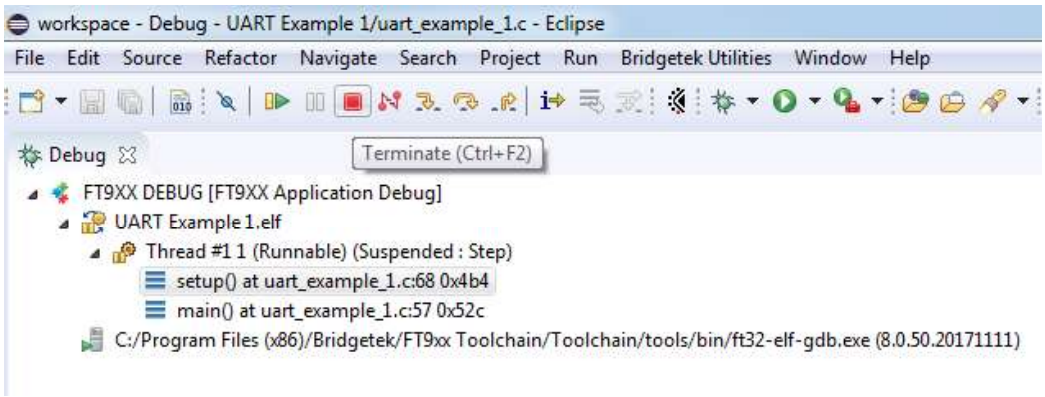

Figure 23 – Terminate Active Debug Session

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 24. Click on the "End Task" button to terminate the task and allow other connections to the device.

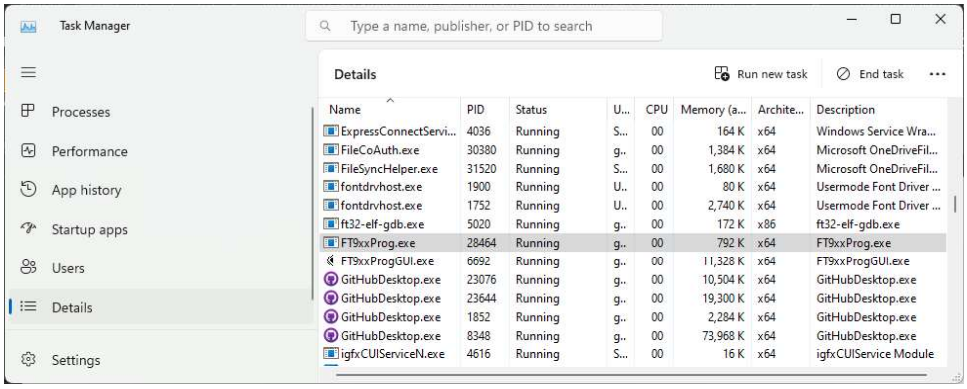

Figure 24 – Terminate Programmer in Task Manager

4.3.1 Programming Utility

The Programming Utility will fail to see the UMFTPD2A Programmer as shown in Figure 20.

4.3.2 Command Line Utility

The command line utility will report UMFTPD2A Module not found as shown in Figure 21.

4.4 FT90x vs FT93x

The FT9xx Toolchain allows the user to build a project for the FT90x or FT93x MCU as shown in Figure 25.

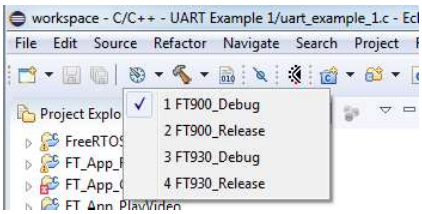

Figure 25 - FT9ox and FT93x Build

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.

4.5 Incorrect Board State

If the UMFTPD2A board or the FT9xx device gets into an incorrect state, then the message in Figure 26.

```
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 26 – Incorrect Board State

To rectify this re-run the programmer and the board and programming board are in a consistent state.

5 Debugging Errors

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

5.1 No MCU Connected

The error shown in Figure 27 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 27 – Debug No MCU Connected

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

5.2 No UMFTPD2A Connected

If the UMFTPD2A is not connected or not enumerated with the PC, the error shown in Figure 28 is shown.

```
**** 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 28 – Debug No UMFTPD2A Connected

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

5.3 Release Build

If the debug session is started when the release build is selected as shown in Figure 29, the debug session will launch but the release build contains no debug symbols.

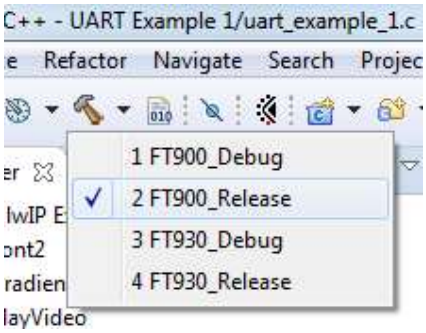

Figure 29 – Release Build

The source code won't be able to be seen in the debug session as shown in Figure 30.

Figure 30 – Release Build No Source

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

5.4 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 can be seen in Figure 31.

```
**** 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 31 – Build Error

5.5 FT90x vs FT93x Debug

The FT9xx Toolchain allows the user to build a project for the FT90x or FT93x MCU as shown in Figure 32.

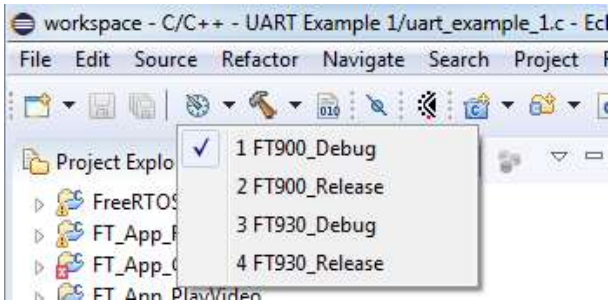

Figure 32 - FT90x and FT93x Build

Care must be taken to select the correct target MCU before launching the debug session or the error shown in Figure 33 will be seen.

```
**** 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 33 - FT90x and FT93x Error

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

5.6 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.

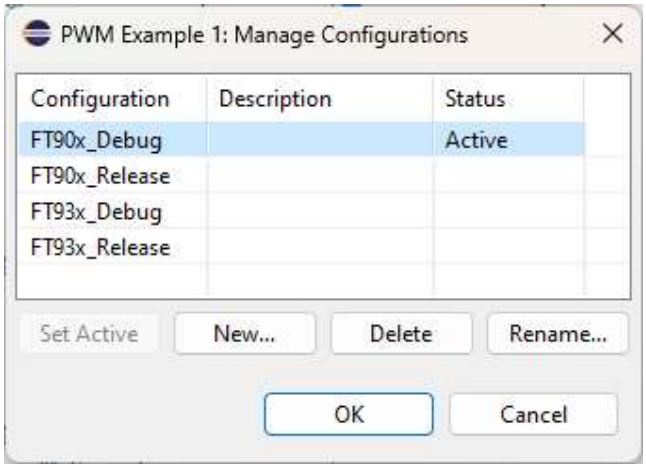

Figure 34 – Manage Configurations

6 USB DFU Programming Errors

Common USB DFU errors are detailed in the forthcoming sections.

6.1 No DFU Device

If there is no USB DFU interface available, the error is shown in Figure 35.

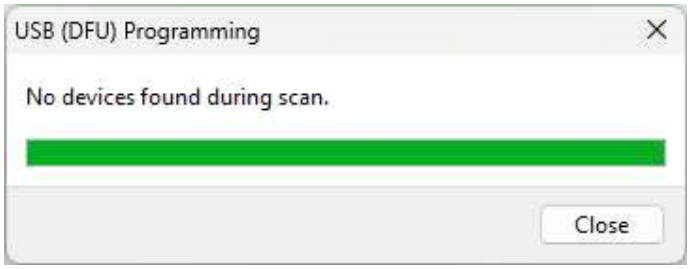

Figure 35 – No DFU Device

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. See Section 7.4 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.

6.2 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 36 will be seen.

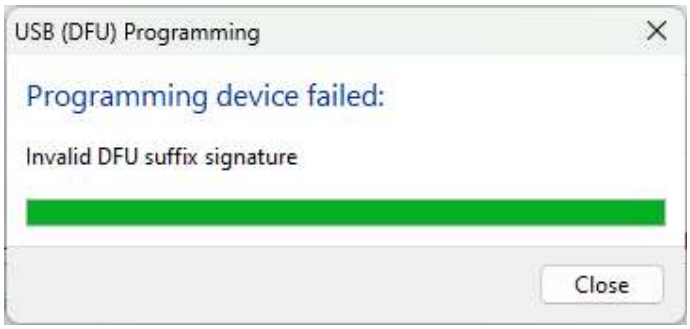

Figure 36 – DFU Caution

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.

7.1 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 37.

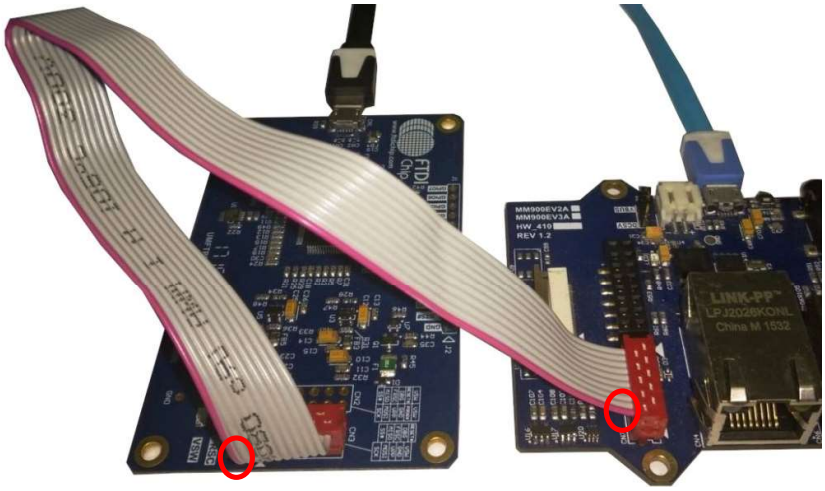

Figure 37 – MicroMatch Connection

7.2 UART connection

The UMFTPD2A board also provides the UART port to monitor/debug. Ensure that the wires are connected properly see the **Figure 38 –** UART Connection. You can use any COM port application to interact with the 3rd "USB Serial port" see the **Figure 39.**

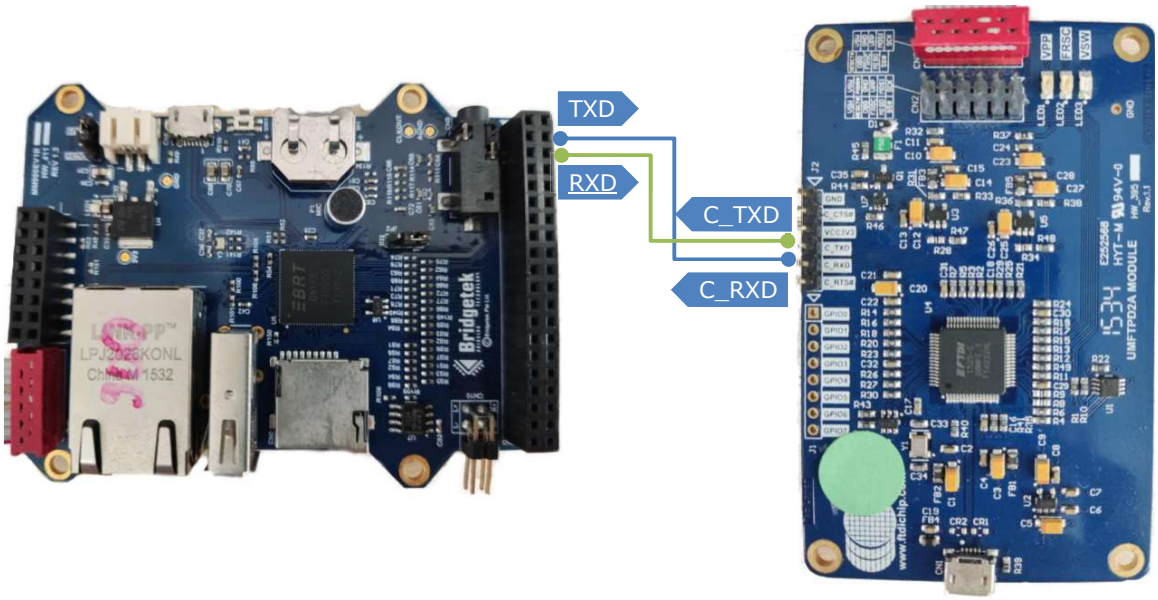

Figure 38 – UART Connection

7.3 UMFTPD2A drivers

Check that the UMFTPD2A drivers have been installed correctly as shown in Windows Device Manager. See **Figure 39**. COM10 to COM13 is the UMFTPD2A in this screenshot. The UMFTPD2A contains an FT4232H IC which is a four port USB device.

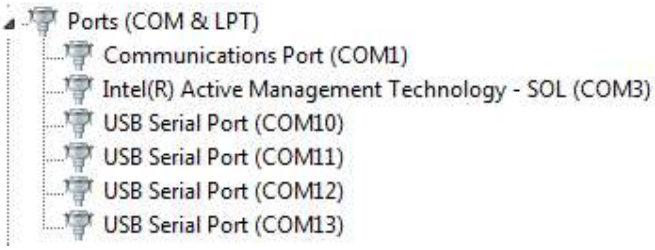

Figure 39 – Device Manager

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 see our Installation Guides.

7.4 Restore the Bootloader

Restoring the bootloader of the FT9xx MCU can be done using the FT9xx Programming Utility and UMFTPD2A only. See **Figure 40**.

This programs the MCU into a default factory state.

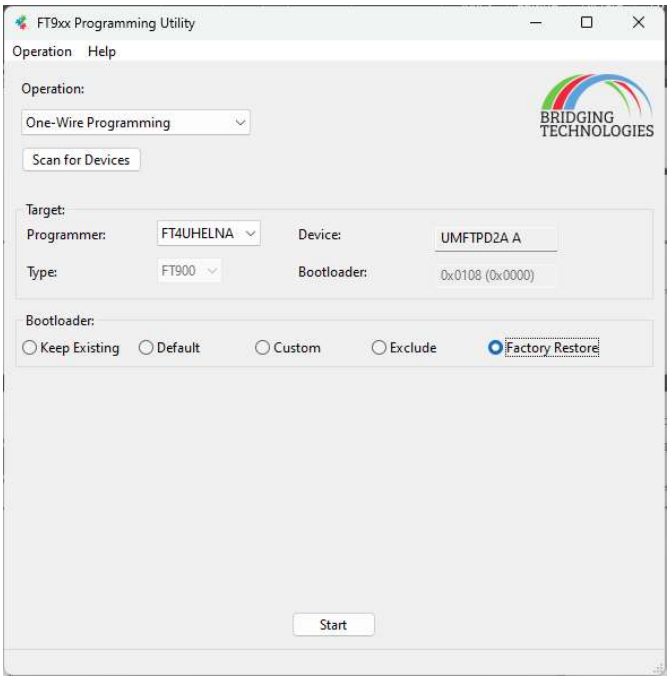

Figure 40 – Restore Bootloader

7.5 Reinstall the FT9xx Toolchain

Reinstalling the FT9xx Toolchain is always a last measure.

Ensure the default Python v2.7.10 is selected during the install. It comes packaged in the toolchain. Our python scripts have also been ported to work with version 3.x.

7.6 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.

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

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