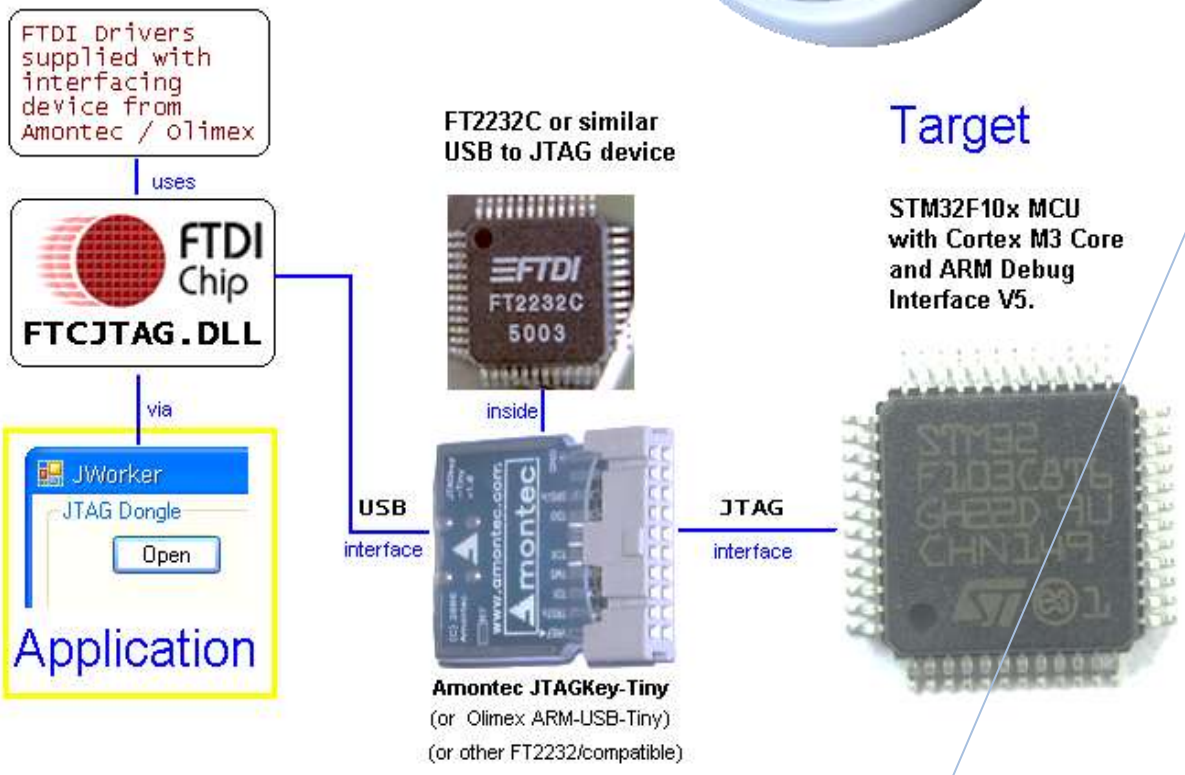


JWorker

Version 1.0.0.11

JWorker is a simple JTAG STM32F10x 'flasher' written in Visual Basic .Net Express 2010 using FTCJTAG.DLL and a low cost JTAG adapter like Amontec JTAGKey-Tiny or Olimex ARM-USB-Tiny.

Bob Seabrook
9/2/2013

Contents

J-Worker

JWorker is a simple JTAG STM32F10x 'flasher' written in Visual Basic .Net Express 2010 using FTCJTAG.DLL and a low cost JTAG adapter like Amontec JTAGKey-Tiny or Olimex ARM-USB-Tiny.

User Interface

Purpose

Easily program binary files to flash on an STM32 via JTAG.

View STM32 targets addressable space via JTAG.

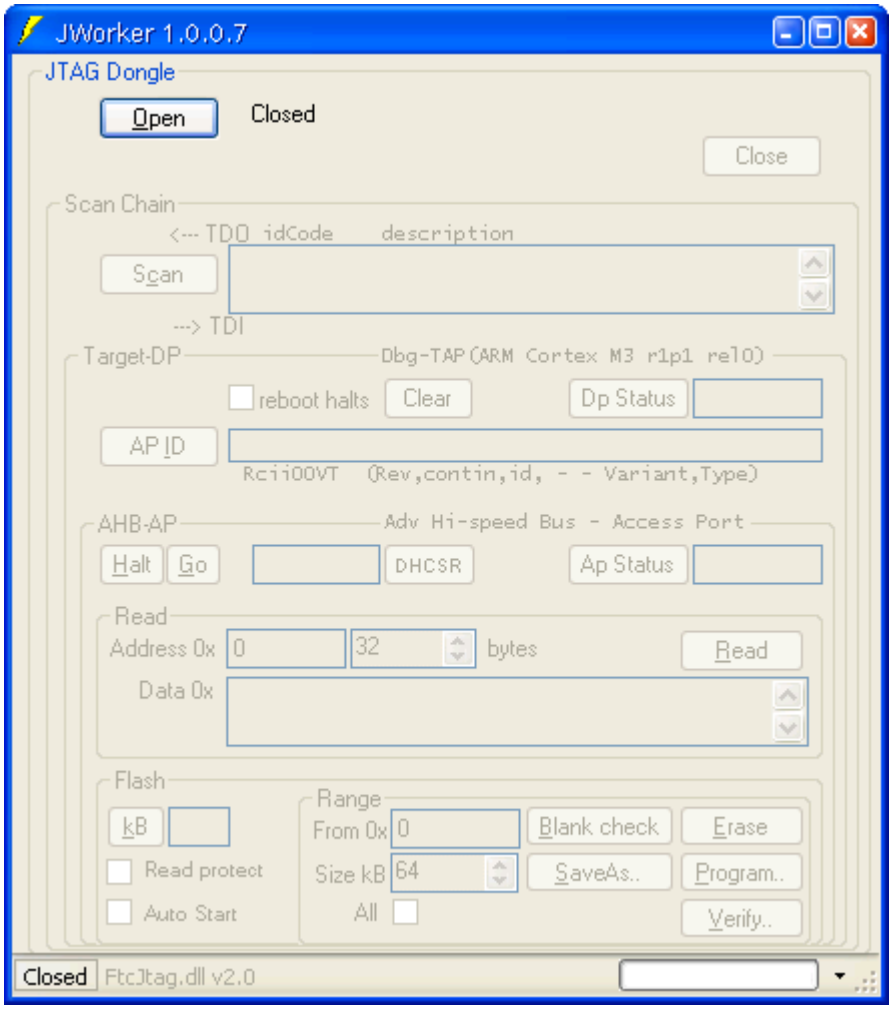
Backup STM32 targets flash to a binary file via JTAG.

Confirm identity of devices in the JTAG scan chain.

Rationale

Click buttons from top to bottom on the left side of the window. As preparation stages complete buttons are un-greyed. Once ready, use the other buttons to do tasks.

Initial Appearance



Quick Preparation Example – Required before other procedures

1. Connect Amontec JTAG-USB-Tiny FTDI/USB based JTAG device between STM32 JTAG interface and PC's USB. Check polarity of connectors.

2. Power up target. Remember to power up USB hub if using one.

3. Run JWorker

4. Click **Open** This connects PC to USB device.

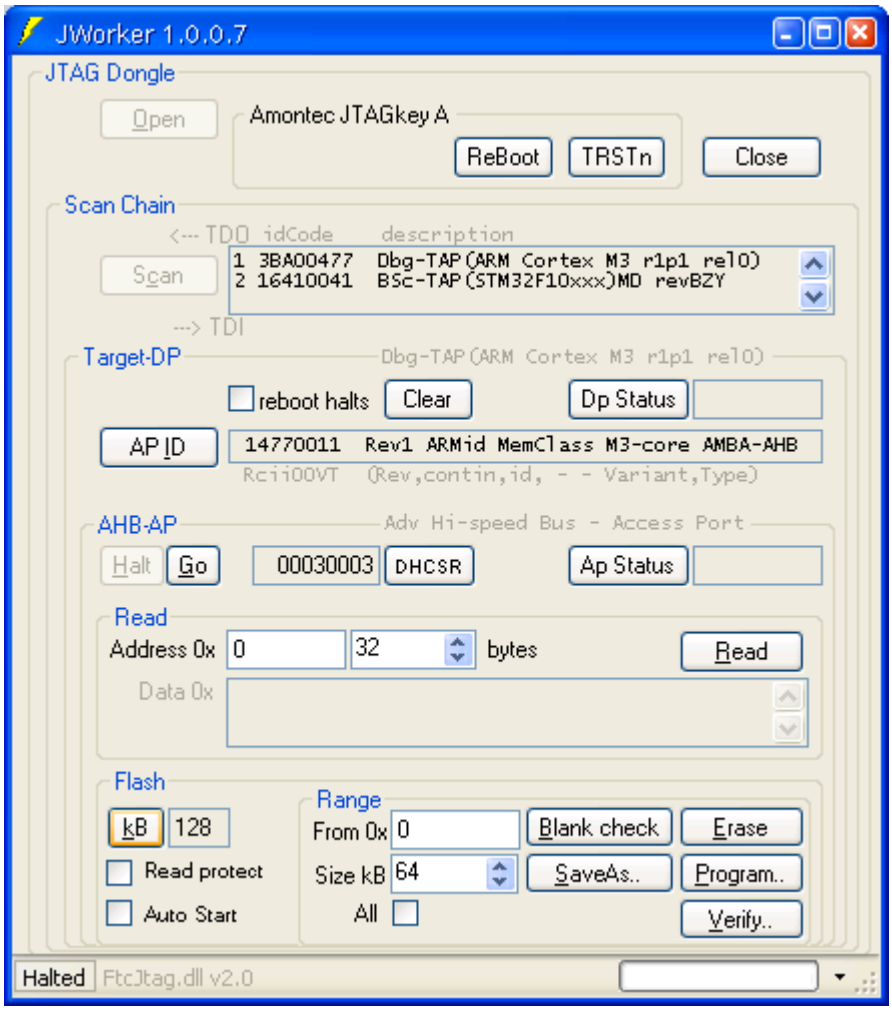
5. Click **Scan**. If target uses JTAG lines for general purpose i/o hold it in reset state while doing so.

6. Click **AP ID** Check: 14770011 or 04770011. 00000000 or FFFFFFFF is an error. Non essential.

7. Click **Halt** - if un-greyed. Core must be halted to un-grey erase/Open(program) buttons.

8. Click **kB** to establish or confirm SIZE of flash in kBytes. You need to know this.

Here is the appearance after preparation last stage is complete.



Now any of the following can be done

Quick Read Example - Looks at any addressable space.

1. Locate the **Read** box.

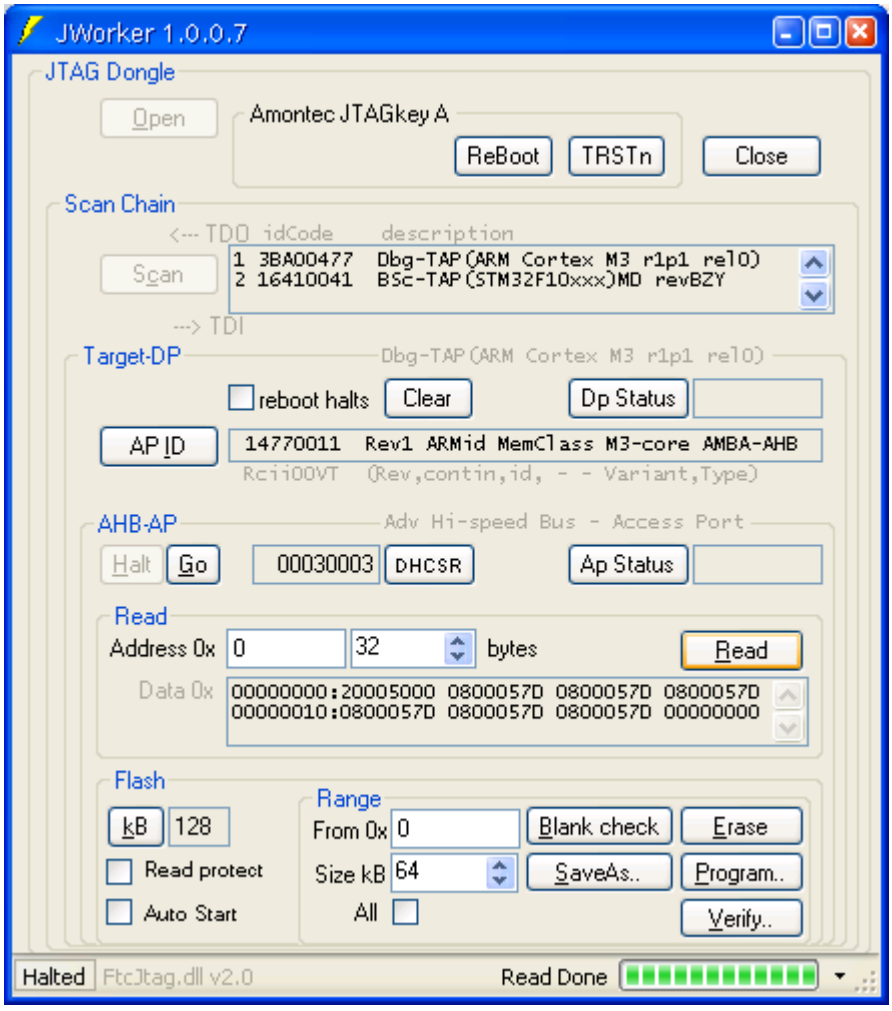
2. If required adjust the **Address** and number of **Bytes**. Default is first 32 bytes of flash.

3. Click **Read**.

4. **Wait** for bar-graph at foot of display to progress to completion.

5. **View** Data in data box. Use scroll bar if necessary.

Here is the appearance [example] after clicking Read;



Notice that whilst flash starts at 0x08000000 the SAME flash also appears at address 0 [the alias], so 0 is used here just because it's simpler.

Quick Erase Example – Use before programming flash

1. Locate the Flash→**Range** box

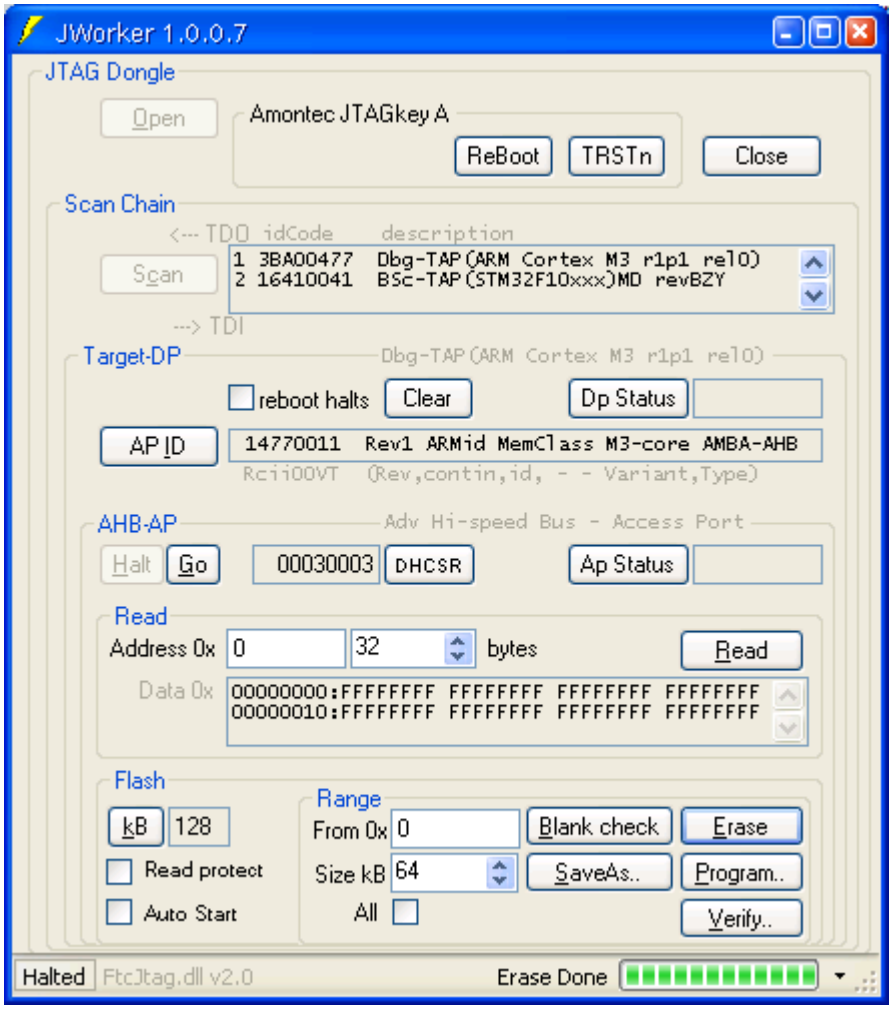
2. Set the start and size of the range using the '**From'** and '**Size'** boxes.

3. Click **Erase**

4. Confirm **OK**.

5. **Wait** for bar-graph at foot of window to reach completion.

Here is the appearance after erasing flash.



Notice Read was clicked again after Erase in order to *show* that the flash was indeed empty (all FF's). After that the flash range was erased AGAIN just to *show* the 'Erase Done' message at the foot of the window.

Quick Programming Example – Programs flash from binary file

1. Locate the Flash→**Range** box

2. Set the start and size of the range with the '**From'** and '**Size'** boxes.

3. Click **Program**, ***or*** drag a binary (.bin) file onto the form and go to step **6**.

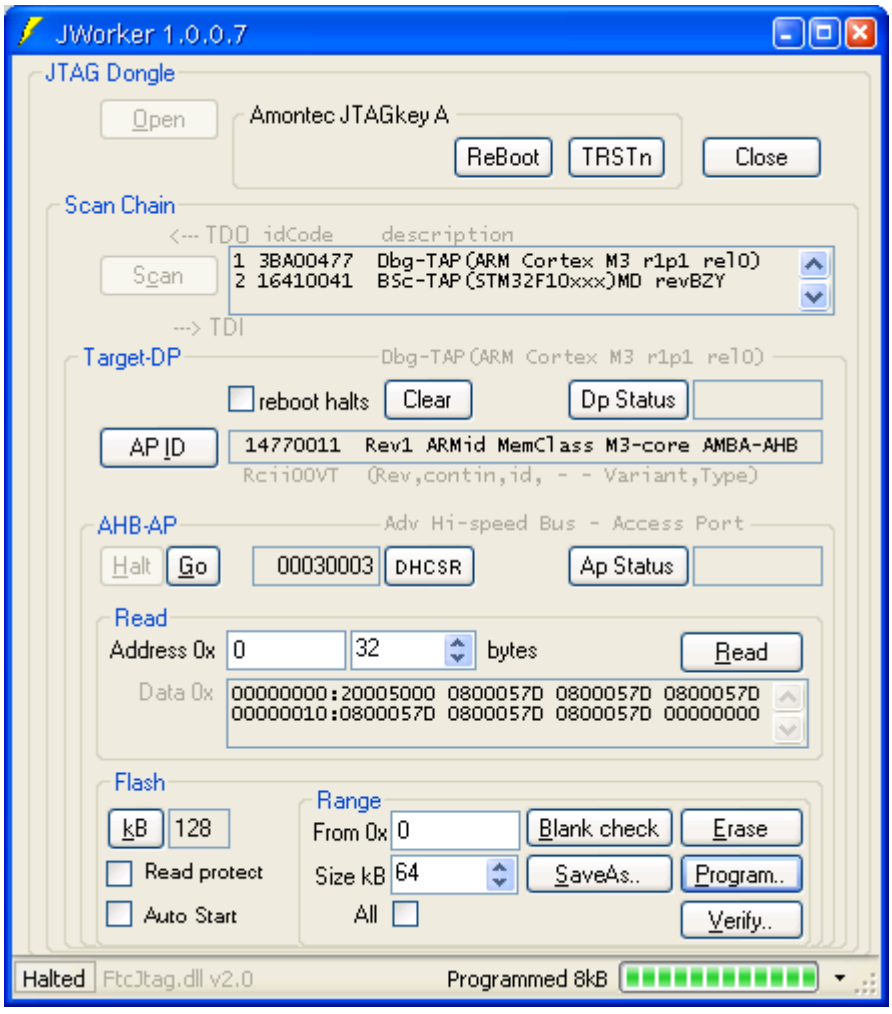
4. Select binary file to program.

5. Click **Open** or double-click the filename in the list-view

6. Confirm **OK** Notice the range programmed is limited if the file is bigger than the range.

7. **Wait** for the bar-graph at the foot of the window to reach completion.

Here is the appearance after completing the above procedure.



Notes. Read was clicked again after Program in order to show that the flash was indeed programmed (not FF's). After that the flash range was programmed yet AGAIN just to show the 'Programmed… message at the foot of the window. Notice also that 8kB programmed because the file was 7392 Bytes.

Quick Save to File Example – Backs-up flash to file

1. Locate the Flash→**Range** box

2. Set the start and size of the range to save with the '**From'** and '**Size'** boxes.

3. Click **SaveAs**..

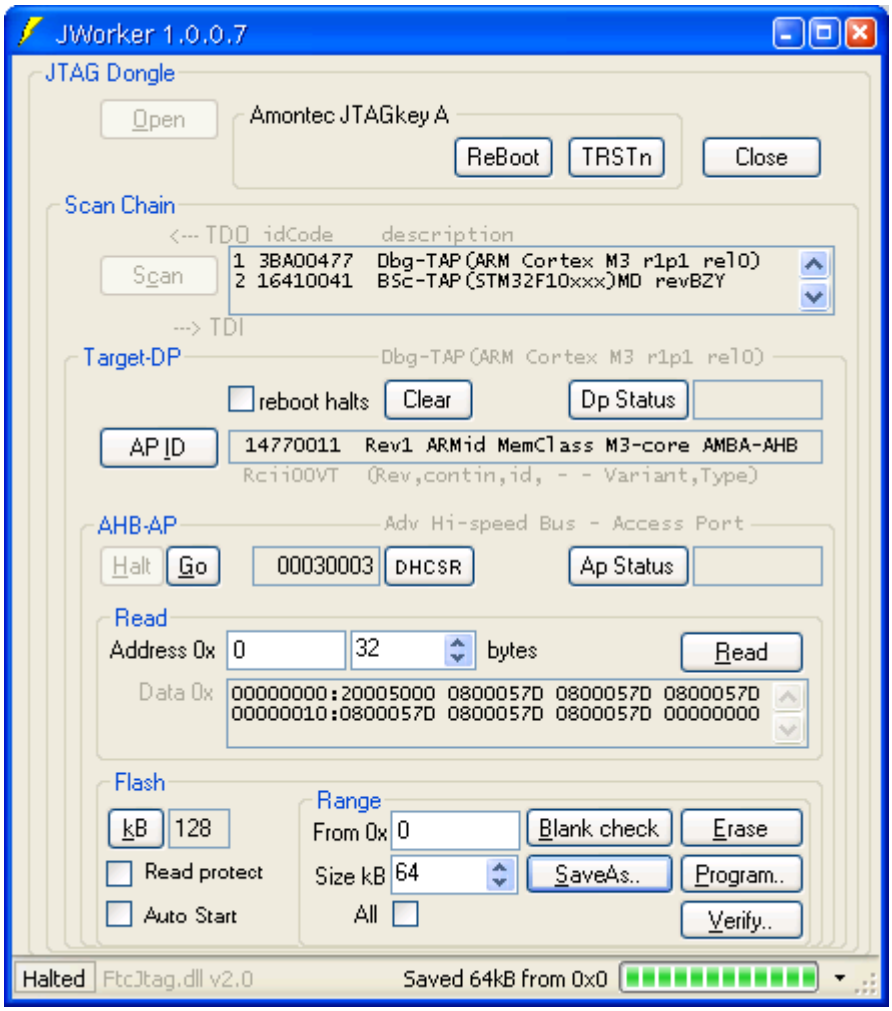
4. Optional: Using the List-View select another folder in which to save the file

5. Type a NEW name for the file at the Filename prompt.

6. Click **Save**

7. **Wait** for the bar-graph at the foot of the window to progress to completion.

Here is the appearance after step 7 above.



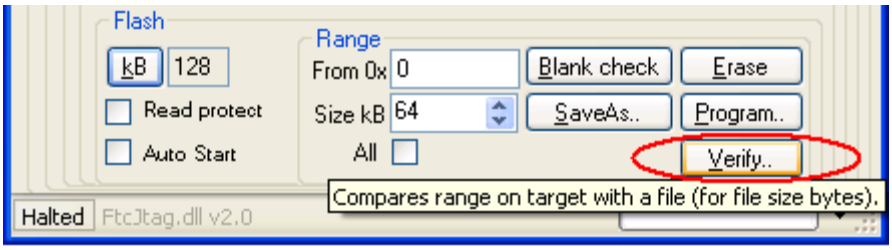
The **SaveAs** button is useful to back-up the flash before re-programming it so that the operation is reversible. Notice that blank flash is saved as byte values of FF.

Quick Verify Example – Compare file with flash

1. Locate the Flash→Range→**From 0x** box.

2. Specify the address start of the range to verify. Example1: **0** [start of flash (alias) in hex].

3. Click **Verify** in the Flash→Range box, ***or*** drag a binary file *onto* the **Verify** button and go to step 5.



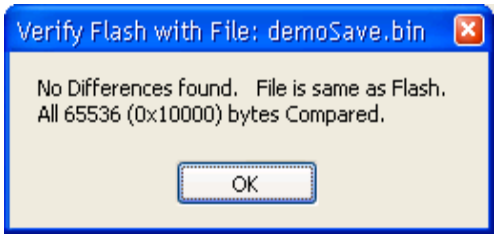
4. In the file selection dialog – locate the file to compare and **double-click files name** to proceed.

5. Click **OK** in response to the confirmation message.

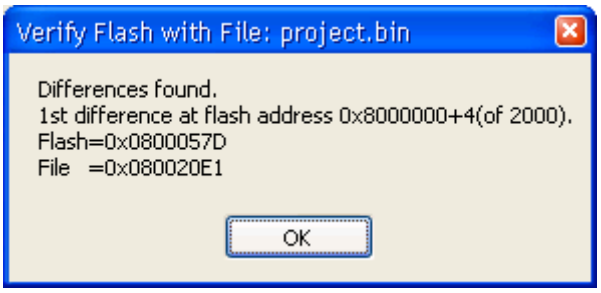
6. **Wait** for the bar-graph at the foot of the window to progress to completion.

7. **Read** the result message displayed upon completion.

Example1: No Differences.



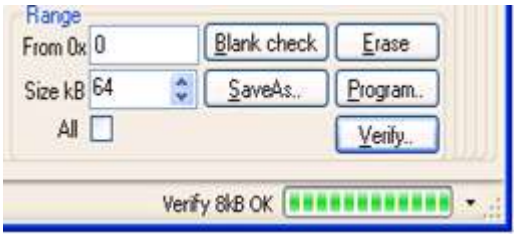
Example2: Differences found.



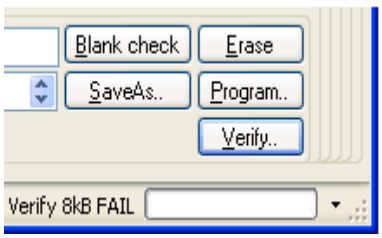
8. Click **OK** in the result message box.

9. Notice the Verify success or failure is also displayed next to the bar-graph, and that the compare stops when the first difference is found – which may be before the bar-graph reaches completion.

Example1:
After No Differences.



Example2:
After Differences Found.



Notice 8kB verified because file was between 7 and 8 kBytes.

Quick Blank-check Example

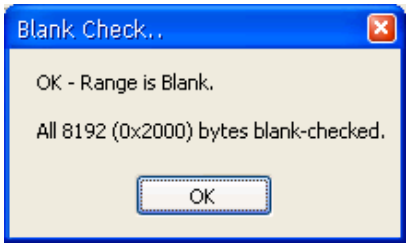
1. Locate the Flash→Range→**From 0x** box.

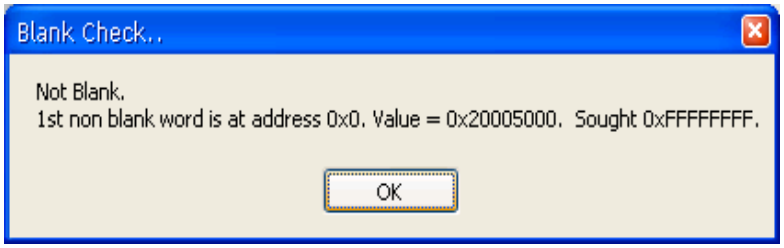
2. Set the start and size of the range to blank-check with the '**From'** and '**Size'** boxes.

3. Click **Blank-check** in the Flash→Range box.

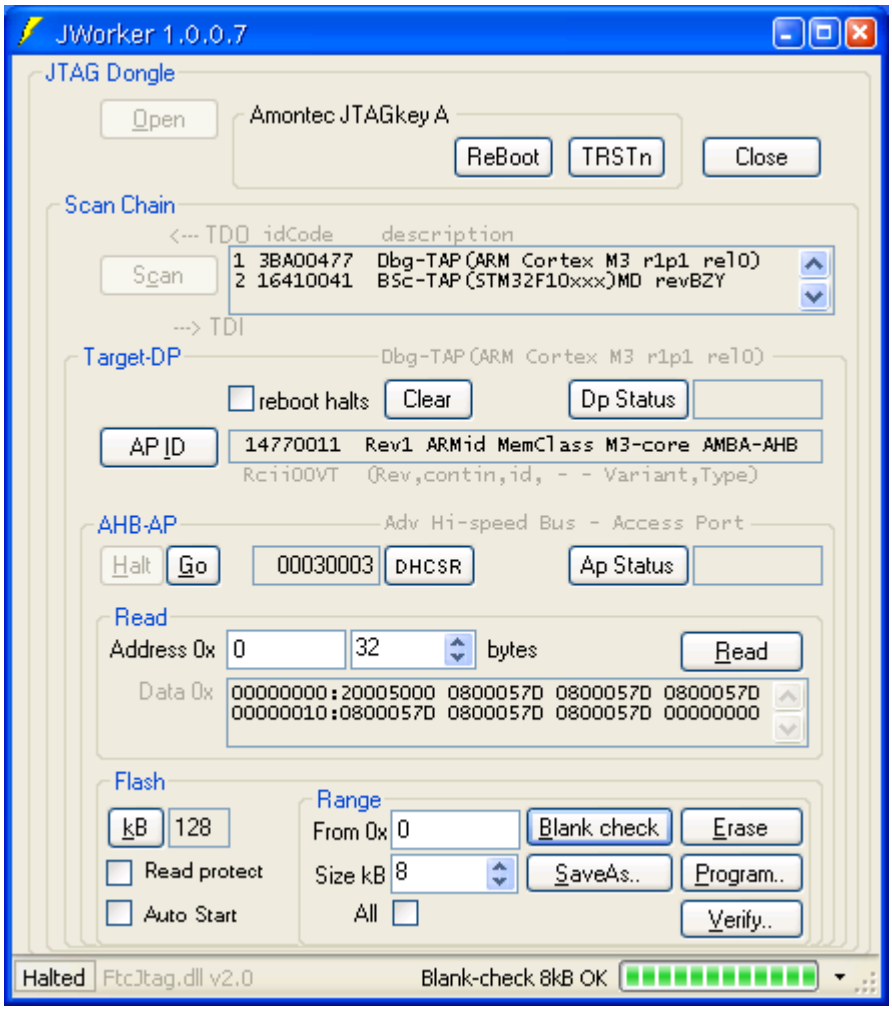
4. **Wait** for the bar-graph at the foot of the window to progress to completion.

5. Notice the result in the message box and click OK when understood.





Here is the appearance after completing the above procedure.



Advanced Preparation Example – When firmware reconfigures JTAG lines

This method requires the ability to manually hold the targets system reset at 0V during some steps.

Basically this method works by ensuring that the firmware – which interferes with JTAG lines - is prevented from running by setting the DEMCR.HaltOnCoreReset bit *whilst under reset* so that on emergence from reset the target halts and JTAG continues to work. See reference [4] section 10.2.

1. Power-down the target, connect JTAG/USB device between target and PC USB. Check polarity.

2. Start Jworker Application.

3. Manually force the system reset line to 0V and hold it there for the next 2 steps.

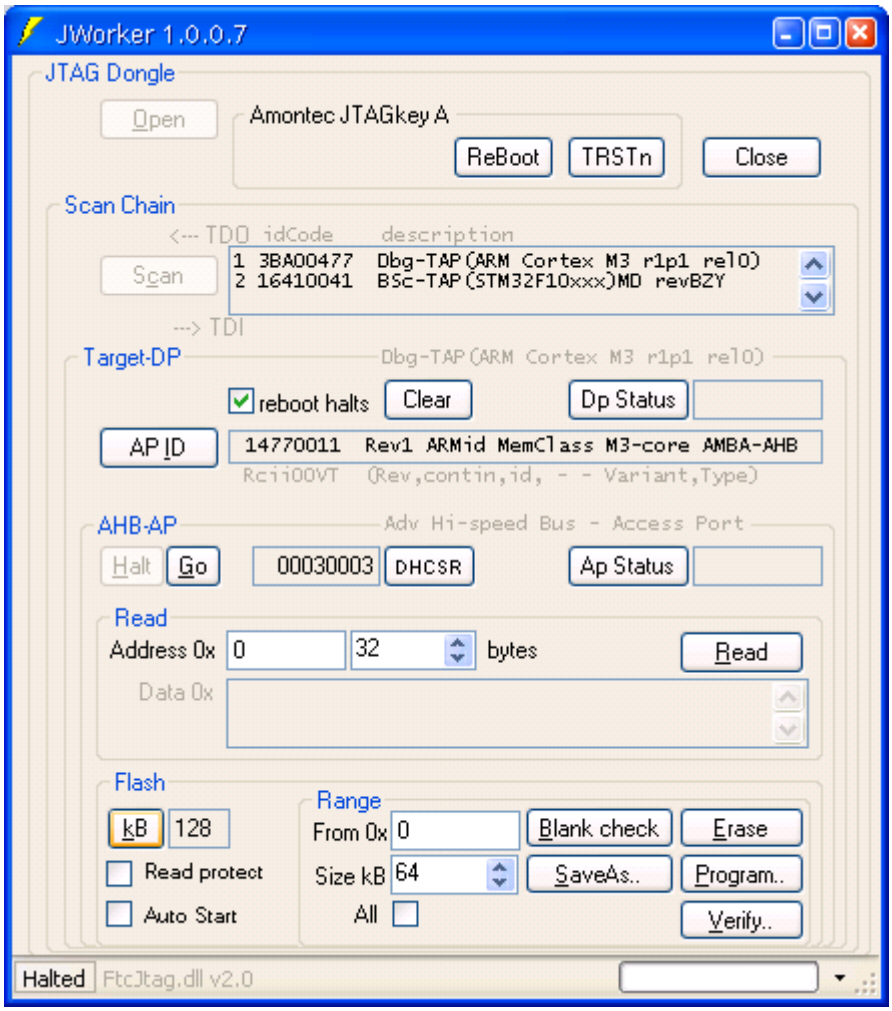
4. Power up the target. Keep reset at 0V.

5. Click **Open**, **Scan**, and tick **Reboot-Halts**. Keep reset at 0V.

6. Release reset. Stop holding it at 0V - allow it to go high.

7. Click **APID**, **DHCSR**, **kB**. State should be halted, Flash buttons ungreyed.

Here is the appearance at the end of this procedure. Notice Halted displayed at foot of the window.



Limitations

Multiple instances of the application are *not* allowed to run at the same time because it is likely that they would find the JTAG unavailable anyway.

Jworker deliberately avoids changing flash write-protection on the basis that its current protection settings are for good reason. It is considered that this particular application *may* be *more* useful if kept simple and secure. Read-protection for the entire flash can be enabled and disabled.

JTAG transfers are not fast. Developed for Windows XP 32bit. No Linux version. Unsigned. Four languages only; English, French, Spanish, Polish. JWorker terminates safely when un-anticipated problems occur.

System Reset: The **ReBoot** button can reboot the system. However trying to hold and maintain the system reset low programmatically during the **Open** and **Scan** phase may not be as effective as manually *shorting* the system reset low to 0V at this time. This is because targets may not gate the debug reset with reset from other contending sources [using a 74vhc08 AND to OR the active low signals]. For this reason the application does not make a futile attempt to hold reset low programmatically during Open and Scan.

The flash erase procedure always erases 1kB blocks even for bigger memories >128kB where the erase blocks might be 2kB or 4kB. This may slightly slow the erase procedure but it guarantees the maximum attainable erase-map resolution for all flash sizes and is only possible because block erasure is not interleaved with block writes.

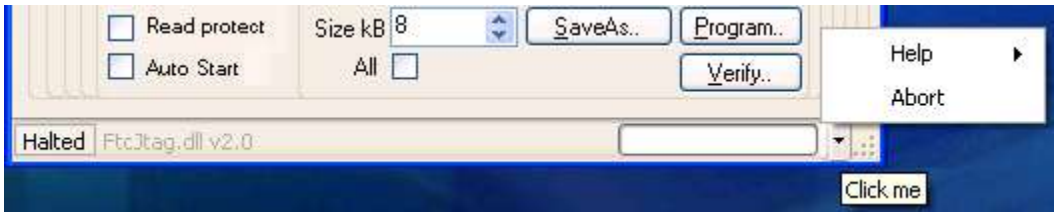
The application should be restarted if the JTAG is disconnected and reconnected during use, otherwise errors will be reported out of context which is misleading.

When using drag and drop, to program and verify, binary files must have ".bin" extension.

Advice

Help – This Document

Click the down-arrow to the right of the bar-graph and click **Help** from the menu.



Abort - Stopping JTAG Transactions

To abort long *ranged* operations; click the down-arrow to the right of the bar-graph and click **Abort**. This practice is not recommended because it can be misleading if only the start of a range is programmed or erased or saved.

Speed – USB Hubs can be faster

Connect through a USB hub for faster data transfers.

Compatibility – Amontec and Olimex

The Amontec JTAGkey-Tiny and the Olimex ARM-USB-Tiny work with this software. Other devices that utilize the FTDI FT2232C compatible range of chips are also likely to work.

Compatibility – With IWDG Firmware Watchdog

An enabled IWDG watchdog is prevented from restarting a core - halted via the debug interface - by setting bit 8 of the DBGMCU_CR register documented in the STM32F10x Rev14 reference. See reference [8].

Reboot Halts Checkbox – Advanced use only

As shown in the Advanced Preparation Example above, only tick this checkbox ***while*** the target is ***under*** reset since before powering up the target. The aim is to ensure that when reset is ***released*** the target halts before running ANY code that would mess up the normal working of the JTAG lines by configuring them for general purpose i/o. In other words it is *not* normally necessary to tick this box if the firmware does not interfere with JTAG lines.

Clear Button – Illegal Address Accesses

Illegal address accesses cause a latching '***sticky error***' in the debug interface which is easily cleared using the **Clear** button illustrated below. Some operations that access a range will automatically clear the sticky error before returning control back to the operator. Nevertheless a sticky error can be present after opening the JTAG port.

Status Button – DP-Control/Status Register view

As illustrated above, when a sticky error is present, data values read as zero and bit 5 of the DP-Control/**Status** register is set. See reference [5]; Arm Debug Interface; section 6.2.3 page 6-10.

DHCSR Button – When Code Space is Blank

An attempt to run code from a blank (FF) flash memory harmlessly sets 'lockup' bit-19 [0x0008:0000] of the MCU's 'Debug-Halting-and-Control-Register' [DHCSR]. This is a latching '***sticky***' bit. The core remains in the running state. Click the **DHCSR** button to update the view of the DHCSR. See reference [4] section 10.2.1 page 10-3.

Auto Start checkbox – Saves time at Start-up

Tick this checkbox so that next time the application is started the Open, Scan AP ID, Halt, kB buttons are automatically clicked all ready to begin flash operations like Erase or Program. If a problem occurs – like no target connected - then the procedure will stop at the point of failure allowing you to fix it and continue manually. The checkbox is only obeyed when the application starts – not every time the port is re-opened.

Read-protect Checkbox – Prevents flash being read via JTAG

This checkbox allows the targets entire flash to be protected or unprotected from reads via the JTAG port. Un-protection also erases the entire flash. This is a feature of the STM32F10x. The checkbox shows the initial and current read-protection status of the targets flash. Changing the checkbox changes the flash protection which takes effect upon the targets next power-on reset. Users are

prompted to apply the power on reset at the appropriate time after changing the checkbox. Do not change this checkbox - either way- without the targets flash image available on file. When read-protected the Flash-Range box and Flash kB button are greyed/disabled. Users wishing to protect their IP may also embrace other STM32 features discussed in this security text.

References

The following external files were essential to *make* this application. The list may be incomplete. The PDF documents are *not* needed to *use* the application. Hypertext links valid on 1st July 2013.

[] Reference	File properties		Author
1 Microsoft Visual Basic 2010 Express. ISO link.	Installed from an	ISO	Microsoft
2 FTCJTAG.DLL Zip file.	Ver1.9. Non Doc.	DLL	FTDI
3 AN_110 Programmers Guide for High Speed FTCJTAG DLL	Ver1.2. 2009	PDF	FTDI
4 Cortex-M3 Revision r1p1 Technical Reference manual.	DDIO337E	PDF	ARM Limited
5 ARM® Debug Interface v5 Architecture Specification	IHI0031A	PDF	ARM Limited
6 IEEE 1149.1 JTAG AND BOUNDARY SCAN TUTORIAL	Texas Instruments	PDF	Dr B Bennets
7 PM0075 Programming manual STM32F10xxx Flash ….	Rev1. CD00283419	PDF	ST
8 RM0008 Reference manual STM32F101xx ….	Rev14. CD00171190	PDF	ST
9 Datasheet for STM32F103x8, STM32F103xB ….	Rev13. CD00161566	PDF	ST

Other Files Provided

8kbinary.bin: This is just a small 8k demo firmware for the STM32F103xx that can be fast programmed onto the target and read after programming to show typical non-blank data in flash. It does configure some general purpose i/o lines as outputs [port C bits 4,5,10] so it *could* damage your hardware if run using the Go or ReBoot button.

Details of 8kbinary.bin are beyond the scope of this document. *Nevertheless* it implements a virtual serial communications [CDC] so Windows 'ding dong' notes play upon USB enumeration] if;

a) the target has been BOTH programmed AND rebooted with 8kloader.bin at address 0
b) the next 8k [at 0x2000 after the 8kbinary] is blank (FF's).
c) the target has a USB peripheral
d) the target has a USB physical connector and circuit outside
e) the targets USB is connected to an available Windows [XP] USB port.
f) Windows [XP] is already setup using the STM32 CDC driver: 'stmcdc.inf' from ST.

Un-install

In windows XP use Add/Remove programs, locate JWorker. Remove it. Use equivalent in other versions of Windows.

Upgrade

To manually upgrade the Jworker application, first make a note of your existing version [in the title bar of the Jworker application window] and browse to
http://www.seabrooks.plus.com/jworker/current-download/
Notice the file v10xx which shows the current version number available.
If this version is later than yours then

- download file publish-jworker.zip. [e.g. 1008 is later than 1003]
- uninstall your existing version as described above.
- Install the new version by running the setup.exe in the zip file.

Troubleshooting

1. Problem - The Program and Erase buttons are **disabled**, yet other flash buttons are enabled,
   Reason – the target is running code, you cannot erase or program flash that is running code.
   Solution - Click the Halt button on the left side of the window.

2. Problem – All flash box controls are **disabled** after Open and Scan, so I can't do anything really.
   Reason – The flash may be read-protected – see the **read-protect checkbox** in the bottom left.
   Caution – Turning off read-protection *immediately* ERASES the entire flash.
   Solution – To reprogram the targets flash you must first turn off read-protection.

3. Problem – The ID Codes in the Scan-Chain box are all **FFFFFFFF** and I can't proceed further.
   Reason – The target STM32 is not connected to the JTAG port [properly] or it is not powered up.
   Solution – Check the target board is powered and connected. Repower and reconnect.

4. Problem – The AP-ID is **00000000** and other values are 0 when I expected non zero values.
   Reason – A latching 'sticky' error occurred in the ARM debug interface so values all read as 0.
   Solution – Click the Clear button and try again.

5. Problem – clicking Scan reports the ID-Codes in the scan chain but I **cannot proceed** further.
   Reason – Application is Incompatible with the target. IDCode "**1** 3BA00**477**" is required.
   Details – More specifically the IDCode for scan-chain entry 1 MUST end with 477.
   Solution – For that target use another application such as the awesome OpenOCD.

6. Problem - The **auto-start checkbox has no effect** when I click the Open button
   Reason – Auto Start only operates when the application is first started.
   Solution – None. Open is not supposed to invoke auto start sequence.

7. Problem – I see **silly ID codes** when I click Scan or AP-ID button.
   Reason – Target firmware may be using running and re-using JTAG lines for general i/o.
   Solution – Try the Advanced Preparation procedure discussed on page 11.

Other Information

Origin

Author Bob Seabrook: Bob@seabrooks.plus.com. Web: www.seabrooks.plus.com/jworker . The author is NOT an authority on this subject. The ware will NOT be maintained after the initial releases issued in summer 2013.

Disclaimer

This software is not suitable for any specific or explicit or implicit purpose or system and as such the author or affiliates or associates or providers or service providers can take no responsibility for losses apparent due to its attempted use in whole or in part.

STM32F10xx's have a **read-protect** and anti-tamper facility which should be used where there are concerns about the exact level of ease with which a chips flash can be copied. Read protection can be disabled once enabled. Disabling read-protection also erases the flash (see page13).

Copyright

Future Developments

To handle other file formats because binary files do not include the target addresses which can be problematic [at build time too] if the image contains a big unused area. This change is unlikely to happen since a build can usually create files of different formats for example using objcopy in the GNU tool-chain.

Add an 'Auto-check' checkbox in the flash range box to automatically do a blank check before programming, automatically erase the range if not blank and automatically verify the range after programming. This change might necessitate a time-stamped log of activity, or at least a completion time of the last action shown in the status bar. Such a log could even access the 96bit target chip identification code so there is no doubt about which chip the actions were performed upon.

Allow users to specify some favourite bin file locations rather than relying on the open/save file dialogues MRU(most recently used) folder pick-list. This has been resolved by being able to drag files onto the form or verify button.

History

1.0.0.0

1. First version

1.0.0.1

1. Added help – reached as shown in the Advice section of this document.
2. Updated this documentation. General application snap-shots remain **not** updated for mods 3,4.
3. Moved the Close button outside of the Amontec box so that it is accessible when using an Olimex dongle.
4. The window title now contains the program name and version rather than name and description.

1.0.0.2

1. Added the **verify** button to compare a file with the targets flash and report on whether they match. The number of bytes compared is usually the number of bytes in the file. To save time 4kB blocks are read from flash and compared before advancing blocks so the operation ends as soon as a difference is found in a 4k block.

1.0.0.3

1. Retains settings, in particular the range-start and size so that the application can be provided with defaults appropriate to a recipient, and users can restart with previously used settings.

2. Renamed Reset button to Clear button so as not to confuse with nSRST, nTRST etc. See also mods 3,10.

3. Changed tooltip text for the Read button; replaced 'Get-ID' with 'Clear' button reference.

4. Internal detail: Changed the size of blocks erased to 1kB regardless of the reported erase block size that is *unreliably estimated* from the flash size. This ensures erase will always work at the possible cost of wasting a small amount of time for big memories where erase blocks are bigger than 1kB. A further advantage is that the maximum attainable erase-map resolution is achieved at all times with a simpler method.

5. Stopped the Running/Halted status appearing to change from Halted to Running during Read/Erase/Save/Verify procedures. It was a result of the status being derived from the Go buttons enable state when *both* halt and go were disabled while busy.

6. Added **blank check** facility and added blank-check documentation to this document.

7. Now that mod 4 is done and the estimated erase-block size is only reported, the reporting of the estimate has been removed too. Consequently all code to with estimation and reporting of erase block size has been removed.

8. Renamed the 'Program..' button from 'Open..'. This has the added advantage of distinguishing it from the 'Open' port button.

9. Tightened up validation of hex start addresses given in the read and flash boxes - at the key-pressed event level.

10. Changed ref to 'Clear' button from Get-Id button in 2 warning messages about sticky errors on accessing illegal addresses.

11. Ensure sticky error is cleared – and an illegal address accessed warning is displayed when the Read button accesses an illegal address.

12. Simplified the 'illegal address accessed' warning from the verbose rubbish that was previously issued.

13. Added **Reboot-Halts** checkbox for advanced programming when JTAG used for general i/o.

14. The Scan button now automatically activates the Clear button function because the reboot-halts checkbox won't work if there is a sticky error – which Clear resolves. This has allowed the Clear button to be removed from the preparation procedure step after the scan button. In this light the Clear button has been moved rightwards since it is no longer one of the essential buttons [on the left]. See rationale.

1.0.0.4

1. Source code; removed de-activated code. Documented FTCJTAG.DLL calls from AN_110 adding copyright notice for FTDI.

2. Documentation; added Upgrade section.

3. Added “How it works” to help/abort menu activated from button in status bar at foot of window.

4. Caught the misleading “dll not found exception” in order to report that the underlying FTDI driver maybe absent [it is provided with the JTAG device e.g. from Amontec/Olimex].

1.0.0.5

1. Updated “How it Works” (internals.pdf) to include an overview diagram and references table at the start in a section entitled “What are we talking about?”

2. Added an icon to the main form and the project which depicts a lightening flash, which is *iconic* [sigh].

3. Upgraded the ftcjtag.dll from 1.9 to 2.0. For 2.0 there are 2 dll files; one for 32bit and the other ftcjtag64.dll for 64 bit windows. The application attempts to discover the o/s [64 or 32bit] and use the appropriate dll. The upgrade has been tested on a 32bit system, but has yet to be tested on a 64bit system. It is understood that the 2.0 upgrade operates with the latest and fastest ft2232 compatible devices. Owing to no hi-speed hardware implementations at hand, no changes to the calling code been made to take advantage of this hi-speed just yet, favouring consistent reliability, but there will be less to do should the need arise.

4. Changed the publish manifest to create a desktop shortcut, since requested by a user.

5. FtcJtag**64** .dll v2.0 is displayed in the status bar at the foot of the window if a 64bit version of Windows is detected – and the 64bit dll is used, otherwise FtcJtag.dll v2.0 is displayed and used.

6. For portability onto 64bit systems; uses system.threading.thread.sleep(ms) instead of kernel32.dll sleep(ms) when programming flash and rebooting target for 200ms. Especially carefully tested.

1.0.0.6

1. Added ‘Auto-Start’ checkbox to automatically run the preparation procedure [it clicks buttons Open Scan, AP ID, Halt, Kb] the next time the application starts. Obviously the state of the checkbox *is* remembered between runs.

2. Accessibility hot keys added. For example; you can select the ‘Blank check’ button with Alt+B. The hot key is underlined in the button text when the Alt button is pressed. It seems to be the prerogative of .Net to makes those underlines disappear when other parts of the form are greyed/ungreyed. Trivial buttons are not hot keyed so that they are not accidentally selected.

3. The addresses of the registers accessed by buttons DHCSR, kB, SR are displayed in those buttons tooltips.

4. The 2 buttons ‘Ap Status’ and ‘Dp Status’ are renamed from ‘Status’.

5. The tooltip for the scan box improved to explain more precisely what FFFFFFFF signifies. i.e. no target connected [or target not powered up].

6. Moved the 'Blank check' button upward so next to – and before - 'Erase'.

7. Changed the menu layout reached from the widget next to the bar-graph so that only Help and Abort are displayed rather than Help, Abort, How-it-works. Under Help we now have User-Guide, How-it-works. Further external online help may be accessed here in later versions without swamping the root menu where the Abort option needs to be identified quickly.

8. Aborted Blank-check and Verify no longer display a message box reporting success or failure so far.

1.0.0.7

1. Added Read-protect checkbox.

2. Removed [hid] the SR [status register] button and value from the flash box to make way for the Read-protect checkbox.

1.0.0.8

1. Internal rationalisation.

   (a) Simplified code that turns on/off read protection. The aim is to remove unnecessary 'just-to-be-safe' code to avoid misleading the maintainer as to what is necessary or not.

   (b) Re-wrote function that tests if target flash is currently read-protected. The aim is to access the FLASH_OBR [option byte register (RO status)] rather than inspecting option-byte0 itself [@0x1fffF800 in the information block for a value of A5 hex] because the latter – being flash - also becomes unreadable when read-protection is applied whereas OBR is a proper peripheral register that's always readable. However even proper registers fail to read [reading as 0] when there is a prior 'sticky' error, so the function now *also* clears any sticky error at the start *before* reading OBR.

2. Shortened the message displayed when the read protection is changed. The aim is to show two main things clearly. (a) What has been done [read protected or not], and (b) to make it clear the user needs to MANUALLY apply a power-on-reset [by repowering or shorting nSRST to 0Volts].

3. Added troubleshooting points 1 to 7 to this document.

4. Added trivial drag and drop to potentially speed-up **program** and **verify** operations.

   Drop a binary file (*.bin) onto the **verify** button to check that file matches the flash range specified.

   Drop a .bin file anywhere on the application window [except the verify button] in order to **program** that file to the flash range already specified.

   In both cases the connection to the target must be opened via the Open and Scan buttons. Read-protect must be off because the write procedure needs to do some confidence checks to ensure that the operation is likely to work.

5. The read box is no longer disabled/greyed when the read-protect is ticked. This is because;

   (a) It is re-assuring for users to confirm for themselves that flash-read attempts simply will not work.

   (b) Some registers and RAM can still be read anyway because flash protection applies to flash whereas the Read button applies to all addressable space.

1.0.0.9

1. Added a **balloon** tool-tip at start-up to show what to click for help. Added another balloon tooltip displayed after scan clicked which informs that a file can be dragged onto the form (if halted) to program it to flash range specified, or onto the verify button (if not halted) to verify it against the flash range.

2. The Scan Chain box **identifies STR71x** TAPs [in addition to ARM Debug TAPs], but because they [STR71x's] are; older/obsolete, the public debug interface doc is lacking, and they are so radically different from the STM32F10x, the STR71x is still not supported and reported as such. Shame.

1.0.0.10

1. Internationalization: Languages supported: English, **French, Spanish**. Translations were performed and checked using Google translate so that the reverse translations made the same (non;)sense as the original English. **Message box titles** were changed to consistently show the name of the application.

1.0.0.11

1. **Polish** translation performed and checked using Google translate so that the reverse translations made the same (non;)sense as the original English.