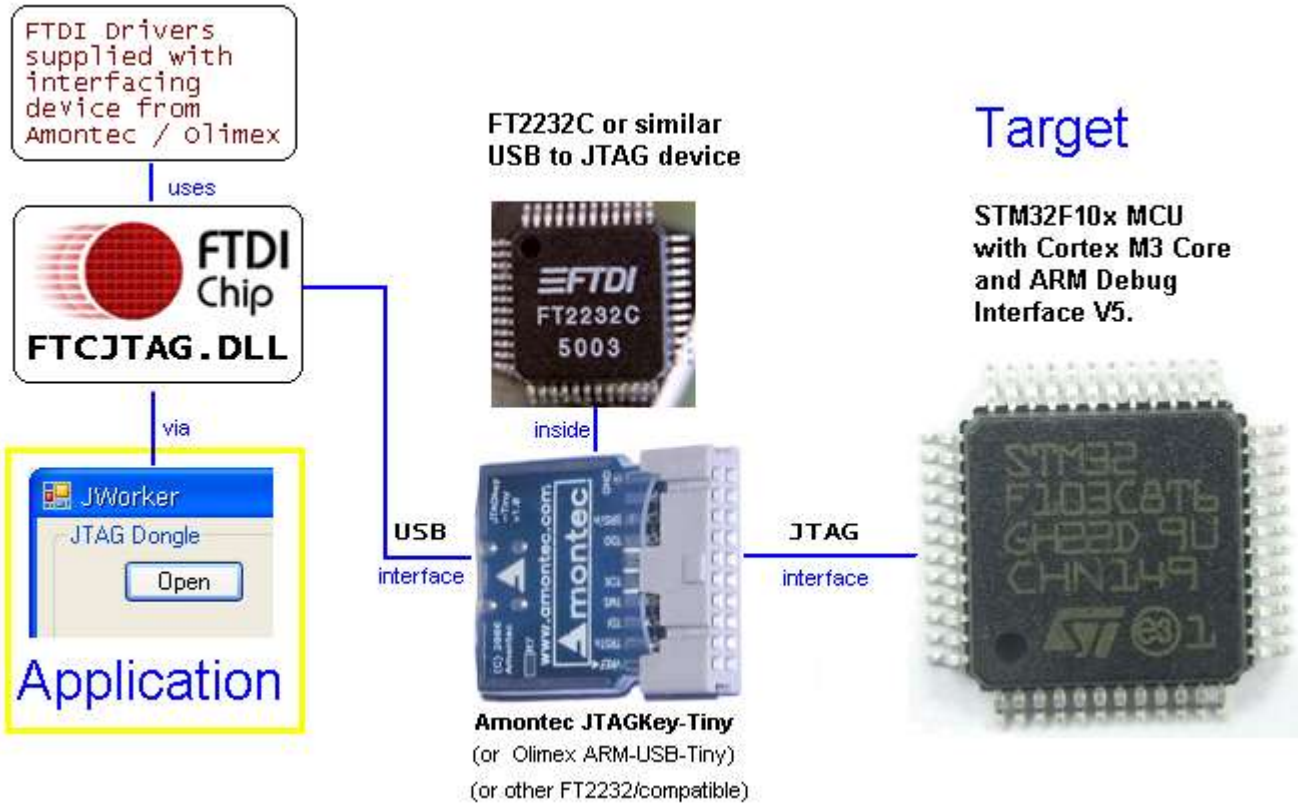
# *Jworker – How it works*

## Context

Jworker is a Windows [XP] application that communicates with a target STM32F10x MCU via its JTAG interface. Its main aim is to *easily* program the targets flash memory.

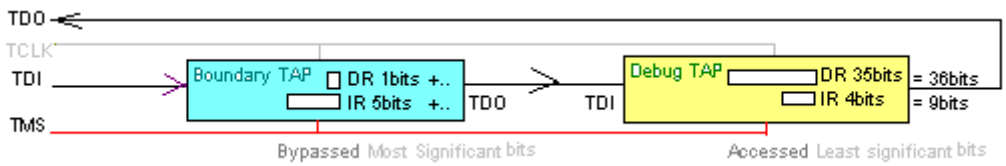
Jworker may be useful to those with low cost Amontec JTAGKey-Tiny or Olimex ARM-USB-Tiny USB-to-JTAG devices, containing FT2232C[-compatible] devices, connected between a Windows computer and an STM32F10x target device.




| Implementation References | File properties | | Author |
|---|---|---|---|
| 1 Microsoft Visual Basic 2010 Express. ISO link. Download. | Installed from an | ISO | Microsoft |
| 2 FTCJTAG.DLL Zip file. Download. | 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 |

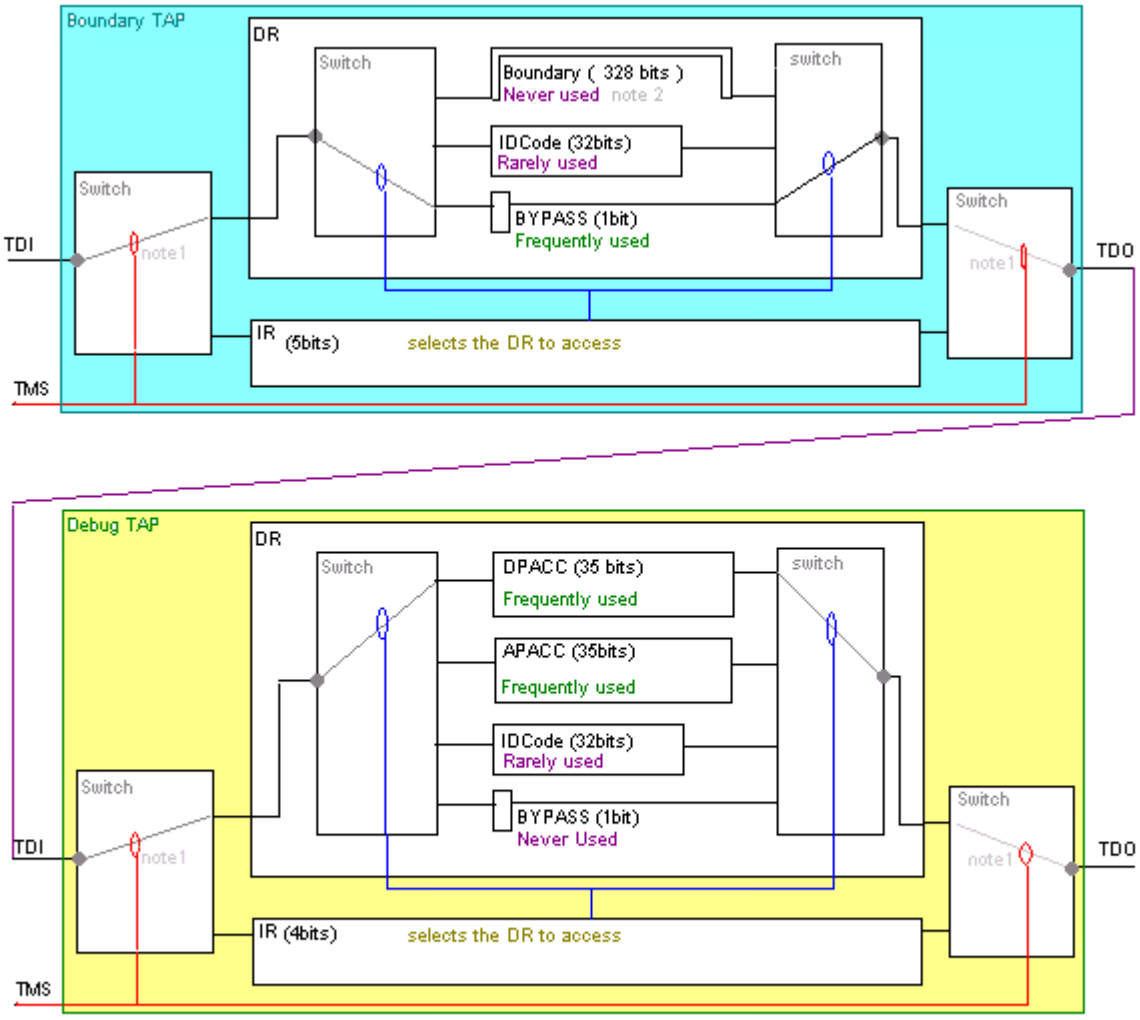
## Basic Operation

The STM32 F10x contains 2 Test-Access-Ports(TAPs) in the 'scan chain' shown below.




For Both TAPs an instruction register(IR) or Data register(DR) is switched between TDI and TDO by a boolean parameter of the JTAG_Read/write functions in FtcJTAG.Dll which in turn sends a command on TMS

to set the IR/DR-switch. Since TMS goes to both TAPs, both TAPs switches move together and are always in the IR or DR position at the same time. This means that we consider the combined IRs, or the combined DRs of both TAPs as if they were one long TAP - with the Debug TAPs register at the least significant end.




note1. selected by the boolean "bInstructionTestData" parameter of the JTAG_read and write functions in FTCJTAG.DLL as documented by FTDI's AN_110_Programmers_Guide_for_High_Speed_FTCJTAG.DLL. version 1.2

note2. the length of the boundary scan was derived from a file called STM32F1_High_density_LQFP100.bsd downloaded from ST. Packages with less pins probably have a shourter boundary register (less than 328 bits). Not that it matters here since we bypass it.

The **IR** for a Boundary TAP is always 5 bits, and always 4 bits for the Debug TAP, the 5+4 bits are combined into 5+4=9bits between TDI and TDO. So FTCJTAG.DLL read and write functions will clock in/out the combined IR between TDI/TDO by calling a JTAG_write function with the 'bInstructionTestData' parameter set to **True**. The values inside the IR pair switch a particular DR pair between TDI and TDO.

For **DR**s, we call the JTAG_Read/Write routines with the same parameter **False,** and consider the combined DRs of both TAPS as one long DR. DRs might have 1+35=36 data bits clocked in via TDI - and out through TDO. 1bit for the bypassed boundary TAP, 35bits for the Debug TAP – provided that its IR previously selected was DPACC (1010 binary) or APACC (1011) which are the most frequently access data registers.

## Bypassing the boundary TAP

BYPASS *really* means the DR for a TAP will be a single bit of value 0 so that it can be easily ignored, especially at the most significant end of a DR pair composed of 1 bypass bit and 35 data bits for the boundary and debug TAPs respectively.

We don't use the Boundary TAP. The 1st IR of all IR pairs sent is always the 5bit BYPASS instruction (11111 binary) which means the 1st DR of all DR pairs is always a 1bit of value of 0 – easily ignored.

## Accessing the debug TAP

The debug TAP is used. It accesses everything addressable with only IR codes of DPACC (1010 binary) and APACC (1011) in the 2nd IR of the IR pair.

## Example - Accessing a Target Address

To read a 32bit word from address 0x0800:0000 of value known to be 20005000 hex.

| Step | IR/DR | Write to TDI + | + | + | | Read from TDO | | | Comment |
|---|---|---|---|---|---|---|---|---|---|
| 1 | IR<br>9bits | Bypass<br>11111 binary | DPACC<br>1010 binary | | | | | | Access the top level debug port |
| 2 | **DR**<br>**36bits** | Bypass<br>0 binary | **Data**<br>**00**000000 hex | APSelect<br>**10** binary | Write<br>0 binary | | | | Select **port0**<br>**bank0** |
| 3 | IR<br>9bits | Bypass<br>11111 binary | APACC<br>1011 binary | | | | | | Allow details to be specified next |
| 4 | **DR**<br>**36bits** | Bypass<br>0 binary | Data CSW<br>A3000022 hex | CSW\|<br>00 binary | Write<br>0 binary | | | | select 32bit data size |
| 5 | **DR**<br>**36bits** | Bypass<br>0 binary | <address><br>**08000000** hex | TAR<br>01 binary | Write<br>0 binary | | | | Set the address to access |
| 6 | **DR**<br>**36bits** | Bypass<br>0 binary | Data. don't care<br>00000000 hex | DRW<br>11 binary | Read<br>1 binary | | | | Dummy read |
| 7 | **DR**<br>**36bits** | Bypass<br>0 binary | Data. don't care<br>00000000 hex | DRW<br>11 binary | Read<br>1 binary | Bypass<br>0 binary | Data sought<br>**20005000** hex | Ack code<br>**010** binary | Access the value at the address |

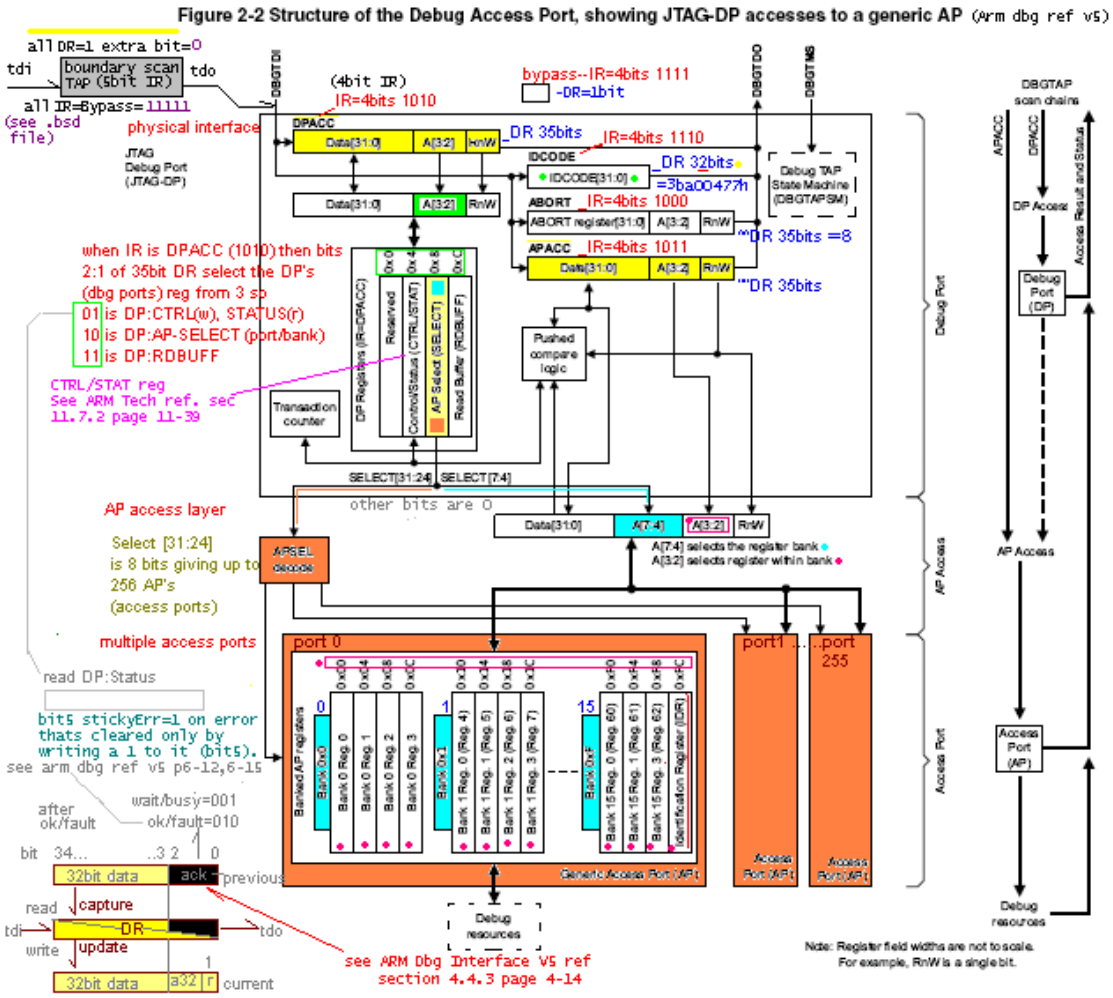
Explanation. Refer also to the colour coded diagram ahead.

There are 2 main parts.

Steps 1&2 above pre-select port0 and bank0 because it contains 3 registers that specify the address to access (TAR), the size of the accessed word[s] (CSW), and of course the data at the address (DRW).

Steps 3-7 populate the 3 registers in that bank to specify the address, its word-size, and whether to write or read the value at the address.

This diagram ahead is figure 2-2 in the ARM Debug Interface V5. With colour coded annotations overlaid. The colour coding will obviously not be helpful on a monochrome hardcopy.




## Accessing flash.

Flash addresses are read like any other. However flash addresses are written and erased through the FPEC (flash programming and erase controller). Flash protection must be off to write to it.

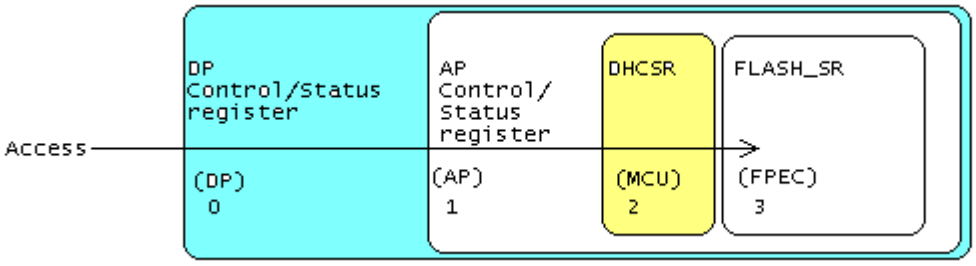
Essentially therefore the Jworker mirrors the flow-charted ST procedures in the **PM0075 Programming manual** except that each address in the FPEC is accessed in the manner described above, and unnecessary re-initializations and unnecessary flag re-assertions have been removed to speed up the procedures commanded through the relatively slow JTAG interface.

Because flash is erased separately from programming JWorker can get away with erasing flash in 1k blocks rather than establishing what the devices erase block size is. If a devices erase blocks are actually 2k bytes then 2k blocks are erased twice as the start address of each 1k is encountered. The penalty is low because erasing an already erased block is much faster and may not affect endurance.

## Conclusion

Using FTCJTAG.dll you do need to *aware* of the “TAP Controller State Table Diagram” only to know there are two basic paths, to set IR or DR, and only one of the six stable resting states is used here– which is the “Run-Test-Idle” standby state.

There are four levels of status registers. **Level1**: DPACC-control/status register, **level2**: APACC-control/Status register, **level3**: DHCSR(debug-halting-control/status register), **level4**: FLASH_SR inside the FPEC. The last two are just addresses inside the AP (Access port), but since DHCSR can be accessed under reset, and must have halted the core before changing flash; it’s arguably a higher level than the FLASH_SR. It is particularly easy to confuse the documentation for the DPACC and APACC control/status registers since they have the same name and title.




# Other Information

## Translation in VB

References

Quick guide: http://code.msdn.microsoft.com/windowsdesktop/VBWinFormLocalization-966546b3#content
Language codes: http://msdn.microsoft.com/en-us/library/ee825488(v=cs.20).aspx
Google translate: http://translate.google.co.uk/

### Method

a) In the main form designer select the forms surface and in its properties window seek the ones starting with **L** i.e. Language and Localizable.
b) Set Localizable to True.
c) Set the forms language property as required; e.g. French. The language is initially 'default'.
d) Re-type all controls text properties in French.
Take extra care to set the language property **BEFORE** re-typing all the texts in another language.
When the application is installed on a French computer the application shows the French text. When installed on an English speaking computer the default English text is shown.
e) In the main forms CODE window, in Sub New(), before the call to InitializeComponent() add the following:

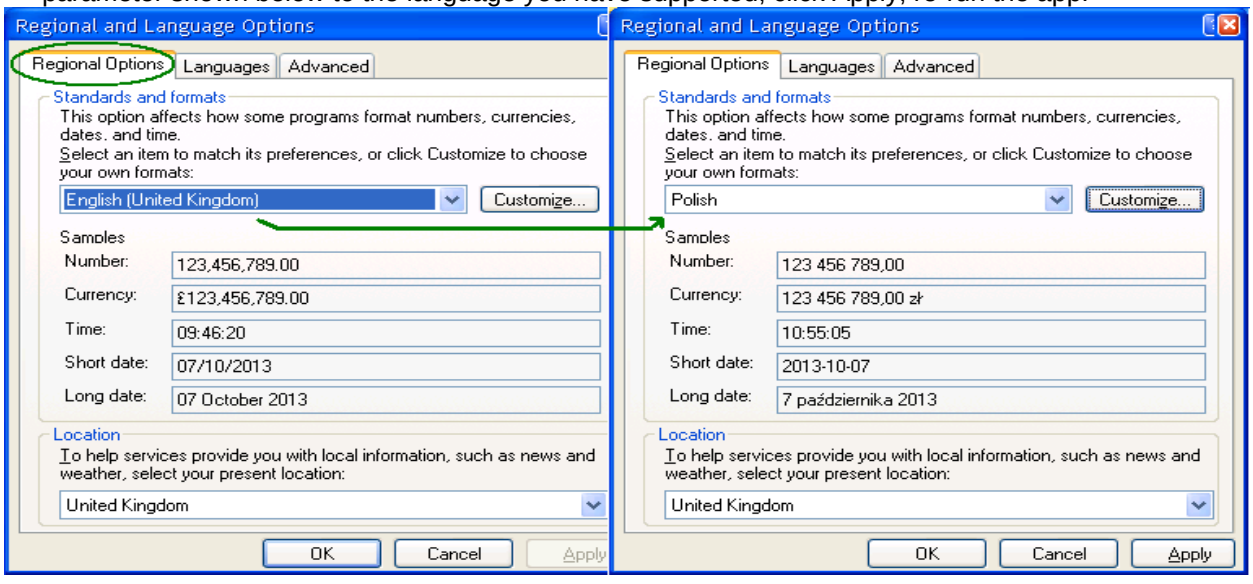
```
With Threading.Thread.CurrentThread
 .CurrentUICulture = .CurrentCulture 'source resource culture from system culture
End With
```

Tip

To translate a message-box text one might place a textbox on an unseen part of the form and source the message box text from it. This way you provide an alternate translation for the textbox as for the other controls so that its automatic language selection extends to the message box.

### Testing

From windows Control panel → Regional and Language Options → 'Regional Options[tab]; change the parameter shown below to the language you have supported, click Apply, re-run the app.



The CurrentCulture is sourced from this parameter rather than a parameter in the languages tab. A busy system may take some time to apply the new settings.

## Origin

Author: Bob Seabrook bob@seabrooks.plus.com www.seabrooks.plus.com
Specific: http://www.seabrooks.plus.com/jworker/current-download