

**inDART<sup>®</sup> -STX**  
**In-Circuit**  
**Debugger/Programmer**  
**for STMicroelectronics**  
**ST72F Family FLASH**  
**Devices**

**User's Manual Addendum**  
**(v. 3.3)**



Copyright © 2006 SofTec Microsystems®

DC01065

## **SofTec Microsystems**

E-mail (general information): [info@softecmicro.com](mailto:info@softecmicro.com)

E-mail (marketing department): [marketing@softecmicro.com](mailto:marketing@softecmicro.com)

E-mail (technical support): [support@softecmicro.com](mailto:support@softecmicro.com)

Web: <http://www.softecmicro.com>

---

### **Important**

SofTec Microsystems reserves the right to make improvements to the inDART® Series In-Circuit Debuggers, their documentation and software routines, without notice. Information in this manual is intended to be accurate and reliable. However, SofTec Microsystems assumes no responsibility for its use; nor for any infringements of rights of third parties which may result from its use.

**SOFTEC MICROSYSTEMS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.**

### **Trademarks**

inDART is a trademark of SofTec Microsystems.

ST is a trademark of STMicroelectronics.

Microsoft and Windows are trademarks or registered trademarks of Microsoft Corporation.

PC is a registered trademark of International Business Machines Corporation.

Other products and company names listed are trademarks or trade names of their respective companies.

# Contents

<b>Contents</b>	<b>3</b>
<b>1. XFlash Devices (without On-Chip Debug Module) Specific Topics</b>	<b>5</b>
Introduction	5
Limitations	6
Breakpoint Notes	6
Program Execution Notes	7
<i>Real-Time Execution</i>	9
<i>Step Mode Execution</i>	9
<i>ST7 Product Identifier</i>	10
Online Assembler Notes	10
Stop Execution Notes	10
Option Bytes	11
Troubleshooting	12
<i>LITE0- and ST7FLITES-Specific: Communication can't be established with inDART-STX</i>	12
<i>When debugging, the program runs too slowly</i>	12
<i>The program execution stops at the beginning of user's code</i>	12
<i>The program seems not to work correctly</i>	13
<i>Interrupt handling routines are not executed</i>	13
<b>2. XFlash Devices (with On-Chip Debug Module) Specific Topics</b>	<b>15</b>
Introduction	15
Limitations	15
Breakpoint Notes	16
<i>Instruction Breakpoints and Advanced Breakpoints</i>	16
<i>Advanced Breakpoints Limitations</i>	18
ST7 Product Identifier	19
Online Assembler Notes	20
Stop Execution Notes	20
Option Bytes	20
Troubleshooting	21
<i>When debugging, the program runs too slowly</i>	21

<i>The program execution stops at an unexpected location</i>	21
<i>The program execution stops at the beginning of user's code</i>	21
<b>3. HDFlash Devices (without On-Chip Debug Module) Specific Topics</b>	<b>23</b>
Introduction	23
Limitations	24
Program Execution Notes	25
<i>Real-Time Execution</i>	26
<i>Step Mode Execution</i>	26
Online Assembler Notes	27
Stop Execution Notes	27
Option Bytes	28
Troubleshooting	28
<i>When debugging, the program runs too slowly</i>	28
<i>The "Stop Program" command doesn't work</i>	29
<i>The program execution stops at the beginning of user's code</i>	29
<i>The program seems not to work correctly</i>	29
<i>Interrupt handling routines are not executed</i>	30
<b>4. HDFlash Devices (with On-Chip Debug Module) Specific Topics</b>	<b>31</b>
Introduction	31
Limitations	31
Breakpoint Notes	32
<i>Instruction Breakpoints and Advanced Breakpoints</i>	32
Online Assembler Notes	34
Stop Execution Notes	35
Option Bytes	35
Troubleshooting	35
<i>When debugging, the program runs too slowly</i>	35
<i>The program execution stops at an unexpected location</i>	36
<i>The program execution stops at the beginning of user's code</i>	36
<b>Appendix A: ICC Mode Entry Details</b>	<b>37</b>
<b>Appendix B: RC Calibration Notes</b>	<b>39</b>

# 1. XFlash Devices (without On-Chip Debug Module) Specific Topics

## Introduction

This chapter details some specific issues that must be known when working with the following devices:

- ST72F260
- ST72F262
- ST72F264
- ST7FLITE02
- ST7FLITE05
- ST7FLITE09
- ST7FLITES2
- ST7FLITES5

These devices are based on XFlash technology (EEPROM-based, byte per byte reprogrammable, byte per byte erasable) and do not feature an on-chip Debug Module. Debugging capabilities are therefore added through a small program (monitor) which is automatically and transparently added to the user code and programmed into the target microcontroller. The monitor code has been developed by SofTec Microsystems.

**Note:** *the behaviour and limitations of inDART-STX described in this chapter only refer to debugging sessions dedicated to the above listed devices, and do not hold for other target devices.*

## Limitations

Some target on-chip resources are wasted for debugging purposes. In particular:

- 7 stack levels (bytes) are wasted.
- 180 bytes are reserved for the monitor (from address FF28H to address FFDCH).
- ICCDATA and ICCCLK lines are reserved for programming and in-circuit debugging. Therefore, in the user application software, associated register bits in the port registers must always be masked to avoid conflict with the debugger.
- The TRAP instruction and the TRAP interrupt vector are reserved for the monitor.
- Peripherals run even in STOP mode.

## Breakpoint Notes

inDART-STX can handle an unlimited number of breakpoints within program memory. When you set a breakpoint on a source code line, inDART-STX automatically (and transparently) replaces the opcode of the instruction where the breakpoint is set with the opcode of the TRAP instruction (this explains why the TRAP instruction—and the TRAP vector—is reserved and should not be used in user applications).

The difference between inDART-STX and a standard emulator is that the latter can set/reset breakpoints under all conditions, while inDART-STX (since it needs to change the opcode of the instruction where the breakpoint is set with the TRAP instruction) needs to program the FLASH memory of the target microcontroller to perform this operation. To program the FLASH memory, inDART-STX needs to enter the ISP mode and, consequently, to reset the target microcontroller. inDART-STX actually programs breakpoints only when an execution command that resets the target microcontroller (e.g., the “**Run**”

command) is issued. In all of the other cases breakpoint information is stored inside the host PC, program execution switches to “step mode” (see below) and breakpoint programming occurs next time an execution command that resets the target microcontroller is issued.

## Program Execution Notes

inDART-STX executes in real-time mode or in step mode.

- Step mode execution occurs when you launch the emulation, the emulation stops due to a breakpoint, then you set/modify breakpoints, and then continue the emulation; step mode also occurs when executing step commands.
- Real-time execution occurs in all other cases.

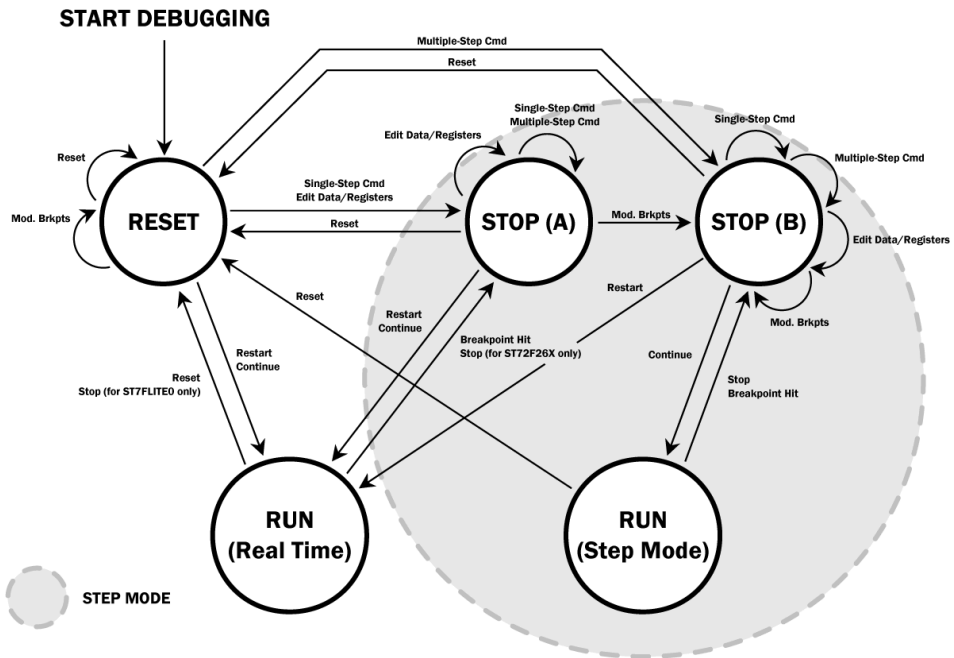
The following table summarizes (for the most common emulation commands) which command causes inDART-STX to execute in real-time mode or in step mode.

Emulation Command	Execution Mode
Run	Real-time
Restart	Real-time
Continue	Real-time if no breakpoints have been set/modified since a breakpoint has been encountered, step mode otherwise
Step Into	Step mode
Step Over	Step mode
Step Out	Step mode
Run to Cursor	Step mode

Execution Mode Summary

The following FSM (Finite State Machine) diagram illustrates in detail the behaviour of inDART-STX. In the diagram, each state transition is caused by one or more events, explained below.

- **Reset.** The Reset event is caused by the “**Chip Reset**” command, an external Reset event, or a peripheral Reset event.
- **Restart.** Include either the “**Run**” command or the “**Restart**” command.
- **Stop.** The Stop event is caused by the “**Stop Program**” command.
- **Breakpoint Hit.** This event is caused by a breakpoint hit.
- **Continue.** Is caused by the “**Continue**” command.
- **Single-Step Command.** Is caused by the “**Step Into**” and “**Step Into ASM**” commands.
- **Multiple-Step Command.** One of the following commands: “**Step Out**”, “**Step Over**”, “**Step Over ASM**”, “**Run To Cursor**”. Please note that, in some cases, the “**Step Over**” and “**Step Over ASM**” commands have the same effect of the “**Step Into**” and “**Step Into ASM**” commands, respectively, depending on whether there is a function call to be executed or not.
- **Modify Breakpoints.** One of the following commands: “**Insert/Remove Breakpoint**”, “**Enable/Disable Breakpoint**”, “**Remove All Breakpoints**”.
- **Edit Data/Registers.** One of the following events: memory editing, registers editing, watch variables editing, peripherals editing, Program Counter editing.



inDART-STX Execution Modes

**Note:** user application *RESET* is not guaranteed to work under all circumstances.

### Real-Time Execution

When inDART-STX runs in real time, instructions are executed just as the microcontroller would without the debugger. inDART-STX executes in real time until a breakpoint is encountered. Subsequent “**Continue**” commands are still performed in real time only if you do not set/modify breakpoints.

### Step Mode Execution

In step mode, program execution is supervised by the host PC and performed (automatically) step by step.

**Note:** *in step mode, peripherals are still truly handled by the target microcontroller. However, interrupts are not handled (that is, interrupt events never occur). This affects execution of instructions like `WFI` and `HALT`—since these instructions require interrupts to work, in step mode their behaviour is different than in real-time mode. In particular, the `WFI` instruction loops forever on itself (no interrupts ever occur), and after executing an `HALT` instruction, program execution cannot continue (same reason).*

**Note:** *in step mode, program execution speed is affected by the host PC speed—in particular, in step mode, program execution is slower than real-time program execution. This can affect the program execution speed when, for example, commands such as “**Step Over**”, “**Step Out**” or “**Run to Cursor**” are executed, since it can take a long time to perform the action (especially when the code contains loops to be executed many times).*

### ST7 Product Identifier

The first execution command occurring after the Reset state causes the first two RAM locations to be loaded with an ST7 Product Identifier. This code is device-specific: the first three nibbles of this identifier give the ST product code, while the fourth nibble gives the System Memory revision for the device.

### Online Assembler Notes

During a debugging sessions, every time assembly instructions are modified in the *Assembly* window (through the Online Assembler), a reset condition occurs (even though the Program Counter arrow still points to the same instruction). Subsequent debugging commands will be therefore executed from a reset state.

### Stop Execution Notes

During debugging, user program execution will stop if one of the following conditions occur:

1. A breakpoint has been encountered (or a `TRAP` instruction has been executed);
2. The microcontroller RESET line has been driven active;
3. A microcontroller internal RESET state has been triggered (by, for example, a Watchdog event, or an internal low-voltage detection);
4. The **“Stop Program”** emulation command has been issued from the inDART-STX user interface.

**Note (ST7FLITE0- and ST7FLITES-specific):** *when in real-time emulation, the “Stop Program” command causes a chip reset.*

## Option Bytes

Particular attention must be paid in correctly setting each of the Option Bytes parameters. Improper settings (and improper setting combinations) may cause the target microcontroller not to work correctly (or not to work at all). For example, “PLLx2 Enabled” programmed with a 16 MHz resonator on the application or “External Source” programmed with a resonator on the application.

If the Option Bytes are programmed with a wrong combination, you will no longer be able to access the device, so you must set the “ICC Entry Mode” parameter to “Ignore Option Bytes” and then, if the application board hasn’t an oscillator or a logic clock, the `OSC_CLK` pin from the ISP connector must be connected to the application board.

For detailed information on the meaning of the “ICC Entry Mode” parameter please refer to the “Appendix A: ICC Entry Mode Details”.

**Note (ST7FLITE0- and ST7FLITES-specific):** *when debugging, the “WDG\_SW (Watchdog Activation)” bit of the Option Bytes must be set to “Software”. After setting this bit to “Hardware”, setting it back to “Software” only succeeds if the “ICC Mode Entry” parameter is set to “Ignore Option Bytes”.*

# 1

## Troubleshooting

This section reports some device-specific problems that may arise when working with the target devices covered in this chapter. Please refer to the inDART-STX user's manual for general troubleshooting hints.

### LITE0- and ST7FLITES-Specific: Communication can't be established with inDART-STX

If the "WDG\_SW (Watchdog Activation)" bit of the Option Bytes has been set to "Hardware" and the "ICC Mode Entry" parameter is set to "Use Option Bytes", you may experience communication problem. In this case, set the "ICC Mode Entry" parameter to "Ignore Option Bytes", and make sure to provide an external clock source to the target microcontroller.

### When debugging, the program runs too slowly

This problem can have two causes:

- If not correctly set (via Option Bytes), the target microcontroller may run at a frequency lower than that specified. In this conditions, the low working frequency affects the ICC communication and, therefore, the inDART-STX user interface performances.
- Under some conditions, program execution is not performed in real time. As a consequence, program execution may slow down.

### The program execution stops at the beginning of user's code

A Reset condition occurred. This can be due to an external Reset condition (microcontroller's RESET line driven low) or an internal Reset condition (e.g., due to a Watchdog event). For more information on causes that can trigger a Reset condition, please refer to the specific ST7 microcontroller device data sheet.

**Note (ST7FLITE0- and ST7FLITES-specific):** *when executing in real-time, a “**Stop Program**” command causes a Reset condition.*

### The program seems not to work correctly

When debugging your program, some target microcontroller's resources are reserved for debugging purposes. In particular, 180 bytes are reserved for the monitor (from address FF28H to address FFDCH). Therefore, you must pay attention that your program doesn't use this range of memory locations. Also, please remember that 7 stack levels are reserved as well.

### Interrupt handling routines are not executed

Under some conditions, program execution is not performed in real time. As a consequence, interrupts are not handled.



## 2. XFlash Devices (with On-Chip Debug Module) Specific Topics

# 2

### Introduction

This chapter details some specific issues that must be known when working with the following devices:

- ST7FDALI
- ST7FLIT1xB
- ST7FLITE10, 15, 19
- ST7FLITE20, 25, 29
- ST7FLITE30, 35, 39
- ST7FLITEBC

These devices are based on XFlash technology (EEPROM-based, byte per byte reprogrammable, byte per byte erasable), and do include an on-chip Debug Module which allows for basic debugging capabilities (breakpoint handling, start/stop user program, etc.).

**Note:** *the behaviour and limitations of inDART-STX described in this chapter refer to debugging sessions dedicated to the above mentioned devices, and do not hold for other target devices.*

### Limitations

Some target on-chip resources are wasted for debugging purposes. In particular:

- 5 stack levels (bytes) are wasted.

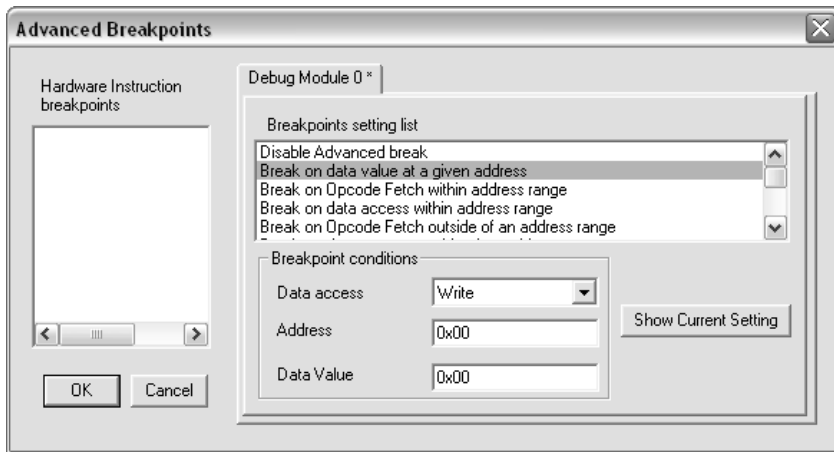
- ICCDATA and ICCCLK lines are reserved for programming and in-circuit debugging.
- The TRAP instruction and the TRAP interrupt vector are reserved for the on-chip Debug Module.
- The DM peripheral (the on-chip Debug Module) is reserved (DM registers must not be written to by the user application; if read from, the data returned is undefined).
- Peripherals run even in STOP mode.
- The “WDG SW” bit in the Option Bytes must be set to “software watchdog”.

### Breakpoint Notes

inDART-STX can handle a maximum of two breakpoints within program memory. Since these two breakpoints are handled by the on-chip Debug Module, inDART-STX always executes in real time.

#### Instruction Breakpoints and Advanced Breakpoints

When working with devices with on-chip Debug Module, advanced breakpoints are available. Advanced breakpoints differ from instruction breakpoints (that is, standard breakpoints) in that they can stop program execution depending on complex situations. Advanced breakpoints allow you, for example, to stop execution when a specific value is written to a specific address, or when two specified opcodes are fetched sequentially. To set advanced breakpoints, choose **Emulator > Advanced Break** from the main menu (a debugging session must have been started). The following dialog box will appear.



The *Advanced Breakpoints* Dialog Box

2

When using breakpoints (either instruction breakpoints or advanced breakpoints), be aware of the following:

- You can use advanced breakpoints or instruction breakpoints, but not a combination of the two;
- As soon as a complex breakpoint is set, previously set instruction breakpoints are automatically disabled—and vice versa;
- Step commands (Step Into, Step Over, Step Out, Run to Cursor) waste one instruction breakpoints. When issuing a step command, all previously set instruction or advanced breakpoints are disabled, the step command is executed, and then previously set instruction or advanced breakpoints are re-enabled.

**Note:** *if an advanced breakpoint is set which specifies anything but an instruction fetch, then program execution stops **after** the instruction during which the breakpoint condition is met is executed.*

The following table summarizes the available advanced breakpoints.

Breakpoint Type	Required Parameters
Break on data value at a given address	The type of data access (write, read, read/write); The address; The data value.
Break on opcode fetch within address range	The address at the beginning of the range; The address at the end of the range.
Break on data access within address range	The type of data access (write, read, read/write); The address at the beginning of the range; The address at the end of the range.
Break on opcode fetch outside of an address range	The address at the beginning of the range; The address at the end of the range.
Break on data access outside of an address range	The type of data access (write, read, read/write); The address at the beginning of the range; The address at the end of the range.
Break on two sequential opcode fetches	The first opcode fetch; The second opcode fetch.
Break on data access at one of two addresses	The type of data access (write, read, read/write); The first of the two addresses; The second of the two addresses.
Break on opcode fetch or data access at given addresses	The type of data access (write, read, read/write); The address where the opcode fetch must occur; The address where the data access must occur.
Break on conditional stack write or opcode fetch at given address	The maximum stack address below which the write operation must occur; The address where the opcode fetch must occur.
Break on conditional stack write or data access at given address	The type of data access (write, read, read/write); The address where the data access must occur; The maximum stack address below which the write operation must occur.

Advanced Breakpoints Summary

### Advanced Breakpoints Limitations

The Advanced Breakpoint dialog box allows you to configure breakpoints with a large range of possibilities and combinations. However, due to current silicon limitations, some restrictions apply.

- Never set a breakpoint on data value/access with the data access type set to Read or Read/Write, as an unexpected break may occur (the user application could be stopped at a location different from the one specified in the breakpoint settings). All the different types of breakpoint on data are concerned:
  - Break on data value at a given address;
  - Break on data access within address range;
  - Break on data access outside of an address range;
  - Break on data access at one of two addresses;
  - Break on conditional stack write or data access at given address.

Workaround: use instruction breakpoints instead.

Note: setting a breakpoint on data value/access with the data access type set to Write does not lead to any unexpected break.

- Never set a breakpoint on data value/access within the stack area or just after the stack area with the data access type set to Read or Read/Write, as an unexpected break may occur (the user application could be stopped at a location different from the one specified in the breakpoint settings). All the different types of breakpoint on data (as above) are concerned.

Workaround: check any stack overflow, use the dedicated advanced breakpoints, break on conditional stack write.

Note: setting a breakpoint on data value/access within the stack area or just after the stack area with the data access type set to Write does not lead to any unexpected break.

## ST7 Product Identifier

The first execution command occurring after the Reset state causes the first two RAM locations to be loaded with an ST7 Product Identifier. This code is device-specific: the first three nibbles of this identifier give the ST product code, while the fourth nibble gives the System Memory revision for the device.

## 2

### Online Assembler Notes

During a debugging sessions, every time assembly instructions are modified in the *Assembly* window (through the Online Assembler), a reset condition occurs (even though the Program Counter arrow still points to the same instruction). Subsequent debugging commands will be therefore executed from a reset state.

### Stop Execution Notes

During debugging, user program execution will stop if one of the following conditions occur:

1. A breakpoint has been encountered;
2. The microcontroller RESET line has been driven active;
3. A microcontroller internal RESET state has been triggered (by, for example, a Watchdog event, or an internal low-voltage detection);
4. The **“Stop Program”** emulation command has been issued from the inDART-STX user interface.

### Option Bytes

Particular attention must be paid in correctly setting each of the Option Bytes parameters. Improper settings (and improper setting combinations) may cause the target microcontroller not to work correctly (or not to work at all). For example, if “OSCRANGE” is programmed with “External Clock: CLKIN on PB4” while the actual clock source is provided on the OSC1 pin, the microprocessor will not start at all.

If the Option Bytes are programmed with a wrong combination, you will no longer be able to access the device, so you must set the “ICC Entry Mode” parameter to “Ignore Option Bytes” and then, if the application board hasn’t an oscillator or a logic clock, the OSC\_CLK pin from the ISP connector must be connected to the application board.

For detailed information on the meaning of the “ICC Entry Mode” parameter please refer to the “Appendix A: ICC Entry Mode Details”.

## Troubleshooting

This section reports some device-specific problems that may arise when working with the target devices covered in this chapter. Please refer to the inDART-STX user's manual for general troubleshooting hints.

2

### When debugging, the program runs too slowly

If not correctly set (via Option Bytes), the target microcontroller may run at a frequency lower than that specified. In this conditions, the low working frequency affects the ICC communication and, therefore, the inDART-STX user interface performances.

### The program execution stops at an unexpected location

Advanced breakpoints are enabled. When advanced breakpoints are enabled, instruction breakpoints are disabled and the program execution stops as soon as an advanced breakpoint condition is met. The stop location may appear to be “random”, but is actually consistent with what specified in the *Advanced Breakpoints* dialog box. To view advanced breakpoints active settings, choose **Emulator > Advanced Break** from the main menu.

### The program execution stops at the beginning of user's code

A Reset condition occurred. This can be due to an external Reset condition (microcontroller's RESET line driven low) or an internal Reset condition (e.g., due to a Watchdog event). For more information on causes that can trigger a Reset condition, please refer to the specific ST7 microcontroller device data sheet.



# 3. HDFlash Devices (without On-Chip Debug Module) Specific Topics

## 3

### Introduction

This chapter details some specific issues that must be known when working with the following devices:

- ST72F321
- ST72F321B
- ST72F324
- ST72F324B
- ST72F324L (low voltage)
- ST72F32A
- ST72F521
- ST72F561
- ST72F621
- ST72F622
- ST72F623
- ST72F63B
- ST72F651
- ST72F652

These devices do not feature an on-chip Debug Module, therefore debugging capabilities are added through a small program (monitor) which is automatically and transparently added to the user code and programmed into the target microcontroller. The monitor code has been developed by SofTec Microsystems.

Additionally, these devices are based on HDFlash technology (FLASH-based, programmable byte per byte, erasable by sector), which only allows for a limited

number of FLASH erase/write operations (typically <100). Each time a debugging session is started, the user code is downloaded to the device FLASH memory, and one FLASH erase/write cycle is wasted.

**Note:** *the behaviour and limitations of inDART-STX described in this chapter refer to debugging sessions dedicated to the above mentioned devices, and do not hold for other target devices.*

3

## Limitations

Some target on-chip resources are wasted for debugging purposes. In particular:

- 7 stack levels (bytes) are wasted.
- 200 bytes are reserved for the monitor (from address FF18H to address FFDFH).
- ICCDATA and ICCCLK lines are reserved for programming and in-circuit debugging. Therefore, in the user application software, associated register bits in the port registers must always be masked to avoid conflict with the debugger.
- SPI peripheral and the SPI interrupt vector are reserved, since the SPI I/O lines share the ICCDATA and ICCCLK lines (this limitation only applies to ST72F321, 324, 324L and 521).
- Global interrupts must be enabled to allow the STOP command to be implemented (this limitation only applies to ST72F321, 324, 324L and 521).
- The TRAP instruction and the TRAP interrupt vector are reserved for the monitor.
- Peripherals run even in STOP mode.

**Note:** *in order to enter the program the FLASH memory of the devices, inDART-STX supplies a 12 V DC voltage to the ICCSEL/V<sub>PP</sub> pin. You should be cautious of this high voltage, since it could damage other circuitry connected to that pin.*

## Program Execution Notes

inDART-STX executes in real-time mode or in step mode.

- Step mode execution occurs when the user program contains one or more breakpoints;
- Real-time execution occurs in all other cases.

The following table summarizes (for the most common emulation commands) which command causes inDART-STX to execute in real-time mode or in step mode.

Emulation Command	Execution Mode
Run	Real-time if no breakpoint is set, step mode otherwise
Restart	Real-time if no breakpoint is set, step mode otherwise
Continue	Real-time if no breakpoint is set, step mode otherwise
Step Into	Step mode
Step Over	Step mode
Step Out	Step mode
Run to Cursor	Step mode

Execution Mode Summary

**Note:** *user application RESET is not guaranteed to work under all circumstances.*

**Tip:** *real-time execution only occurs when no breakpoints are set. However, you can set “low-level” breakpoints (and still get real-time execution) by putting the **TRAP** instruction in your source code. When a **TRAP** instruction is executed, the program stops just as if a breakpoint was set. The advantage of using this technique is that you can have as many **TRAP** instructions (and therefore set as many “low-level” breakpoints) and still the program will be executed in real-time. However, every time you insert/remove a **TRAP** instruction you need to recompile your program.*

#### Real-Time Execution

When inDART-STX runs in real time, instructions are executed just as the microcontroller would without the debugger.

#### Step Mode Execution

In step mode, program execution is supervised by the host PC and performed (automatically) step by step.

**Note:** *in step mode, peripherals are still truly handled by the target microcontroller. However, interrupts are not handled (that is, interrupt events never occur). This affects execution of instructions like **WFI** and **HALT**—since these instructions require interrupts to work, in step mode their behaviour is different than in real-time mode. In particular, the **WFI** instruction loops forever on itself (no interrupts ever occur), and after executing an **HALT** instruction, program execution cannot continue (same reason).*

**Note:** *in step mode, program execution speed is affected by the host PC speed—in particular, in step mode, program execution is slower than real-time program execution. This can affect the program execution speed when, for example, commands such as “**Step Over**”, “**Step Out**” or “**Run to Cursor**” are executed, since it can take a long time to perform the action (especially when the code contains loops to be executed many times).*

## Online Assembler Notes

During a debugging sessions, you should not modify assembly instructions in the *Assembly* window (through the Online Assembler), because, due to the HDFlash memory technology, the byte-by-byte programming is only successful if the device is blank. Undesired results may arise otherwise. The same holds for the *Memory* window.

## Stop Execution Notes

During debugging, user program execution will stop if one of the following conditions occur:

1. A breakpoint has been encountered (or a TRAP instruction has been executed);
2. The microcontroller RESET line has been driven active;
3. A microcontroller internal RESET state has been triggered (by, for example, a Watchdog event, or an internal low-voltage detection);
4. The “**Stop Program**” emulation command has been issued from the inDART-STX user interface. Please note that, during real-time execution, the “**Stop Program**” command only works if interrupts are enabled in the user application. This note only applies to ST72F321, 324, 324L and 521.

**Note (for ST72F321, 324, 324L and 521):** *when executing in real-time mode, the “**Stop Program**” command only works if interrupts are enabled on the user program (the **RIM** instruction must be present at the beginning of user’s code). The **RIM** instruction is only needed if users want to stop the program by using the “**Stop Program**” from the user interface—and does not affect any other debugging feature. In particular, the **RIM** instruction does not affect breakpoints.*

**Note (for ST72F621, 622, 623, 63B, 651 and 652):** *when in real-time emulation, the “**Stop Program**” command causes a chip reset.*

## Option Bytes

Particular attention must be paid in correctly setting each of the Option Bytes parameters. Improper settings may cause the target microcontroller not to work correctly (or not to work at all).

On some devices, if the Option Bytes are programmed with a wrong combination, you will no longer be able to access the device. If this happens, you must set the “ICC Entry Mode” parameter to “Ignore Option Bytes” (note: this setting is not available for all devices) and then, if the application board hasn’t an oscillator or a logic clock, the OSC\_CLK pin from the ISP connector must be connected to the application board.

For detailed information on the meaning of the “ICC Entry Mode” parameter please refer to the “Appendix A: ICC Entry Mode Details”.

## Troubleshooting

This section reports some device-specific problems that may arise when working with the target devices covered in this chapter. Please refer to the inDART-STX user’s manual for general troubleshooting hints.

When debugging, the program runs too slowly

This problem can have two causes:

- If not correctly set (via Option Bytes), the target microcontroller may run at a frequency lower than that specified. In this conditions, the low working frequency affects the ICC communication and, therefore, the inDART-STX user interface performances.
- Under some conditions, program execution is not performed in real time. As a consequence, program execution may slow down.

**3**

### The “Stop Program” command doesn’t work

ST72F321-, 324-, 324L- and 521-specific: when executing in real-time mode, the **“Stop Program”** command only works if interrupts are enabled on the user program (the **RIM** instruction must be present at the beginning of user’s code). The **RIM** instruction is only needed if users want to stop the program by using the **“Stop Program”** from the user interface—and does not affect any other debugging feature. In particular, the **RIM** instruction does not affect breakpoints.

Also make sure that your program doesn’t use the SPI peripheral and that the I/O bits corresponding to the ICCDATA and ICCCLK signals are set to input mode.

### The program execution stops at the beginning of user’s code

A Reset condition occurred. This can be due to an external Reset condition (microcontroller’s RESET line driven low) or an internal Reset condition (e.g., due to a Watchdog event). For more information on causes that can trigger a Reset condition, please refer to the specific ST7 microcontroller device data sheet.

ST72F621-, 622-, 623-, 63B-, 651- and 652-specific: additionally, when executing in real-time, a **“Stop Program”** command causes a Reset condition.

### The program seems not to work correctly

When debugging your program, some target microcontroller’s resources are reserved for debugging purposes. In particular, 200 bytes are reserved for the monitor (from address FF18H to address FFDFH). Therefore, you must pay attention that your program doesn’t use this range of memory locations. Also, please remember that 7 stack levels are reserved as well.

#### **Interrupt handling routines are not executed**

Under some conditions, program execution is not performed in real time. As a consequence, interrupts are not handled.

## 4. HDFlash Devices (with On-Chip Debug Module) Specific Topics

### Introduction

This chapter details some specific issues that must be known when working with the following devices:

- ST72F325
- ST7FLCD1
- ST7FMC1
- ST7FMC2

These devices are based on HDFlash technology (FLASH-based, programmable byte per byte, erasable by sector), which only allows for a limited number of FLASH erase/write operations (typically <100). Each time a debugging session is started, the user code is downloaded to the device FLASH memory, and one FLASH erase/write cycle is wasted.

**Note:** *the behaviour and limitations of inDART-STX described in this chapter refer to debugging sessions dedicated to the above mentioned devices, and do not hold for other target devices.*

### Limitations

Some target on-chip resources are wasted for debugging purposes. In particular:

- 5 stack levels (bytes) are wasted.
- 216 bytes are reserved for the monitor (from address FF08H to address FFDFH).

- ICCDATA and ICCCLK lines are reserved for programming and in-circuit debugging.
- The TRAP instruction and the TRAP interrupt vector are reserved for the on-chip Debug Module.
- The DM peripheral (the on-chip Debug Module) is reserved (DM registers must not be written to by the user application; if read from, the data returned is undefined).
- Peripherals run even in STOP mode.
- The “WDG SW” bit in the Option Bytes must be set to “software watchdog”.

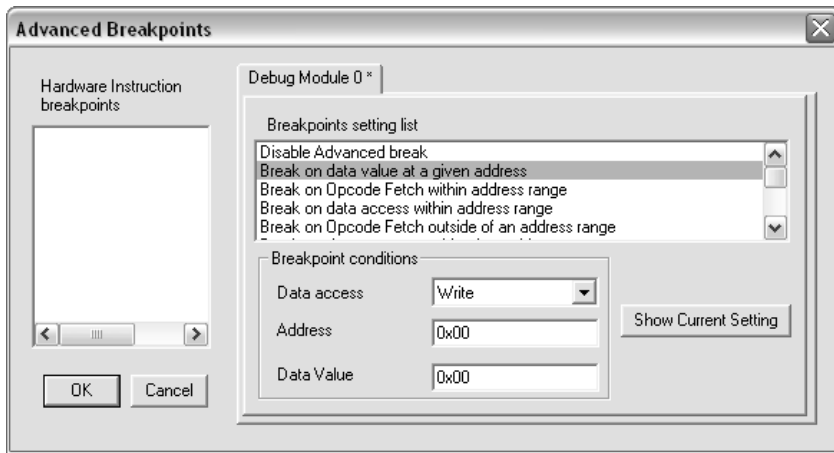
**Note:** *in order to enter the program the FLASH memory of the devices, inDART-STX supplies a 12 V DC voltage to the ICCSEL/V<sub>PP</sub> pin. You should be cautious of this high voltage, since it could damage other circuitry connected to that pin.*

## Breakpoint Notes

inDART-STX can handle a maximum of two breakpoints within program memory. Since these two breakpoints are handled by the on-chip Debug Module, inDART-STX always executes in real time.

### Instruction Breakpoints and Advanced Breakpoints

When working with devices with on-chip Debug Module, advanced breakpoints are available. Advanced breakpoints differ from instruction breakpoints (that is, standard breakpoints) in that they can stop program execution depending on complex situations. Advanced breakpoints allow you, for example, to stop execution when a specific value is written to a specific address, or when two specified opcodes are fetched sequentially. To set advanced breakpoints, choose **Emulator > Advanced Break** from the main menu (a debugging session must have been started). The following dialog box will appear.



The *Advanced Breakpoints* Dialog Box

4

When using breakpoints (either instruction breakpoints or advanced breakpoints), be aware of the following:

- You can use advanced breakpoints or instruction breakpoints, but not a combination of the two;
- As soon as a complex breakpoint is set, previously set instruction breakpoints are automatically disabled—and vice versa;
- Step commands (Step Into, Step Over, Step Out, Run to Cursor) waste one instruction breakpoint. When issuing a step command, all previously set instruction or advanced breakpoints are disabled, the step command is executed, and then previously set instruction or advanced breakpoints are re-enabled.

**Note:** *if an advanced breakpoint is set which specifies anything but an instruction fetch, then program execution stops **after** the instruction during which the breakpoint condition is met is executed.*

The following table summarizes the available advanced breakpoints.

## 4. HDFlash Devices (with On-Chip Debug Module) Specific Topics

---

# 4

Breakpoint Type	Required Parameters
Break on data value at a given address	The type of data access (write, read, read/write); The address; The data value.
Break on opcode fetch within address range	The address at the beginning of the range; The address at the end of the range.
Break on data access within address range	The type of data access (write, read, read/write); The address at the beginning of the range; The address at the end of the range.
Break on opcode fetch outside of an address range	The address at the beginning of the range; The address at the end of the range.
Break on data access outside of an address range	The type of data access (write, read, read/write); The address at the beginning of the range; The address at the end of the range.
Break on two sequential opcode fetches	The first opcode fetch; The second opcode fetch.
Break on data access at one of two addresses	The type of data access (write, read, read/write); The first of the two addresses; The second of the two addresses.
Break on opcode fetch or data access at given addresses	The type of data access (write, read, read/write); The address where the opcode fetch must occur; The address where the data access must occur.
Break on conditional stack write or opcode fetch at given address	The maximum stack address below which the write operation must occur; The address where the opcode fetch must occur.
Break on conditional stack write or data access at given address	The type of data access (write, read, read/write); The address where the data access must occur; The maximum stack address below which the write operation must occur.

Advanced Breakpoints Summary

## Online Assembler Notes

During a debugging sessions, every time assembly instructions are modified in the *Assembly* window (through the Online Assembler), a reset condition occurs (even though the Program Counter arrow still points to the same instruction). Subsequent debugging commands will be therefore executed from a reset state.

## Stop Execution Notes

During debugging, user program execution will stop if one of the following conditions occur:

1. A breakpoint has been encountered;
2. The microcontroller RESET line has been driven active;
3. A microcontroller internal RESET state has been triggered (by, for example, a Watchdog event, or an internal low-voltage detection);
4. The **“Stop Program”** emulation command has been issued from the inDART-STX user interface.

**4**

## Option Bytes

Particular attention must be paid in correctly setting each of the Option Bytes parameters. Improper settings (and improper setting combinations) may cause the target microcontroller not to work correctly (or not to work at all).

On some devices, if the Option Bytes are programmed with a wrong combination, you will no longer be able to access the device. If this happens, you must set the “ICC Entry Mode” parameter to “Ignore Option Bytes” (note: this setting is not available for all devices) and then, if the application board hasn’t an oscillator or a logic clock, the OSC\_CLK pin from the ISP connector must be connected to the application board.

For detailed information on the meaning of the “ICC Entry Mode” parameter please refer to the “Appendix A: ICC Entry Mode Details”.

## Troubleshooting

This section reports some device-specific problems that may arise when working with the target devices covered in this chapter. Please refer to the inDART-STX user’s manual for general troubleshooting hints.

**When debugging, the program runs too slowly**

If not correctly set (via Option Bytes), the target microcontroller may run at a frequency lower than that specified. In this conditions, the low working

frequency affects the ICC communication and, therefore, the inDART-STX user interface performances.

### The program execution stops at an unexpected location

Advanced breakpoints are enabled. When advanced breakpoints are enabled, instruction breakpoints are disabled and the program execution stops as soon as an advanced breakpoint condition is met. The stop location may appear to be “random”, but is actually consistent with what specified in the *Advanced Breakpoints* dialog box. To view advanced breakpoints active settings, choose **Emulator > Advanced Break** from the main menu.

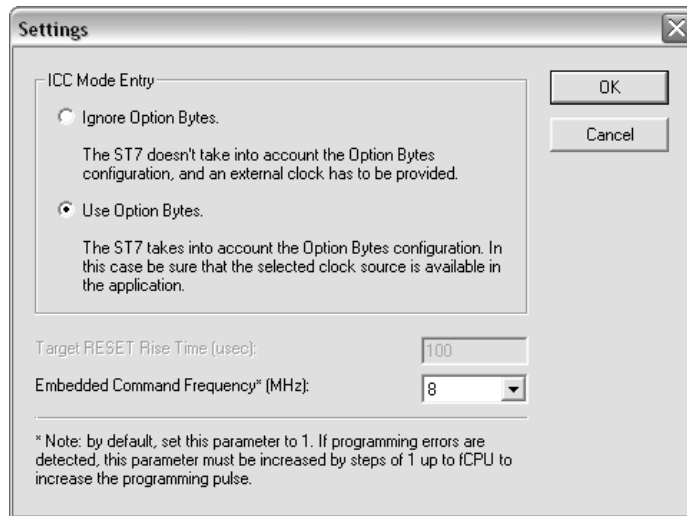
4

### The program execution stops at the beginning of user’s code

A Reset condition occurred. This can be due to an external Reset condition (microcontroller’s RESET line driven low) or an internal Reset condition (e.g., due to a Watchdog event). For more information on causes that can trigger a Reset condition, please refer to the specific ST7 microcontroller device data sheet.

# Appendix A: ICC Mode Entry Details

The “ICC Mode Entry” parameter allows you to specify how inDART- STX will enter the ICC mode. You can choose to use the Option Bytes programmed into the device or to bypass them and use a “forced” value instead. The explanation which follows replaces the one given in the printed inDART-STX for ST7 user’s manual.



The *Settings* Dialog Box

In the former case (“Use Option Bytes”), you must provide the required clock source (specified in the Option Bytes value programmed into the device)—otherwise the device won’t work.

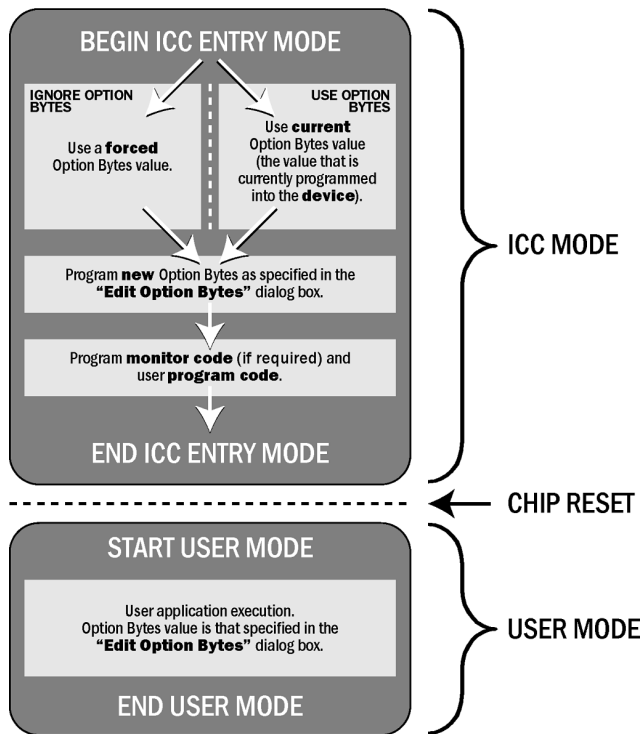
In the latter case (“Ignore Option Bytes”), a “forced” Option Bytes value is used, thus enabling you to recover from an previous incorrect setting. This means, among other things, that a “safe” clock source is used (typically an external clock must be provided). This option (which is used by inDART-STX by default) is useful when the Option Bytes value programmed into the device specifies

**A**

incorrect start-up parameters; you can bypass them and start-up from a safe condition.

The following diagram illustrates what happens when starting the debug.

### START DEBUGGING SEQUENCE



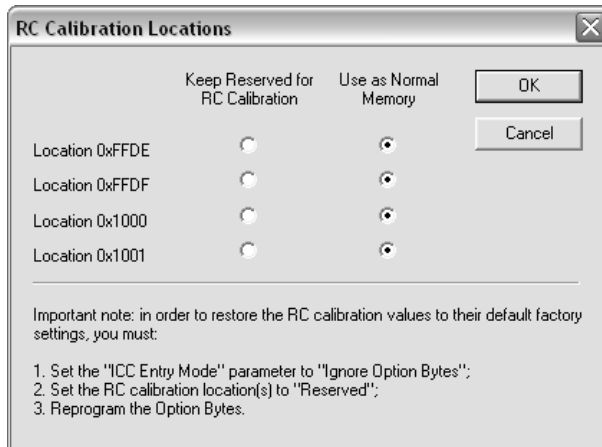
Start Debugging Sequence



# Appendix B: RC Calibration Notes

Some devices (ST7FLITE0, LITE1, LITE2, LITES, DALI) contain an internal RC oscillator. This oscillator must be calibrated to obtain the frequency required in the application. This is done by software writing a calibration value in the RCCR (RC Control Register). Whenever the microcontroller is reset, the RCCR returns to its default value (FFh)—i.e., each time the device is reset, the calibration value must be loaded in the RCCR. Predefined calibration values are stored in some FLASH and/or EEPROM locations.

If RC calibration is not used, these locations can be used as “normal” memory. The *RC Calibration Locations* dialog box, available by clicking the “RC Calibration Locations” button in the *Settings* dialog box (this button is available only for devices featuring an internal RC oscillator) allows you to specify, for every RC calibration location available, whether to keep it reserved for calibration purposes or to use it as a normal memory location.



The *RC Calibration Locations* Dialog Box

**B**

