freescale™
Enabled

indart® | one

User's Manual

SofTec® MICROSYSTEMS

Development Tools for the Embedded World

inDART-One

In-Circuit Programmer and Debugger for Freescale 8- and 16-bit Microcontrollers

User's Manual

Revision 2.0

SofTec MICROSYSTEMS

Development Tools for the Embedded World

DC01076

We want your feedback!

SofTec Microsystems is always on the look-out for new ways to improve its Products and Services. For this reason feedback, comments, suggestions or criticisms, however small, are always welcome.

Our policy at SofTec Microsystems is to comply with all applicable worldwide safety and EMC/EMI regulations. Our products are certified to comply to the European New Approach Directives and the CE mark is applied on all our products.

This product as shipped from the factory has been verified to meet with requirements FCC as a CLASS A product.

This product is designed and intended for use as a development platform for hardware or software in an educational or professional laboratory.

In a domestic environment, this product may cause radio interference in which case the user may be required to take adequate prevention measures.

Attaching additional wiring to this product or modifying the product operation from the factory default as shipped may effect its performance and cause interference with other apparatus in the immediate vicinity. If such interference is detected, suitable mitigating measures should be taken.

SofTec Microsystems

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

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

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

Web: http://www.softecmicro.com

Important

SofTec Microsystems reserves the right to make improvements to the inDART-One In-Circuit Programmer/Debugger, its 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

SofTec Microsystems is a registered trademark of SofTec Microsystems, Spa.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc.

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.

Written by Paolo Xausa

Contents

Index of Figures

Contents

Index of Tables

0 Before Starting

0.1 Important Notice to Users

While every effort has been made to ensure the accuracy of all information in this document, SofTec Microsystems assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accidents, or any other cause.

0.2 Required Skills

In order to beneficially use the inDART-One In-Circuit Programmer/Debugger, you should be acquainted with certain skills, ranging from hardware design to software design. In particular, you should possess knowledge of the following:

- Microcontroller systems;
- HC08, HCS08, RS08, S12 or S12X architecture knowledge;
- Programming knowledge (Assembly and C).

1 Overview

1.1 What is inDART-One?

The inDART-One In-Circuit Programmer/Debugger is a powerful programming and debugging tool for Freescale HC08-, HCS08-, RS08-, S12- and S12X-based systems.

1.1.1 In-Circuit Debugger

inDART-One takes advantage of the CodeWarrior Development Studio Special Edition (which groups an Editor, Assembler, C Compiler and Debugger) and the Freescale MON08 and BDM interfaces, which allow the download and debug of the user application into the target microcontroller's FLASH memory.

Together with CodeWarrior, inDART-One provides you with everything you need to write, compile, download, in-circuit emulate and debug user code. Full-speed program execution allows you to perform hardware and software testing in real time. inDART-One is connected to the host PC through a USB port.

inDART-One offers you the following debugging features:

- Real-time code execution and in-circuit debugging without probes—works with all packages (MON08- and BDM-compatible connectors);
- 1.8 V to 5.5 V devices supported;
- Standard chip used—no bondouts, 100% electrical characteristics guaranteed;
- Working frequency up to the target microcontroller's maximum;
- Jumperless hardware mode setting;
- MON08 automatic target frequency detection;
- MON08 automatic V_{DD} and V_{TST} voltage generation;
- Target power in and power out connectors for manual or automatic MON08 target power switching;

- Hardware self diagnostic test;
- USB connection to the PC;
- CodeWarrior IDE (the same user interface of all Freescale tools), with editor, assembler, C compiler and debugger.

1.1.2 Single Programmer

Additionally, inDART-One is a full-featured programmer, thanks to the provided DataBlaze programming utility.

1.1.3 Multiple Programmer

Up to 32 inDART-One instruments can be connected (using USB hubs) to the same PC, allowing for multiple (gang) programming sessions. A specific multiple programming utility, MultiBlaze, is provided.

1.1.4 inDART Programming Library

The inDART Programming Library (IPL-One) is a DLL which includes all of the low-level functions that allow you to set up the instrument and perform all of the programming commands and functions of the DataBlaze and MultiBlaze programming utilities from within your own Windows application.

The IPL-One Programming Library contains C written routines, and can be used to interface the instrument from within, for example, a Microsoft Visual C or Visual Basic application, as well as any other programming language that supports the DLL mechanism.

1.2 Package Contents

The inDART-One package includes the following items:

- inDART-One in-circuit programming/debugging unit;
- A USB cable;
- A 16-conductor MON08 cable;
- A 6-conductor BDM cable;

- Two 2-conductor target power cables;
- SofTec Microsystems inDART-One “System Software” CD-ROM;
- CodeWarrior Development Studio Special Edition CD-ROMs;
- A QuickStart Tutorial poster;
- This user’s manual;
- A registration card.

1.3 Optional HC08 Fast Programming Algorithms

Fast programming algorithms reduce significantly the amount of time needed to program HC08 devices. Fast programming algorithms can be used both in single and multiple programming.

Device	Memory Size	Programming Times
MC68HC908GP32	32KB, fCPU = 4 MHz	Program = 1.5 s Program + Verify = 2.9 s
MC68HC908QB8	8KB, fCPU = 4 MHz	Program = 1.1 s Program + Trimming + Verify = 1.5 s
MC68HC908AZ60A	60KB, fCPU = 4 MHz	Program = 2.7 s Program + Verify = 5.4 s

Table 1.1: Fast Programming Times for Some HC08 Devices

See “Unlocking Fast Programming Algorithms” on page 32 for information on how to purchase and setup these features.

1.4 Hardware Overview

The following figure shows the various inDART-One connectors.

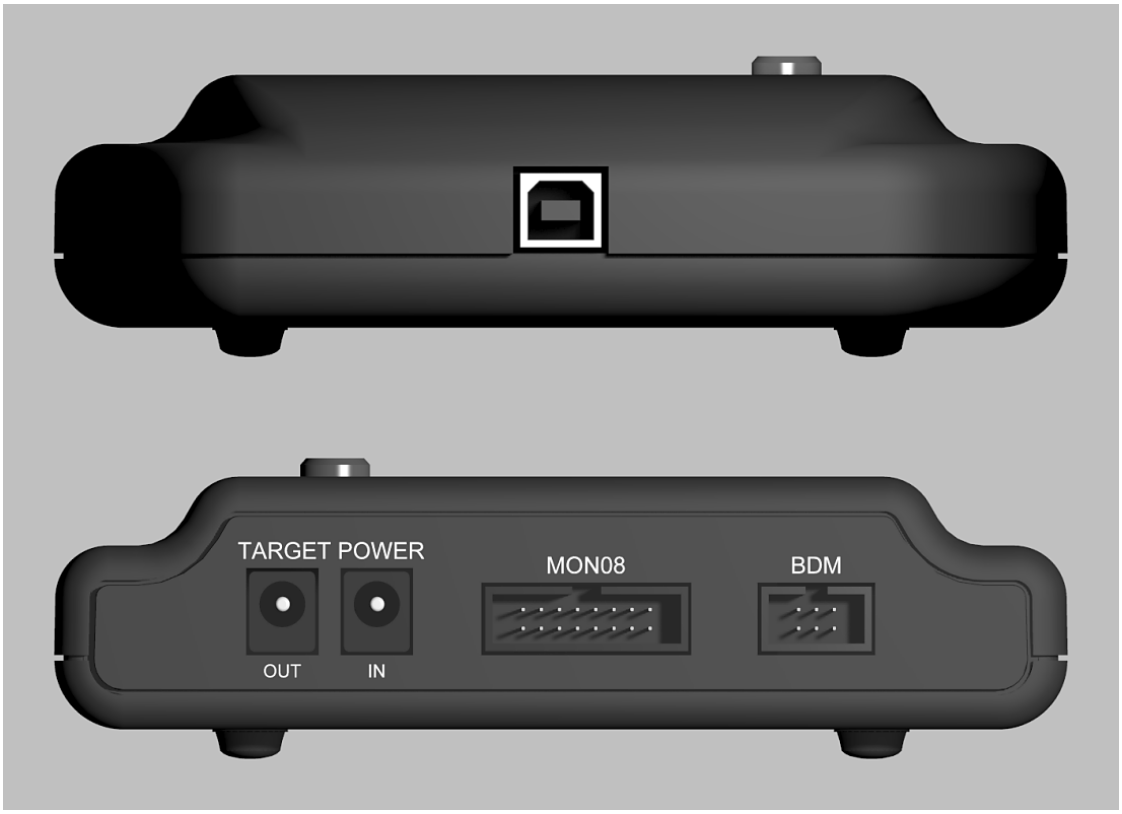

Figure 1.1: inDART-One Connectors

1.4.1 USB Connector

The USB connector is used to connect inDART-One to the host PC. When connecting more then one inDART-One to a host PC, USB hubs can be used.

inDART-One is USB 2.0 compliant and uses a high speed connection, but can be connected to USB 1.1 systems as well.

inDART-One is powered by the USB bus voltage.

1.4.2 MON08 Connector

inDART-One uses a 16-pin MON08 connector to program and debug HC08 devices. This connector can be configured to work as a standard (Multilink compatible) MON08 connector or as an enhanced (SofTec Microsystems compatible) MON08 connector. The enhanced MON08 connector allows the

target microcontroller to free some lines after entering the monitor mode at reset.

For more information about how to provide the appropriate MON08 connector on your target board, see “MON08 Target Connections” on page 63.

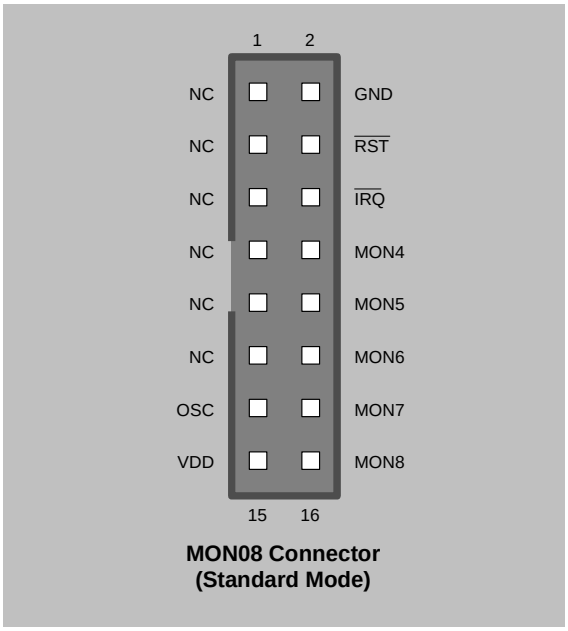

Figure 1.2: MON08 Connector Signals (Standard Mode)

Pin	Signal Name	Description
2	**GND**	System ground.
4	**RST#**	MCU reset; held at V_{TST} (or V_{DD}, depending on the target microcontroller) out of reset. No other target-system logic should be tied to this signal.
6	**IRQ#**	MCU interrupt; held at V_{TST}.
8, 10, 12, 14, 16	**MON4 ... MON8**	I/O pins connected to target microcontroller.
13	**OSC**	This signal can be used as an auxiliary clock source, and is particularly useful when the target microcontroller requires an external clock which is not available on the target board.
15	**VDD**	The target V_{DD} line needs to be driven correctly at reset. When using the MON08 connector in standard mode, inDART-One can automatically generate this signal.
1, 3, 5, 7, 9, 11	**NC**	Not connected.

Table 1.2: MON08 Connector Signals (Standard Mode)

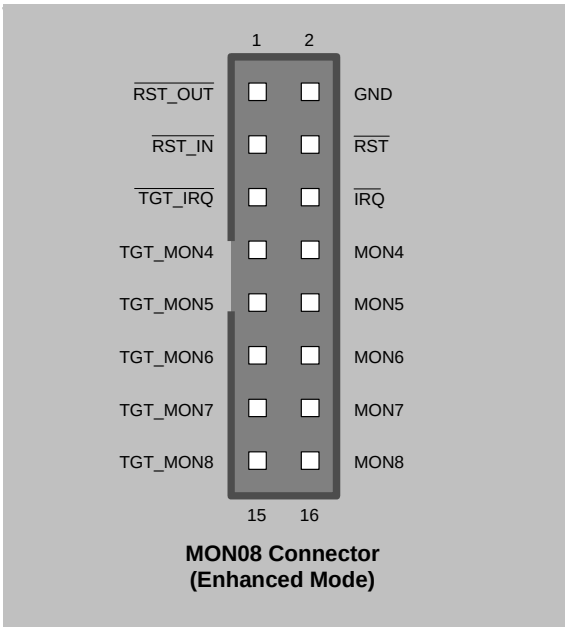

Figure 1.3: MON08 Connector Signals (Enhanced Mode)

Pin	Signal Name	Description
1	**RST_OUT#**	Reset signal to target system: GND or open drain output reflecting the state of the MCU RST# and RST_IN# signals.
2	**GND**	System ground.
3	**RST_IN#**	Reset signal from target system: GND to V_{DD} input to control the state of the MCU RST# and RST_OUT# signals.
4	**RST#**	MCU reset; held at V_{TST} (or V_{DD}, depending on the target microcontroller) out of reset. No other target-system logic should be tied to this signal.
5	**TGT_IRQ#**	Interrupt signal from target system: GND to V_{DD} input to control the state of the MCU IRQ# signal.
6	**IRQ#**	MCU interrupt; held at V_{TST} when the TGT_IRQ# signal is not asserted.
7, 9, 11, 13, 15	**TGT_MON4 ... TGT_MON8**	I/O pins connected to target application.
8, 10, 12, 14, 16	**MON4 ... MON8**	I/O pins connected to target microcontroller.

Table 1.3: MON08 Connector Signals (Enhanced Mode)

Each of the MON08 connector lines must be connected to the appropriate pins of the specific target microcontroller used. Once a target microcontroller has been selected, the “Communication Settings” dialog box (available in CodeWarrior, DataBlaze and MultiBlaze) automatically shows you how to connect that specific device to the MON08 connector.

1.4.3 BDM Connector

inDART-One uses the standard, 6-pin BDM connector defined by Freescale to program and debug HCS08, RS08, S12 and S12X devices. You must therefore provide such connector (see the diagram below) on your target board.

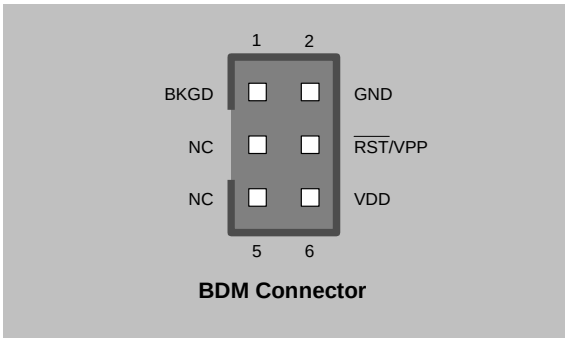

Figure 1.4: BDM Connector Signals

Pin	Signal Name	Description
1	**BKGD**	Single-wire background interface pin.
2	**GND**	System ground.
3	**NC**	Not connected.
4	**RST#/VPP**	Reset signal to target system, or V_{PP}.
5	**NC**	Not connected.
6	**VDD**	Power supply voltage from target. This pin is used by inDART-One for signal conditioning.

Table 1.4: BDM Connector Signals

1.4.4 Target Power Connectors

These two connectors allow you control how to power the target board, depending on the target device selected and on the target communication settings.

For more information, see the chapter relative to the target family you are working with, later in this manual.

1.4.5 Status LEDs

The status LEDs on the instrument turn on/off during the various stages of debugging and programming.

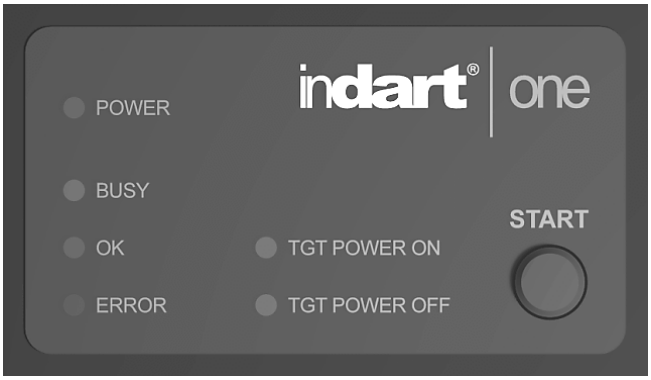

Figure 1.5: Status LEDs

The following table explains the meaning of each LED.

LED Name	Description
POWER	Indicates that the instrument is powered on. Turns on when connecting the instrument to the USB bus.
BUSY	Indicates that the instrument is busy, either during programming or debugging.
OK	Turns on at the end of the programming if no errors occurred.
ERROR	Turns on at the end of the programming if it could not be performed successfully.
TGT POWER ON	Indicates that you must power on the target board. Only for MON08 devices, when manual target powering is selected.
TGT POWER OFF	Indicates that you must power off the target board. Only for MON08 devices, when manual target powering is selected.

Table 1.5: Status LEDs

1.4.6 "START" Push-Button

The "START" push-button can be used to start programming the target device, when using the DataBlaze or MultiBlaze programming utilities.

Additionally, when using MON08 devices with manual target powering, the "START" button can be used to confirm the target board's powering on/off, (see "Power Settings" on page 57).

1.5 Software Overview

1.5.1 CodeWarrior Development Studio Special Edition

The inDART-One In-Circuit Programmer/Debugger comes with CodeWarrior Development Studio Special Editions for the various Freescale HC08, HCS08, RS08, S12 and S12X families.

CodeWarrior Development Studio is a powerful and easy-to-use tool suite designed to increase your software development productivity. Its Integrated Development Environment (IDE) provides unrivaled features such as Processor Expert application design tool, full chip simulation, Data Visualization and project manager with templates to help you concentrate on the added value of your application.

The comprehensive, highly visual CodeWarrior Development Studio for Freescale Microcontrollers enables you to build and deploy Freescale systems quickly and easily. This tool suite provides the capabilities required by every engineer in the development cycle, from board bring-up to firmware development to final application development.

Without a license key, the product will run in a 1 KB code-size limited demonstration mode.

To break the 1 KB limit, you have two options:

1. Contact Metrowerks to request an unlimited period, free license key to increase the code size limit to 16 KB (HC08, HCS08, RS08) or 32 KB (S12, S12X);
2. Contact Metrowerks to request a 30-day limited, free license key to run the compiler without limitations.

This documentation covers the basic setup and operation of CodeWarrior Development Studio, but does not cover all of its functions. For further information, please refer to the CodeWarrior on-line help and on-line documentation provided.

1.5.2 DataBlaze Programming Utility

DataBlaze is a full-featured programming utility. DataBlaze offers the following advanced features:

- Memory editing;
- Blank check/erase/program/verify/read operations;
- Project handling;
- One-button, multiple-operations programming (“Auto” feature);
- Serial numbering.

1.5.3 MultiBlaze Programming Utility

MultiBlaze is an easy-to-use multiple programming utility suitable for production environments. MultiBlaze offers the following features:

- Easy-to-use programming interface;
- Blank check/erase/program/verify/read operations;
- One-button, multiple-operations programming (“Auto” feature);
- Statistics and logging.

1.5.4 inDART-One Control Panel

The inDART-One Control Panel utility allows you to:

- Perform an instrument self-diagnostic hardware test;
- Install/enable purchased HC08 fast algorithms.

1.5.5 Software Upgrades

The latest version of the inDART-One system software is always available free of charge from our website: *http://www.softecmicro.com.*

When installing the inDART-One system software you have the option to electronically register the product. If you register the product, you will be automatically notified by e-mail every time a new version of the inDART-One system software is available.

1.6 Recommended Reading

This documentation describes how to use the inDART-One In-Circuit Programmer/Debugger, how to set up basic debugging sessions with CodeWarrior, how to use the DataBlaze and MultiBlaze programming utilities, and how to use the IPL-One Programming Library. Additional information can be found in the following documents:

- CodeWarrior additional documentation— available from the CodeWarrior IDE.
- Freescale datasheets.
- Freescale application notes.

1.7 Getting Technical Support

For technical assistance, documentation and information about products and services, please refer to your local SofTec Microsystems partner.

SofTec Microsystems offers its customers a technical support service at *support@softecmicro.com*. Before getting in contact with us, we advise you to check that you are working with the latest version of the inDART-One system software (upgrades are available free of charge at *http://www.softecmicro.com*) and to run the diagnostic test (see “Diagnostic Test” on page 131).

2 Setup

2.1 Software Setup

Note: *before connecting the inDART-One board to the PC, it is recommended that you install all of the required software first (see below), so that the inDART-One USB driver will be automatically found by Windows when you connect the board.*

2.1.1 Host System Requirements

The following hardware and software are required to run the CodeWarrior user interface and the DataBlaze and MultiBlaze programming utilities together with inDART-One:

1. A 500-MHz (or higher) PC compatible system running Windows 98, Windows 2000 or Windows XP;
2. 256 MB of available system RAM plus 1 GB of available hard disk space;
3. A USB port;
4. CD-ROM drive for installation.

2.1.2 CodeWarrior Setup

To install the CodeWarrior Development Studio Special Edition, insert the correct CodeWarrior CD-ROM (the one supporting the device family you will work with) into your computer's CD-ROM drive. A startup window will automatically appear. Follow the on-screen instructions.

2.1.3 inDART-One Utilities Setup

The inDART-One utilities setup install all of the other required components to your hard drive. These components include:

- The inDART-One USB driver and DLLs;
- DataBlaze programming utility;
- MultiBlaze multiple programming utility;
- inDART-One control panel;
- CodeWarrior examples;
- inDART Programming Library examples;
- Documentation in PDF format.

To install the inDART-One utilities, insert the SofTec Microsystems **"System Software"** CD-ROM into your computer's CD-ROM drive. A startup window will automatically appear. Choose **"Install Instrument Software"** from the main menu. A list of available software will appear. Click on the **"inDART-One Utilities"** option. Follow the on-screen instructions.

Note: *to install the inDART-One utilities on Windows 2000 or Windows XP, you must log in as Administrator.*

2.2 Hardware Setup

2.2.1 PC Connection

inDART-One connects to the host PC through a USB port. When connecting more than one inDART-One to a host PC, USB hubs can be used.

inDART-One is USB 2.0 compliant and uses a high speed connection, but can be connected to USB 1.1 systems as well.

inDART-One is powered by the USB bus voltage, and requires a USB port capable of supplying 350 mA.

The first time inDART-One is connected to the PC, Windows recognizes the instrument and starts the **"Found New Hardware Wizard"** procedure, asking you to specify the driver to use for the instrument.

1. On Windows XP (SP2) the following dialog box will appear, asking you to search for a suitable driver on the web.

Figure 2.1: New Hardware Wizard, Step 1

   Select the **"No, not this time"** option and click the **"Next >"** button.
2. The following dialog box will appear.

Figure 2.2: New Hardware Wizard, Step 2

Click the **"Next >"** button.

3. Depending on your Windows settings, the following warning may appear.

Figure 2.3: New Hardware Wizard, Step 3

Note: *this warning is related to the fact that the USB driver used by inDART-One is not digitally signed by Microsoft, and Windows considers it to be potentially malfunctioning or dangerous for the system. However, you can safely ignore the warning, since every kind of compatibility/security test has been carried out by SofTec Microsystems.*

Click the **"Continue Anyway"** button.

4. Windows will install the driver files to your system. At the end of the installation, the following dialog box will appear.

Figure 2.4: New Hardware Wizard, Step 4

Click the **"Finish"** button to exit from the **"Found New Hardware Wizard"** procedure.

5. The Starter Kit's USB driver is now installed on your system.

2.2.2 Target Connection

inDART-One connects to your target board either via the MON08 connector or the BDM connector, depending on your hardware's target device. Additionally, you can take advantage of the Target Power connectors to supply your target board. For detailed information about these two

connectors and how to use them in conjunction with your target board, please refer to the following chapters:

- “Working with HC08 Devices”, on page 55
- “Working with HCS08 Devices”, on page 85
- “Working with RS08 Devices”, on page 87
- “Working with S12(X) Devices”, on page 89

depending on the hardware you are working with.

2.2.3 Communication Settings

After physically connecting inDART-One to your target hardware, inDART-One must be configured properly so that communication with the target device can be established correctly.

Communication settings are defined through the “Communication Settings” dialog box, available both in CodeWarrior and in the DataBlaze and MultiBlaze programming utilities.

Each target device/hardware configuration requires specific settings, detailed later in this manual in the chapter relative to the target family you are working with.

2.3 Unlocking Fast Programming Algorithms

Fast HC08 programming algorithms can be purchased and enabled (in one or more instruments) in order to reduce significantly the amount of time needed to program HC08 devices. To purchase fast HC08 programming algorithms, please contact SofTec Microsystems (*info@softemicro.com*).

Note: *once an instrument is unlocked for a particular fast algorithm, all of the other instruments connected to the same PC are automatically considered unlocked for the same fast algorithm, as long as the unlocked instrument is also connected. The unlocked instrument works as a "master", temporarily unlocking all of the other instruments connected to the same PC at the same time.*

Once you have the unlocking code(s), do the following:

1. Open the inDART-One Control Panel. Select **"Start > Programs > SofTec Microsystems > inDART-One > Control Panel"**. The following dialog box will appear.

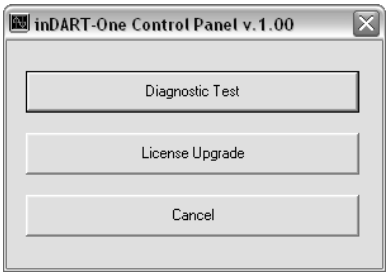

Figure 2.5: inDART-One Control Panel

2. Click the **"License Upgrade"** button and follow the on-screen instructions.

3 Debugging

3.1 inDART-One Working Principles

inDART-One is an in-circuit debugger as well as a programming tool. It programs files into the target microcontroller and offers debugging features like real-time code execution, stepping, and breakpoint. Its debugging features are achieved thanks to the microcontroller's integrated debug module.

The integrated debug module communicates with the host PC through a bi-directional, command-based protocol via some dedicated lines of the microcontroller (which are therefore reserved during debugging sessions). The same lines are also used during device programming.

Contrariwise to traditional in-circuit emulation (where the target application is executed and emulated inside the emulator), inDART-One uses the very same target microcontroller to carry on in-circuit execution. This means that all microcontroller's peripherals (timers, A/D converters, I/O pins, etc.) are not reconstructed or simulated by an external device, but are the very same target microcontroller's peripherals. Moreover, the inDART-One debugging approach ensures that the target microcontroller's electrical characteristics (pull-ups, low-voltage operations, I/O thresholds, etc.) are 100% guaranteed. The trade-off, however, is that the target microcontroller must be properly configured and ready to execute target applications.

3.2 Working with CodeWarrior

3.2.1 Using the Project Wizard to Create Your Application Skeleton

CodeWarrior helps you get started with your own application by including a project wizard specific for inDART-One.

To create a new project with CodeWarrior for HC08/HCS08/RS08:

1. From the main menu, select **“File > New Project”**.
2. A Project Wizard dialog box will appear. Follow the Project Wizard steps, making sure you select the correct microcontroller derivative you are working with and that the **“SofTec”** target connection is used.

To create a new project with CodeWarrior for S12(X):

1. From the main menu, select **“File > New”**.
2. A dialog box will appear. Select **“HC(S)12 New Project Wizard”**.
3. Follow the Project Wizard steps, making sure you select the correct microcontroller derivative you are working with and that the **“SofTec”** target connection is used.

3.2.2 Starting your First Debugging Session

The first time you enter a debugging session (by selecting **“Project > Debug”** from the CodeWarrior’s main menu) the **“MCU Configuration”** dialog box will open, asking you to select the debugging hardware connected to the PC. Make sure that the hardware code is set to **“inDART-One”**.

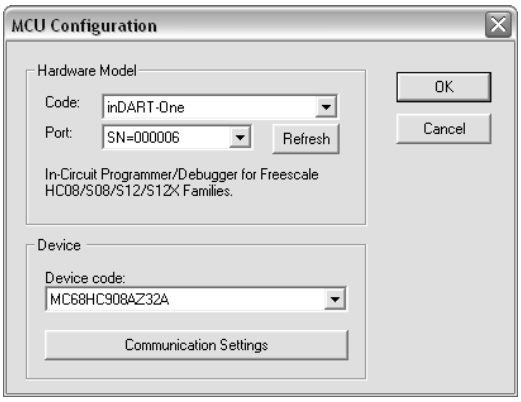

Figure 3.1: The “MCU Configuration” Dialog Box

3.2.3 Using Existing Projects with inDART-One

If your project has been targeted to an emulator/simulator other than inDART-One and you wish to use inDART-One as the debugger for your project, please do the following:

1. From the CodeWarrior debugger interface, select **"Component > Set Connection"**. The "Set Connection" dialog box will appear.

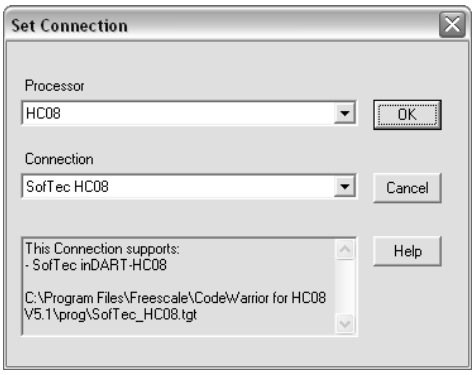

Figure 3.2: The "Set Connection" Dialog Box

2. Choose **"HC08"**, **"HCS08"**, **"RS08"** or **"HC12"** (depending on your target device) as processor and **"SofTec"** as connection. Click the **"OK"** button.
3. The **"MCU Configuration"** dialog box will appear allowing you to select inDART-One as the hardware debugger.

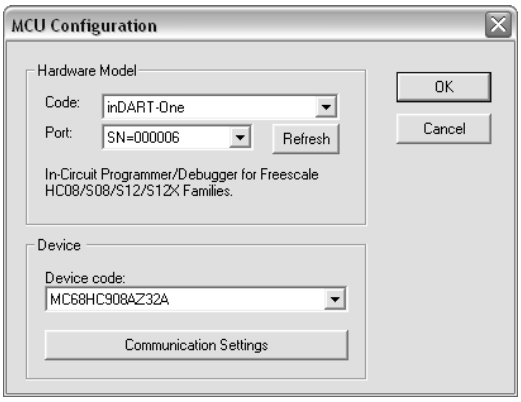

Figure 3.3: The "MCU Configuration" Dialog Box

4. On the CodeWarrior debugger interface a new menu (**“inDART-HC08”**, **“SofTec-HCS08”**, **“SofTec-RS08”** or **“inDART-HCS12”**) will be created. From this menu, select **“Load”** and locate the object file your project is based on.

3.2.4 Breakpoints and Trace

CodeWarrior offers a variety of tools for analyzing the program flow: breakpoints (both simple and complex), watchpoints and a trace buffer. All these features are implemented by taking advantage of the target microcontroller’s debug peripheral.

Note: *when setting an instruction breakpoint on a RAM location, a software breakpoint is set (the opcode present at that location is automatically replaced by the **BGND** Assembly instruction). Therefore, no hardware breakpoints are wasted.*

Note: *the Single Step command (in a C source code) and the Step Over and Step Out commands (both in a C and Assembly source code) use one hardware breakpoint.*

3.3 HC08 Notes and Tips

3.3.1 Stop Command Handling

The **“Halt”** debugging command will not work unless the IRQ interrupt is properly handled. In particular, the following precautions must be taken in the application’s source code.

1. Global interrupts must be enabled (use the “cli” instruction);
2. The IRQ interrupt must be enabled;

3. The IRQ interrupt vector must be handled;
4. The IRQ handling routine must include the following code:

```
irq_isr:  bil irq_isr    ; Waits for the IRQ signal to go high
          swi            ; Jumps to monitor code
          rti
```

5. Under these conditions, the TGT_IRQ# line is reserved; when it is driven low, a **"Halt"** debugging command is automatically recognized.

3.3.2 Breakpoints and Swi Instruction

The HC08's on-chip debug module only handles one hardware breakpoint. However, you can force the program execution to stop at other specific locations by inserting the **"swi"** Assembly instruction on your source code.

3.3.3 Reading Peripheral Status

Care must be taken when reading some peripheral's status/data registers, since a reading operation may cause the clearing of flags. This may happen when the "Memory" window or the "Data" window is open, since these windows read microcontroller's resources during refresh operations.

3.3.4 Interrupt Execution during Steps

When issuing stepping instructions (Single Step, Step Over, etc.) and there are pending interrupts, inDART-One will not step inside the interrupt handling routine, but the whole interrupt handling routine is executed. An exception is when you single step on an Assembly instruction which branches to itself: in this case, interrupts which may occur are not handled.

3.3.5 Peripheral Status during Steps

When single stepping on an Assembly instruction which branches to itself, peripheral status is frozen.

3.4 HCS08, RS08 and S12(X) Notes and Tips

3.4.1 Entering Debug Session with CodeWarrior

When entering a debug session, the target microcontroller's FLASH memory is automatically erased, unsecured, programmed with the user application, and the trimming value (if trimming is available for the selected microcontroller) is automatically calculated and programmed (in the location suggested by Freescale).

Note: *When programming the microcontroller with the user application (after having unsecured the device), CodeWarrior ignores (doesn't program) the security bits. As a result, when entering a debug session, the device is always unsecured, regardless of other user settings.*

3.4.2 Reading Peripheral Status

Care must be taken when reading some peripheral's status/data registers, since a reading operation may cause the clearing of flags. This may happen when the "Memory" window or the "Data" window is open, since these windows read microcontroller's resources during refresh operations.

3.4.3 Breakpoints and BGND Instruction

The BGND Assembly instruction forces the target microcontroller to enter the Active Background Debug mode, stopping program execution. CodeWarrior recognizes this event as a breakpoint and updates the contents of registers, memory, etc. Successive commands (Start/Continue, Single Step, etc.) will continue the execution of the program from the next instruction.

3.4.4 Real-Time Memory Update

During program execution, it is possible to view/edit the contents of the "Memory" window and "Data" window in real time (edit operations are only

available for RAM locations). For example, it is possible to set the periodical refresh of the “Memory” window contents by choosing **“Mode > Periodical”** from the pop-up menu which appears by right-clicking on the “Memory” window.

4 Programming

4.1 DataBlaze Programming Utility

4.1.1 Overview

DataBlaze is a full-featured programming utility. DataBlaze offers the following advanced features:

- Memory editing;
- Blank check/erase/program/verify/read operations;
- Project handling;
- One-button, multiple-operations programming (“Auto” feature);
- Serial numbering.

4.1.2 Using DataBlaze

To start the DataBlaze utility select **“Start > Programs > SofTec Microsystems > inDART-One > DataBlaze Programmer”**.

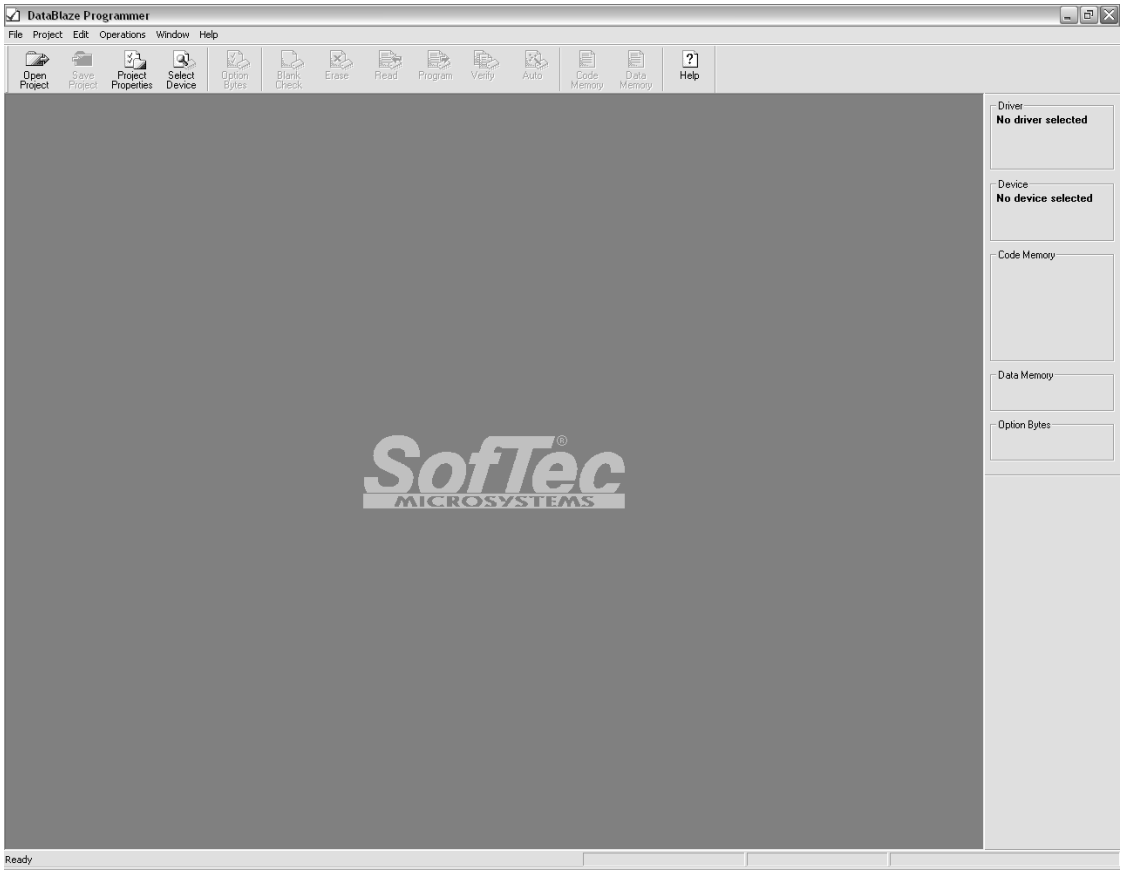

Figure 4.1: The DataBlaze User Interface

First, select the target device you are working with. To do so, select **"Operations > Select Device"** from the DataBlaze main menu. The following dialog box will appear.

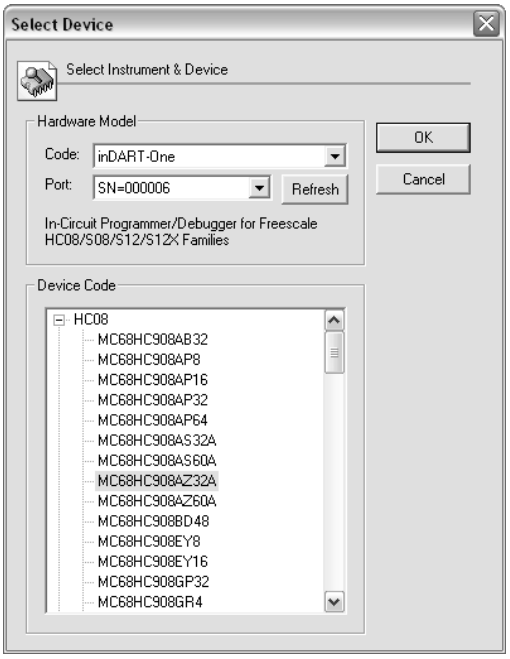

Figure 4.2: Device Selection

If you have more than one inDART-One instrument connected to the USB hub, you can specify which one to use with DataBlaze.

Select the device you are working with in the **"Device Code"** device list and click the **"OK"** button.

Next, open the "Communication Settings" dialog box (**"Operations > Communication Settings"**), and specify target device-specific settings. For more information, please refer to the "Communication Settings" section in the chapter relative to the target family you are working with, later in this manual.

The "Communication Settings" dialog box can be recalled at any time by selecting **"Operations > Communication Settings"** from the DataBlaze main menu.

For more information about the DataBlaze user interface, please refer to the DataBlaze online help (**"Help > Contents"**).

4.1.3 Using HC08 Fast Algorithms

Fast programming algorithms (available as option) reduce significantly the amount of time needed to program HC08 devices.

To unlock fast programming algorithms, you must purchase the appropriate license. See “Unlocking Fast Programming Algorithms” on page 32 for more information.

Once fast programming algorithms are unlocked, they are immediately available to be selected in the “Communication Settings” dialog box.

4.2 MultiBlaze Gang Programming Utility

4.2.1 Overview

MultiBlaze is an easy-to-use multiple programming utility suitable for production environments. MultiBlaze offers the following features:

- Easy-to-use programming interface;
- Blank check/erase/program/verify/read operations;
- One-button, multiple-operations programming (“Auto” feature);
- Statistics and logging.

4.2.2 Starting MultiBlaze

To start the MultiBlaze gang programming utility select **“Start > Programs > SofTec Microsystems > inDART-One > MultiBlaze Programmer”**. The following dialog box will appear.

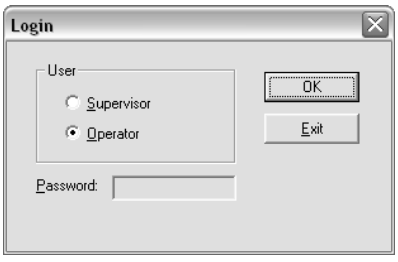

Figure 4.3: MultiBlaze Login

MultiBlaze requires you to log in, either as supervisor or operator, with your username and password. Logging in as supervisor allows you to set up a

project and perform program sessions, while logging in as operator only allows you to perform programming sessions based on an existing project.

Note: *logging in as supervisor requires a password. The default password is* ***"admin"****. You can enter a different password after you have logged in. The default password,* ***"admin"****, can still be used to login, should you forget your own password.*

After you log in, the MultiBlaze main window will appear.

Figure 4.4: MultiBlaze Main Window

4.2.3 Creating a Project

Before to start a programming session, you must first create a project. To create a project, click the **"New Project"** button. You will be guided through a wizard that will help you select your target device, the file to be programmed, and the inDART-One instruments to be used.

Note: *creating or editing a project is only possible if you have logged in as supervisor.*

On the first wizard step, you must specify a name for the project and the path where the project file will be saved. You must also specify the target device you are going to program, and you must specify the communication settings with your target board by clicking the **“Communication Settings”** button, which will open the “Communication Settings” dialog box.

The “Communication Settings” dialog box is specific for the target device you selected. For more information, please refer to the “Communication Settings” section in the chapter relative to the target family you are working with, later in this manual.

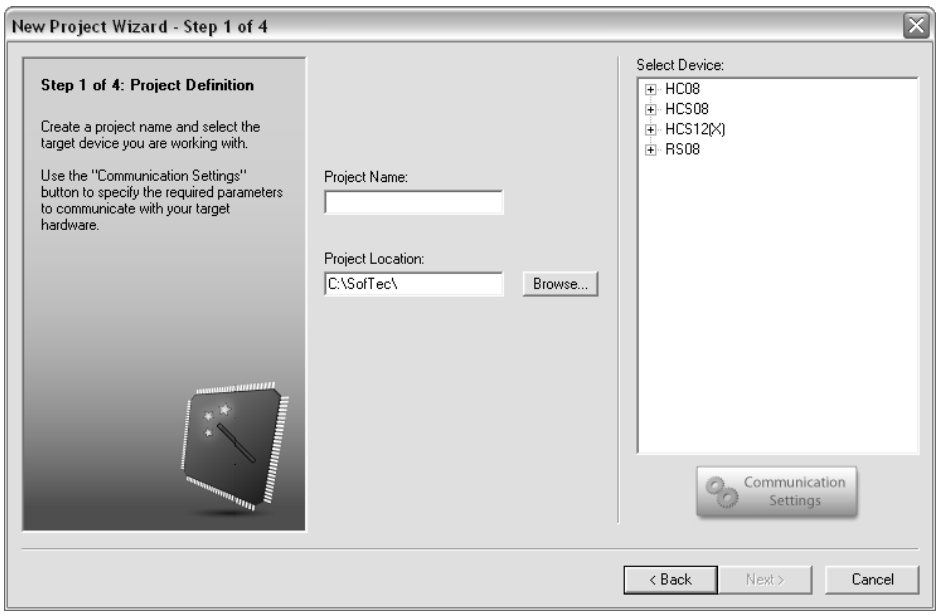

Figure 4.5: MultiBlaze Project Wizard, Step 1

On the second wizard step, you must specify the filename to be programmed into the target device, its format, and the file and buffer offsets, if different than zero.

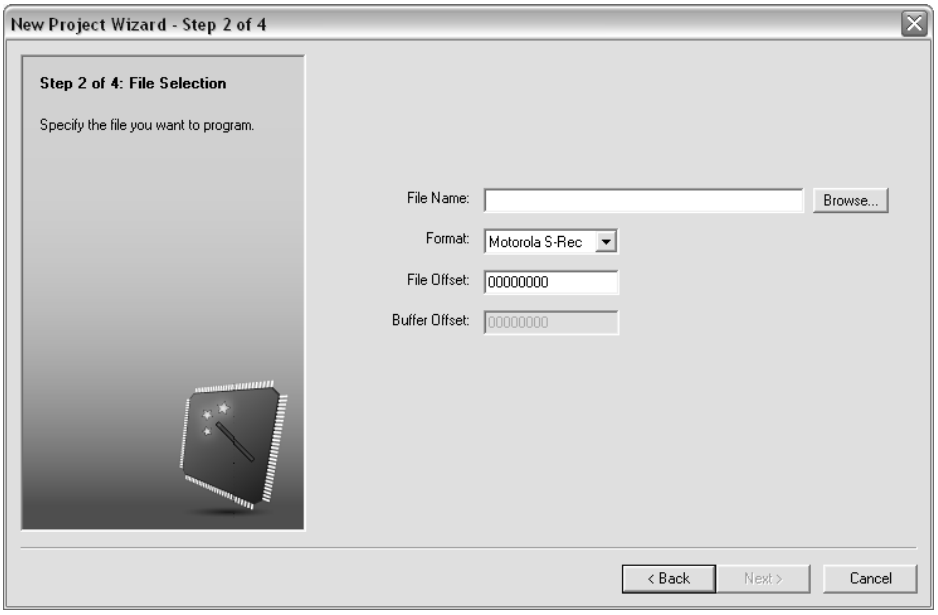

Figure 4.6: MultiBlaze Project Wizard, Step 2

On the third wizard step, you can specify which programming steps will be performed, in sequence, during device programming.

Additionally, you can specify how to start programming. Programming can be started by either clicking the **"Start"** button on the Program dialog box (see below) or by pressing the **"Start"** button on each inDART-One instrument.

Note: *clicking the **"Start"** button in the Program dialog box causes all instruments to start programming simultaneously. Pressing the **"Start"** button on any inDART-One instrument causes only that instrument to start programming.*

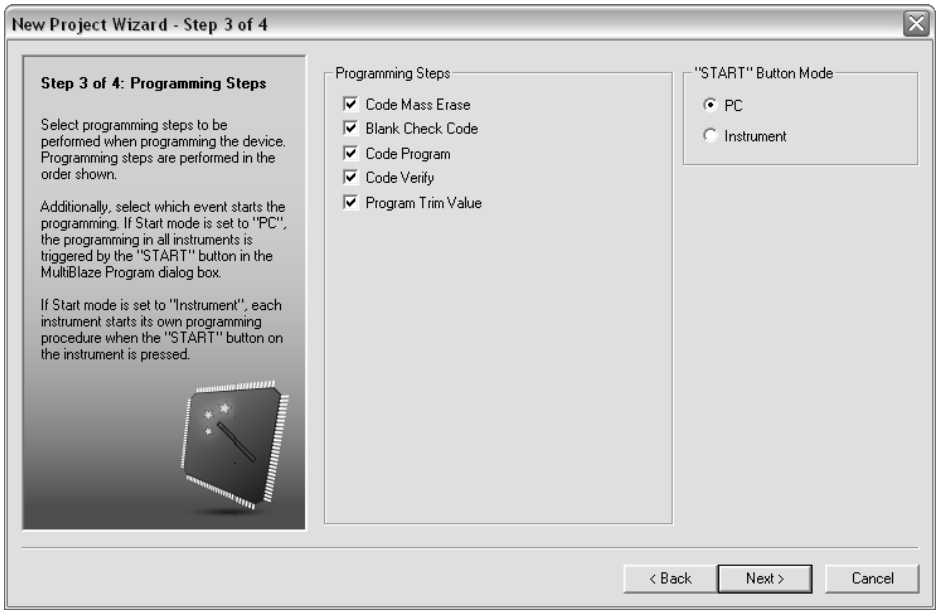

Figure 4.7: MultiBlaze Project Wizard, Step 3

Finally, on the fourth wizard step, you can assign each inDART-One instrument to a “programming node”. Different inDART-One instruments are identified by their serial numbers: you can manually associate their serial numbers to the programming node of choice.

The **“Autofill”** button can be used to automatically detect each inDART-One instrument connected to the USB bus and assign it to a programming node.

Only inDART-One instruments specified in this wizard step will be used during programming sessions.

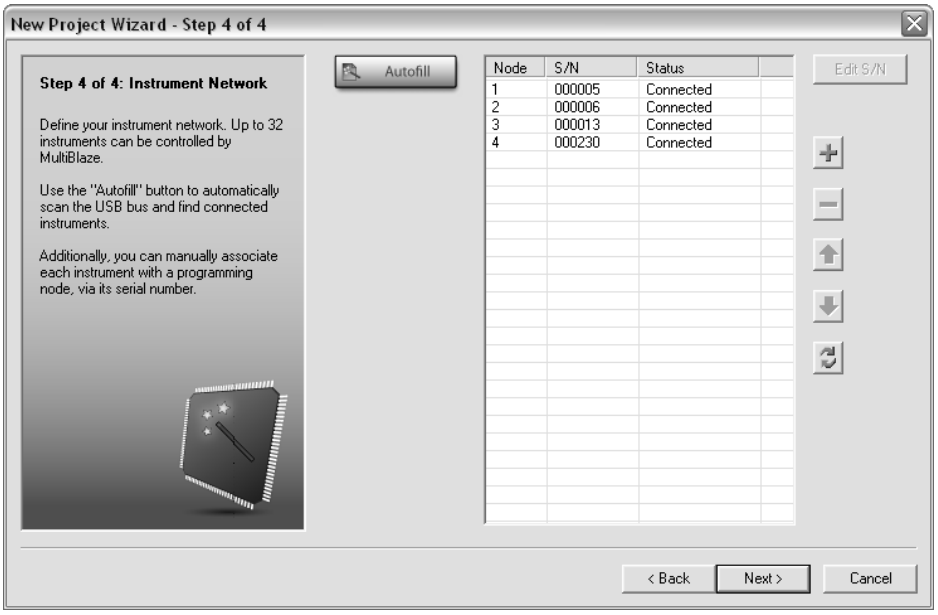

Figure 4.8: MultiBlaze Project Wizard, Step 4

At the end of the wizard, a project file is automatically created with the name and in the location you specified.

4.2.4 Programming

Once a project is loaded (via the **"Open Project"** button) or after a project has been created (via the **"New Project"** button) programming sessions can be performed.

To start a programming session, click the **"Program"** button. The following dialog box will appear.

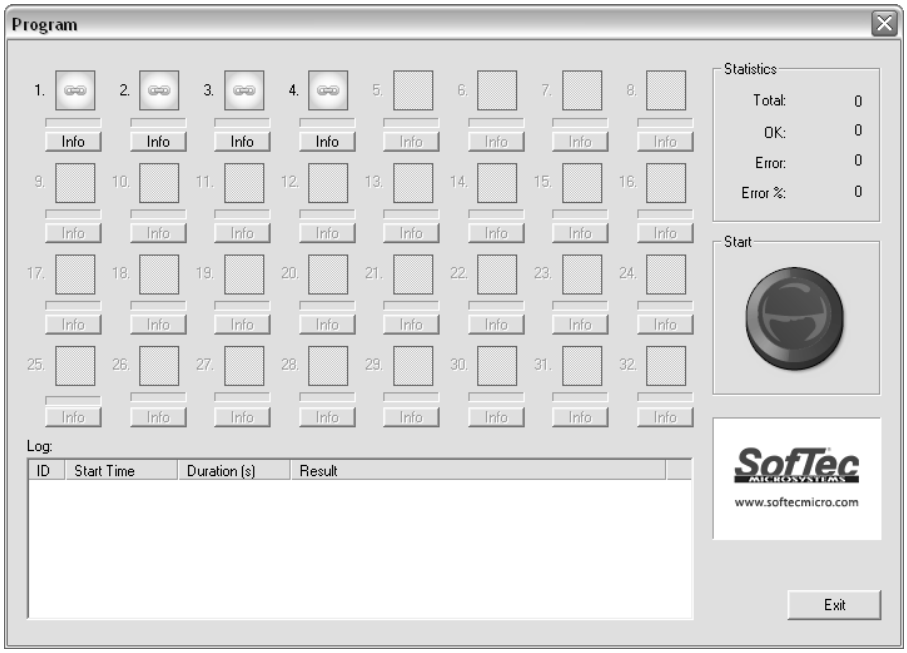

Figure 4.9: MultiBlaze Programming Window

For each of the inDART-One instruments specified in your project, an icon is present which indicates the status of that instrument. Possible states are listed in the table below.

Icon	State	Description
	Connected	The instrument is connected and waiting for a start programming command.
	Not Connected	No link could be established with the instrument.
	Busy	The instrument is programming the target device.
	OK	The instrument has successfully performed all of the programming steps and is waiting for a new start programming command.
	Error	The instrument could not complete all of the programming steps successfully and is waiting for a new start programming command.
	Disabled	The instrument will not take part in the programming (as defined in the current project).

Table 4.1: MultiBlaze Programming States

Depending on the project settings, you can start programming by either clicking the **"Start"** button in the "Program" dialog box (all instruments start programming simultaneously) or press the **"Start"** button on any inDART-One instrument (only that instrument will perform programming).

A log file with the details of the programming session is automatically saved in the project directory.

4.2.5 Using HC08 Fast Algorithms

Fast programming algorithms (available as option) reduce significantly the amount of time needed to program HC08 devices.

To unlock fast programming algorithms, you must purchase the appropriate license. See "Unlocking Fast Programming Algorithms" on page 32 for more information.

Once fast programming algorithms are unlocked, they are immediately available to be selected in the "Communication Settings" dialog box.

4.3 BDM Programming Notes

- The “Mass Erase” operation always blanks the device (even if the device is protected or secured) and “unsecures” the device (the FLASH Options/Security Byte location is programmed with 0xFE).
- The “Blank Check” operation doesn’t blank check the FLASH Options/Security Byte location.
- The “Program” operation automatically verifies the programmed data, by reading back the programmed data and checking it against the buffer sent to the target device. The “Verify” operation is much more secure (but slower), since it reads back the programmed data and checks it against the data buffer present in the host PC.
- In case of verifying error, please verify the value programmed to the FLASH Options/Security Byte location. The bit 0 of this byte is always programmed to 0, so any attempt to program it to 1 will cause a verifying error.
- The “Read”, “Program” and “Verify” operations are performed (when possible) by setting the target microcontroller’s PLL peripheral so that the maximum BDM communication speed is achieved.
- In the “Auto” operation, a “Run” option is available which, if enabled, resets the microcontroller and runs the user application at the end of programming.

5 Working with HC08 Devices

5.1 Debugging Limitations

Since inDART-One is based on the in-chip debugging features of the HC08 family of microcontrollers, some hardware and software limitations apply. The main ones are listed below; for the complete list of limitations please refer to the microcontroller's data sheet.

- The pin dedicated to the host communication is reserved—in particular, the corresponding bit in the Data Direction Register must not be changed (must be left to input);
- The Break Module peripheral is reserved, and only one hardware breakpoint is available—however, you can insert a **"swi"** instruction into your code to generate a software breakpoint;
- The **"swi"** instruction is reserved and can be only used to generate a software breakpoint;
- Step commands which involve the execution of two or more Assembly instructions waste one hardware breakpoint—therefore, if one breakpoint was already set by the user, the step command cannot be executed;
- The **"Halt"** debugging command (in the CodeWarrior HC08 user interface) will not work unless the IRQ interrupt is properly handled;
- 13 bytes of stack are wasted by the on-chip monitor—therefore the addresses from SP-13 to SP are reserved;
- Registers which affect the target microcontroller's clock speed must not be changed during debugging sessions.

5.2 Communication Settings

inDART-One must be configured properly so that MON08 communication with the target device can be established correctly.

Communication settings are defined through the “Communication Settings” dialog box, available both in CodeWarrior and in the DataBlaze and MultiBlaze programming utilities.

The dialog box is divided into four sections: “MON08 Configuration”, “Power Settings”, “Programming” and “Trimming”. All of the parameters must be carefully set, otherwise unsuccessful operations will result.

5.2.1 MON08 Configuration

This section shows you what lines of the target microcontroller must be tied to the MON4-MON8 lines of the MON08 connector. Depending on the microcontroller, either the MON4 or MON5 line is dedicated to the single-wire communication (which is reserved during debugging).

The other MON lines must be driven at specific values at reset. inDART-One automatically drives all of the required lines upon reset, and releases them soon afterwards (except for the single-wire communication line which is always reserved). Please note that each microcontroller implements the MON08 interface on different ports and pins.

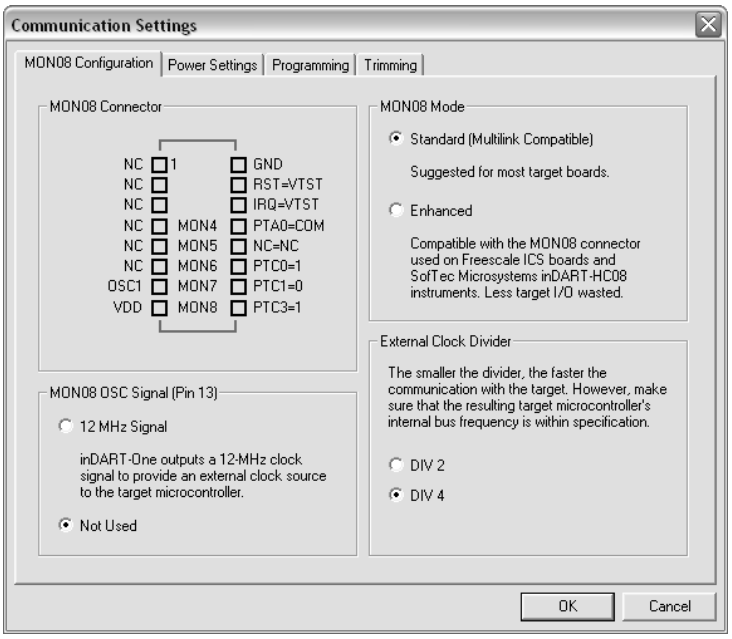

Figure 5.1: MON08 Communication Settings: MON08 Configuration

inDART-One's MON08 connector can behave as a standard (Multilink compatible) MON08 connector or as an enhanced MON08 connector. The former is suggested for most target boards, while the latter is compatible with SofTec Microsystems inDART-HC08 and can be specifically used in conjunction with Freescale ICS boards or with SofTec Microsystems demo boards. In this case less target I/O lines will be wasted.

For device-specific information, see "MON08 Target Connections" later in this chapter.

The **"MON08 OSC Signal"** parameter (available in the standard MON08 mode) allows you to select whether inDART-One generates a clock signal at the pin 13 of the MON08 connector. This signal can be used as an auxiliary clock source, and is particularly useful when the target microcontroller requires an external clock which is not otherwise available on the target board.

The **"External Clock Divider"** parameter should always be set to **"DIV 4"** in order to guarantee that, in monitor mode, all of the target microcontroller's peripherals run at the same speed they do in user mode. If, however, you need to speed up program execution, you can set this parameter to **"DIV 2"**, therefore doubling the microcontroller's speed in monitor mode. Note, however, that not all peripherals will work at this doubled speed.

5.2.2 Power Settings

The parameters in this group are used by inDART-One to determine how to force the target microcontroller to enter the monitor mode.

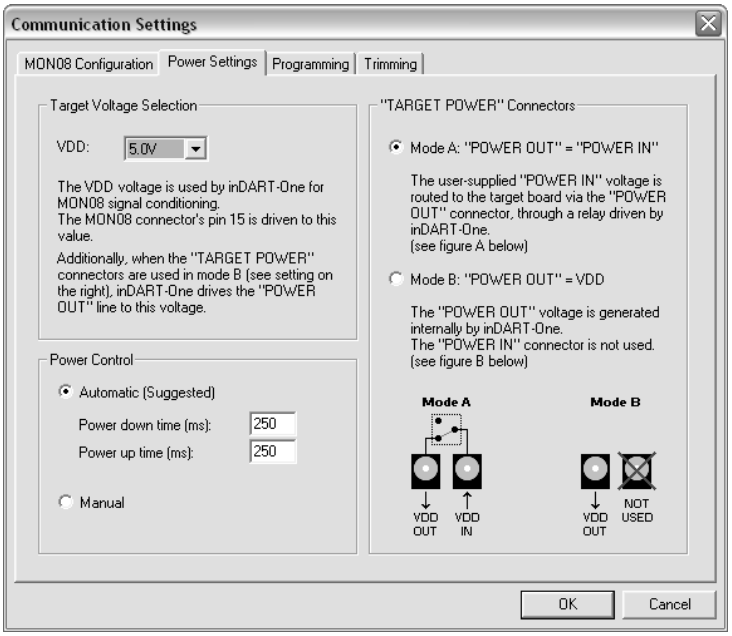

Figure 5.2: MON08 Communication Settings: Power Settings

5

The **"Target Voltage Selection"** parameter must be set to the actual operating voltage of the target microcontroller. This parameter is fundamental for proper operations, since it affects (among other things) the signal conditioning of MON08 lines.

The **"Power Control"** parameter specifies how the target board is powered off/on. MON08 debugging and programming requires that the target microcontroller is powered off and on in order to enter the monitor mode.

Depending on your target hardware, two possibilities are available:

- **Automatic.** Select this option if you want inDART-One to automatically turn off and on the target system in order to enter the monitor mode. In order for this option to work, the target system must be powered through the "TARGET POWER" connectors (see below).
- **Manual.** Select this option if your target system cannot be powered by the "TARGET POWER" connectors. You will be asked, when necessary, to power off and on the target board manually.

When **"Automatic"** is selected, you must additionally set two parameters:

- **"Power down time"**. The time it takes for the power voltage to drop below 100 mV.
- **"Power up time"**. The time it takes for target microcontroller to get ready to communicate after the V_{DD} voltage becomes high.

The **"TARGET POWER Connectors"** parameter specifies how the two "TARGET POWER" connectors work.

When the first option ("Mode A") is selected, the user-supplied voltage at the "TARGET POWER IN" connector is routed to the target board via the "TARGET POWER OUT" connector, through an internal relay driven by inDART-One.

When the second option ("Mode B") is selected, the voltage at the "TARGET POWER OUT" connector is internally generated by inDART-One, and the "TARGET POWER IN" connector is not used. The generated voltage is as specified by the **"Target Voltage Selection"** parameter.

Note: *when "Mode B" is selected, only the central pin of the "TARGET POWER OUT" connector is used, while the outer sleeve is not connected. The GND reference is taken from the "MON08" cable.*

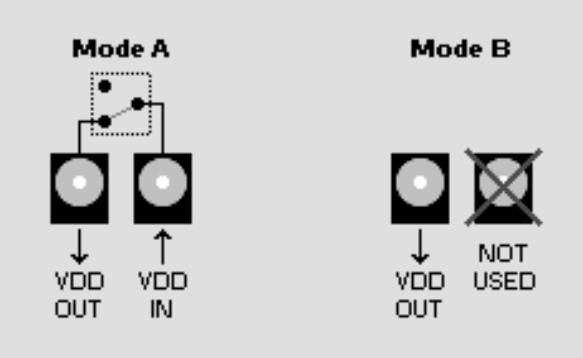

Figure 5.3: MON08 Communication Settings: Target Power Connectors Modes

Note: *when the MON08 connector is configured as* ***"Standard (Multilink Compatible)"*** *connector, the target voltage is also generated by inDART-One at the pin 15 of the MON08 connector.*

5.2.3 Programming

Programming parameters mainly determine the speed at which programming operations are performed.

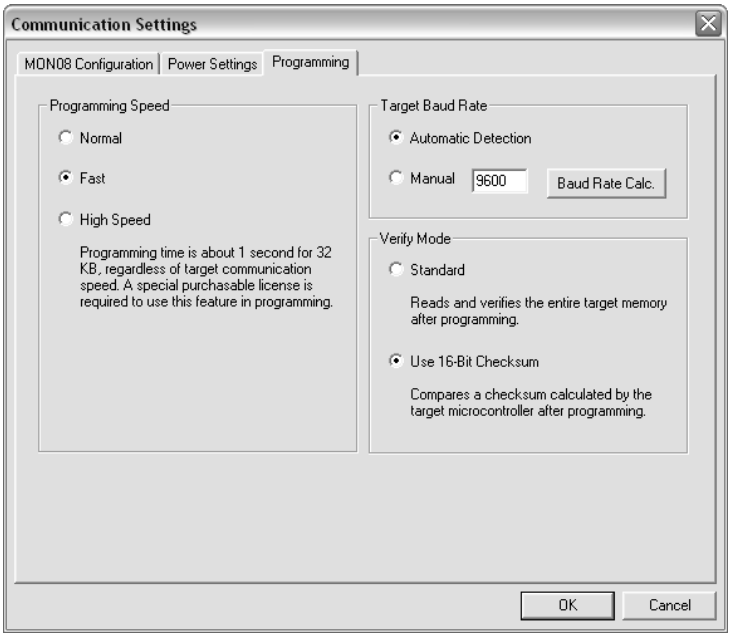

Figure 5.4: MON08 Communication Settings: Programming

The **"Programming Speed"** parameter allows you to select among four programming speeds.

Note: *the* ***"High Speed"*** *option is only available after purchasing the appropriate license. See "Unlocking Fast Programming Algorithms" on page 32 for more information.*

The **"Target Baud Rate"** parameter specifies the MON08 serial line's baud rate used to enter the monitor mode. inDART-One can automatically determine the correct target baud rate in the most common situations; the **"Automatic Detection"** is the default option. However, if for some reasons the automatic detection doesn't work, you still have the possibility to select the baud rate manually.

Note: *when using the manual baud rate selection, keep in mind that the correct target baud rate value to use critically depends on the target hardware; different target configurations require different baud rate values. It is highly recommended that you consult the data sheet of the target microcontroller (the section about entering the monitor mode) in order to calculate the correct baud rate to set. An improper target baud rate value may result in programming errors.*

As an aid, a baud rate calculator is provided. Just click on the **"Baud Rate Calc."** button and enter the microcontroller external clock frequency and the appropriate baud rate will be calculated for you. Please note, however, that this method only works if the IRQ line of the microcontroller is tied to the IRQ line of the MON08 connector. In all other cases, again, you must consult the data sheet of the target microcontroller.

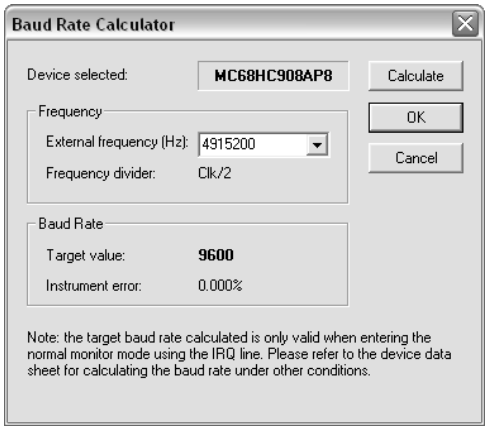

Figure 5.5: MON08 Communication Settings: Baud Rate Calculator

The **“Verify Mode”** parameter specifies how the verifying of the programmed data will be performed. The standard verify mode reads back all of the programmed data and verifies it with the data in the PC buffer, while the checksum mode is based on a checksum of the programmed data.

The standard verify mode is the safest mode, while the checksum verify mode is the fastest.

Note: *the **“Verify Mode”** parameter is only available when using the DataBlaze or MultiBlaze user interface.*

5.2.4 Trimming

inDART-One allows you to enable the target microcontroller’s internal oscillator calibration (trimming).

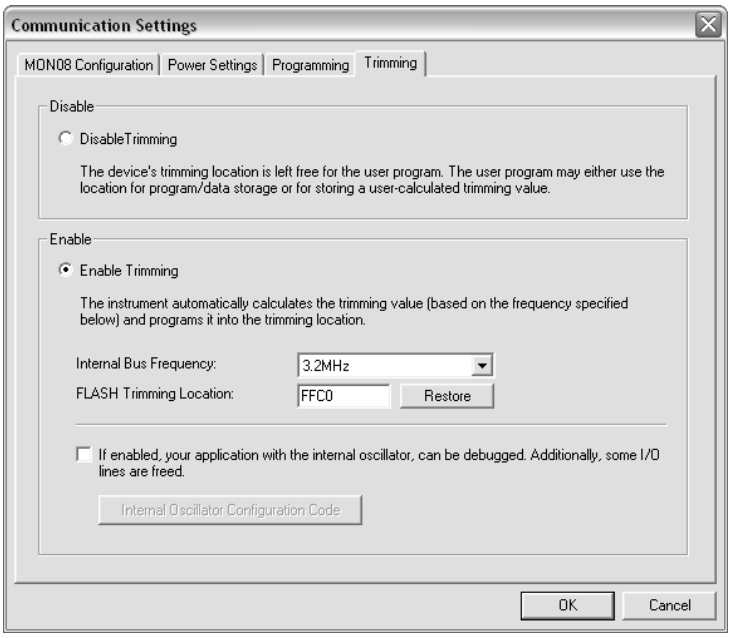

Figure 5.6: MON08 Communication Settings: Trimming

The trimming parameters are specific for the selected device. If the selected device incorporates the OSC module, then you are allowed to select the OSC frequency to be calibrated, through the **“Internal Bus Frequency”** parameter (if more than one internal bus clock frequency is available on that device).

If, instead, the selected device incorporates the ICG module, then you are allowed to select the ICG frequency to be calibrated (**“ICG Frequency”** parameter), with the corresponding multiplier.

In both cases, it is possible to specify whether to save the calculated trimmed value in the default location (the location suggested either by Freescale or SofTec Microsystems, restorable at any moment via the **“Restore”** button) or into a different location (**“Flash Trimming Location”** parameter).

5.3 MON08 Target Connections

5.3.1 Standard MON08 Connections

The following diagram illustrates a typical connection between your target microcontroller and the MON08 connector, when inDART-One is configured to use the standard (Multilink compatible) MON08 connector.

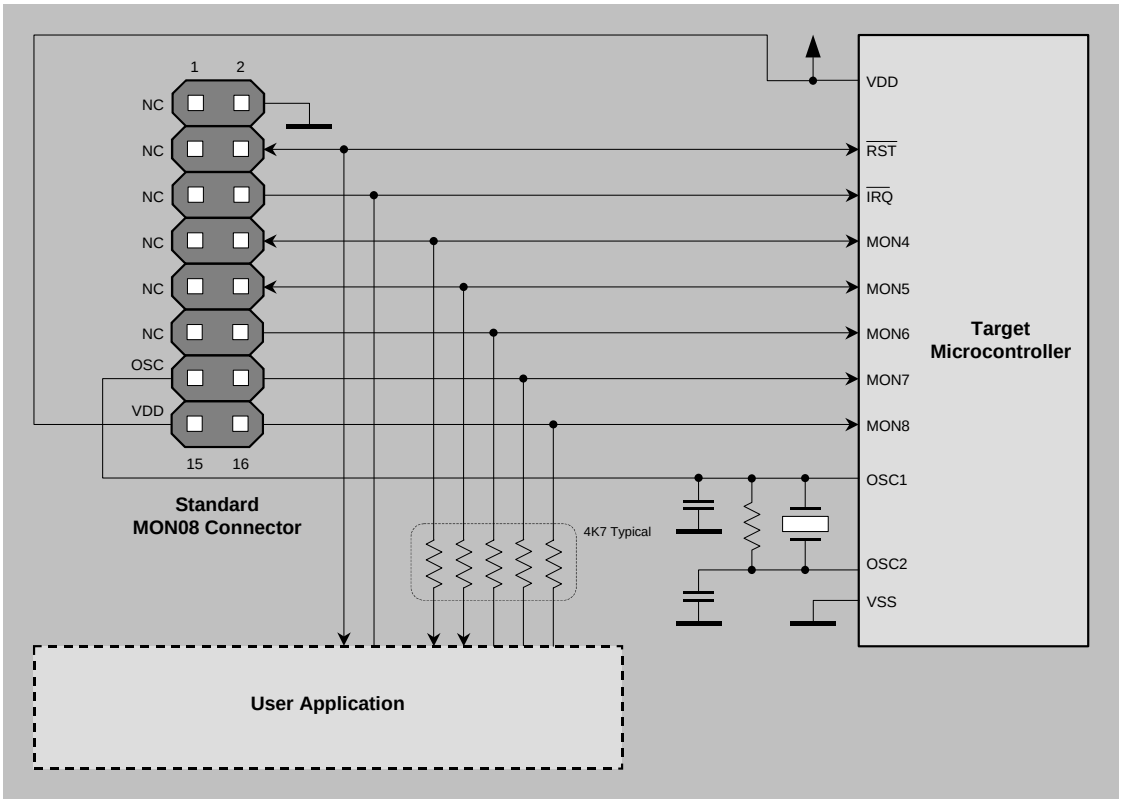

Figure 5.7: Typical MON08 Target Connections (Standard Mode)

Note: *the target V_{DD} line needs to be driven correctly at reset. When using the MON08 connector in standard mode, inDART-One can automatically supply the connector's pin 15 with the appropriate voltage.*

Note: *the MON08 OSC signal (pin 13 of the MON08 connector) can be used instead as an auxiliary clock source, and is particularly useful when the target microcontroller requires an external clock which is not available on the target board.*

Note: *each microcontroller implements the MON08 interface on different ports and pins. MON4-MON8 lines must be therefore tied to the appropriate pins of the specific target microcontroller.*

5.3.2 Enhanced MON08 Connections

The following diagram illustrates a typical connection between your target microcontroller and the MON08 connector, when inDART-One is configured to use the enhanced (SofTec Microsystems) MON08 connector.

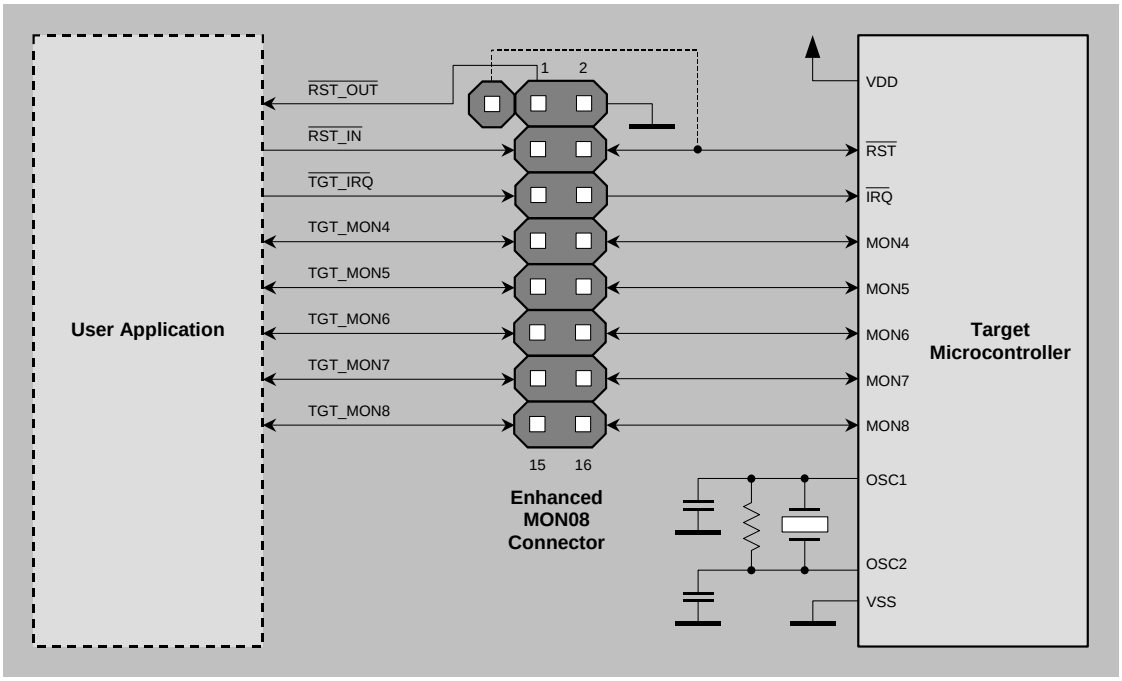

Figure 5.8: Typical MON08 Target Connections (Enhanced Mode)

Note: *the target V_{DD} line needs to be driven correctly at reset. When using the MON08 connector in enhanced mode, the correct voltage can be taken from inDART-One's "TARGET POWER" connectors.*

Note: *each microcontroller implements the MON08 interface on different ports and pins. MON4-MON8 lines must be therefore tied to the appropriate pins of the specific target microcontroller.*

If your target board implements the enhanced MON08 connector, to work without inDART-One connected then the MON08 connector must be jumpered as shown below on the left.

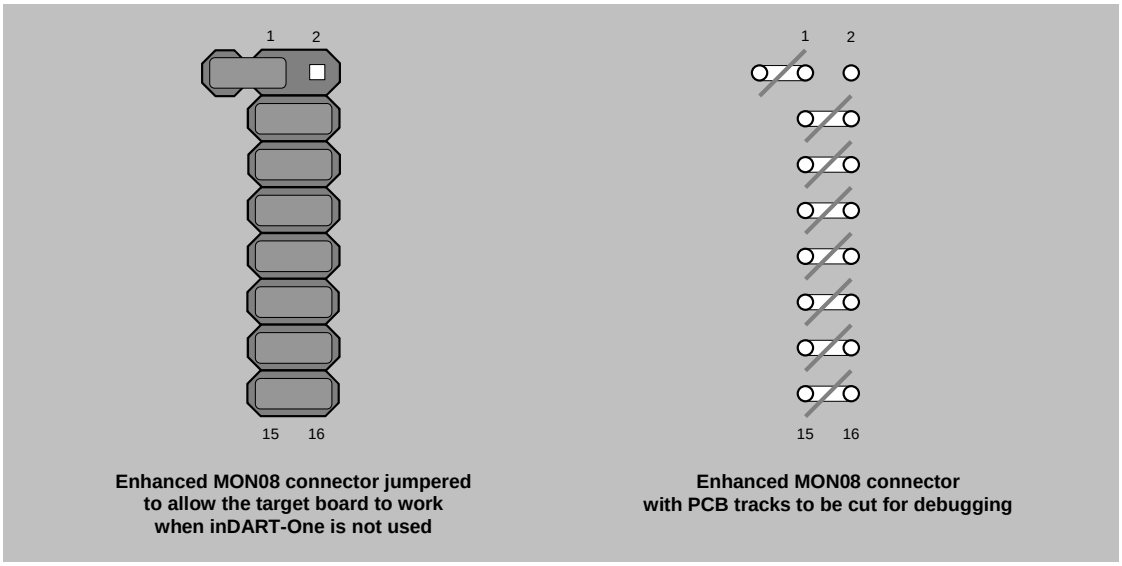

Figure 5.9: Jumpered Enhanced MON08 Connector

Alternatively, PCB tracks can be routed as shown on the right. In the target board you use for debugging, cut the tracks as indicated.

5.3.3 MC68HC908AB Family Connections

Standard MON08 connections for MC68HC908AB family devices are shown in the figure below.

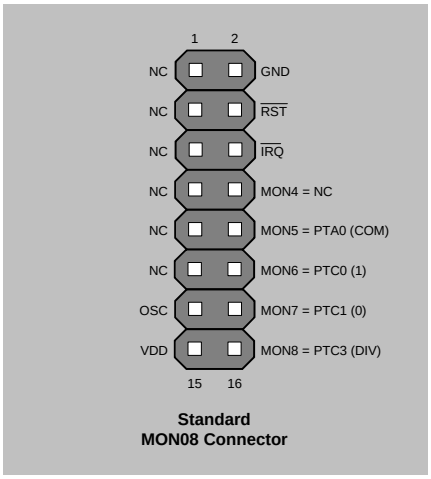

Figure 5.10: Standard MON08 Connections for the MC68HC908AB Family

5.3.4 MC68HC908AP Family Connections

Standard MON08 connections for MC68HC908AP family devices are shown in the figure below.

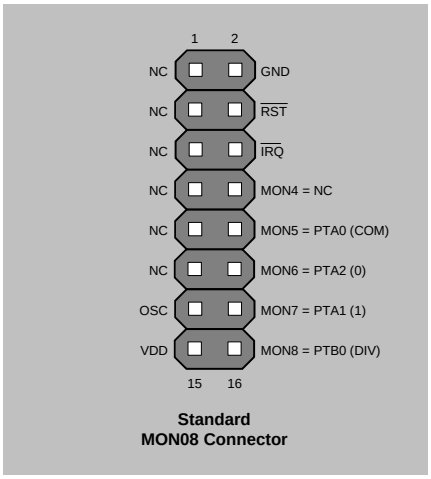

Figure 5.11: Standard MON08 Connections for the MC68HC908AP Family

5.3.5 MC68HC908AS Family Connections

Standard MON08 connections for MC68HC908AS family devices are shown in the figure below.

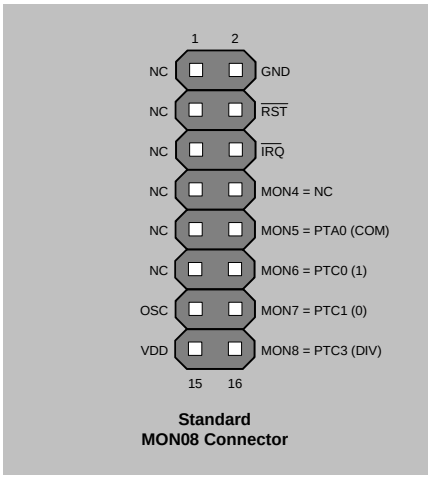

Figure 5.12: Standard MON08 Connections for the MC68HC908AS Family

5.3.6 MC68HC908AZ Family Connections

Standard MON08 connections for MC68HC908AZ family devices are shown in the figure below.

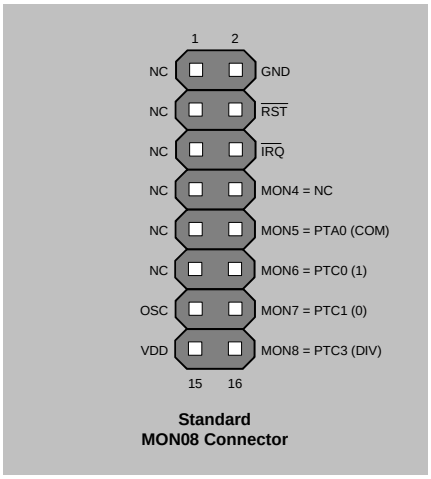

Figure 5.13: Standard MON08 Connections for the MC68HC908AZ Family

5.3.7 MC68HC908BD Family Connections

Standard MON08 connections for MC68HC908BD family devices are shown in the figure below.

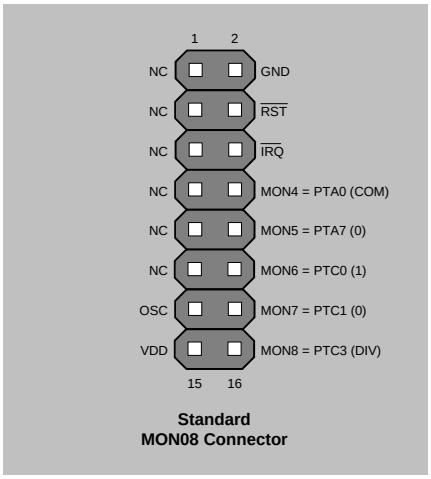

Figure 5.14: Standard MON08 Connections for the MC68HC908BD Family

5.3.8 MC68HC908EY Family Connections

Standard MON08 connections for MC68HC908EY family devices are shown in the figure below.

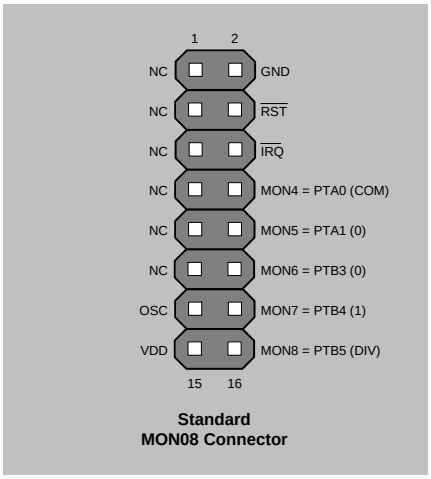

Figure 5.15: Standard MON08 Connections for the MC68HC908EY Family

5.3.9 MC68HC908GP Family Connections

Standard MON08 connections for MC68HC908GP family devices are shown in the figure below.

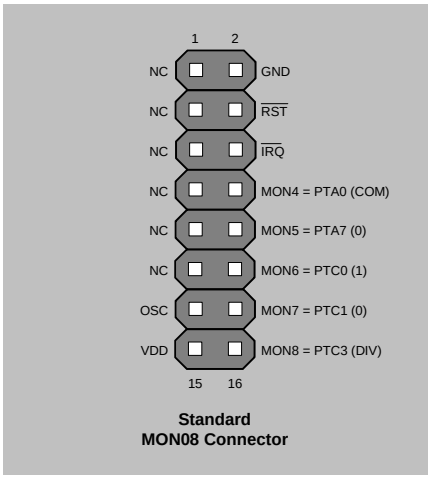

Figure 5.16: Standard MON08 Connections for the MC68HC908GP Family

5.3.10 MC68HC908GR4/4A/8/8A Connections

Standard MON08 connections for MC68HC908GR4/4A/8/8A devices are shown in the figure below.

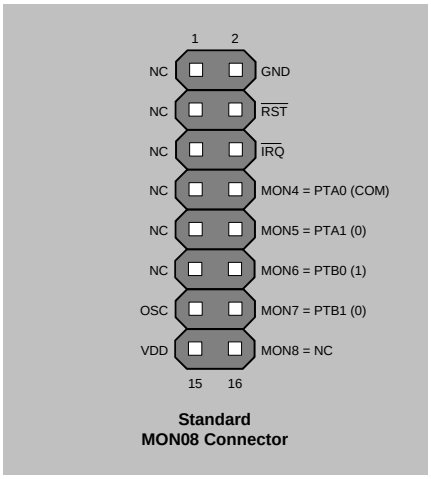

Figure 5.17: Standard MON08 Connections for MC68HC908GR4/4A/8/8A Devices

5.3.11 MC68HC908GR16 Connections

Standard MON08 connections for the MC68HC908GR16 device are shown in the figure below.

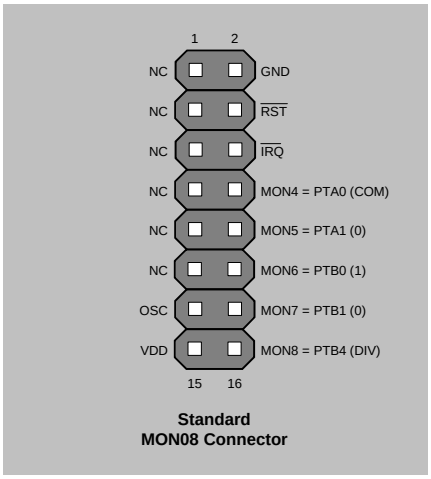

Figure 5.18: Standard MON08 Connections for the MC68HC908GR16 Device

5.3.12 MC68HC908GR16A/32A/48A/60A Connections

Standard MON08 connections for MC68HC908GR16A/32A/48A/60A devices are shown in the figure below.

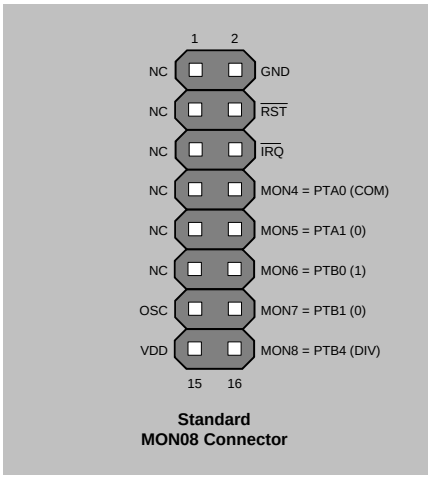

Figure 5.19: Standard MON08 Connections for MC68HC908GR16A/32A/48A/60A Devices

5.3.13 MC68HC908GT Family Connections

Standard MON08 connections for MC68HC908GT family devices are shown in the figure below.

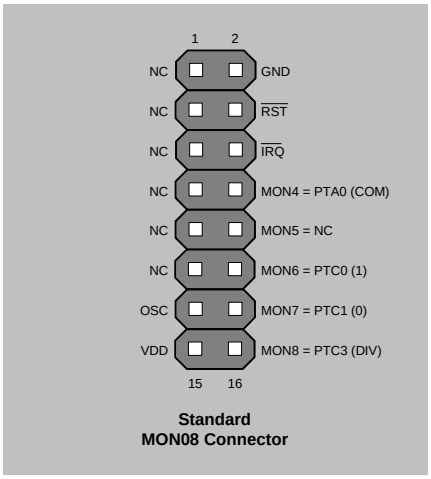

Figure 5.20: Standard MON08 Connections for the MC68HC908GT Family

5.3.14 MC68HC908GZ Family Connections

Standard MON08 connections for MC68HC908GZ family devices are shown in the figure below.

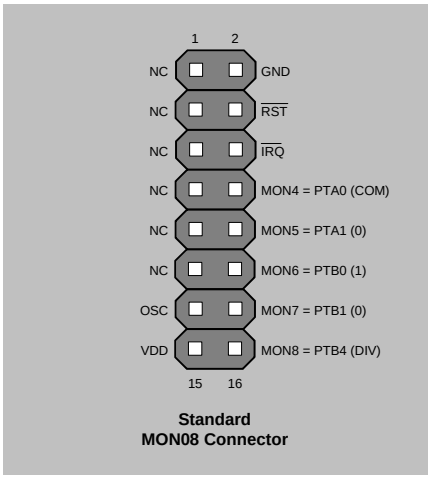

Figure 5.21: Standard MON08 Connections for the MC68HC908GZ Family

5.3.15 MC68HC908JB8 Connections

Standard MON08 connections for the MC68HC908JB8 device are shown in the figure below.

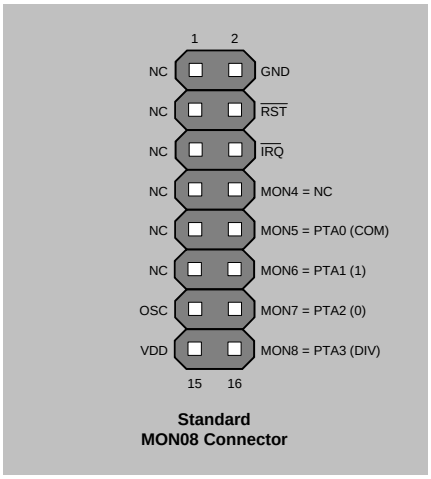

Figure 5.22: Standard MON08 Connections for the MC68HC908JB8 Device

5.3.16 MC68HC908JB12/16 Connections

Standard MON08 connections for MC68HC908JB12/16 devices are shown in the figure below.

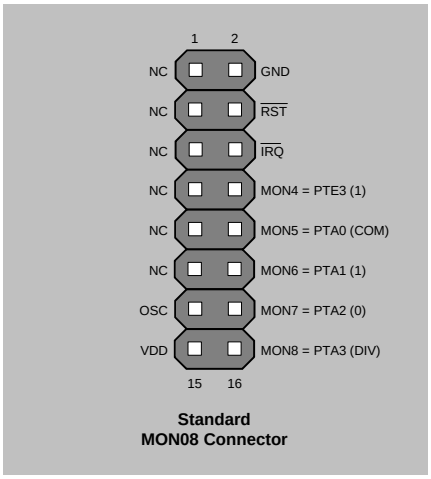

Figure 5.23: Standard MON08 Connections for MC68HC908JB12/16 Devices

5.3.17 MC68HC908JG Connections

Standard MON08 connections for MC68HC908JG family devices are shown in the figure below.

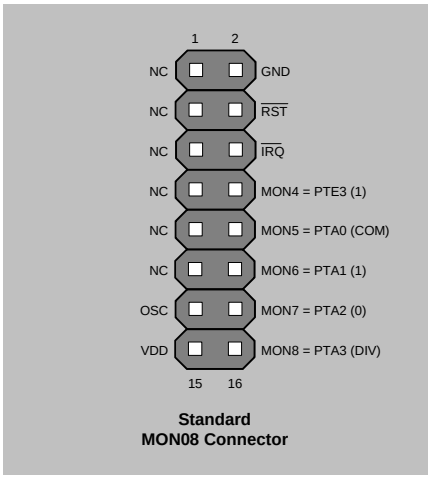

Figure 5.24: Standard MON08 Connections for the MC68HC908JG Family

5.3.18 MC68H(L)C908JK Family Connections

Standard MON08 connections for MC68H(L)C908JK family devices are shown in the figure below.

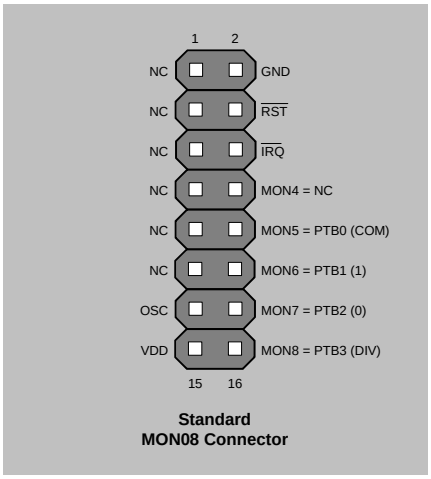

Figure 5.25: Standard MON08 Connections for the MC68H(L)C908JK Family

5.3.19 MC68H(L)C908JL Family Connections

Standard MON08 connections for MC68H(L)C908JL family devices are shown in the figure below.

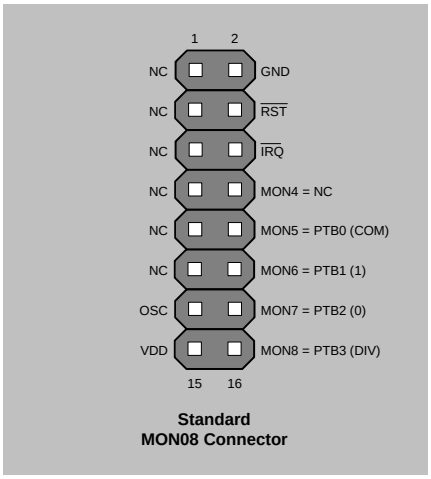

Figure 5.26: Standard MON08 Connections for the MC68H(L)C908JL Family

5.3.20 MC68HC908JW Family Connections

Standard MON08 connections for MC68HC908JW family devices are shown in the figure below.

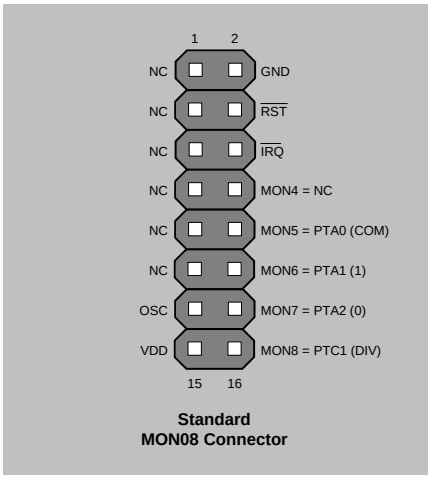

Figure 5.27: Standard MON08 Connections for the MC68HC908JW Family

5.3.21 MC68HC908KX Family Connections

Standard MON08 connections for MC68HC908KX family devices are shown in the figure below.

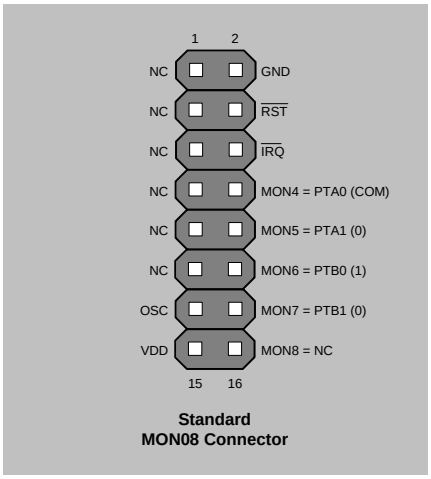

Figure 5.28: Standard MON08 Connections for the MC68HC908KX Family

5.3.22 MC68HC908LB Family Connections

Standard MON08 connections for MC68HC908LB family devices are shown in the figure below.

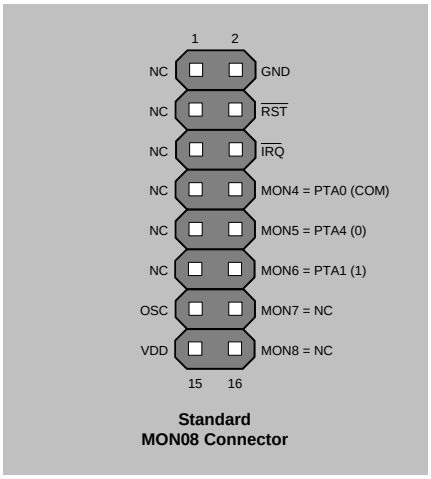

Figure 5.29: Standard MON08 Connections for the MC68HC908LB Family

5.3.23 MC68HC908LD Family Connections

Standard MON08 connections for MC68HC908LD family devices are shown in the figure below.

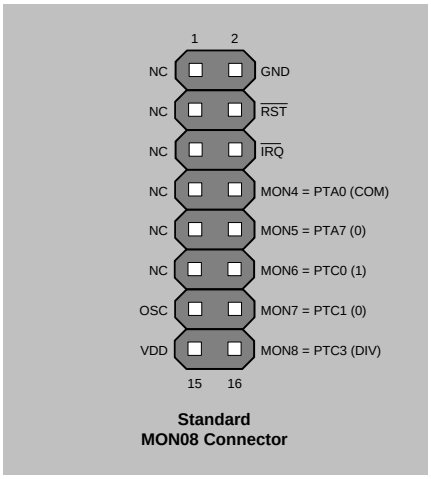

Figure 5.30: Standard MON08 Connections for the MC68HC908LD Family

5.3.24 MC68HC908LJ Family Connections

Standard MON08 connections for MC68HC908LJ family devices are shown in the figure below.

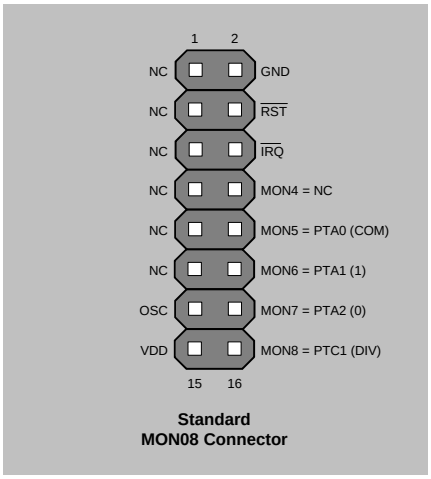

Figure 5.31: Standard MON08 Connections for the MC68HC908LJ Family

5.3.25 MC68HC908LK Family Connections

Standard MON08 connections for MC68HC908LK family devices are shown in the figure below.

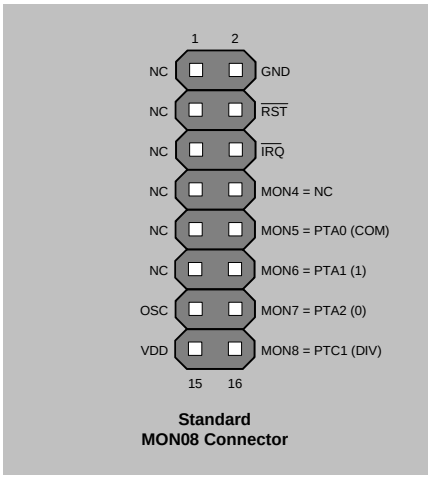

Figure 5.32: Standard MON08 Connections for the MC68HC908LK Family

5.3.26 MC68HC908LT Family Connections

Standard MON08 connections for MC68HC908LT family devices are shown in the figure below.

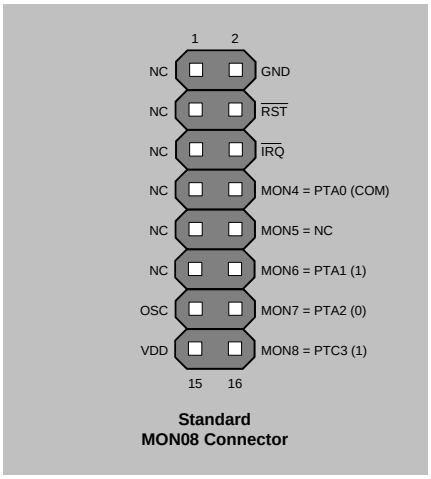

Figure 5.33: Standard MON08 Connections for the MC68HC908LT Family

5.3.27 MC68HC908LV Family Connections

Standard MON08 connections for MC68HC908LV family devices are shown in the figure below.

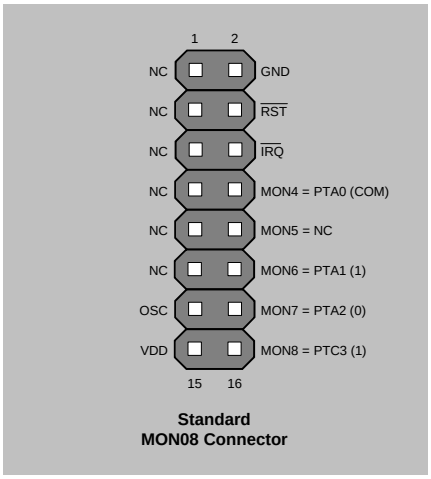

Figure 5.34: Standard MON08 Connections for the MC68HC908LV Family

5.3.28 MC68HC908MR Family Connections

Standard MON08 connections for MC68HC908MR family devices are shown in the figure below.

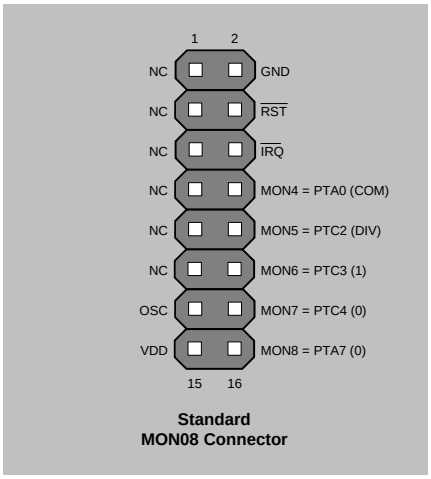

Figure 5.35: Standard MON08 Connections for the MC68HC908MR Family

5.3.29 MC68HC908QB Family Connections

Standard MON08 connections for MC68HC908QB family devices are shown in the figure below.

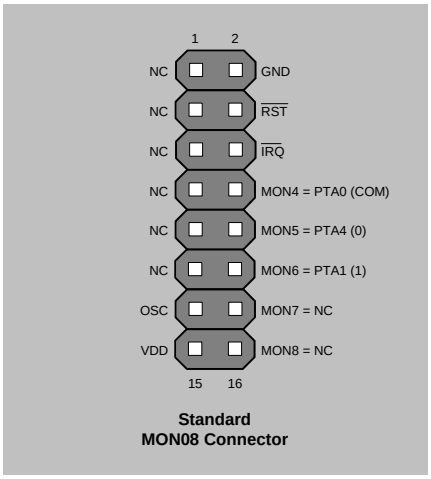

Figure 5.36: Standard MON08 Connections for the MC68HC908QB Family

5.3.30 MC68HC908QC Family Connections

Standard MON08 connections for MC68HC908QC family devices are shown in the figure below.

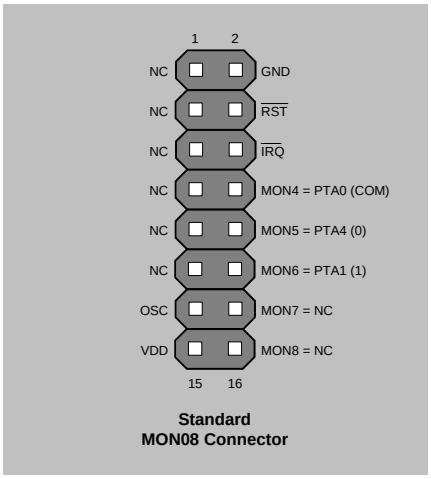

Figure 5.37: Standard MON08 Connections for the MC68HC908QC Family

5.3.31 MC68HC908QF Family Connections

Standard MON08 connections for MC68HC908QF family devices are shown in the figure below.

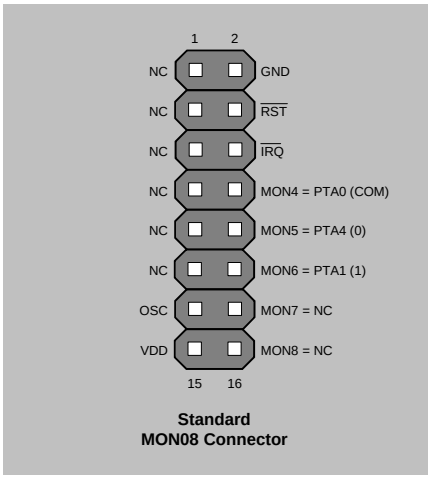

Figure 5.38: Standard MON08 Connections for the MC68HC908QF Family

5.3.32 MC68HC908QL Family Connections

Standard MON08 connections for MC68HC908QL family devices are shown in the figure below.

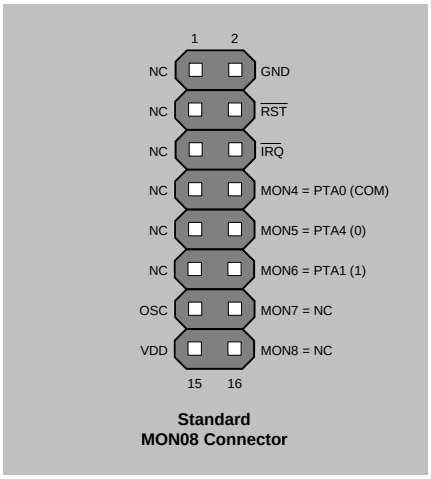

Figure 5.39: Standard MON08 Connections for the MC68HC908QL Family

5.3.33 MC68H(L)C908QT Family Connections

Standard MON08 connections for MC68H(L)C908QT family devices are shown in the figure below.

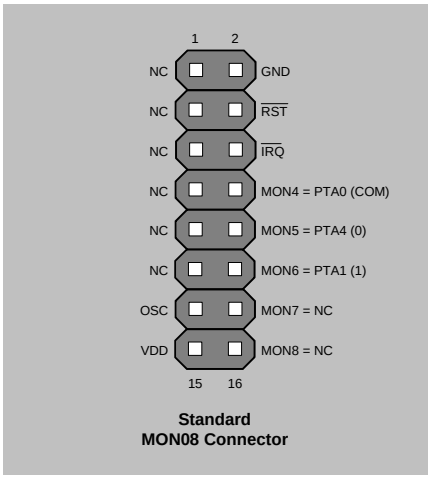

Figure 5.40: Standard MON08 Connections for the MC68HC908QT Family

5.3.34 MC68H(L)C908QY Family Connections

Standard MON08 connections for MC68H(L)C908QY family devices are shown in the figure below.

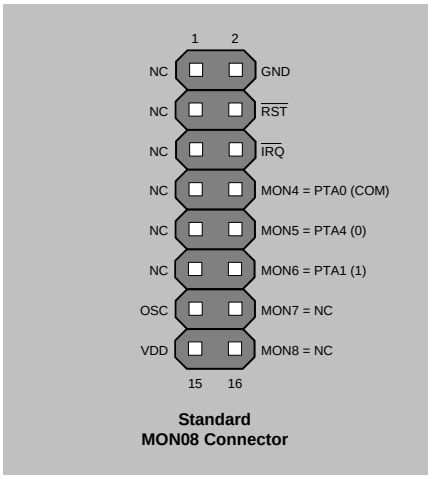

Figure 5.41: Standard MON08 Connections for the MC68HC908QY Family

5.3.35 MC68HC908RF Family Connections

Standard MON08 connections for MC68HC908RF family devices are shown in the figure below.

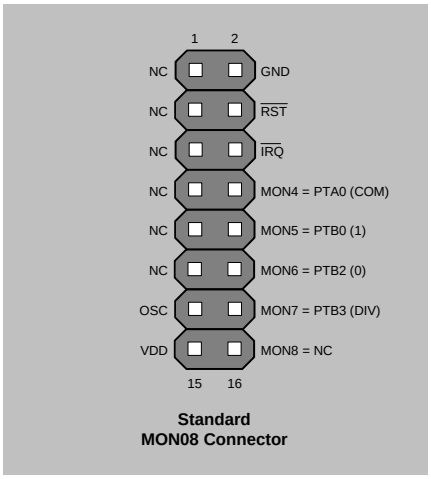

Figure 5.42: Standard MON08 Connections for the MC68HC908RF Family

5.3.36 MC68HC908RK Family Connections

Standard MON08 connections for MC68HC908RK family devices are shown in the figure below.

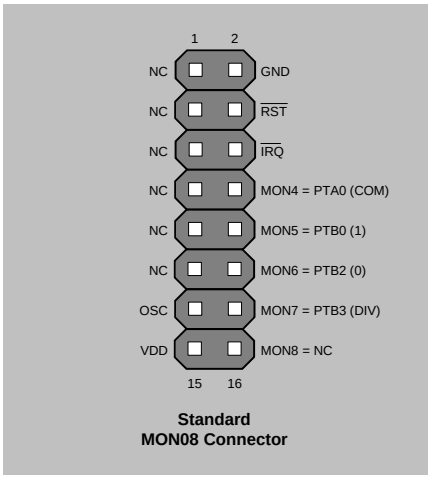

Figure 5.43: Standard MON08 Connections for the MC68HC908RK Family

5.3.37 MC68HC908SR Family Connections

Standard MON08 connections for MC68HC908SR family devices are shown in the figure below.

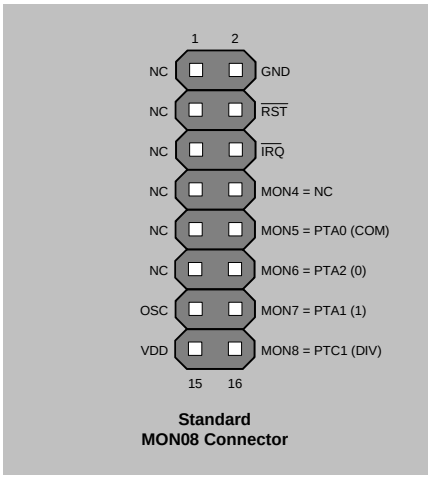

Figure 5.44: Standard MON08 Connections for the MC68HC908SR Family

6 Working with HCS08 Devices

6.1 Communication Settings

inDART-One must be configured properly so that BDM communication with the target device can be established correctly.

Communication settings are defined through the “Communication Settings” dialog box, available both in CodeWarrior and in the DataBlaze and MultiBlaze programming utilities.

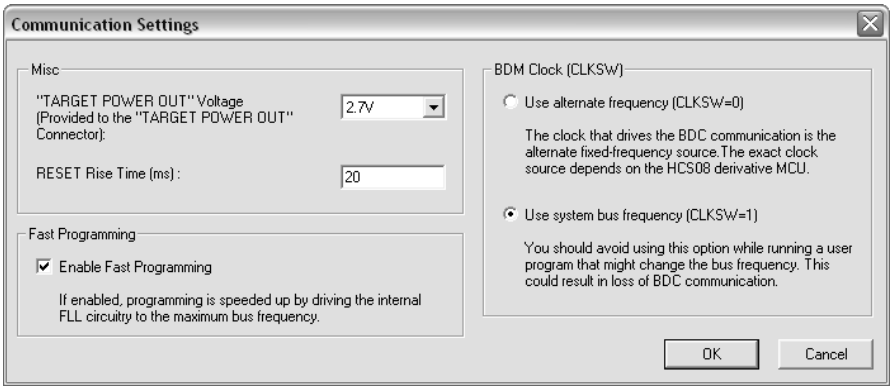

Figure 6.1: The BDM Communication Settings Dialog Box for HCS08 Devices

6.1.1 BDM Clock

The BDM communication speed depends on a clock source which, in turn, is selected by the CLKSW bit in the Status register. If the CLKSW bit is set to 1, the BDM communication clock source is the microcontroller’s bus frequency; if the CLKSW bit is set to 0, the BDM communication clock source is a constant clock source, which can vary depending on the specific HCS08 derivative. In the case of the MC9S08GB60, for example, this constant clock source is a 8 MHz internal clock. Other derivatives may use the external crystal frequency.

Which CLKSW setting to use depends on the target system and on the user application. The idea is to set the CLKSW bit so that the highest (and less subject to changes) clock frequency is used for the BDM communication. A

low clock frequency will result in slow BDM communication (and therefore slow debugging and slow programming), while a clock frequency which changes frequently (as in the case of the user application modifying the FLL peripheral) may result in loss of BDM communication.

6.1.2 Fast Programming

The **"Enable Fast Programming"** parameter (available on some devices), if selected, speeds up programming by driving the microcontroller's internal PLL circuitry to the maximum settings.

6.1.3 Trimming

Some devices can have their internal oscillator calibrated (trimmed) through the **"VCO Bus Frequency"** parameter (please note that other devices may present different calibration parameters).

6.1.4 Other Settings

The **"TARGET POWER OUT Voltage"** parameter specifies the voltage provided by inDART-One on the "TARGET POWER OUT" connector, which can be used to power up the target board.

Note: *only the central pin of the "TARGET POWER OUT" connector is used, while the outer sleeve is not connected. The GND reference is taken from the "BDM" cable.*

The **"RESET Rise Time"** parameter specifies the time, in milliseconds, needed for the target RESET signal to go high. This value depends on the target system, and is used by inDART-One to generate the appropriate delay before to drive the BKGD line correctly in order to enter the special single chip mode. The default value of 20 ms is appropriate for most systems.

7 Working with RS08 Devices

7.1 Communication Settings

inDART-One must be configured properly so that BDM communication with the target device can be established correctly.

Communication settings are defined through the “Communication Settings” dialog box, available both in CodeWarrior and in the DataBlaze and MultiBlaze programming utilities.

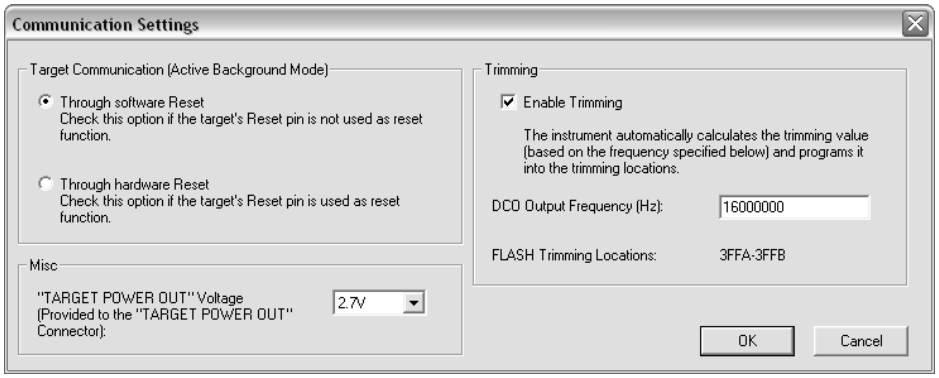

Figure 7.1: The BDM Communication Settings Dialog Box for RS08 Devices

Note: *this chapter describes communication settings with the only RS08 device so far released by Freescale (MC9RS08KA2). Other devices may present different configuration parameters.*

7.1.1 Target Communication

The **“Target Communication”** parameter specifies how to enter the target device’s Active Background Mode. Communication with the target device can be established via either a hardware or software Reset, depending on whether the Reset pin is used as Reset function or not.

7.1.2 Trimming

The target device's internal oscillator can be calibrated (trimmed) through the **"DCO Output Frequency"** parameter (please note that other devices may present different calibration parameters).

7.1.3 Other Settings

The **"TARGET POWER OUT Voltage"** parameter specifies the voltage provided by inDART-One on the "TARGET POWER OUT" connector, which can be used to power up the target board.

Note: *only the central pin of the "TARGET POWER OUT" connector is used, while the outer sleeve is not connected. The GND reference is taken from the "BDM" cable.*

8 Working with S12(X) Devices

8.1 Communication Settings

inDART-One must be configured properly so that BDM communication with the target device can be established correctly.

Communication settings are defined through the “Communication Settings” dialog box, available both in CodeWarrior and in the DataBlaze and MultiBlaze programming utilities.

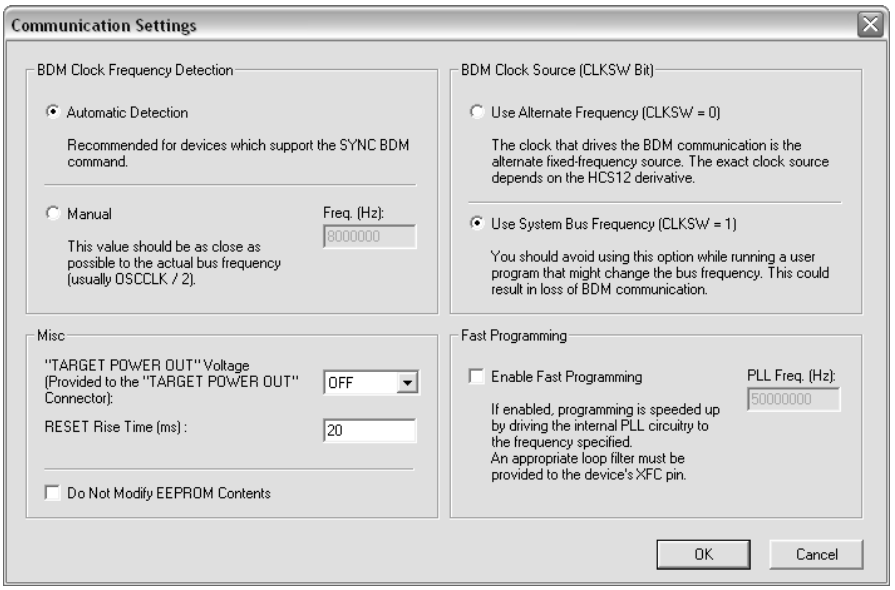

Figure 8.1: The BDM Communication Settings Dialog Box for S12(X) Devices

The first parameter, **“BDM Clock Frequency Detection”**, specifies how to detect the target microcontroller’s BDM clock frequency. There are two options:

- **“Automatic Detection”:** inDART-One automatically detects the target microcontroller’s BDM frequency. This setting is highly recommended for devices which support the SYNC BDM command, since in such

devices the frequency detection is totally accurate. For devices which do not support the SYNC command, the automatic frequency detection may not be accurate.

- **"Manual":** alternatively, you can manually specify the microcontroller's BDM frequency. Usually, this value is half that of the external oscillator frequency. Since this frequency is also used by the programming algorithms, it is important to carefully specify this parameter. Programming the device with an incorrect BDM frequency may result in stress to the FLASH/EEPROM memory cells, thus reducing the lifetime of the memory.

Another parameter, **"BDM Clock Source"**, specifies the link between the microcontroller's bus frequency and the BDM clock frequency during debugging. The BDM clock source can be selected by the CLKSW bit in the BDM Status register.

- If the CLKSW bit is set to 1, the BDM communication clock source is the microcontroller's bus frequency.
- If the CLKSW bit is set to 0, the BDM communication clock source is a constant clock source (not dependent on the bus frequency), which can vary depending on the specific S12 derivative. In the case of the MC9S12DP256, for example, this constant clock source is half the frequency of the external oscillator.

Which CLKSW setting to use depends on the target system and on the user application. The idea is to set the CLKSW bit so that the highest (and less subject to changes) clock frequency is used for the BDM communication. A low clock frequency will result in slow BDM communication (and therefore slow debugging), while a clock frequency which changes frequently (as in the case of the user application modifying the PLL peripheral) may result in loss of BDM communication.

The **"Enable Fast Programming"** parameter, if selected, speeds up programming by driving the microcontroller's internal PLL circuitry to the maximum settings. In order for this feature to work, an appropriate loop filter circuitry must be provided to the device's XFC pin.

The **"TARGET POWER OUT Voltage"** parameter specifies the voltage provided by inDART-One on the "TARGET POWER OUT" connector, which can be used to power up your target board.

Note: *only the central pin of the "TARGET POWER OUT" connector is used, while the outer sleeve is not connected. The GND reference is taken from the "BDM" cable.*

The **"RESET Rise Time"** parameter specifies the time, in milliseconds, needed for the target RESET signal to go high. This value depends on the target system, and is used by inDART-One to generate the appropriate delay before to drive the BKGD line correctly in order to enter the special single chip mode. The default value of 20 ms is appropriate for most systems.

9 inDART Programming Library

9.1 Introduction

This documentation deals with low-level interfacing between user written PC applications and the inDART-One In-Circuit Programmer/Debugger. This section assumes you have already read the previous sections of this user's manual and got acquainted with the instrument. All of the examples provided in this documentation are written in C, unless otherwise reported.

9.2 The inDART Programming Library (IPL)

The IPL-One Programming Library is a DLL which includes all of the low-level functions that allow you to set up the instrument and perform most of the programming commands and functions of the DataBlaze and MultiBlaze user interfaces from within your own application.

Dynamic-link libraries (DLL) are modules that contain functions and data. A DLL is loaded at run time by its calling modules (.exe or .dll). When a DLL is loaded, it is mapped into the address space of the calling process.

The inDART Programming Library contains C written routines, and can be used to interface the instrument from within, for example, a Microsoft Visual C or Visual Basic application, as well as any other programming language that supports the DLL mechanism. For details on how to call DLL functions from within your application, please refer to the your programming language's documentation.

9.3 Installation

Before to start working with the inDART Programming Library, you must set up your system with all the required files and drivers. The files to be installed are:

- The **"sftdrv01.sys"** file in "Windows\System32\Drivers";

- The **“udart01.inf”** file in “Windows\Inf”;
- The **“IPL-One.dll”**, **“BL-One.dll”**, **“PI-One.dll”** and all of the **“Drv*.dll”** files into your application’s directory.

These files are automatically installed by the inDART-One Utilities setup, as described in “inDART-One Utilities Setup” on page 28.

9.4 Programming Library Reference

9.4.1 Using the Interface Library Functions

When you control one or more inDART-One within your own application, you will typically follow the steps indicated below:

1. **Initialize the programming session.**
   The **`IPL_StartSession()`** function must be called prior to any other IPL function. This function detects all inDART-One instruments connected to the USB bus and determines whether fast programming algorithms are unlocked.
2. **Initialize the instrument(s).**
   To communicate with the target device you need to initialize the instrument(s) with target device information. Initialization functions include:

   - **`IPL_GetDeviceList()`**
   - **`IPL_SetDevice()`**
   - **`IPL_SetCommunicationSettings()`**
   - **`IPL_LoadFileIntoBuffer()`**

3. **Program.**
   Once the instrument(s) has/have been set up, you can begin programming, with the **`IPL_StartProgramming()`** function.
4. **Close the communication with the instrument(s).**
   This is done by the **`IPL_EndSession()`** function. Closing the

communication with the instrument(s) frees resources used during communication.

The figure below illustrates a typical working flow.

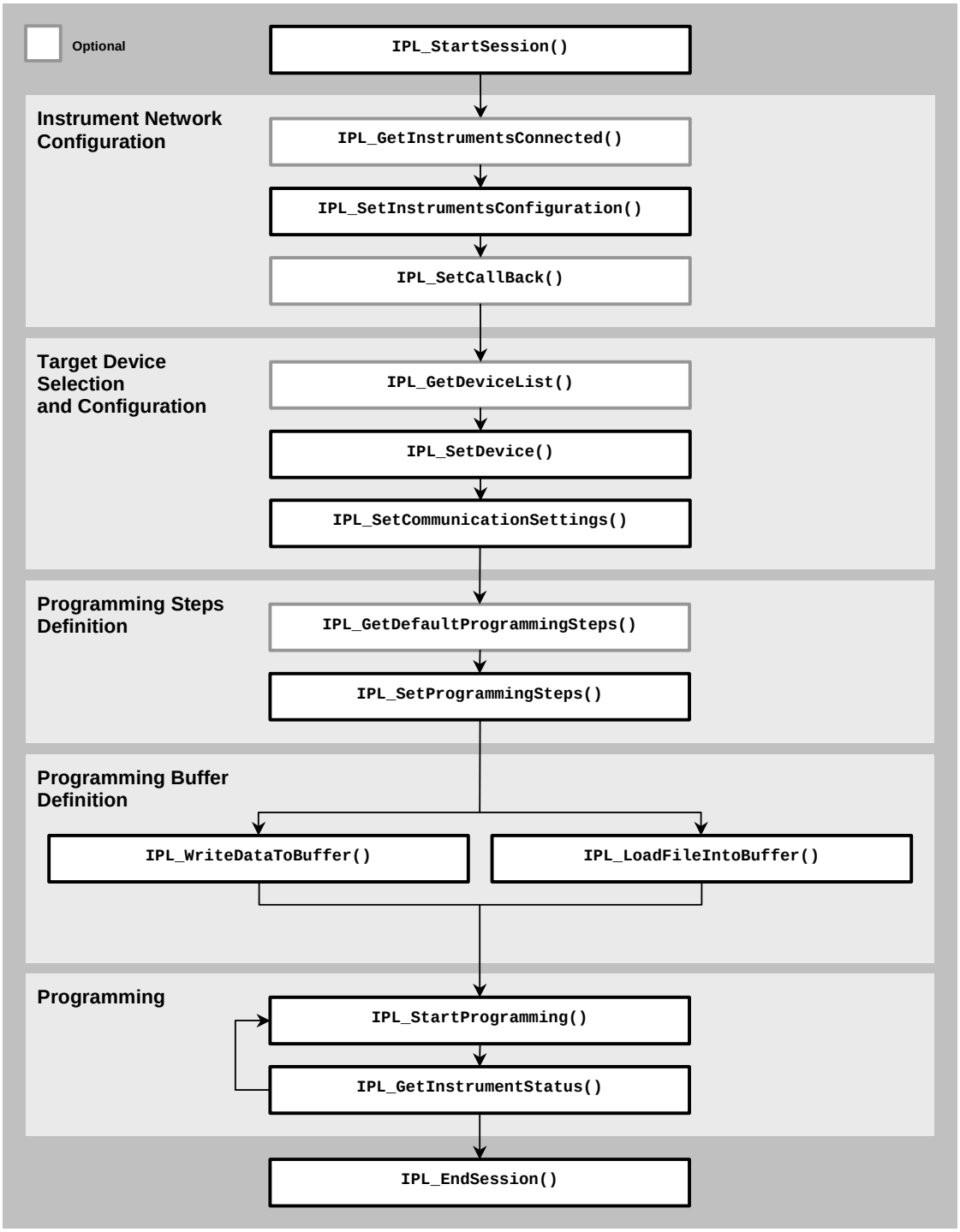

Figure 9.1: Typical IPL Workflow

9.4.2 Return Values of the Programming Library Functions

Most of the inDART Programming Library functions return a **BOOL** value which indicates whether the function was successfully executed (return value = **TRUE**) or not (return value = **FALSE**). In the latter case it is possible to get extended error information by calling the function **IPL_GetError**:

```
void IPL_GetError(char *error_string);
```

The **error_string** parameter will be filled with a text message explaining the cause of the problem.

9.4.3 Programming Buffer

A local (PC) programming buffer is automatically handled by the Programming Library. Between an **IPL_StartSession()** and **IPL_EndSession()** block, the Programming Library maintains a buffer which can be used to write data to/read data from the target device.

This buffer is filled via the **IPL_LoadFileIntoBuffer()**, **IPL_ReadDeviceMemoryIntoBuffer()** and **IPL_WriteDataToBuffer()** functions and is read via the **IPL_ReadDataFromBuffer()** function.

The **IPL_StartProgramming()** function uses the programming buffer contents to program the target device.

The programming buffer can be thought of as a one-to-one mapping of the target device memory into a PC memory area.

The figure below illustrates how the programming buffer is used.

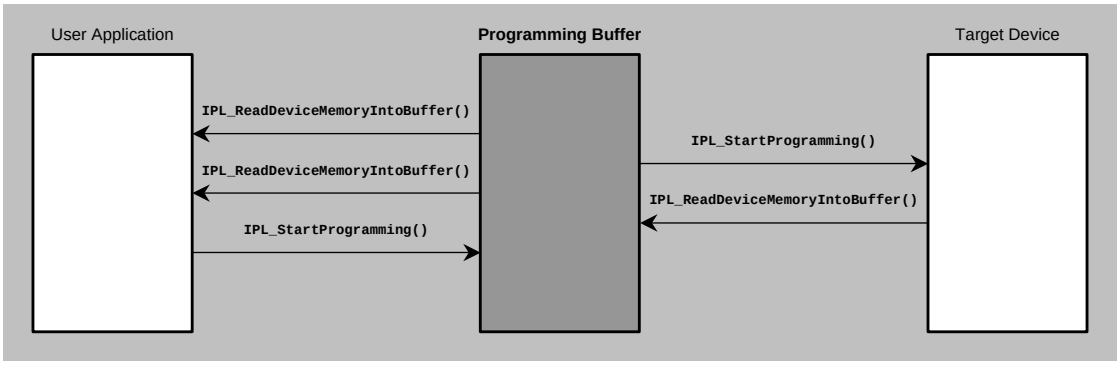

Figure 9.2: Programming Buffer

Note: *the programming buffer size is automatically set to the target device's memory size, after the* ***IPL_SetDevice()*** *function is called.*

9.5 Function Reference

Each Programming Library function is listed alphabetically and explained in the following pages.

9.5.1 Typedefs and Structures

Listed below are the various typedefs and structures used by the IPL-One functions.

```
typedef enum {
      IPL_FORMAT_BINARY                 = 0,
      IPL_FORMAT_IHEX                   = 1,
      IPL_FORMAT_S19                    = 2,
} IPL_FILE_FORMAT;
```

```
typedef enum {
      IPL_STEP_CODE_BLANK                = 0,
      IPL_STEP_CODE_ERASE                = 1,
      IPL_STEP_CODE_PROGRAM              = 2,
      IPL_STEP_CODE_VERIFY               = 3,
      IPL_STEP_DATA_BLANK                = 4,
      IPL_STEP_DATA_ERASE                = 5,
      IPL_STEP_DATA_PROGRAM              = 6,
      IPL_STEP_DATA_VERIFY               = 7,
      IPL_STEP_TRIM_PROG                 = 8,
      IPL_STEP_RUN                       = 9,
      IPL_STEP_CODE_READ                 = 10,
      IPL_STEP_DATA_READ                 = 11,
      IPL_STEP_NOT_SUPPORTED             = 12,
      IPL_STEP_TARGET_OFF                = 13,
} IPL_PROG_STEP;

typedef enum {
      IPL_STATUS_IDLE                    = 0,
      IPL_STATUS_WORKING                 = 1,
      IPL_STATUS_ERR_COMMUNICATION       = 2,
      IPL_STATUS_ERR_INTERNAL            = 3,
      IPL_STATUS_ERR_BLANKCHECK_DEVICE   = 4,
      IPL_STATUS_ERR_READ_DEVICE         = 5,
      IPL_STATUS_ERR_ERASE_DEVICE        = 6,
      IPL_STATUS_ERR_PROGRAM_DEVICE      = 7,
      IPL_STATUS_ERR_VERIFY_DEVICE       = 8,
      IPL_STATUS_ERR_PROTECTION_DEVICE   = 9,
      IPL_STATUS_ERR_INVALID_DEVICE      = 10
} IPL_INST_STATUS_MODE;
```

```
typedef enum {
      IPL_EVENT_PROG_START                = 0,
      IPL_EVENT_PROG_END                  = 1,
      IPL_EVENT_PROG_STEP                 = 2,
      IPL_EVENT_ERROR                     = 3,
} IPL_EVENT_TYPE;

typedef struct _IPL_DEVICE_LIST {
      char device_code[50];
      char family_code[20];
} IPL_DEVICE_LIST;

typedef struct _IPL_INST_INFO {
      unsigned int SN;
      BOOL driver_busy;
} IPL_INST_INFO;

typedef struct _IPL_INST_CONFIG {
      unsigned int SN;
      int ID;
} IPL_INST_CONFIG;

typedef struct _IPL_INST_STATUS {
      IPL_INST_STATUS_MODE mode;
      char message[MAXLEN_ERROR_MSG];
} IPL_INST_STATUS;

typedef struct _IPL_EVENT_PROG_STEP_DATA {
      IPL_PROG_STEP cmd;
      unsigned long perc;
} IPL_EVENT_PROG_STEP_DATA;
```

```
void (*IPL_CALLBACK)(
       int inst_ID,
       IPL_EVENT_TYPE event_type,
       void *event_data);
```

9.5.2 IPL_EndSession()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_EndSession(void);
```

Parameters:

None.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Ends a programming "session". All of the Programming Library functions (exception made for the **IPL_GetVersion()** function) must be called within a **IPL_StartSession()** and **IPL_EndSession()** function.

9.5.3 IPL_GetBufferChecksum()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_GetBufferChecksum(unsigned long *chk);
```

Parameters:

chk:	pointer to an unsigned integer that will receive the checksum of the programming buffer.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Calculates and returns the checksum of the programming buffer.

9.5.4 IPL_GetButtonStatus()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_GetButtonStatus(
        int inst_ID,
        BOOL *pressed,
        IPL_INST_STATUS *status);
```

Parameters:

inst_ID:	ID of the inDART-One instrument (previously set via the **IPL_SetInstrumentConfiguration()** function).
pressed:	**TRUE** if the "START" button is pressed, **FALSE** otherwise.
status:	pointer to an **IPL_INST_STATUS** structure that will receive instrument status information (see the **IPL_GetInstrumentStatus()** function).

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

This function reads the status of the "START" button of the specified inDART-One instruments. If the "START" button is pressed, the **pressed** parameter will be **TRUE**.

Note: *calling this function when the specified inDART-One instrument is busy programming will always result in the* ***pressed*** *parameter to be* ***FALSE****.*

9.5.5 IPL_GetDefaultProgrammingSteps()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_GetDefaultProgrammingSteps(
        IPL_PROG_STEP *prog_steps,
        int n_items,
        int *items_found);
```

Parameters:

prog_steps:	pointer to the array that will be filled with the default programming steps for the currently selected target device.
n_items:	number of programming steps to retrieve.
items_found:	the actual number of default programming steps available for the currently selected target device.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Fills an array (**prog_steps**) with the default programming steps for the currently selected target device. The function returns **n_items**. The **items_found** variable will contain the actual number of programming steps.

Tip: you can first call:

```
IPL_GetDefaultProgrammingSteps(NULL, 0,
       *no_of_prog_steps);
```

to get the number of programming steps, and then call:

```
IPL_GetDefaultProgrammingSteps(*tot_steps,
       no_of_prog_steps, NULL);
```

to retrieve them all.

9.5.6 IPL_GetDeviceList()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_GetDeviceList(
        IPL_DEVICE_LIST *device_list,
        int n_items,
        int *items_found);
```

Parameters:

device_list:	pointer to the array that will be filled with the device list.
n_items:	number of devices to retrieve.
items_found:	the actual number of devices present in the Programming Library's device list.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Fills an array (**device_list**) with devices supported by the IPL-One Programming Library. The function extracts **n_items** from the Programming Library device list. The **items_found** variable will contain the actual number of devices present in the Programming Library's device list.

Tip: you can first call:

```
IPL_GetDeviceList(NULL, 0, *no_of_devices);
```

to get the number of devices, and then call:

```
IPL_GetDeviceList(*tot_list, no_of_devices, NULL);
```

to retrieve them all.

9.5.7 IPL_GetError()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
void IPL_GetError(char *error_string);
```

Parameters:

`error_string`: this parameter will be filled with a text message explaining the cause of the problem.

Return value:

None.

Description:

Most of the Programming Library functions return a **`BOOL`** value which indicates whether the function was successfully executed (return value = **TRUE**) or not (return value = **FALSE**). In the latter case it is possible to get extended error information by calling the **`IPL_GetError()`** function.

Note: *this function must be called within an* ***`IPL_StartSession()`*** */* ***`IPL_EndSession()`*** *block.*

9.5.8 IPL_GetInstrumentsConnected()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_GetInstrumentsConnected(
        IPL_INST_INFO *inst_info,
        int n_items,
        int *items_found);
```

Parameters:

inst_info:	pointer to the array that will be filled with information about inDART-One instruments connected to the USB bus.
n_items:	number of instruments to retrieve.
items_found:	the actual number of instruments found.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Fills an array (**inst_info**) with information about inDART-One instruments connected to the USB bus. The function returns **n_items**. The **items_found** variable will contain the actual number of inDART-One instruments found.

Tip: you can first call:

```
IPL_GetInstrumentsConnected(NULL, 0, *no_of_inst);
```

to get the number of all of the inDART-One instruments connected to the USB bus, and then call:

```
IPL_GetInstrumentsConnected(*tot_inst, no_of_inst,
       NULL);
```

to retrieve information about all of them.

9.5.9 IPL_GetInstrumentStatus()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_GetInstrumentStatus(
        int inst_ID,
        IPL_INST_STATUS *status);
```

Parameters:

inst_ID:	ID of the inDART-One instrument (previously set via the **IPL_SetInstrumentConfiguration()** function).
status:	pointer to an **IPL_INST_STATUS** structure that will receive status information.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Returns status information for the specified instrument. This function is useful to retrieve the result of reading (**IPL_ReadDeviceMemory()** function) or programming (**IPL_StartProgramming()** function) operations.

9.5.10 IPL_GetVersion()

Include file:

```
#include “IPL-One.h”
```

Function prototype:

```
void IPL_GetVersion(char *ver);
```

Parameters:

ver: this parameter will be filled with a string containing the version of the IPL-One DLL and related DLLs.

Return value:

None.

Description:

Call this function to get version information about the IPL-One Programming Library and related DLLs.

9.5.11 IPL_LoadFileIntoBuffer()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_LoadFileIntoBuffer(
        const char *filename,
        IPL_FILE_FORMAT file_format,
        unsigned long file_offset,
        unsigned long buffer_offset);
```

Parameters:

filename: file to load into the programming buffer.

file_format: file format. Can be one of the following constants:

- **IPL_FORMAT_BINARY** (for binary files);
- **IPL_FORMAT_IHEX** (for Intel-Hex files);
- **IPL_FORMAT_S19** (for Motorola S-Record files).

file_offset: offset (in bytes) from the beginning of the file. That is, the first **file_offset** bytes read from the file are discarded. Valid only when **file_format** is **IPL_FORMAT_BINARY**.

buffer_offset: offset (in bytes) from the beginning of the programming buffer. That is, the programming buffer is filled starting from the **buffer_offset** address. Valid only when **file_format** is **IPL_FORMAT_BINARY**.

Return value:

TRUE: the function was successful.

FALSE:	an error occurred. Call the **`IPL_GetError()`** function to get extended error information.

Description:

Loads the contents of an external file into the programming buffer.

9.5.12 IPL_ReadDataFromBuffer()

Include file:

```
#include “IPL-One.h”
```

Function prototype:

```
BOOL IPL_ReadDataFromBuffer(
        unsigned long addr,
        unsigned char *dest_data,
        int len);
```

Parameters:

addr:	programming buffer start address.
dest_data:	pointer to a destination buffer which will contain the data copied from the programming buffer.
len:	number of bytes to copy.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Reads data from the programming buffer. Since the programming buffer is a one-to-one mapping of the target device memory, the **addr** parameter corresponds to a target device memory address.

9.5.13 IPL_ReadDeviceMemory()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_ReadDeviceMemory(int inst_ID);
```

Parameters:

inst_ID:	ID of the inDART-One instrument (previously set via the **IPL_SetInstrumentConfiguration()** function) to read from.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Reads the memory of target device connected to the **inst_ID** instrument and copies it into the PC programming buffer. The programming buffer can then be read with the **IPL_ReadDataFromBuffer()** function.

To check the status of the reading operation, call the **IPL_GetInstrumentStatus()** function.

9.5.14 IPL_SetCallback()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_SetCallback(IPL_CALLBACK callback);
```

Parameters:

`callback`: callback function.

Return value:

TRUE: the function was successful.

FALSE: an error occurred. Call the **`IPL_GetError()`** function to get extended error information.

Description:

Sets the callback function that will be called by the Programming Library each time a meaningful event occurs. The callback function prototype is:

```
void (*IPL_CALLBACK)(
        int inst_ID,
        IPL_EVENT_TYPE event_type,
        void *event_data);
```

where **`inst_ID`** is the instrument that generated the event, **`event_type`** is the type of event and **`event_data`** is additional information regarding the event (if available).

Possible events are summarized in the table below.

Event (`event_type`)	Description
`IPL_EVENT_PROG_START`	Occurs after the **`IPL_StartProgramming()`** function is called. No event information is passed to the callback function (**`event_data`** is **`NULL`**).
`IPL_EVENT_PROG_END`	Occurs at the end of programming, i.e. after all of the programming steps are performed. No event information is passed to the callback function (**`event_data`** is **`NULL`**).
`IPL_EVENT_PROG_STEP`	Repeatedly occurs when a programming step is being performed. Programming step's information (programming step type and completion percentage) is passed to the callback function (**`event_data`** is a pointer to a **`IPL_EVENT_PROG_STEP_DATA`** structure).
`IPL_EVENT_ERROR`	Occurs when there is a programming error. Call the **`IPL_GetInstrumentStatus()`** function to get error information. No event information is passed to the callback function (**`event_data`** is **`NULL`**).

Table 9.1: IPL Callback Events

Note: *in order to avoid recursion when handling events, do not call IPL functions that can cause other events to happen.*

9.5.15 IPL_SetCommunicationSettings()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_SetCommunicationSettings(
        char *settings,
        BOOL show_window);
```

Parameters:

settings:	pointer to a buffer containing the initialization string.
show_window:	if set to **TRUE**, opens the "Communication Settings" dialog box, where you can specify all of the target's communication settings. Upon exiting the "Communication Settings" dialog box, the **settings** parameter is filled with the information coming from the dialog box. If set to **FALSE**, the **settings** parameter must be manually specified.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Initializes inDART-One with target device information. This initialization procedure must be done before calling any other function that reads from/writes to the target device.

9.5.16 IPL_SetDevice()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_SetDevice(const char *device_code);
```

Parameters:

device_code: the string containing the complete device code.

Return value:

TRUE: the function was successful.

FALSE: an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Sets the target device to be programmed. To retrieve a list of supported devices, call the **IPL_GetDeviceList()** function.

9.5.17 IPL_SetInstrumentConfiguration()

Include file:

```
#include “IPL-One.h”
```

Function prototype:

```
BOOL IPL_SetInstrumentsConfiguration(
        IPL_INST_CONFIG *config,
        int n_items);
```

Parameters:

config:	pointer to the array that specifies instrument configuration.
n_items:	number of instruments to configure.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Configures one or more inDART-One instruments. The **config** parameter is a pointer to an array of **IPL_INST_CONFIG** structures which associate each instrument’s serial number to a logical ID.

Note: *this function can be called only once between an **IPL_StartSession()** and **IPL_EndSession()** block.*

9.5.18 IPL_SetProgrammingSteps()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_SetProgrammingSteps(
        IPL_PROG_STEP *prog_steps,
        int n_items);
```

Parameters:

prog_steps:	pointer to the array that specifies the programming steps.
n_items:	number of programming steps in the array.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Sets the programming steps to be performed when calling the **IPL_StartProgramming()** function. To retrieve the default programming steps available for the currently selected target device, call the **IPL_GetDefaultProgrammingSteps()** function.

Note: *programming steps will be performed in a fixed, pre-defined order, regardless of their order in the* ***prog_steps*** *array. E.g., a blank check operation will be always performed before an erase operation, a verify operation will be always performed after a program operation, etc.*

9.5.19 IPL_StartProgramming()

Include file:

```
#include "IPL-One.h"
```

Function prototype:

```
BOOL IPL_StartProgramming(int inst_ID);
```

Parameters:

inst_ID:	ID of the inDART-One instrument (previously set via the **IPL_SetInstrumentConfiguration()** function).

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Starts the programming in the specified instrument. The instrument will perform the programming steps specified with the **IPL_SetProgrammingSteps()** function.

To check the status of the programming, call the **IPL_GetInstrumentStatus()** function.

9.5.20 IPL_StartSession()

Include file:

```
#include “IPL-One.h”
```

Function prototype:

```
BOOL IPL_StartSession(void);
```

Parameters:

None.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Starts a programming “session”. All of the Programming Library functions (exception made for the **IPL_GetVersion()** function) must be called within an **IPL_StartSession()** and **IPL_EndSession()** function.

The **IPL_StartSession()** function detects all inDART-One instruments connected to the USB bus and determines whether fast programming algorithms are unlocked.

9.5.21 IPL_WriteDataToBuffer()

Include file:

```
#include “IPL-One.h”
```

Function prototype:

```
BOOL IPL_WriteDataToBuffer(
        unsigned long addr,
        unsigned char *src_data,
        int len);
```

Parameters:

addr:	programming buffer start address.
src_data:	pointer to a source buffer containing the data to be copied to the programming buffer.
len:	length (in bytes) of the source buffer.

Return value:

TRUE:	the function was successful.
FALSE:	an error occurred. Call the **IPL_GetError()** function to get extended error information.

Description:

Writes data to the programming buffer. The contents of the programming buffer will be programmed into the target device with the **IPL_StartProgramming()** function. Since the programming buffer is a one-to-one mapping of the target device memory, the **addr** parameter corresponds to a target device memory address.

10 Troubleshooting

10.1 Common Problems and Solutions

This section reports some common problems that may arise during general use. Please be aware, however, that working with a specific target device may cause device-specific issues.

10.1.1 USB Driver Problems

If you connected inDART-One to the PC before installing the inDART-One utilities, the instrument's USB driver may not have been correctly installed on your system. Unplugging and replugging the USB cable is of no use, since Windows has marked the device as "disabled". As a consequence, the PC cannot communicate with inDART-One.
To restore the USB driver (provided the inDART-One utilities have been installed), perform the following steps under Windows XP:

1. Connect inDART-One to the PC.
2. Open the Control Panel (**"Start > Settings > Control Panel"**).
3. Open the **"System"** options.
4. Select the **"Hardware"** tab.
5. Click the **"Device Manager"** button.
6. The **"inDART-One"** device will be shown with an exclamation mark next to it. Double click on this device.
7. In the **"General"** tab, click the **"Reinstall Driver"** button. Follow the on-screen instructions.

10.1.2 Communication Can't Be Established with inDART-One

1. Make sure that inDART-One is connected to the PC and powered on. inDART-One is powered by the USB connection.

2. Make sure that the target board is powered on and the target microcontroller is working. Programming and debugging rely on a MON08/BDM communication between inDART-One and the target board. This means that, in order to work correctly, the target microcontroller must be running. In particular, make sure that:

   - The MON08/BDM cable is connected to the target board.
   - All of the required MON08/BDM connector signals are correctly tied to the target microcontroller.

3. Make sure you are working with the correct inDART hardware model.

   a) To view/change the inDART hardware model in use in CodeWarrior, choose **"Connect"** (or **"Communication"**) from the **"inDART-HC08"**, **"SofTec-HCS08"**, **"SofTec-RS08"** or **"inDART-HCS12"** menu (depending on the target device currently selected) in the CodeWarrior debugger window.

   b) To view/change the inDART hardware model in the DataBlaze programming utility, select the **"Operations > Select Device"** menu item.

   c) To view/change inDART-One instruments in the MultiBlaze gang programming utility, choose the **"New Project"** or **"Modify Project"** options.

10.1.3 CodeWarrior-Specific: Stepping Execution is Slow

When the **"Memory"** window is open, step commands may execute slower, since the **"Memory"** window contents need to be refreshed after every step.

10.1.4 HC08-Specific: Peripheral Speed is Low

The **"Frequency divider"** parameter (in the **"Communication Settings"** dialog box) should always be set to 4 in order to guarantee that, in monitor mode, all of the target microcontroller's peripherals run at the same speed they do in user mode. If, however, you set this parameter to 2 not all peripherals will work at this doubled speed.

10.1.5 HCS08-Specific: Communication Lost During Debugging

This problem may have several causes:

1. The microcontroller's bus frequency has been changed by the user application and the CLKSW bit is set to 1. In this case, set the CLKSW bit to 0 (in the **"Communication Settings"** dialog box).
2. The BKGDPE bit in the SOPT (System Option Register) register has been set to 0, configuring the BKGD pin as a generic I/O pin. The BKGD pin must be reserved for the BDC communication (BKGDPE = 1).
3. A microcontroller reset has occurred. Among other things, this may be caused by the COP peripheral. After reset, the COP is enabled, so your program must either disable it or reset its timer.

10.1.6 HCS08-Specific: STOP Assembly Instruction Causes a Microcontroller Reset

If the STOPE bit in the SOPT register is not set, the STOP instruction is recognized as an illegal opcode, causing the microcontroller to reset. The SOPT register can be written only once after reset.

10.2 Diagnostic Test

inDART-One has built-in self-test capabilities. This means that you can verify by yourself, at any time, the correct operation of the instrument's hardware. To perform the diagnostic test:

1. Open the inDART-One Control Panel. Select **"Start > Programs > SofTec Microsystems > inDART-One > Control Panel"**. The following dialog box will appear.

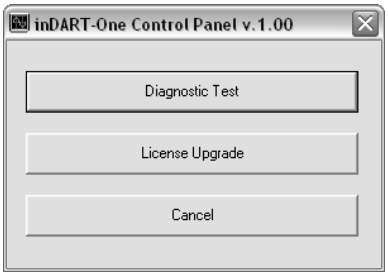

Figure 10.1: inDART-One Control Panel

2. Click the **"Diagnostic Test"** button and follow the on-screen instructions.

Note: *before to start the diagnostic test, make sure that only one instrument is connected to the PC and that no target system is connected to the instrument.*

10.3 Getting Technical Support

For technical assistance, documentation and information about products and services, please refer to your local SofTec Microsystems partner.

SofTec Microsystems offers its customers a free technical support service at *support@softecmicro.com*. Before getting in contact with us, we advise you to check that you are working with the latest version of the inDART-One system software (upgrades are available free of charge at *http://www.softecmicro.com*). Additional resources can be found on our online discussion forums (*http://www.softecmicro.com/forum*).

11 Technical Specifications

Parameter	Value
General	
Operating Voltage	5 V DC (provided by the USB bus)
Power Consumption	350 mA (max)
MON08 Section	
“TARGET POWER” connectors maximum accepted ratings (when “POWER OUT” = “POWER IN”)	42 V DC, 5 A
“TARGET POWER OUT” connector maximum ratings (when “POWER OUT” = VDD)	5.0 V DC, 200 mA 3.3 V DC, 100 mA 2.7 V DC, 100 mA 1.8 V DC, 100 mA
MON08 connector VDD signal maximum ratings (standard MON08 mode, pin 15)	5.0 V DC, 200 mA 3.3 V DC, 100 mA 2.7 V DC, 100 mA 1.8 V DC, 100 mA
MON08 connector OSC signal (standard MON08 mode, pin 13)	12 MHz, GND to V_{DD}, 50% Duty Cycle
BDM Section	
BDM connector VDD signal (pin 6)	1.8 V DC to 5.5 V DC, 5 mA max
BDM connector BKGD signal (pin 1)	50 MHz max at 5 V DC

Table 11.1: Electrical Specifications

Parameter	Value
Dimensions	140 x 102 x 40 mm
MON08 connector type	16-pin, 2.54-mm pitch, dual row header (male)
BDM connector type	6-pin, 2.54-mm pitch, dual row header (male)
Target power connectors type	Coaxial power jack, center pin (positive) = 2.1 mm, outer sleeve = 5.5 mm
Operating temperature	0°C to 50°C
Storage temperature	0°C to 70°C
Humidity	90% (without condensation)

Table 11.2: Physical and Environmental Specifications